Linux Mint и его Cinnamon

Очерки применителя

Алексей Alv Федорчук

Аннотация

Эта книга посвящена дистрибутиву Linux Mint и одной из его главных рабочих сред рабочих сред — десктопу Cinnamon. В ней рассмотрены их особенности, средства настройки дистрибутива и среды, системы управления пакетами, некоторые прикладные программы. Затронут также ряд более специальных вопросов, имеющих отношение не только к Linux Mint, такие, как командная оболочка Zsh, использование программных RAID, технологии LVM и системы размещения данных ZFS. Заключительным аккордом в ней прозвучит рассказ о создании собственных индивидуализированных сборок на базе Linux Mint и Cinnamon.

Книга не является систематическим руководством по данному дистрибутиву и, тем более, по Linux вообще. Она ориентирована исключительно на любопытствующих применителей любого уровня подготовки, имеющих склонность к экспериментам, с одной стороны, и к изящной словесности — с другой.

Об этой книге

В основу этой книги легли заметки, посвящённые дистрибутиву Linux Mint и десктопу Cinnamon, которые на протяжении последнего года размещались на нашем Блогосайте. В ней использованы также фрагменты цикла статей на ту же тему, публиковавшиеся на сайте IBM developerWorks (правда, цикл этот по не зависящим от меня обстоятельствам не был закончен). При подготовке книги эти статьи и заметки были систематизированы, структурированы, отредактированы и актуализированы в соответствии с текущими реалиями, а также дополнены новыми материалами.

Автор не ставил себе целью написать последовательное и законченное руководство по Linux Mint и Cinnamon. Это — именно очерки, посвящённые тем аспектам устройства и применения данного тандема, которые а) не очень подробно освещёны в имеющихся источниках и б) интересны лично автору. И вследствие последнего обстоятельства носят вполне субъективный характер.

Дистрибутив Mint, по моему мнению, пригоден для применителей любого уровня начальной подготовки. Соответственно и книга о нём не предполагает у читателя наличия каких-то особенных познаний, кроме самого обычного любопытства. А также понимания того, что это — именно очерки, а не сборник документации.

При этом читатель книги не обязан быть применителем именно Linux Mint. Дистрибутив этот является дериватом Ubuntu и сохраняет с ней практически полную бинарную совместимость. Так что ряд описанных в книге вещёй (например, про управление пакетами) — общие для всего семейства Ubuntu'идов, а отчасти и для всех deb based дистрибутивов, включая даже прародительский Debian. Кое-что же, скажем, включение поддержки LVM,

softRAID, ZFS, имеет силу для всех дистрибутивов Linux вообще.

Ну а сведения, содержащиеся в очерках про Cinnamon, приложимы к любому дистрибутиве, в котором этот десктоп может быть установлен, и в котором обеспечивается его корректная работа — список их постоянно расширяется, и нынче включает в себя и не только Linux-, но и BSD-системы.

Во время работы над книгой затронутые в ней вопросы обсуждались с моими товарищами и коллегами: Владимиром Поповым, Сергеем Голубевым, Владимиром Родионовым, Станиславом Шрамко, а также многочисленными участниками социальной сети Juick, форумов Linux Mint Росинка и Matuntu, для поимённого перечисления которых не хватит никаких ресурсов. Всем именованным и подразумеваемых лицам автор выражает свою признательность.

Отдельная благодарность — поэту-линуксоиду Ирине Киттель aka Алиса Деева, вдохновлявшей меня на трудовые свершения своими стихами, и моим детям — Ольге и Виктору, ставшими применителями Linux'а по собственной инициативе.

Наконец, автор искренне признателен команде разработчиков дистрибутива Mint и интегрированной среды Cinnamon, и лично Клементу Лефевру aka Clem за создание прекрасного дистрибутива и его рабочего окружения.

Примечание: книга в основных форматах для чтения в оффлайне доступна в Библиотеке Блогосайта. Для неё, как и всех остальных книг **Библиотеки**, разрешается свободное использование и некоммерческое распространение в электронной форме при условии указания авторства и активной индексируемой ссылки на первоисточник. Все книги **Библиотеки** доступны бесплатно, хотя автор не отказывается от аморального (то есть финансового) содействия своим проектам, оказать которое можно, заглянув на эту страницу.

Введение в Linux Mint

Разговор о Mint и его Cinnamon логично начать с рассказа о том, что же такое, товарищи, Mint, и что такое, братья, Cinnamon. Начну с этого и я.

Лирическое вступление

> Ведь в науке, в основном, происходят вещи посредственные — и я приучил себя к посредственному.
> *Владимир Савченко,* ***Открытие себя***

В долгой истории дистрибутивов Linux до сего дня было два знаковых события (если не считать самого факта начала Linux-дистрибуции). Был период «бури и натиска» — рубеж тысячелетий, когда, вслед за первой версией Mandrake (имевшей, как ни странно, номер 5.1), поднялась волна так называемых «дистрибутивов, дружественных к пользователю» (они же — «с человеческим лицом»). И был «год великого перелома» — 2005, когда Ubuntu 5.10 Breezy Badger, став пригодной для применения, в одночасье обрела невероятную популярность среди применителей всех стран и народов, заставив разработчиков всех остальных популярных дистрибутивов пересмотреть свою политику в отношении этих самых применителей.

С тех пор в мире Linux-дистрибуции происходили события более или менее

заурядные. Новостные ленты заполнились сообщениями: вышел релиз W.W дистрибутива Имя рек с ядром X.XX, KDE Y.Y.Y и GNOME Z.Z.Z, со списком прочих изменений — длинным, но мало чего дающим применителю. Конечно, каждый такой релиз содержал усовершенствования. Хотя в последние годы первое действие при знакомстве со многими из них было: посмотреть — а чего они на этот раз поломали?

Началось привыкание к заурядности происходящих событий. Настолько сильное, что, когда в последний день весны 2014 года, произошло событие действительно незаурядное — оно имело все шансы остаться практически незамеченным. А, между тем, по своей значимости оно вполне сопоставимо с выходом Mandrake 5.1 и Ubuntu 5.10. Это событие — релиз Mint 17 Qiana в сборке с рабочей средой Cinnamon 2.2. О которых и пойдёт речь в дальнейшем — но уже в актуализованном тандеме, появившемся в ноябре 2014 года и включающем Mint 17.1 Rebecca и Cinnamon 2.4.

Что такое Mint

Полное имя первого из героев этого цикла — Linux Mint. Однако, поскольку ясно, о дистрибутиве какой операционной системы идёт речь, первый компонент я в дальнейшем буду опускать.

Дистрибутив Mint, начиная с 2011 года, стабильно занимает первое место в рейтинге Distrowatch. Конечно, это не значит, что он является самым распространённым или самым популярным — рейтинг этот, как и все подобные измерители… животов, вещь достаточно условная. Но безусловно свидетельствует о широкой известности дистрибутива в узких кругах применителей Linux. Так что я могу ограничиться очень краткой его характеристикой.

Дистрибутив Mint был создан в 2006 году Клементом Лефевром (Clement Lefebvre), который поставил своей целью создание идеального десктопа «для народа» — домашних пользователей и малого бизнеса. Он представлял собой дериват Ubuntu — термины клон или форк в данном случае не применимы. То есть Mint в базовой своей части, вплоть до Xorg, основан на кодовой базе Ubuntu, и все соответствующие пакеты берутся из её репозиториев без всяких изменений. Однако он имеет и собственный небольшой репозиторий (около 500 пакетов), содержащий дистрибутив-специфические компоненты.



Клемент Лефевр

В сентябре 2010 года было объявлено о выходе другого дистрибутива проекта Mint — Linux Mint Debian Edition (LMDE). Как можно догадаться из его имени, он был основан на кодовой базе не Ubuntu, а Debian. В качестве таковой выступала его ветка testing, и потому релиз-цикла у LMDE нет — его «плавающие» версии маркировались годом и месяцем. Впрочем, в этом цикле речи о них не будет — этот дистрибутив заслуживает отдельного рассказа, время для которого ещё не наступило.

Немного истории

Завязка сюжета относится к 2011 году. До этого момента в качестве рабочего окружения в Mint использовался GNOME текущей версии — той же, что в базовой Ubuntu. Правда, GNOME был в нём главным, но не единственным десктопом. Чуть ли не со дня основания Mint существовала и его сборка с KDE, позднее к ней присоединились варианты с рабочими средами Xfce (2007 год) и LXDE (2010 год), на протяжении 2008-2010 годов существовал даже вариант с оконным менеджером Fluxbox. Однако они имели не вполне официальный статус, и появлялись, как правило, несколько позже сборок «генеральной линии». И иногда пропадали с горизонта вообще, как случилось с редакциями LXDE и Fluxbox.

Но весной 2011 года, с одной стороны, Ubuntu переходит на среду Unity, с другой — появляется релиз GNOME 3 с оболочкой GNOME Shell, поддержка же GNOME 2 разработчиками этого десктопа официально прекращается.

Оба новых десктопа, по ряду причин, оказались для разработчиков Mint неприемлемыми. И потому они, с одной стороны, включили в свой дистрибутив десктоп MATE, а с другой — занялись разработкой новой оболочки для GNOME 3, которая в декабре 2011 года была анонсирована под именем Cinnamon. И это — второй герой настоящего цикла.

Возвращаясь к современности

Однако надо возвратиться к первому герою и обрисовать современное положение дел. Вышедший 31 мая релиз 17 Qiana был представлен, как уже

говорилось, основными десктопными редакциями с Cinnamon и MATE в качестве рабочих сред, в сборках для 32- 64-разрядных архитектур.

Постепенно к ним присоединялись другие варианты дистрибутива. Так, обе базовые редакции получили так называемые образы nocodecs и oem. Как нетрудно догадаться, первые предназначены для стран, признающих патенты на алгоритмы, вторые — для предустановки на новые компьютеры. А затем к базовым редакциям присоединились редакции с рабочими средами KDE и Xfce.

С появлением в ноябре 2014 года релиза 17.1 Rebecca перечисленные образы сменились одноимёнными редакциями с актуальными на данный момент рабочими средами Cinnamon и MATE, к которым опять-таки чуть позже присоединились KDE- и Xfce-сборки. Выход же второй версии дистрибутива LMDE запланирован на март 2015 года.

Все перечисленные варианты представляют Live-образы, которые могут быть записаны либо на DVD-диски (объём их 1,3-1,4 ГБ), либо на твердотельные носители типа USB Flash или SD-карт. Во втором случае это можно сделать и специализированными утилитами типа UNetbootin, и прямой командой dd. Программа инсталляции запускается из Live-режима любого диска, альтернативных, то есть «чисто установочных», вариантов ни для одной редакции Mint не предусмотрено.

Скачать образы можно по ссылкам с официального сайта проекта, где приведён список многочисленных зеркал, физически находящихся в разных странах. В российских условиях целесообразно обратиться на зеркало Яндекса.

На этом мы временно попрощаемся с дистрибутивом Mint. Следующий очерк будет посвящён общему знакомству со второй героиней нашего повествования — одной из двух основных его рабочих сред, которая носит имя Cinnamon.

Введение в Cinnamon

Здесь будет рассказано о интегрированной рабочей среде Cinnamon — её истории, особенностях, распространении и поддержке в других дистрибутивах.

История

Cinnamon — самая молодая из «уже действующих» интегрированных рабочих сред (иначе — декстопов): проект был анонсирован 20 декабря 2011 года, а уже 23 декабря он стал доступен для скачивания, и сразу в виде релиза 1.1.2 — версии с меньшими номерами предназначались только для тестирования.

Далее развитие проекта происходило стремительно: 23 января следующего года появляется релиз 1.3, в середине марта — 1.4, а затем, в сентябре — релиз 1.6. После чего устанавливается полугодовой релиз-цикл — релиз 1.8 выходит в свет 5 мая 2013 года, после серии релизов корректирующих. В октябре того же года появляется релиз 2.0, в апреле 2014 года — релиз 2.2. И, наконец, герой нашего рассказа, релиз 2.4, увидел свет 1 ноября 2014 года.

Все релизы среды опережали версии Mint, для которых они предназначались, примерно на месяц — для дополнительного тестирования среды силами энтузиастов и притирки её к целевому дистрибутиву. Что, как показала практика, давало весьма положительный результат. О чём могу свидетельствовать по собственному опыту для версий Cinnamon 2.2 и 2.4: в релизы Mint 17 и 17.1, соответственно, они были включены в существенно доработанном виде по сравнению с первоначально представленными сборками.

Смена версий Cinnamon отражает специфичность его судьбы. Что же происходило при этом? В предыдущем очерке упоминалось, что история этого десктопа началась с появлением GNOME 3. Говорить о кипении страстей, связанных с этим событием, здесь не уместно. Достаточно сказать, что для многих применителей ряда дистрибутивов, включавших GNOME 2 в качестве штатного десктопа, его «осовремененная» версия, в частности, «очень прогрессивная» оболочка GNOME Shell, оказалась неприемлемой.

В частности, Mint был очень крепко связан с десктопом GNOME 2. Однако GNOME 3, особенно в своём первозданном виде, в концепцию его развития не вписывался, а основа его предшественника, библиотека Gtk 2, перестала поддерживаться разработчиками. Ситуация требовала кардинального решения.

Первое решение носило косметический характер. Это был набор MGSE (Mint GNOME Shell Extensions), объединяющий дополнения к GNOME Shell, которые могли обеспечить не только его традиционный интерфейс, но и восполнить недостающий функционал за счёт внешних модулей, таких, как панель Bottompanel, система переключения между окнами Windowlist и меню приложений Menu. Результатом стал выход в ноябре 2011 года релиза Mint 12 Lisa, включавшего в качестве десктопа по умолчанию GNOME 3 с MGSE.

Однако, видимо, майнтайнерам Mint изначально было ясно, что MGSE — не более, чем паллиатив, и потому, с одной стороны, включили в свой дистрибутив альтернативный десктоп — MATE (первыми среди майнтайнеров распространённых дистрибутивов). А с другой стороны, можно догадаться, что где-то за кадром Клемент Лефевр (Clement Lefebvre), основатель и основной майнтайнер дистрибутива, уже ковал основу совершенно новой оболочки для GNOME 3. Которая стала доступной буквально через месяц после выхода Mint 12 Lisa и получила имя Cinnamon.

Отступление. Словом cinnamon, восходящим к латыни, в английском языке называют, во-первых, коричное дерево, коричник цейлонский (Cinnamomum zeylanicum) — вечнозелёное растение семейства лавровых. Второе его значение — корица, название пряности, изготовляемой из коры этого дерева. Наконец, слово cinnamon применяется и для именования коричного масла, добываемого из листьев коричника, и используемого в медицине, парфюмерии и косметике.

Основу Cinnamon'а составил оконный менеджер Muffin — форк аналогичной программы Mutter из GNOME 3. Главное отличие новой оболочки от связки GNOME 3 и MGSE состояло в том, функционал внешних расширений последнего был включён непосредственно в её состав. Это предоставило средства управления взаимодействием между дополнительными функциями и определения порядка их загрузки. В результате были реализованы добавление пиктограмм в область уведомлений, система уведомлений в стиле GNOME 2, возможность изменения позиции панели и и её автоматического

скрытия.

После серии основных и корректирующих релизов, стремительно следующих друг за другим, Cinnamon 1.4 UP1, появившийся 14 мая 2012 года, был включён в качестве штатного десктопа в Mint 13 Maya, анонсированный десять дней спустя. С тех пор выход его версий и стал привязан к релиз-циклу этого дистрибутива.

Всё это время Cinnamon представлял собой просто оболочку к GNOME 3, надстраивающую «форкнутый» менеджер окон и замещающую собой его штатный GNOME Shell. Он включал все базовые приложения GNOME 3 — терминал, файловый менеджер, текстовый редактор — в неизменном виде. Однако во время подготовки релиза GNOME 3.6, в котором предполагалось существенное ограничение функционала файлового менеджера Nautilus, разработчики Cinnamon начали работы над форком его версии 3.4, назвав её Nemo. Который и попал в релиз Mint 14 Nadia, хотя сначала в качестве альтернативного. Но уже в версии Cinnamon 1.8.X он был интегрирован с этой средой. Кроме того, в этой версии отказались от Центра управления GNOME 3.

Версии Cinnamon 1.8 суждено было стать последним «чистым» форком GNOME 3 — в это же время полным ходом шла подготовка релиза 2.0. Суть её заключалась в полной замене базовых компонентов GNOME 3 собственными аналогами. То есть — в создании полностью обособленного окружения, не пересекающегося с GNOME 3 и не связанного с ним внешними зависимостями. В результате чего Cinnamon из оболочки для GNOME, вроде GNOME Shell и Unity, превращался в полноценное рабочее окружение. Итог этой деятельности был вынесен на суд общественности 10 октября 2013 года, в виде релиза 2.0.

Особенности

Так каковы же особенности нового десктопа, приобретённые им в версии 2.0 и получившие дальнейшее развитие в версиях последующих? Важнейших — три.

Первая — в Cinnamon гармонично сочетаются старые добрые элементы управления, такие, как главное меню в стиле кнопки Пуск, и элементы модерна, столь привлекающие в Unity, такие, как строка поиска, подобная Dash — но без излишеств последнего, то есть без средств поиска в Интернете всякого рода «парнухи».

Вторая особенность — достигнутая в Cinnamon гармония между простотой конфигурирования и богатством возможностей последнего. Если настройки в KDE, при их изобилии, приобретают всё более необозримый вид, а в GNOME 3, напротив разубоживаются, в нашем десктопе они почти столь же просты, как в Xfce, и почти столь же изобильны, как в KDE. И, в отличие от Ubuntu, выполняются исключительно штатными средствами, а не бесчисленными твикерами с полуофициальным и вообще неофициальным статусом.

Третья особенность — аскетизм Cinnamon в отношении штатных приложений. В существующем виде к таковым можно отнести только файловый менеджер Nemo. Обычно богатство приложений считается достоинством интегрированных сред (на то они и зовутся интегрированными). И бедность Cinnamon в этом плане можно было бы отнести к числу её недостатков. Если бы не оборотная сторона медали — отсутствие приложений в штате среды позволяет легко и без избыточности подобрать оптимальный

набор рабочих инструментов «для себя».

Наконец, весь традиционализм Cinnamon'а покоится на весьма современном базисе в виде библиотек Gtk+ 3, что должно обеспечить спокойное развитие этого десктопа в обозримом будущем. При этом сохраняется и совместимость с приложениями на основе Gtk+ 2, до сих пор широко распространёнными и не имеющими адекватных аналогов.

Распространение

Тем не менее, несмотря на многочисленные достоинства, десктоп Cinnamon долго не получал широкого распространения в дистрибутивах Linux. И после знакомства с его историей легко понять, почему. В сущности, модель разработки его оказалась противоположно направленной по сравнению со всеми остальными интегрированными средами. Если KDE, Xfce, GNOME, а позднее LXDE и Razot-qt создавались командами разработчиков, более или менее независимыми от майнтайнеров отдельных дистрибутивов, и лишь потом начинали использоваться последними в своих сборках, если MATE представлял собой попытку сохранить наработки GNOME 2, то Cinnamon с первых дней своего существования выглядел «привязанным» к прародительскому Mint. Почти так же, как это имеет место для Ubuntu и Unity — или, по крайней мере, как это воспринимается для последней пары так называемой общественностью.

На самом деле эта «привязка» была кажущейся, хотя команды разработчиков Mint и Cinnamon действительно были множествами сильно пересекающимися. И политика разработчиков этого десктопа не требует от сторонних разработчиков, скажем, передачи им имущественных прав на свою продукцию, как это практикует фирма Canonical при приёме патчей для Ubuntu и Unity. Однако можно предполагать, что майнтайнеры большинства дистрибутивов отнеслись к Cinnamon настороженно. Тем не менее, некоторые из них поддержали нашу героиню.

Поддержка

Если через поиск Distrowatch попытаться найти дистрибутивы, поддерживающие Cinnamon, получится список из 16, на момент сочинения этих строк, позиций. Он не вполне соответствует действительности — с одной стороны, на официальных сайтах некоторых проектов явных упоминаний о поддержке этого десктопа не обнаруживается, с другой — некоторые дистрибутивы, в которых он есть заведомо (например, openSUSE), в нём отсутствуют. Однако, с учётом этого и с исключением явной экзотики из Южной Африки, Андалузии или Непала, оказывается, что Cinnamon поддерживается в десятке распространённых дисрибутивов, среди которых:

- Fedora и её клон — Korora;
- Sabayon — дружественный к пользователю клон Gentoo;
- Archlinux и его клон Manjaro;
- openSUSE, где он присутствует в полуофициальном (Semi official) репозитории;

- siduction — дистрибутив, основанный на unstable-ветке Debian;

- PC-BSD, которая, конечно, не Linux, но Cinnamon поддерживает на стадии установки.

Я перечислил только те дистрибутивы, в которых поддержка Cinnamon может быть задействована на стадии их установки, и реализацию её проверял лично. Кроме того, Cinnamon нынче можно найти в портах FreeBSD и пакетах DragonFly, в тестовоой ветке Debian, а главное — в ppa-репозиториях Ubuntu. До недавнего времени в последних она поддерживалась Гвендалем Ле Бианном (Gwendal Le Bihan). И, хотя весной 2014 года он отказался от сборки стабильной ветки этой среды, ограничившись экспериментальными «ночными», эстафета была немедленно подхвачена другими майнтайнерами.

Как я уже сказал, большая часть перечисленных выше систем проверялась мной на предмет поддержки Cinnamon собственноручно. И этот личный опыт показывал, что до недавнего времени качество таковой у всех представителей списка (кроме Ubuntu) оставляло желать лучшего. Это касалось мелких, но часто существенных для применителях деталей, таких, как настройка раскладок клавиатуры и их переключателей.

Однако буквально в последние месяцы, после выхода Cinnamon 2.4, ситуация изменилась кардинальным образом. И теперь этот десктоп безукоризненно поддерживается в большинстве дистрибутивов первого эшелона — в Archlinux'e и Manjaro, в openSUSE и Fedora, по прежнему хорошо — в Ubuntu, с оговорками на счёт отставания версии — в Debian testing (опять же, говорю только о тех, в которых проверял сам).

Однако по прежнему наиболее эффективно Cinnamon поддерживается в Mint — за счёт интеграции его с дистрибутив-специфическим системным инструментарием последнего. Подобно тому, как испокон веков KDE было лучше всего интегрировано с openSUSE, GNOME органично срастался с Fedora, а Xfce был «роднее» лёгким клонам Slackware (таким, как Zenwalk и Salix). Так что самый простой способ ознакомиться со средой Cinnamon во всёй её красе — установка соответствующей редакции Mint 17.1 Rebecca, о чём и пойдёт речь в следующем очерке.

Linux Mint: установка

Сравнение инсталлятора дистрибутива с театральной вешалкой настолько старо, что никто уже не помнит автора (вашему покорному слуге почему-то кажется, что это был именно он). И за минувшие пятнадцать лет оно не столько даже затёрлось, сколько потеряло силу. В наши дни любой дистрибутив, претендующий на внимание так называемых конечных пользователей, располагает удобной, более или менее функциональной программой установки самого себя, иногда даже красивой. Тем более что удачные решения в этой области имеют обыкновение расползаться по всяким клонам и ремиксам.

Именно так случилось некогда с инсталлятором Ubuntu — десять лет назад его установка в пять кликов сыграла не последнюю роль и в распространении этого дистрибутива по пользовательским десктопам, и в образовании многочисленных прямых, косвенных и откровенно примазавшихся родственников, вплоть до «дистрибутивов с нескучными обоями».

В числе родственников... нет, не примазавшихся, а настоящих, но пошедших другим путём, был и дистрибутив Mint. Сейчас не время обсуждать его взаимоотношения с прародительской Ububtu, но программу установки он унаследовал от неё практически без изменений. По крайней мере, до недавнего времени макроскопических различий в инсталляторах этих систем не наблюдалось.

Честно говоря, их не наблюдается и сейчас, с выходом версии Mint 17, а затем и Mint 17.1 — формально отличий в установке с Ubuntu 14.04 нет. Так что о чём тут говорить, спросите вы меня? Отвечу: говорить действительно не о чем — это нужно видеть. Точнее, делать своими руками — и наблюдать за результатом. Ибо даже скриншоты, которыми я на этой странице практически и ограничусь, не в силах передать того ощущения лёгкости и плавности, которое испытывает истинный применитель при установке этого дистрибутива.

Итак, установка Mint'а запускается из Live-режима его работы, являющегося следствием загрузки с соответствующего носителя — DVD или, что предпочтительно, флешки (или SD-карты). Кстати, не обязательно слушать того, кто будет убеждать вас закатать iso-образ на твердотельный носитель с помощью всяких специальных утилит. С этой задачей прекрасно справляется такая команда:

```
# dd if=path3/linuxmint-17.1-cinnamon-64bit.iso

of=/dev/sd? bs=8M
```

Символ решётки тут символизирует, что она должна быть дана от имени администратора (то есть в Ubuntu и её клонах — в форме

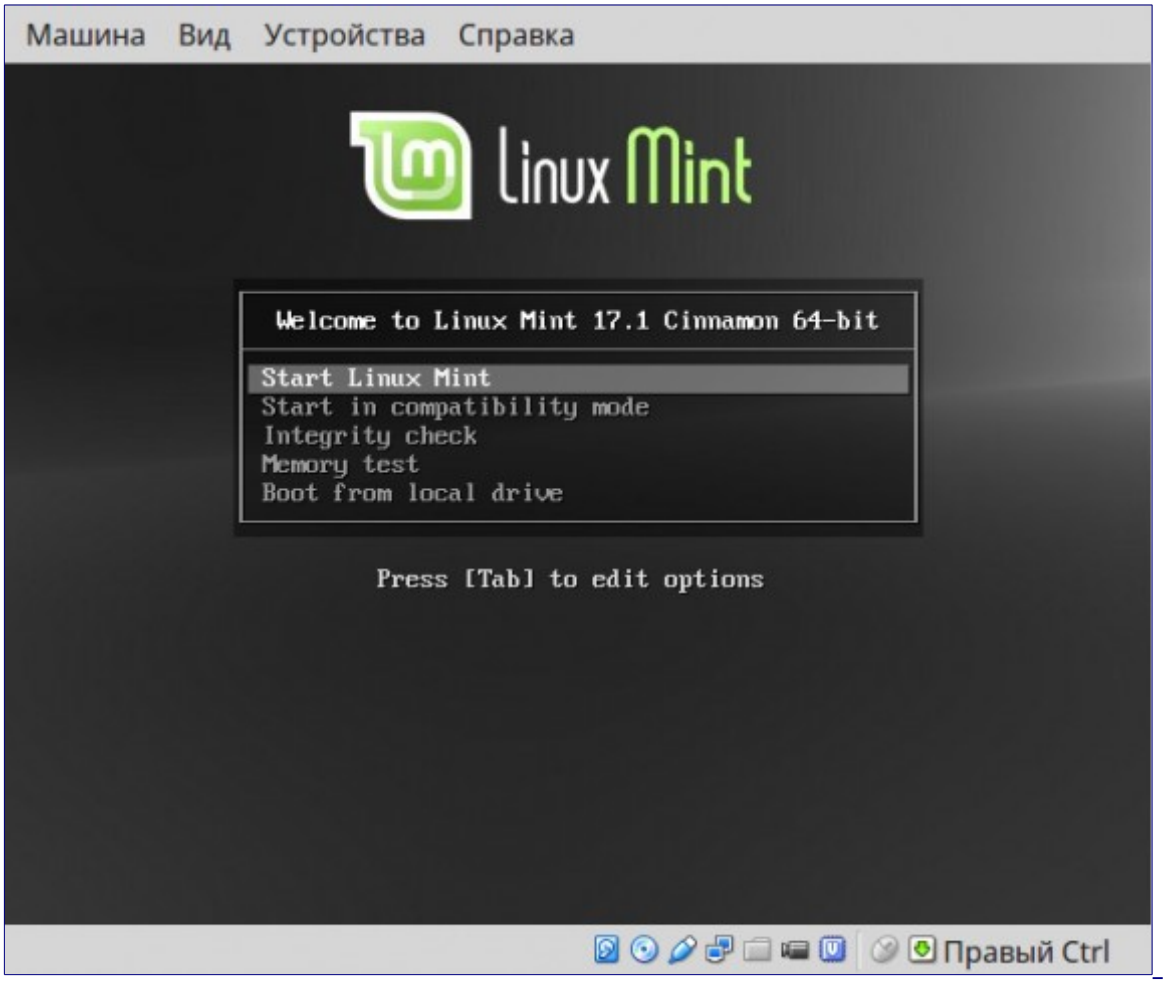
```
$ sudo dd...
```

и так далее. Имя входящего файла образа выбрано так потому, что дальнейший рассказ у меня пойдёт исключительно о Cinnamon-редакции соответствующего релиза. Хотя, надо отметить, что установка MATE- и Xfce-редакций ничем не отличается, только инсталлятор редакции с KDE имеет некоторую специфику,

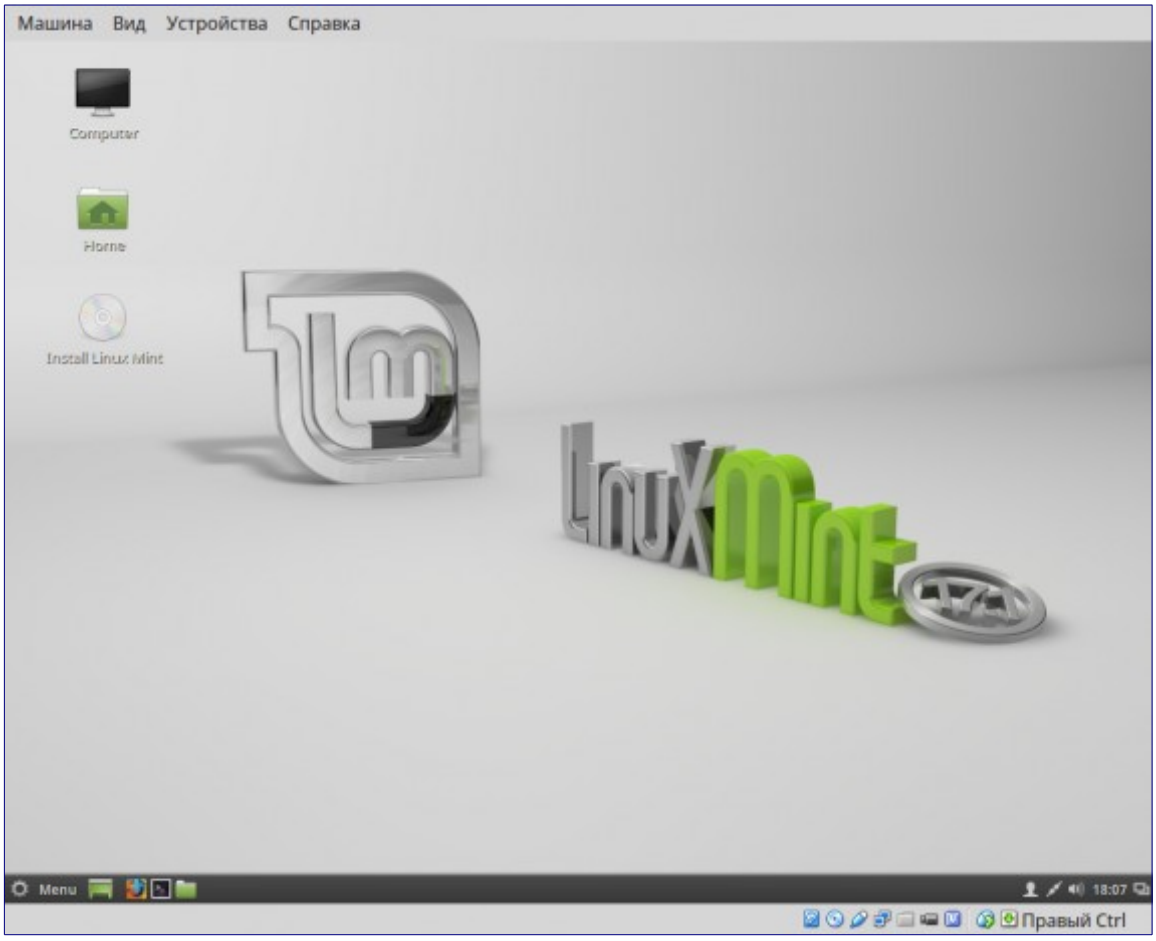
Вместо знака вопроса следует подставить литеру своего твердотельного устройства — подчёркиваю, именно устройства целиком, а не раздела на нём. Ну а значение bs (размер блока записи) взято с потолка: если не задать этот параметр, запись будет идти блоками по 512 байт, и это будет очень медленно и печально.

Конечно, использование для записи iso-образа на флешку/карточку всякого рода графических утилит тоже не возбраняется. Важно только помнить, что многие из них — дистрибутив-специфичны. Этому вопросу на Блогосайте посвящена специальная статья: Дистро в твёрдом теле.

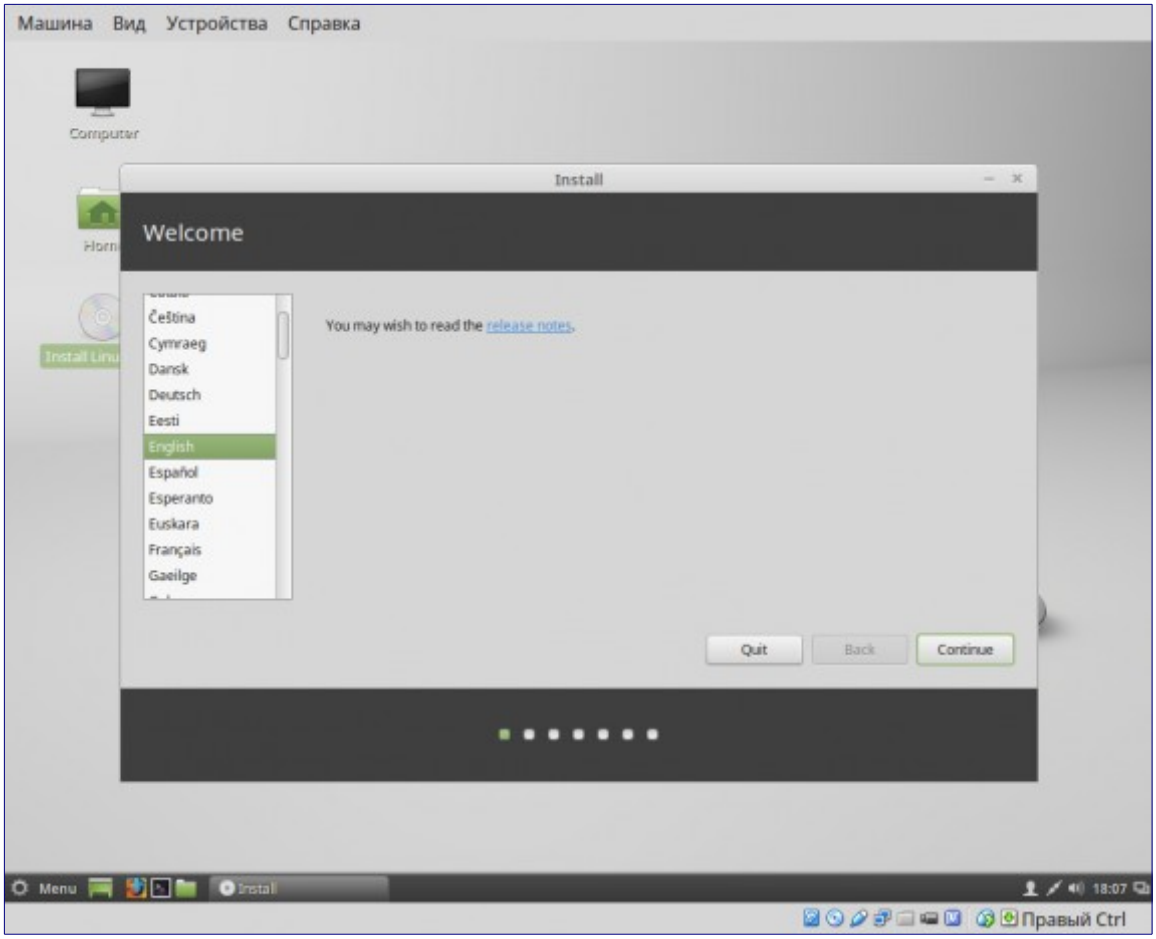
Так или иначе, но флешка будет записана и с неё произойдёт загрузка системы — автоматически, если в ходе её ничего не делать, или, если в течении 10 секунд нажать любую клавишу, из такого меню:



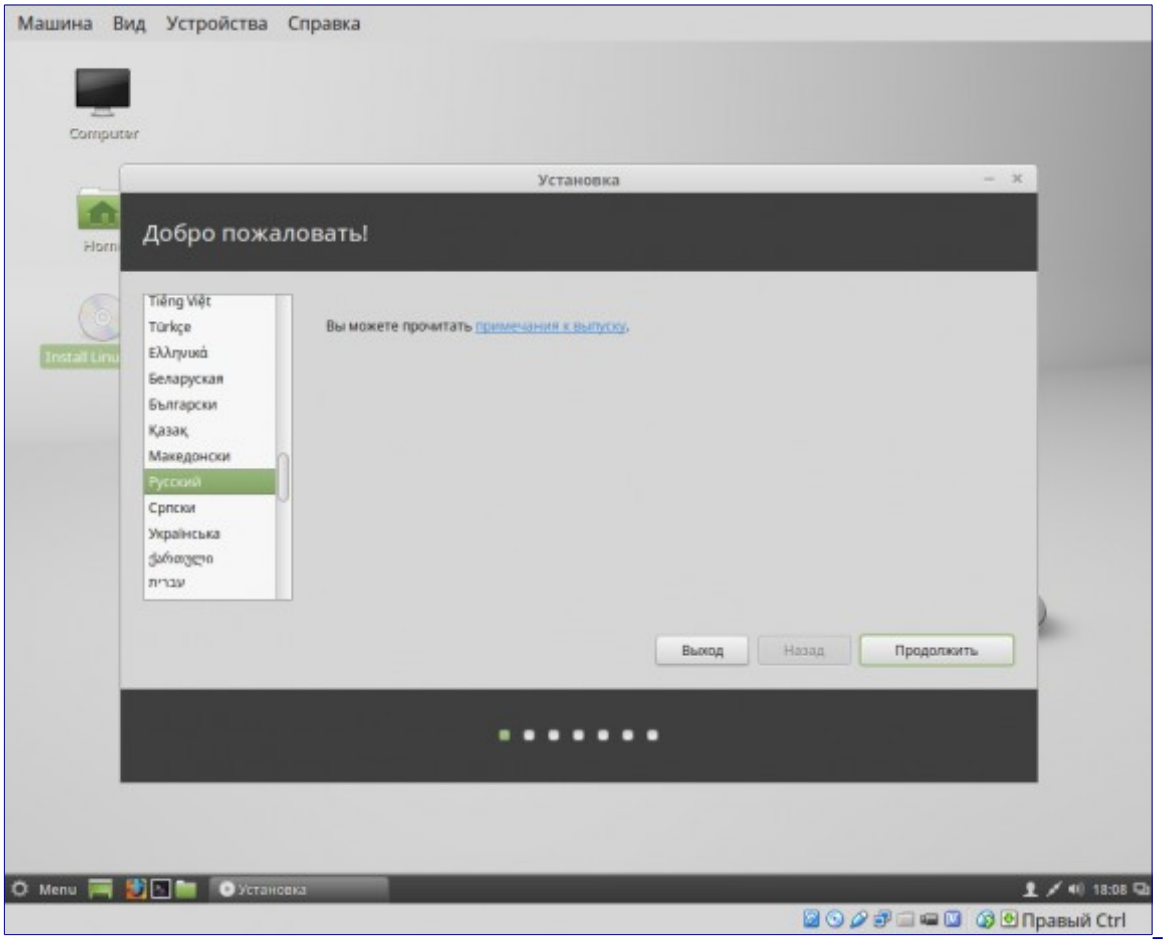
После чего мы оказываемся в live-среде Cinnamon — пора бросить на неё взгляд, пока беглый:



Вдаваться в её рассмотрение сейчас не стоит — для этого у нас будет много времени после установки, к которой и приступаем. Как нетрудно догадаться, запуск инсталлятора осуществляется щелчком на пиктограмме с соответствующей подписью — Install Linux Mint. И начинается процесс с предложения выбрать язык установки — по умолчанию английский:

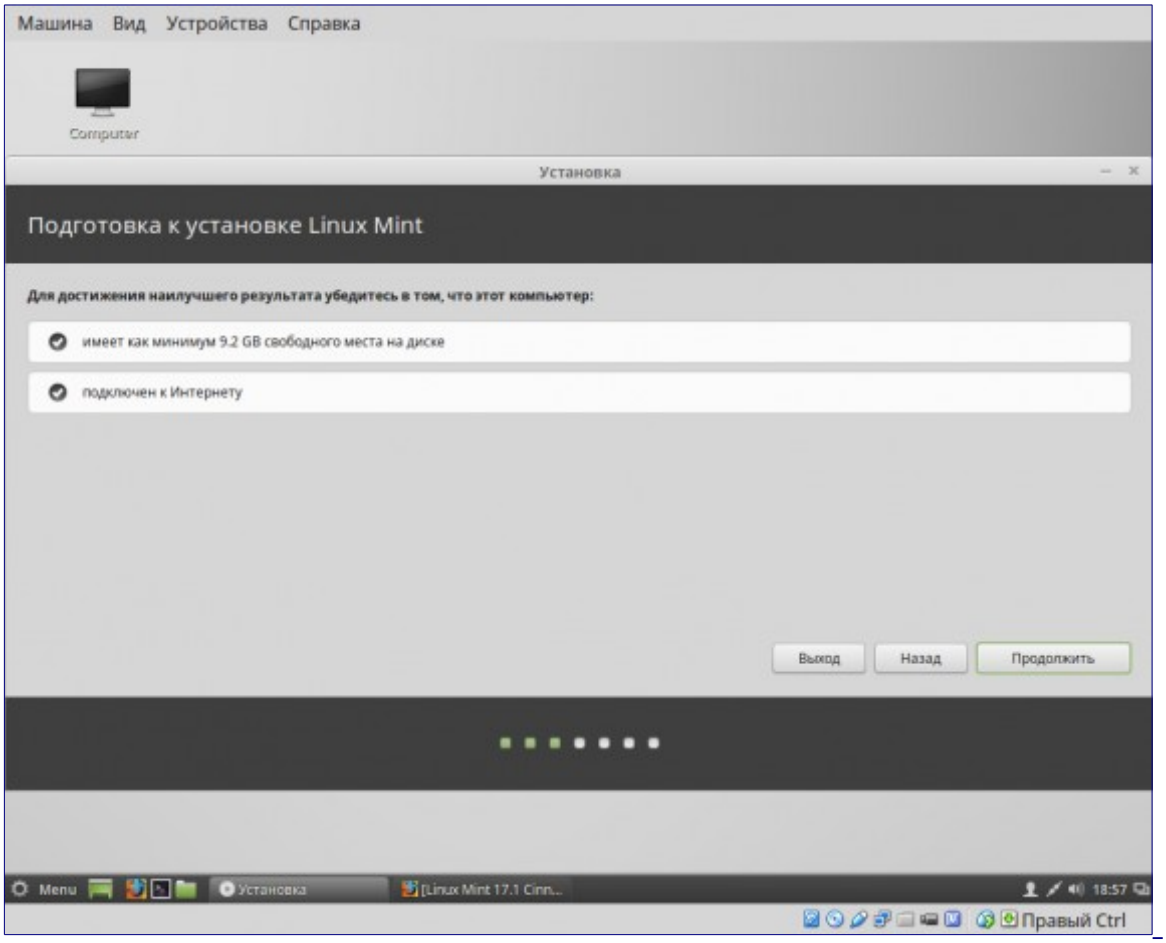


Однако, если у вас нет непреодолимого отвращения к языку родных осин, но есть желание без лишних трудозатрат иметь корректно локализованную систему, выбирайте русский, не пожалеете. Это будет язык не только интерфейса инсталлятора, но и грядущей инсталлированной системы:

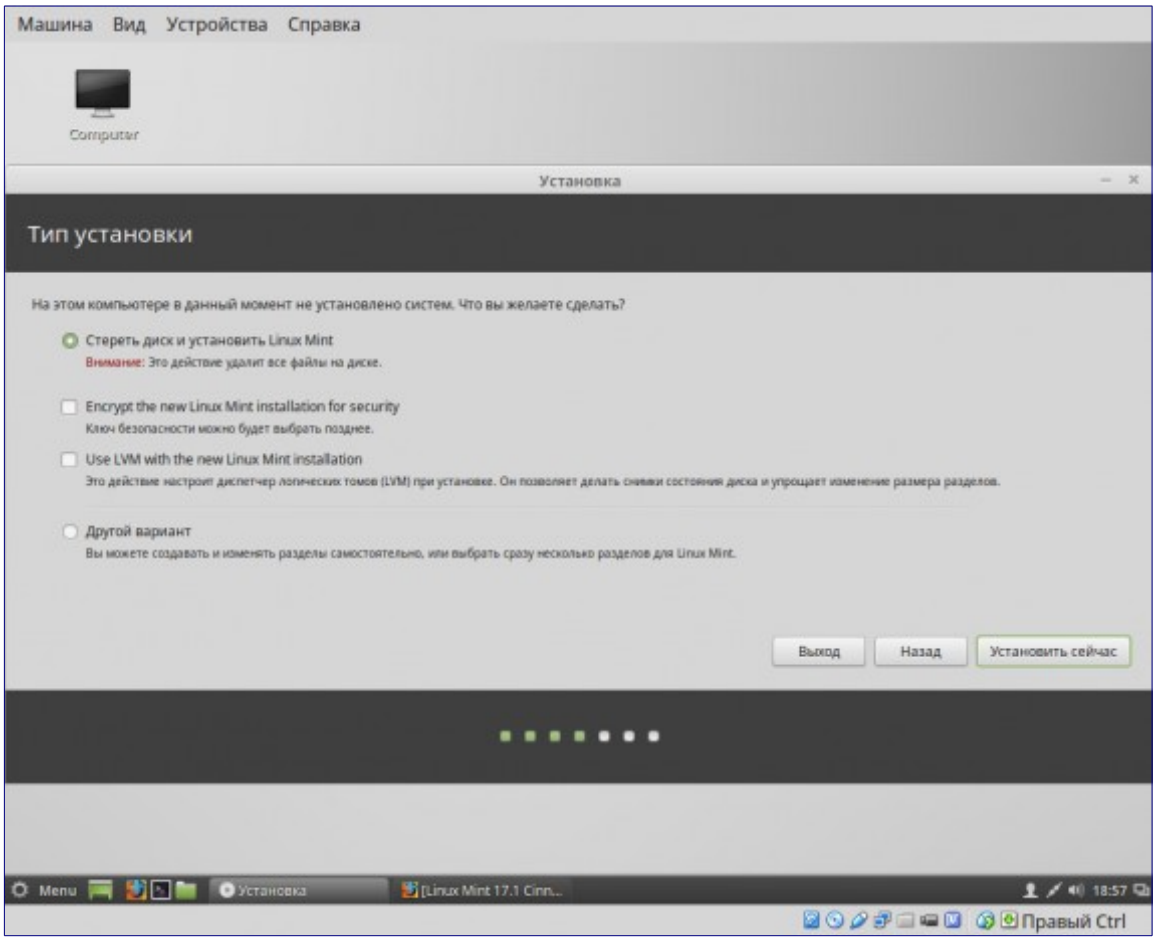


Русификация Cinnamon до недавнего времени была далека от идеала, и со смесью нижегородского с оксфордским приходилось сталкиваться сталкиваться довольно часто. Однако в версии 2.4 ситуация резко улучшилась, и языковый микст можно увидеть только в единичных случаях.

Далее выдвигаются претензии к объёму свободного места и подключению к Интернету — приведённой цифрой первого и потребностью во втором нынче испугать трудно:



Следующий этап — разметка диска. Здесь надо действовать по амбициям и амуниципям, так что просто прокомментирую пукты соответствующего меню:

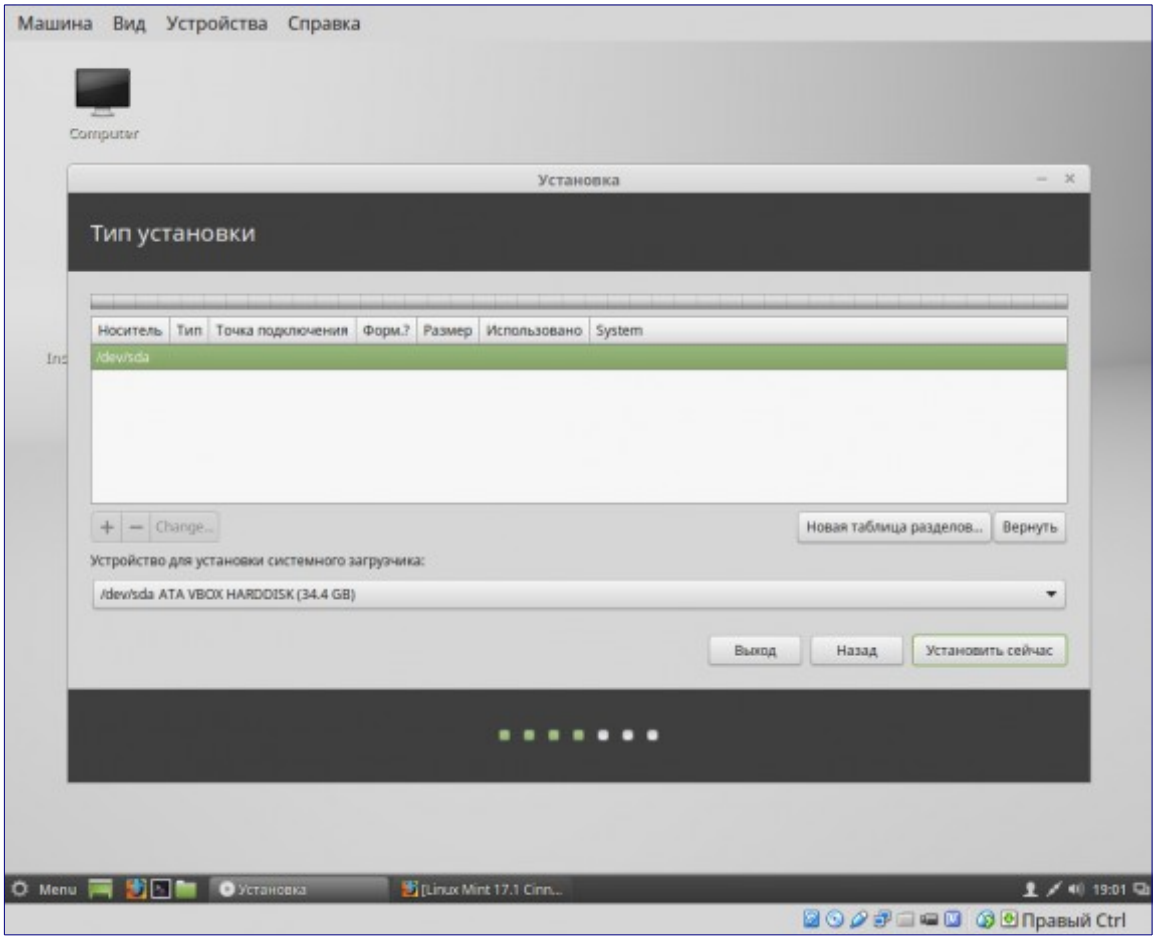


Очевидно, что отмеченный по умолчанию пункт Стереть диск и установить

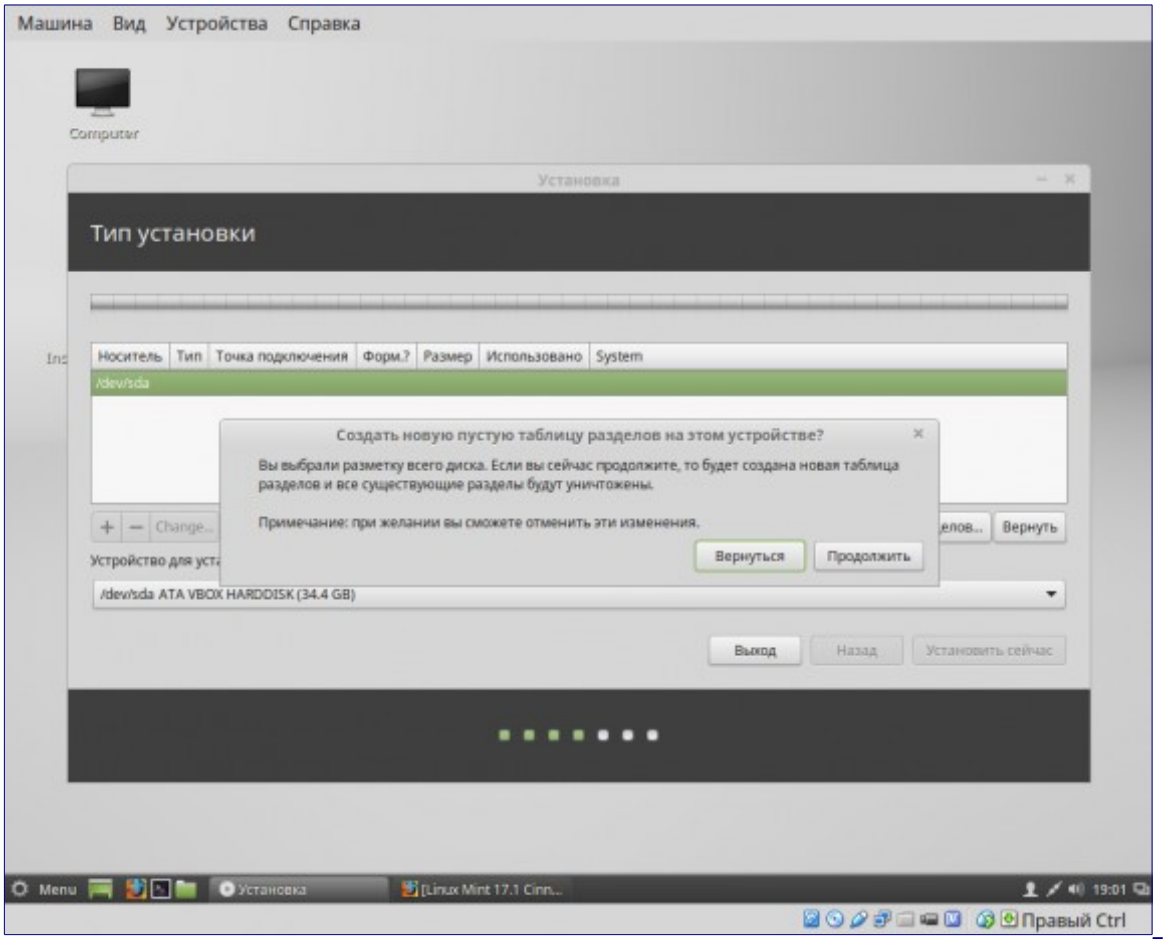
Linux Mint подходит только для установки на «чистый» носитель или такой, содержимым которого можно безболезненно пожертвовать. Причём именно всего устройства, а не какого-либо его раздела, ибо в этом варианте диск будет переразмечен полностью, начиная с таблицы разделов.

Об установке системы на шифрованный раздел (пункт второй) сказать ничего не могу, так как никогда не пробовал (и сомневаюсь в необходимости в обычных «настольных» условиях). Установка с использованием LVM (Logical Volume Manager) показалась мне не очень прозрачной, и я её тоже не опробовал. Но со временем расскажу, как задействовать LVM в уже инсталлированной системе. И потому я всегда выбираю Другой вариант: он подразумевает разметку диска вручную. Как её выполнить — говорено и переговорено на бесчисленных ресурсах в Сети: тут всё зависит от ситуации, потребностей и возможностей. Так что просто приведу несколько скриншотов, иллюстрирующих самый простой случай — разметку одиночного «чистого» на два раздела — корневой и под будущий домашний каталог (каковой, кстати, в умолчальном первом варианте не создаётся).

Итак, перед нами чистый, неразмеченный диск:



Можно видеть, что ни одна из кнопок манипуляции разделами (+, -, Change) не активизирована, ибо диск наш лишён таблицы разделов. Которую и надо в первую очередь создать, нажав соответствующую кнопку. Ответом будет предупреждение о том, что все ранее существовавшие разделы будут уничтожены. Поскольку их всё равно нет, соглашаемся на это:



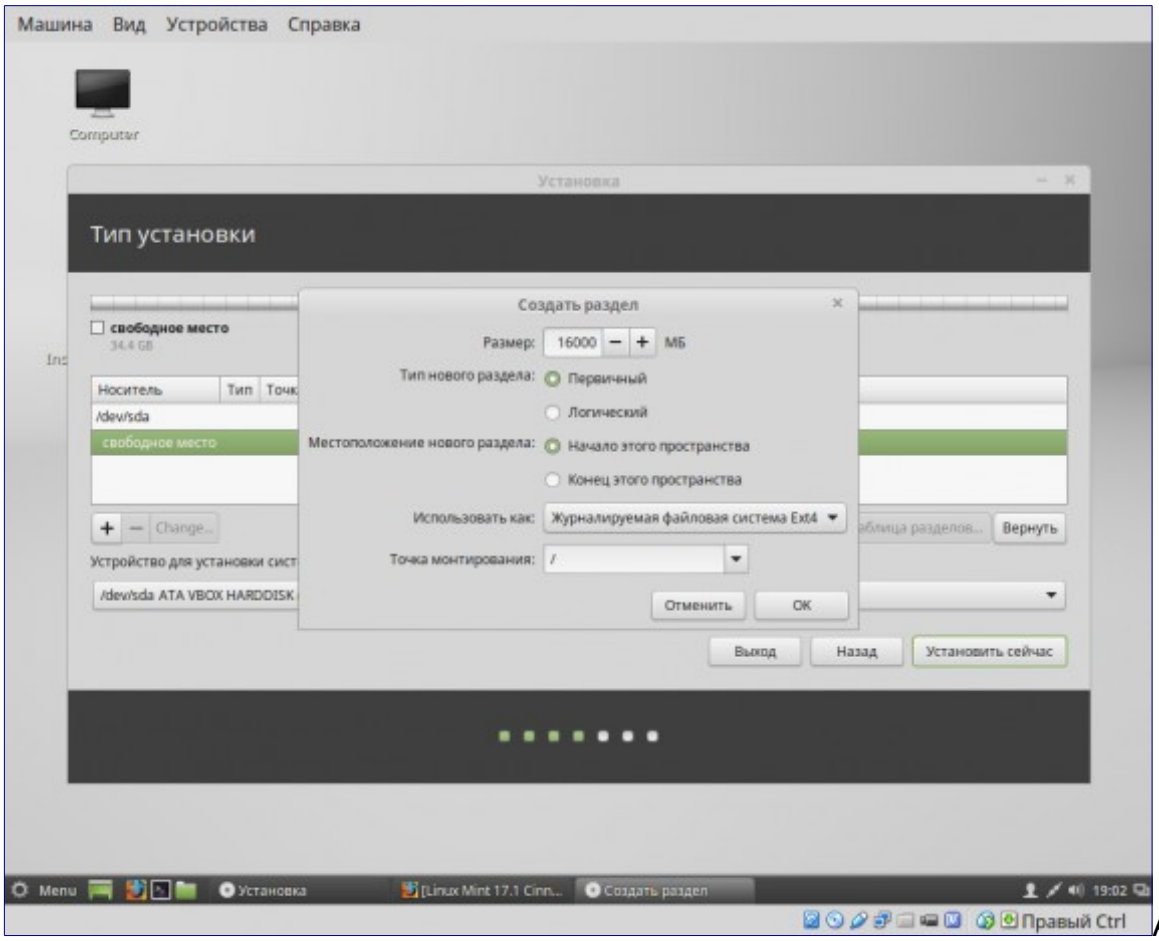
В результате будет создана таблица разделов в стиле MSDOS. Создание GPT-разметки не предусмотрено, но если диск был предварительно размечен в этом стиле — он прекрасно воспримется инсталлятором.

Теперь активизируется кнопка с плюсиком — очевидно, что она служит для создания разделов. Однако, прежде чем заняться этим делом, бросим взгляд на строку Устройство для установки системного загрузчика, каковым на скриншоте выступает /dev/sda. То есть загрузчик будет установлен в MBR первого диска, что для «чистой» однодисковой (и к тому же виртуальной) конфигурации вполне естественно. А вот в случае какой-либо уже установленной системы, особенно при наличии двух и более дисков, этот момент требует внимания.

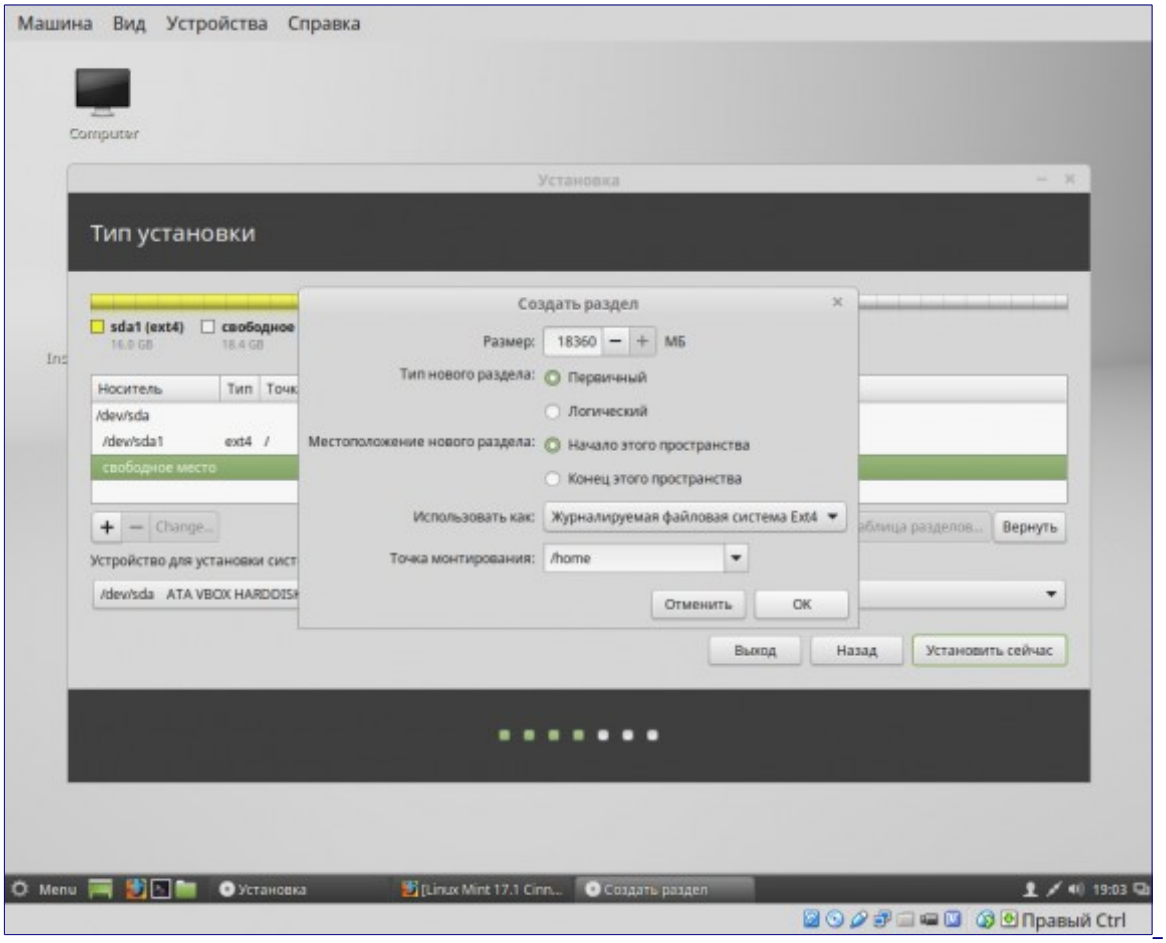
Дело, однако, в том, что первый диск (то есть устройство /dev/sda) будет по умолчанию целевым для установки загрузчика вне зависимости от того, на какой диск и или раздел устанавливается Mint, что в большинстве таких случаев не только не нужно, но и прямо вредит здоровью сосуществующих систем. И потому надо ни в коем случае не забыть выбрать из выпадающего меню нужное устройство — MBR диска или PBR раздела, целевых при установке Mint.

К сожалению, вообще отказаться от установки загрузчика нельзя, даже когда он заведомо лишний, и Mint предполагается загружать через загрузчик какой-либо иной системы. Эта багофича унаследована от Ubuntu, где существует испокон веков. Но можно обмануть установщик, подсунув ему в качестве целевого MBR какого-либо «левого» устройства, например, старой ненужной флешки или SD-карты, каковые потом просто извлекаются, как-будто так и было.

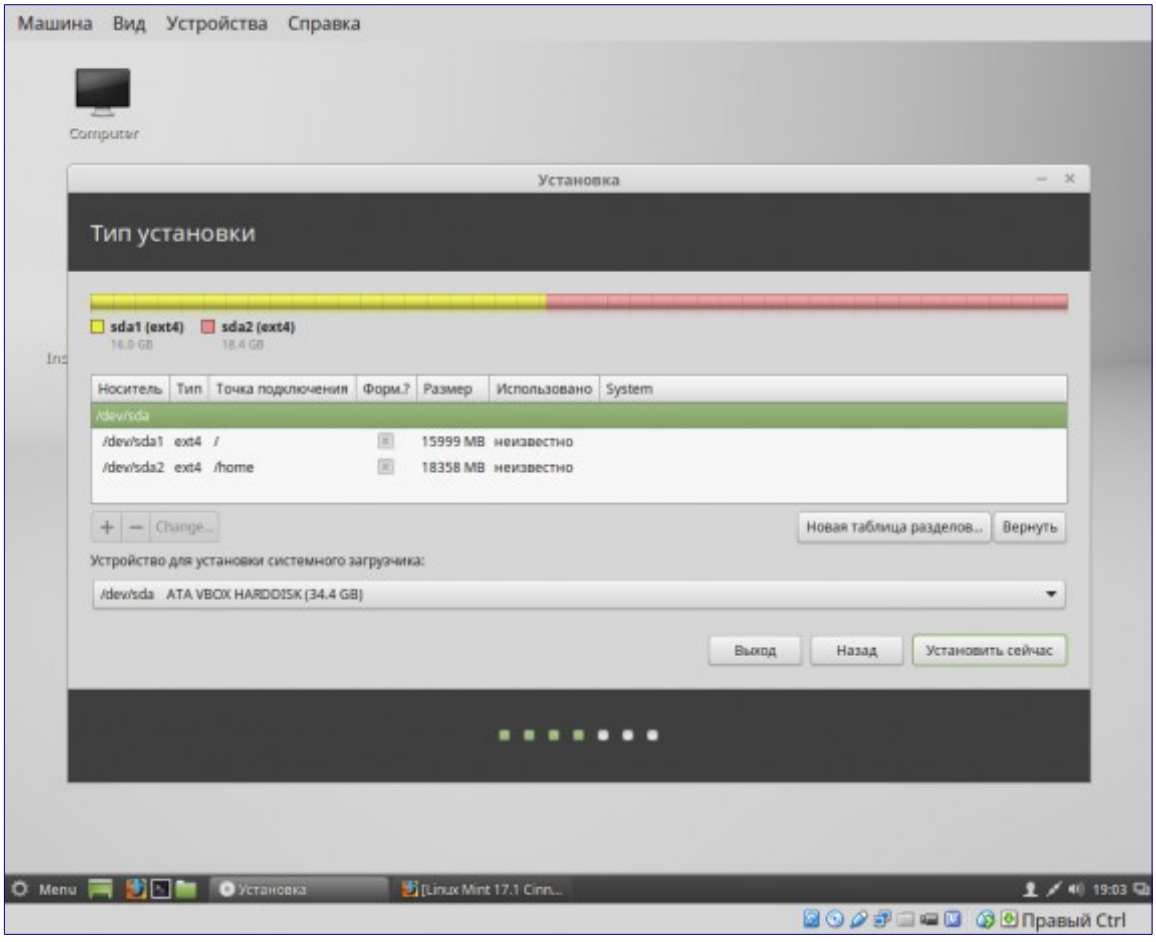
Вот теперь со спокойной душой можно приступать к созданию разделов, нажав кнопку с плюсиком. Проделываем эту процедуру сначала для корневого раздела:



А затем для раздела под будущий каталог /home:

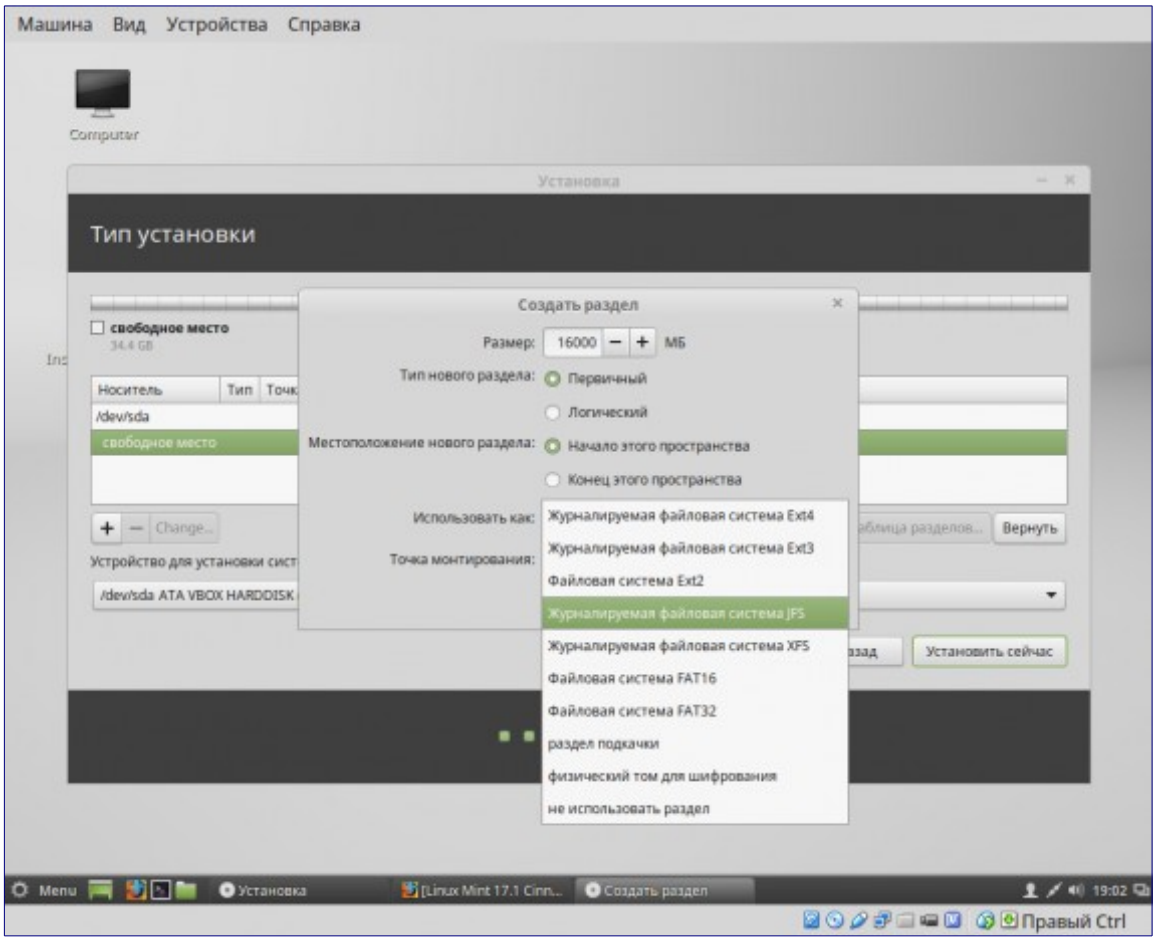


И в результате получаем вот такую картину:



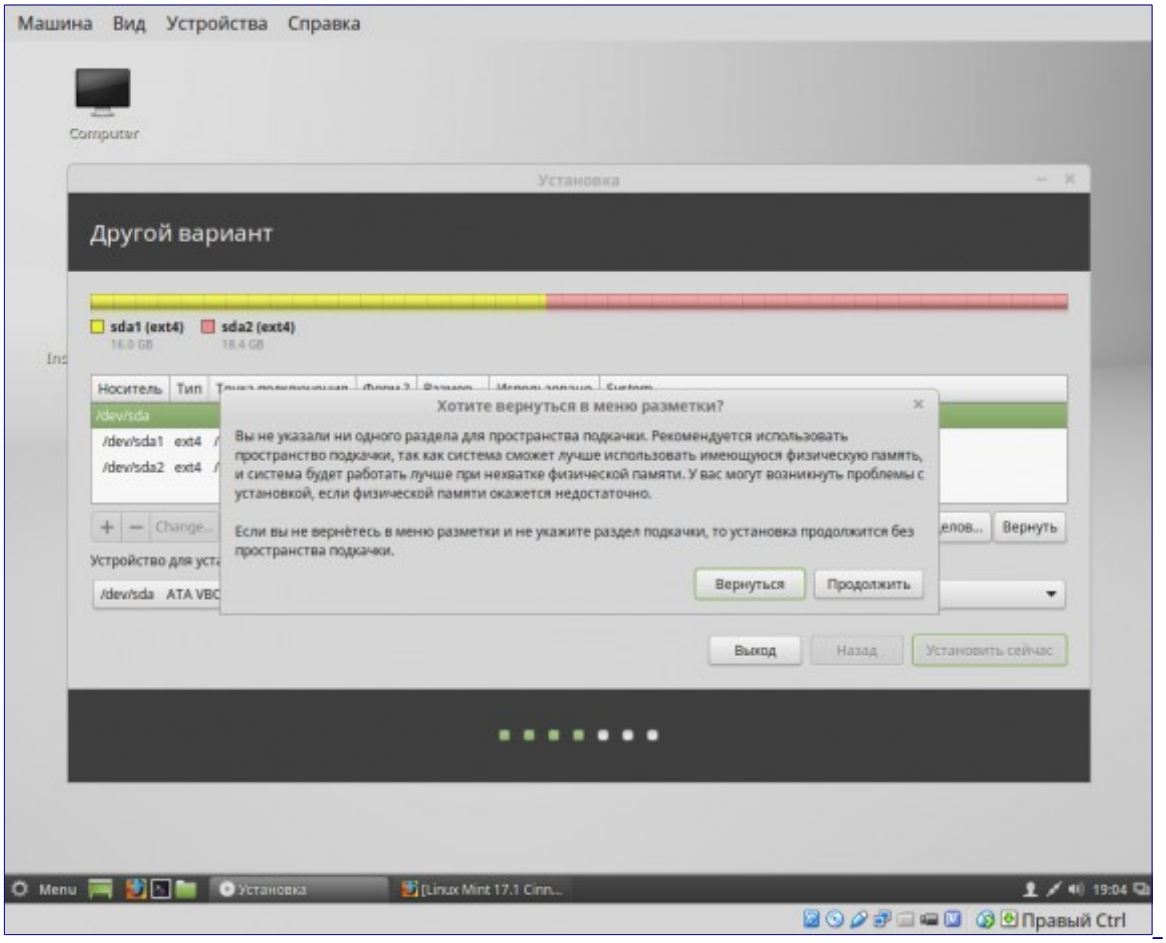
По умолчанию в обоих случаях предлагается файловая система Ext4. Если она почему-либо не устраивает — можно ознакомиться со списком доступных,

он включает все нативные файловые системы Linux'а, кроме ReiserFS:

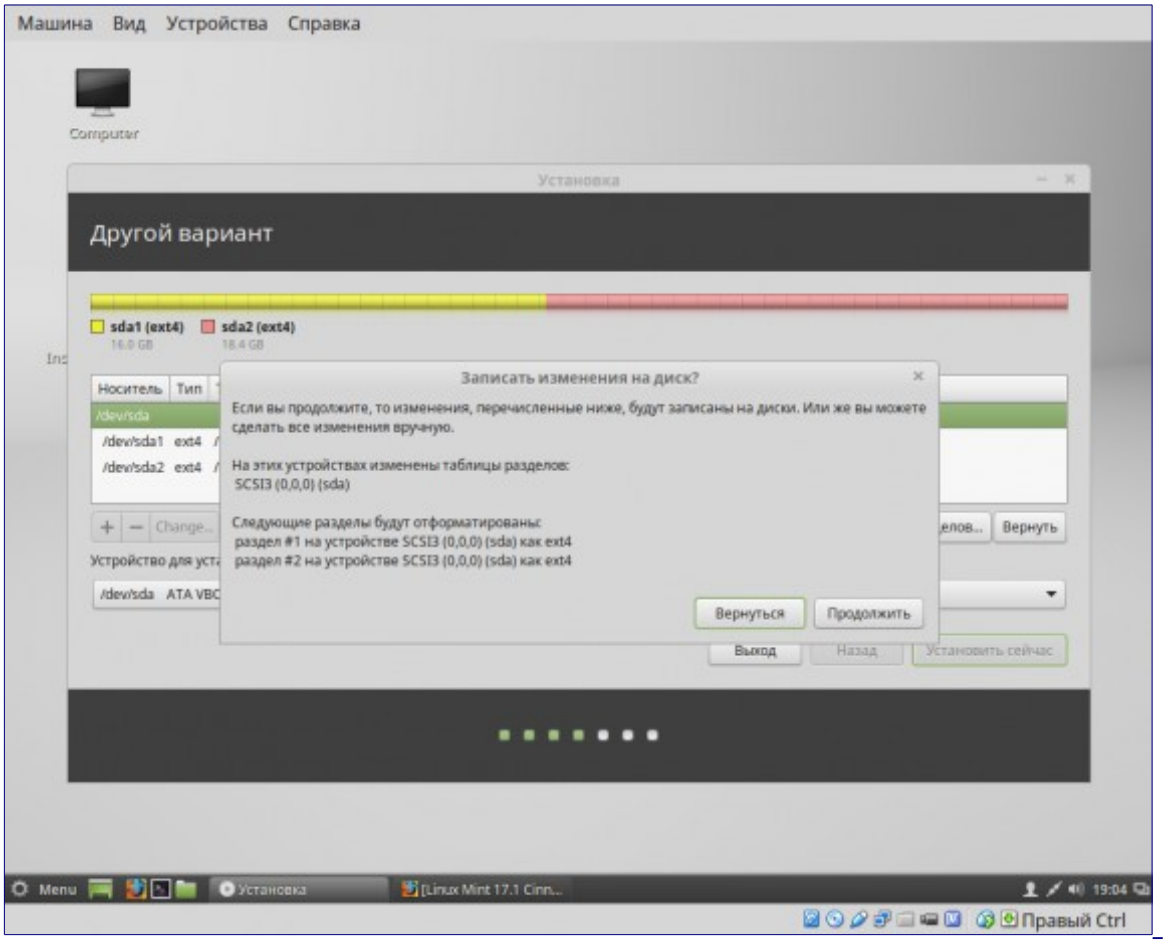


Должен заметить, что на стадии установки в Mint не только нельзя создать, скажем, программный RAID (в отличие от Ubuntu, в которой это уже предусмотрено): нельзя даже установить систему на softRAID, созданный заблаговременно, инсталлятор его просто не увидит. Впрочем, в дальнейшем, после установки, ни подключить существующий RAID, ни слздать его заново труда не составит, о чём будет говориться в соответствующем очерке.

Как можно было видеть, в приведённой схеме разметки отсутствует раздел подкачки. О чём и будет сообщено на следующей стадии:



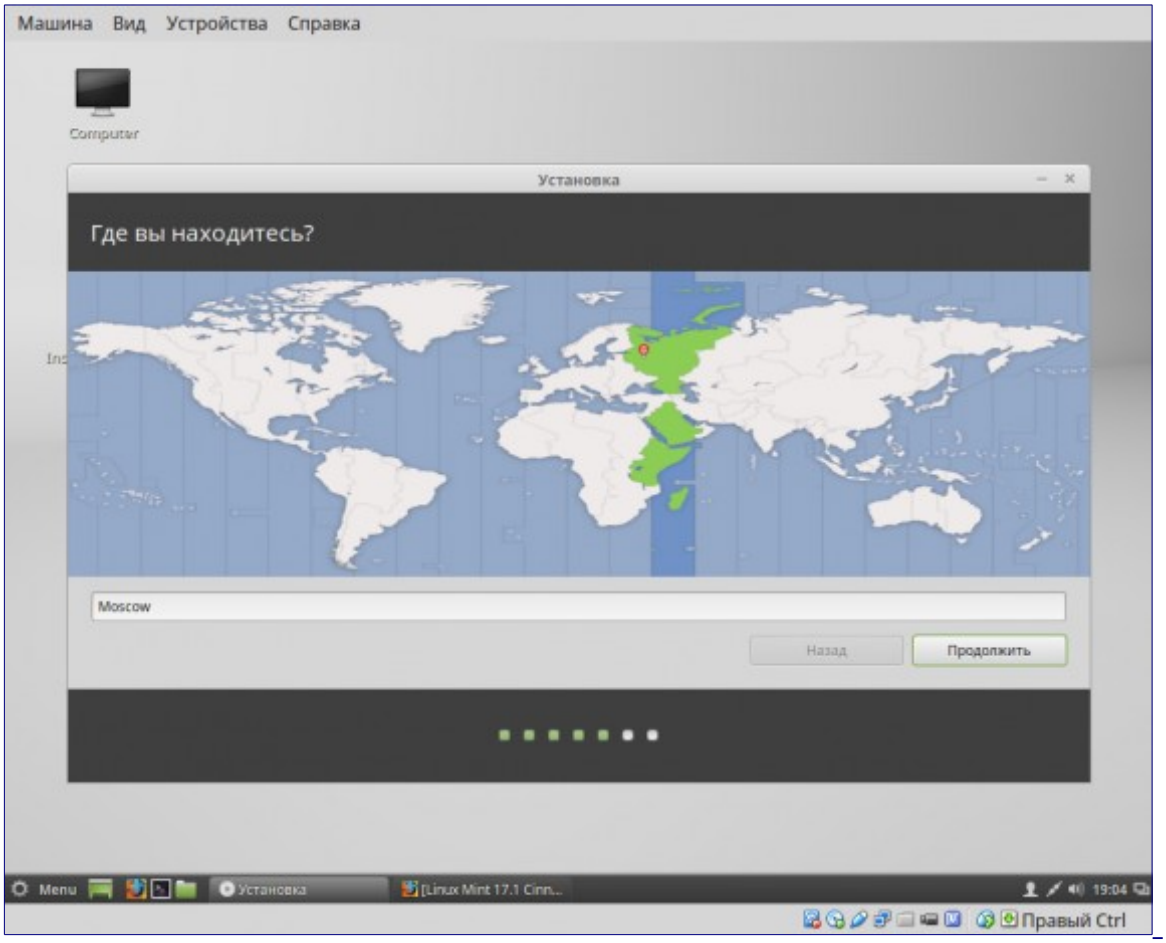
Нужен такой или нет — вопрос спорный и многократно обсуждавшийся. На мой взгляд, при памяти от 4 ГБ в большинстве случаев, кроме некоторых специальных, не нужен. Так что можно продолжать — это действие вызовет последнее китайское предупреждение:



Смысл его в русском переводе может показаться не вполне внятным. Он сводится к тому, что все действия по разметке диска будут выполнены только сейчас, но после этого возврат к прошлому станет невозможным — до сих пор установку можно было в любой момент прервать, сохранив исходное состояние диска.

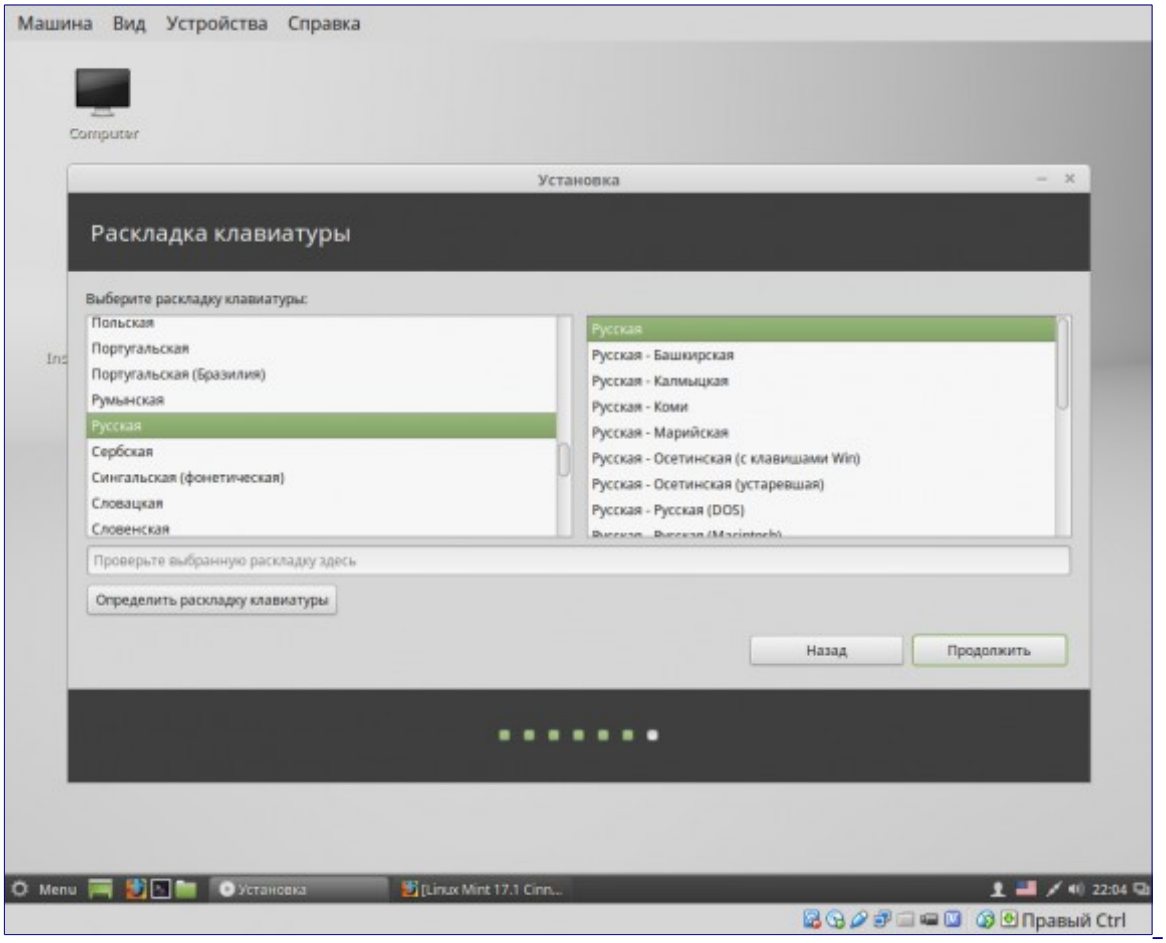
Поскольку бояться нам нечего, нажимаем кнопку Продолжить. И, после некоторого времени, требующегося на разметку и создание файловых систем, совершаем последние манипуляции — настройку часов, клавиатуры и создание пользовательского аккаунта.

С выбором часового пояса всё понятно без комментариев — при выборе русского языка инсталляции он будет московским и подразумевать установку «железных» часов по UTC:

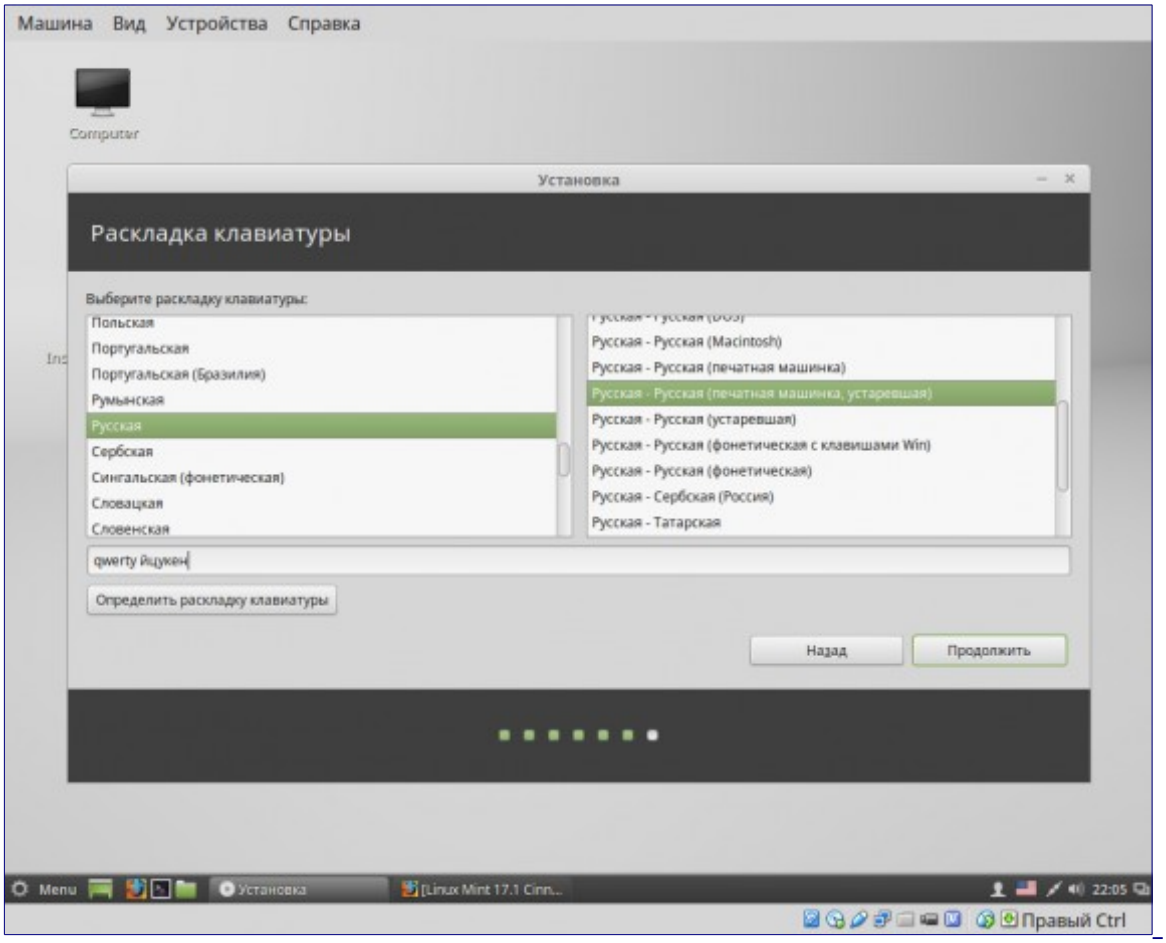


Так что обитателям более иных городов и весей нашей необъятной Родины следует не забыть внести нужные изменения — это можно сделать, просто ткнув курсором мыши в район Петропавловска Камчатского (Улан-Удэ, Новосибирска — нужное дописать).

А вот вопрос с раскладкой внимания заслуживает: выбранная сейчас, она останется и в инсталлированной системе. Опять же при выборе русского языка инсталлятора по умолчанию и раскладка клавиатуры предлагается русская — в данном случае под этим понимается вариант winkeys:

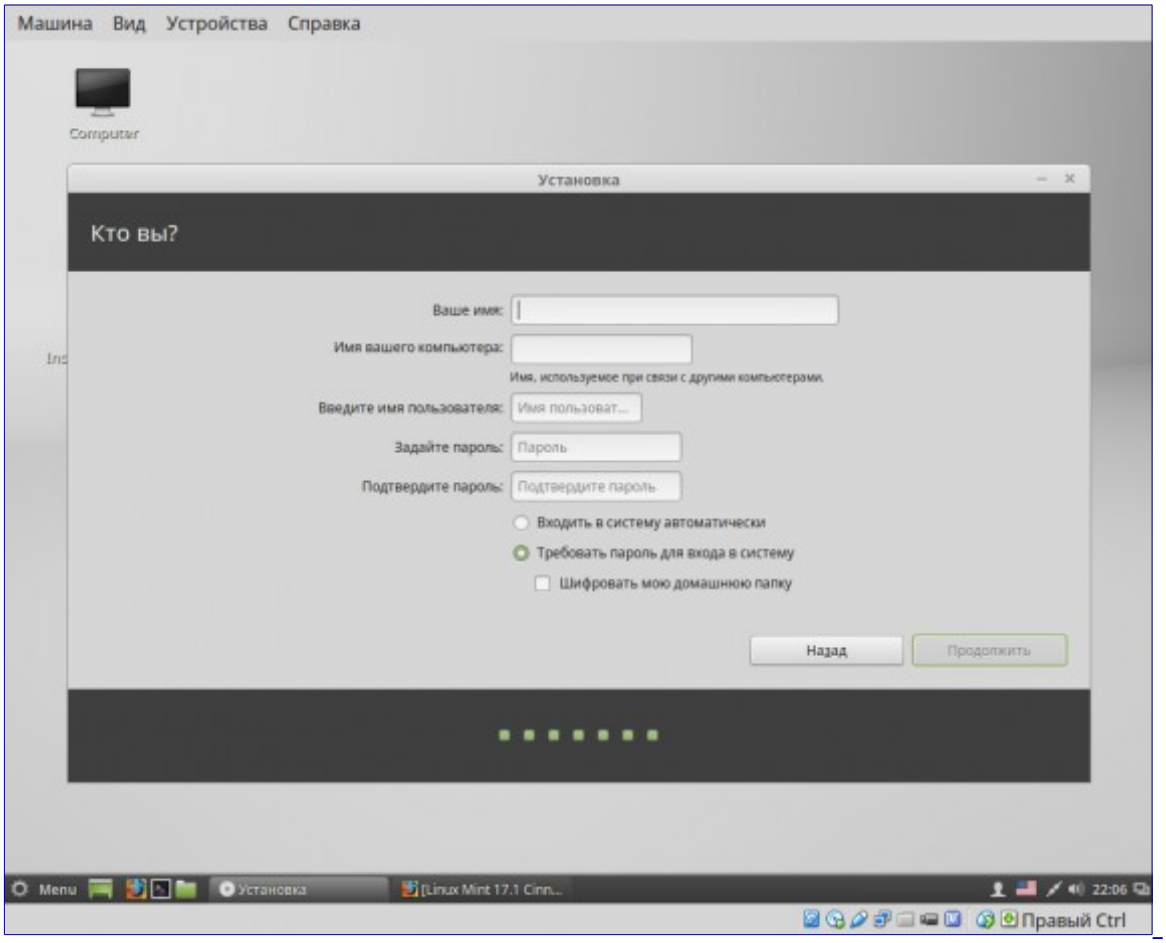


Он устроит большинство применителей — но не таких старых «печатно-машинистов», как автор этих строк. Что же, никто не запрещает выбрать вариант Typewriter Legacy — это раскладка советских пишущих машинок, почему-то объявленная устаревшей. И даже опробовать его в деле, переключаясь с латиницы на кириллицу через Alt+Shift:

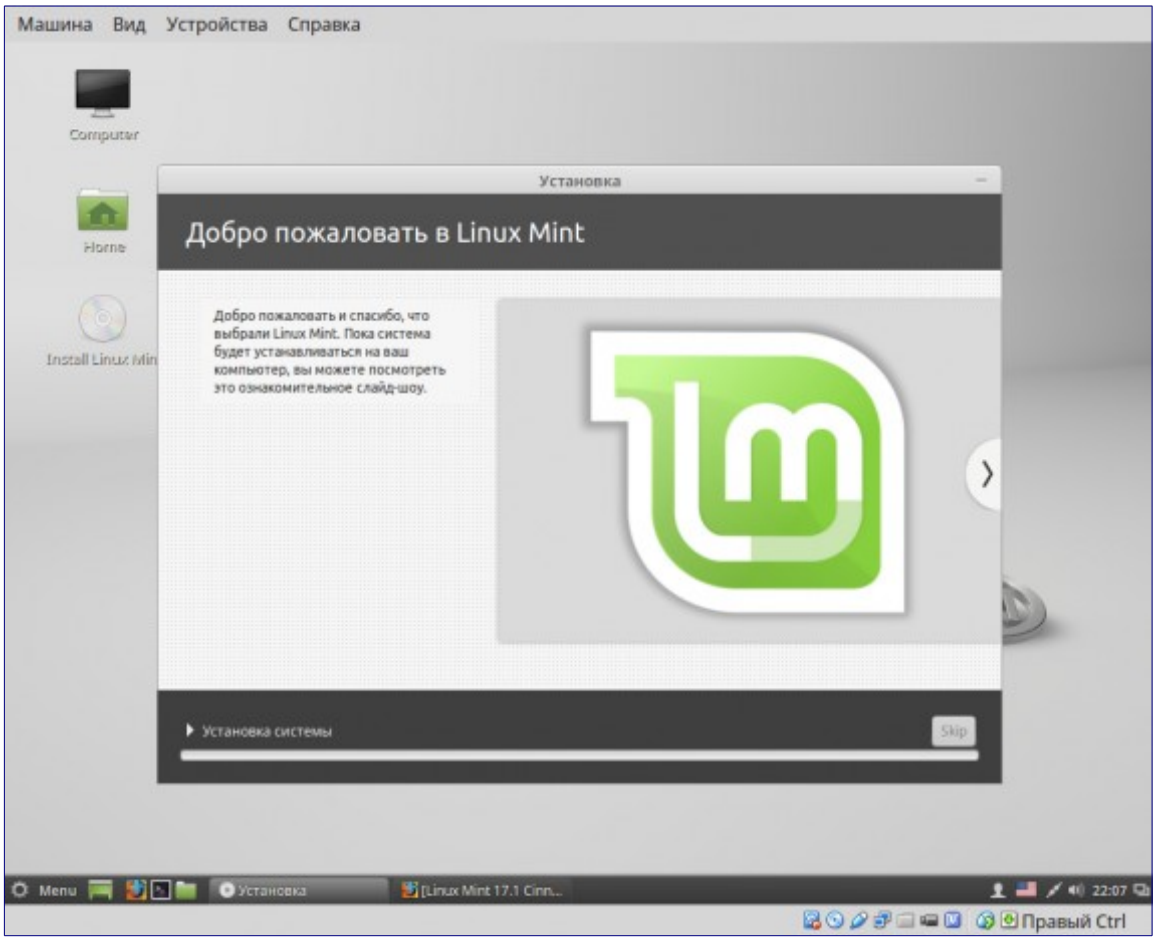


Смены переключателя раскладок на стадии установки не предусмотрено — её можно будет поменять потом, средствами Cinnamon. Зато установленная сейчас раскладка сохранится после установки не только в Иксах, но и в «голой» консоли. Причём даже такая нетипичная, как при моём выборе. Это чуть ли не уникальный случай — в других дистрибутивах для консоли раскладку Typewriter Legacy мне приходилось изготавливать самому.

В деле создания пользовательского аккаунта всё очевидно. Замечу только, что раскрывать своё подлинное имя здесь не обязательно — это поле можно просто оставить пустым. А вот задать имя компьютера необходимо — без этого не активизируется кнопка Продолжить. Впрочем, в большинстве случаев оно может быть произвольным. Ну а требовать ли от самого себя пароль для входа в систему или нет — дело чести, подвига и геройства личной паранойи:

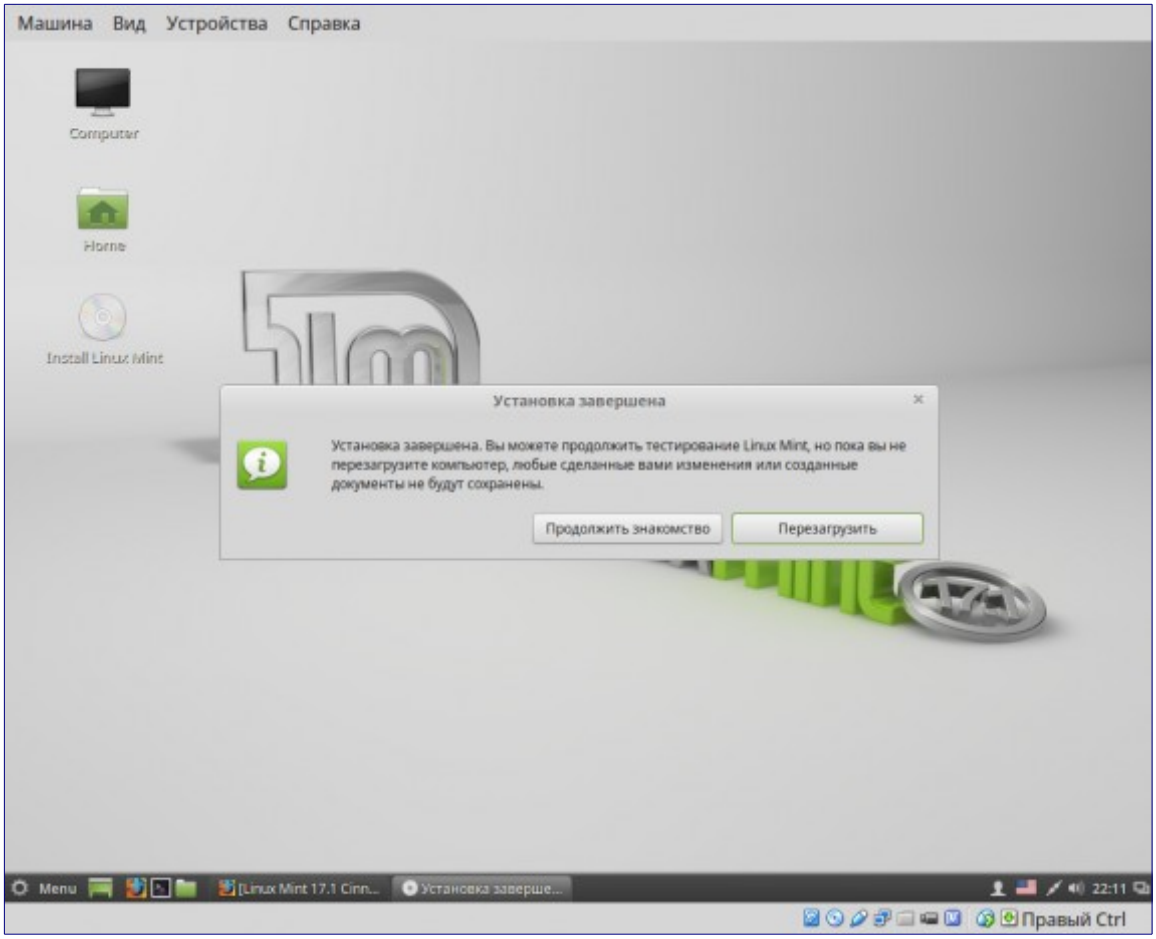


После этого начинается собственно установка, то есть перенос системы на целевой носитель:

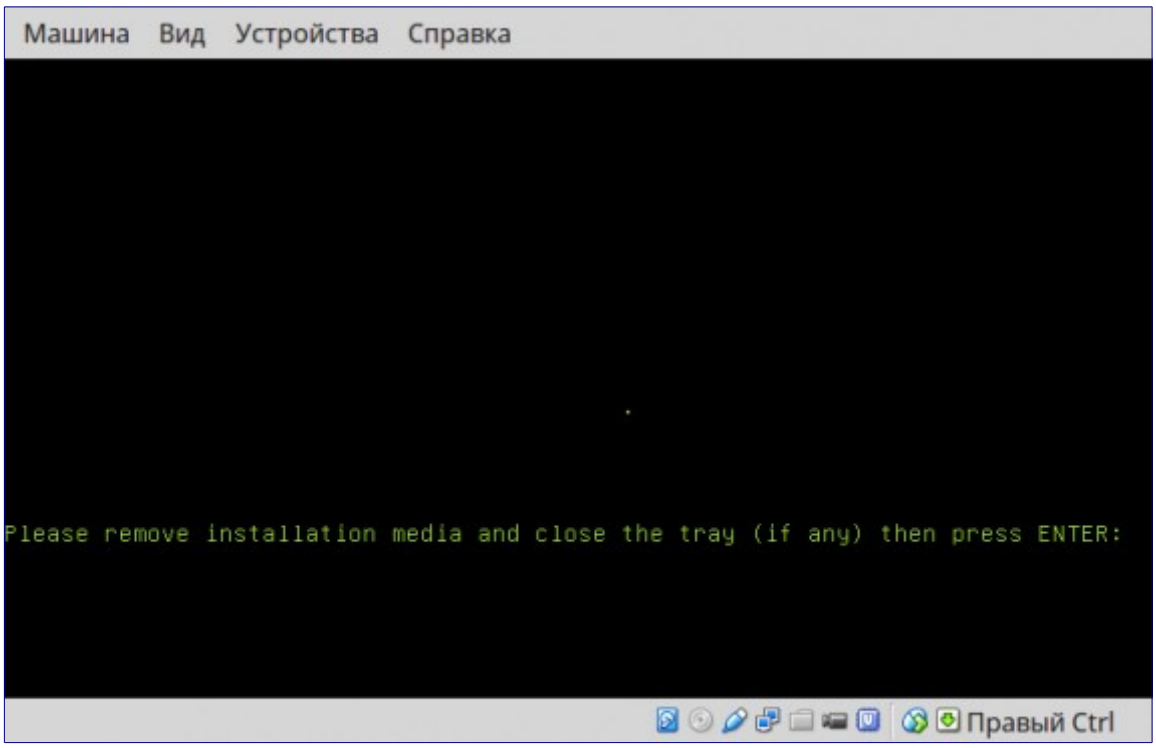


В Ubuntu это делается весьма неторопливо. В инсталляторе же Mint едва

успеваешь заварить себе кофий или выкурить сигарету, как обнаруживаешь на экране вот такое сообщение:



После которого только и остаётся, что перезагрузиться. Правда, после этого будет ещё одно сообщение:

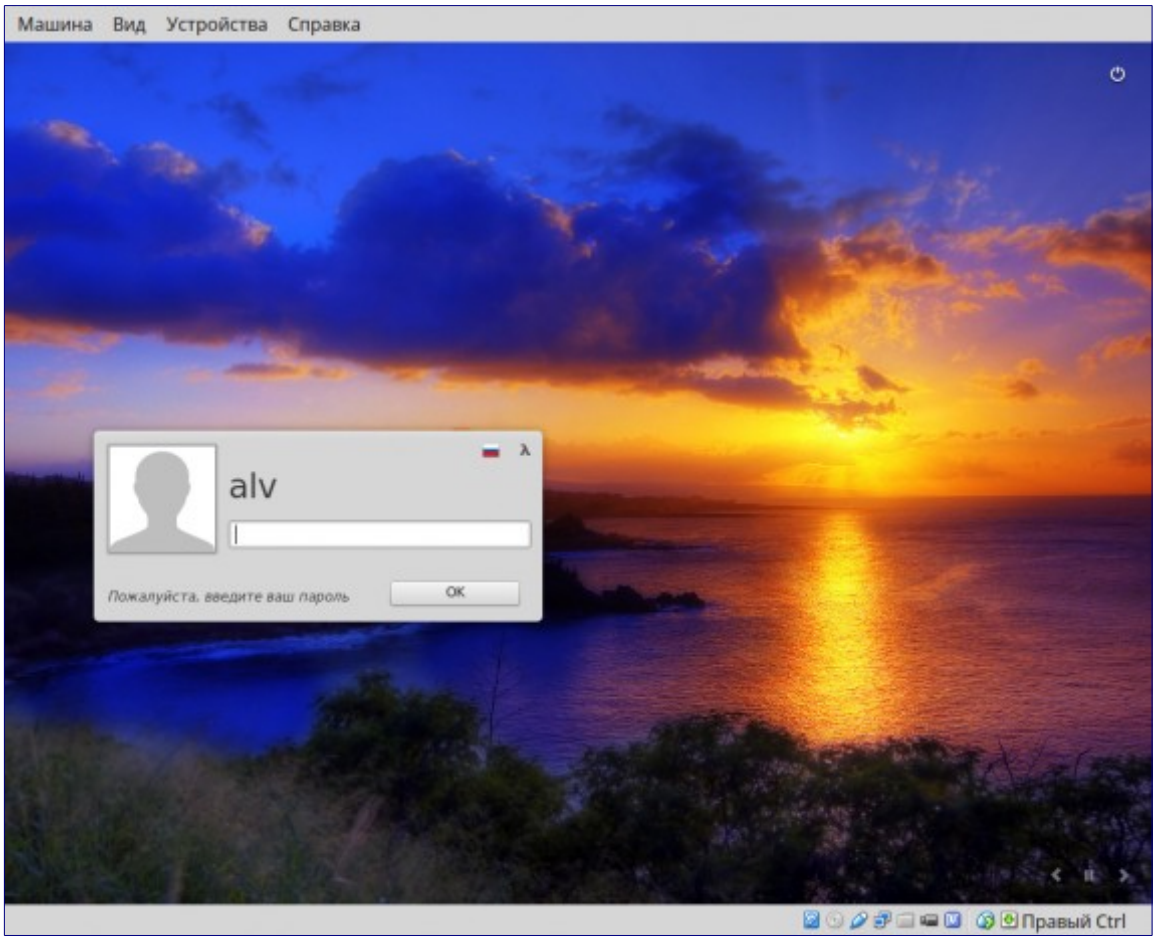


Если установка проводилась с DVD-диска, он будет извлечён из привода автоматически, и Enter можно нажимать сразу. Если с флешки или SD-карты — соответствующее устройство желательно предварительно удалить. Ну а что

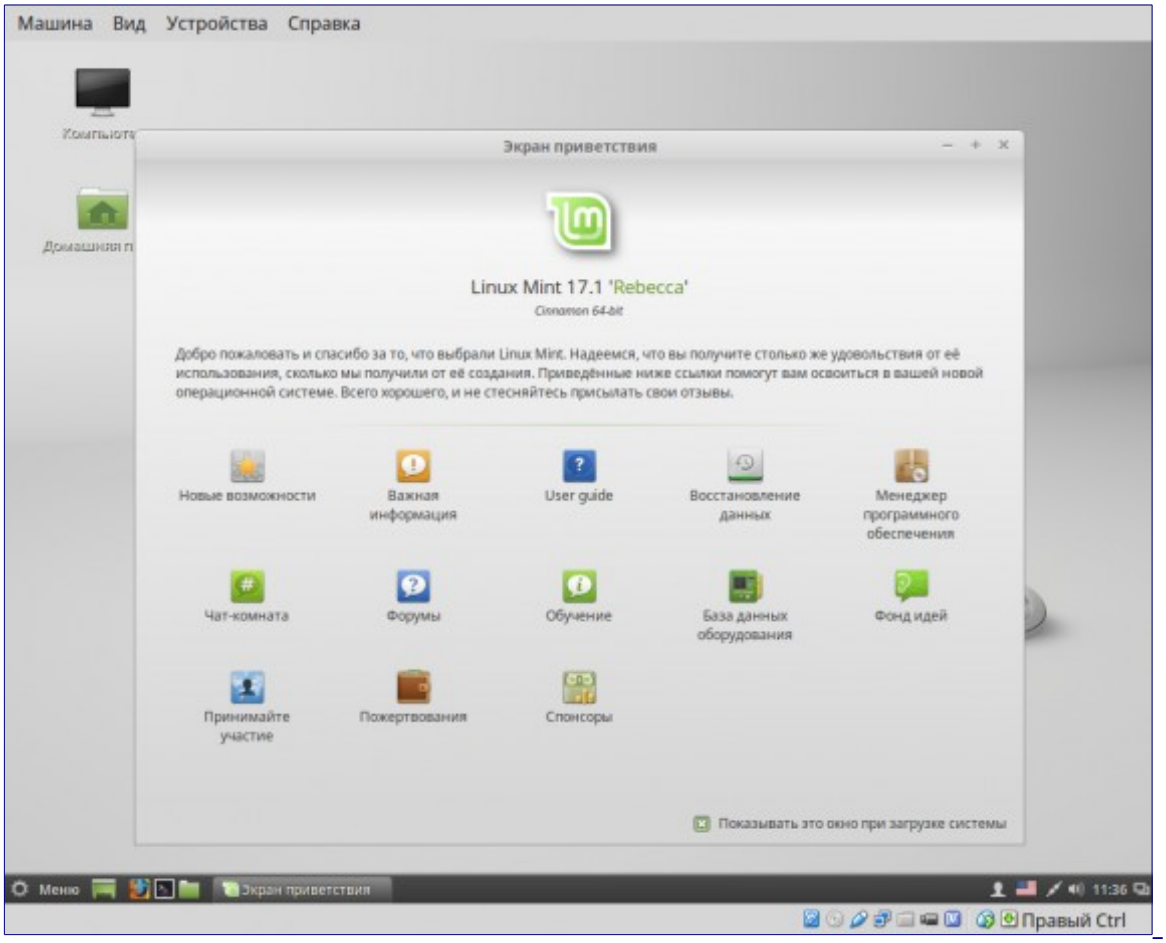
получилось в результате инсталляции — будет рассказано в следующем очерке.

Mint и Cinnamon: обзор среды

Предыдущий очерк мы завершили тем, что после удачной установки отправили машину на перезагрузку. После которой, проскочив меню GRUB'а (в случае одной системы по умолчанию оно не выводится), оказываемся в дисплейном менеджере MDM с предложением авторизоваться:



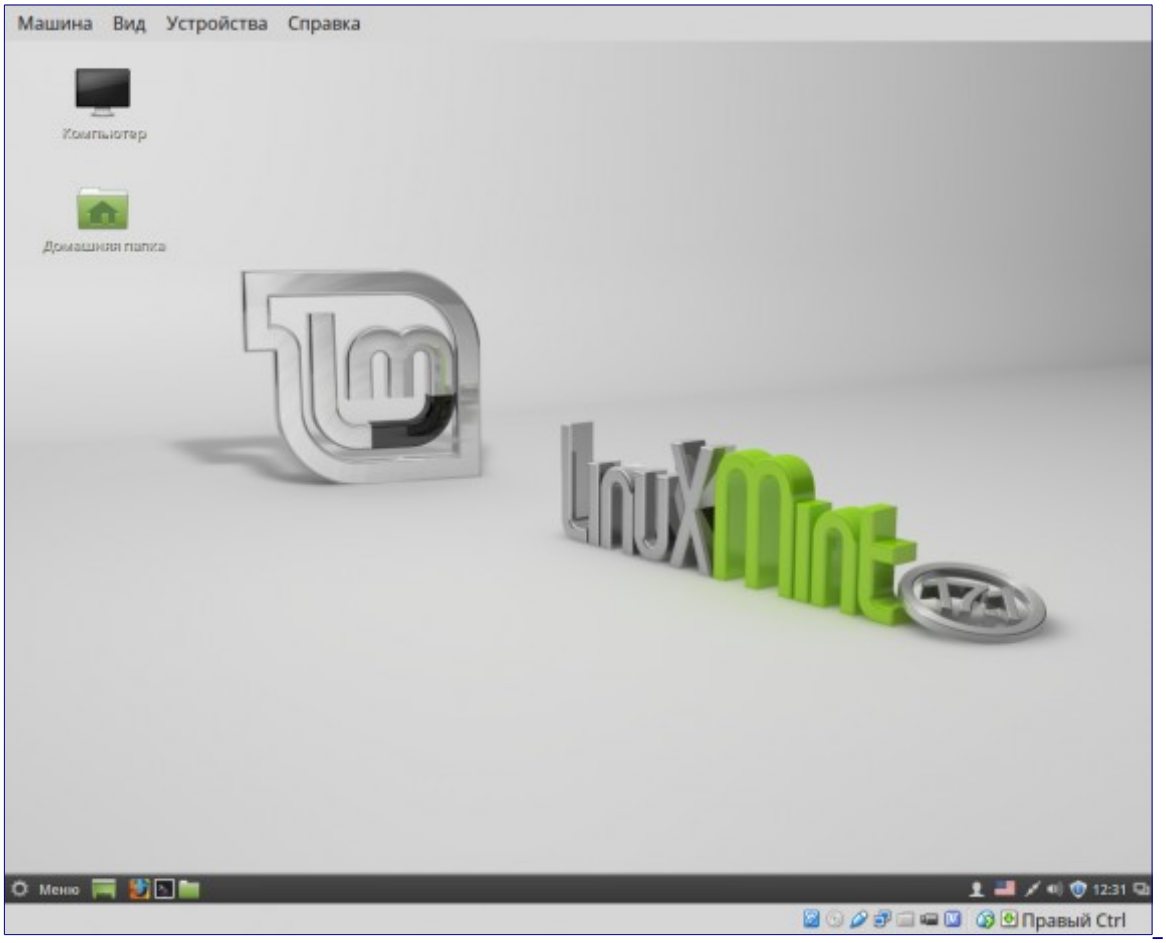
По принятии этого предложения перед нами предстаёт рабочий стол Cinnamon с экраном приветствия:



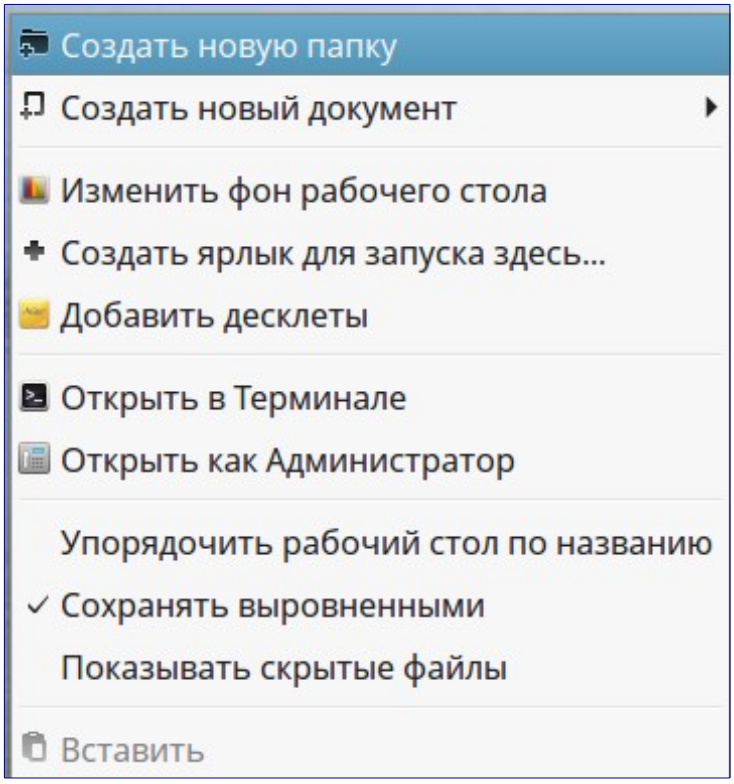
И о дисплейном менеджере, и об экране приветствия случай поговорить ещё будет. Нынешний эе очерк будет посвящён интерфейсу рабочего стола Cinnamon, так как именно там в основном будет проходить деятельность применителя, избравшего соответствующую редакцию дистрибутива Mint.

Cinnamon. Общий вид

При первом своём запуске Cinnamon выглядит более чем традиционно – перед нами самый обычный рабочий стол с управляющей панелью, на которой имеется кнопка с подписью Меню (или Menu, в зависимости от локализации):



В отличие от GNOME Shell'а или Unity, здесь сразу ясно, что делать дальше. Во-первых, можно щёлкнуть правой кнопкой мыши по рабочему столу, чтобы увидеть его контекстное меню:



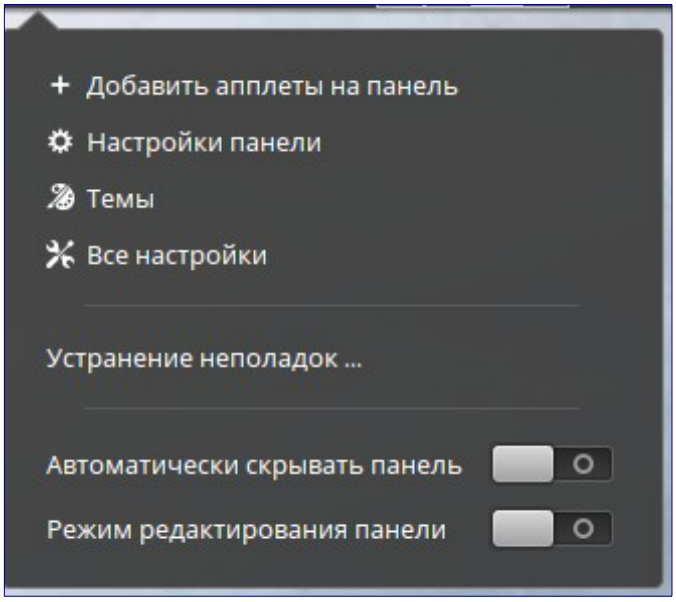
Здесь почти всё понятно без комментариев, пару слов можно сказать только про два пункта:

- Добавить десклеты – добавление на рабочий стол мини-приложений

(подробнее об этом будет сказано в следующем очерке);

• Открыть как администратор – вызов, после ввода пароля, файлового менеджера Nemo с правами суперпользователя.

Во-вторых, щёлкнув правой же кнопкой мыши по свободному полю управляющей панели, можно заняться её настройками (о чём будет разговор в соответствующем очерке):

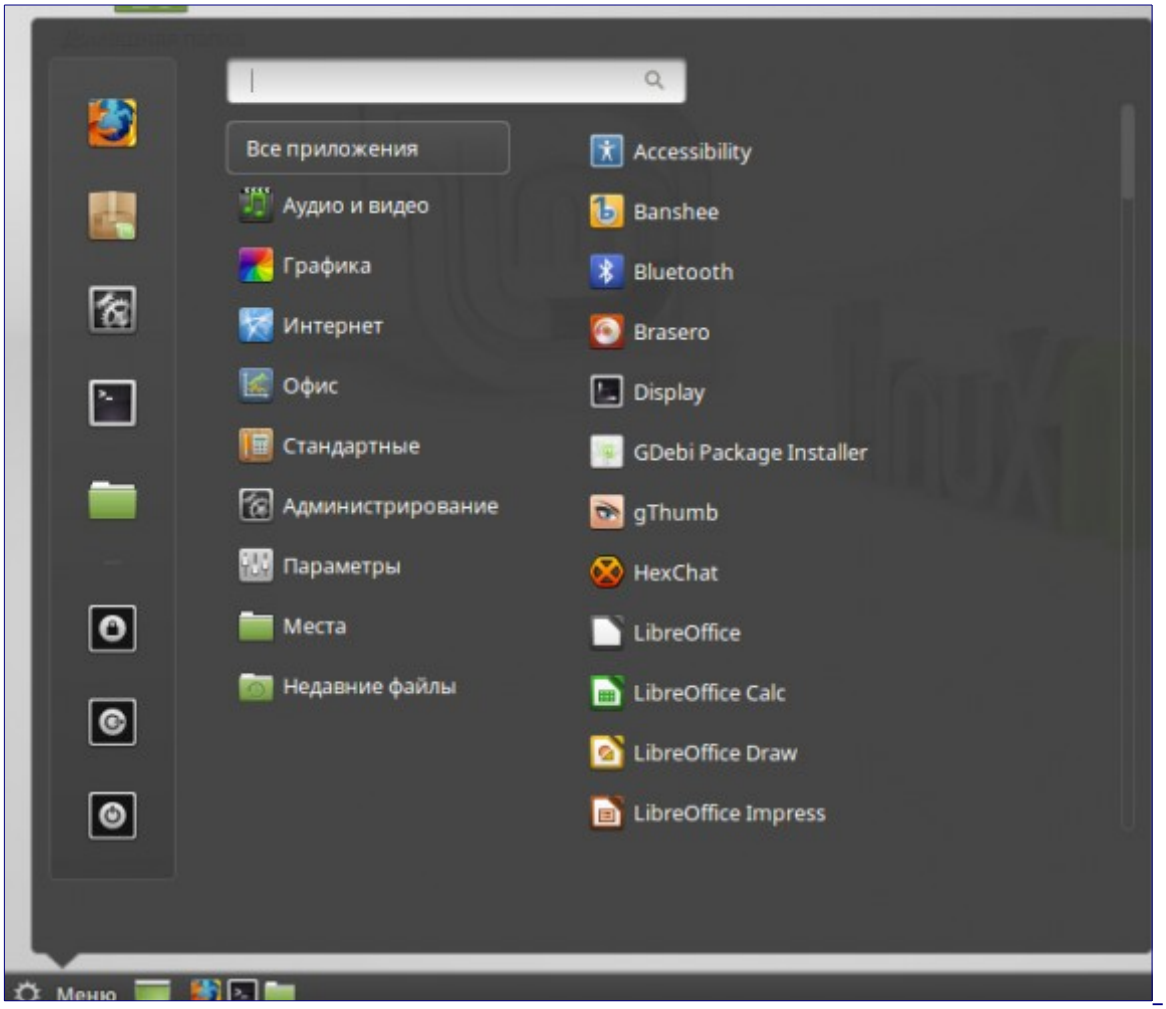


Или, уже с помощью левой кнопки, вызвать одно из приложений, пиктограммы запуска которых уже имеются на панели. По умолчанию их не густо – браузер Firefox, унаследованный от GNOME терминал и файловый менеджер Nemo:



Но пополнить панель иконками приложений первой необходимости труда не составит – и со временем я расскажу, как.

Наконец, в-третьих, можно обратиться к главному меню Cinnamon для знакомства со всем изобилием установленного софта:



Но обзор штатных приложений дистрибутива будет дан в соответствующем очерке.

Как и во всех современных рабочих средах, в Cinnamon'е можно задействовать несколько виртуальных рабочих столов. В терминах этой среды они называются рабочими областями (Workspaces), и по умолчанию их два. Переключение между рабочими областями – комбинациями клавиш Control+Alt+Right/Left.

Есть и другие способы переключения между рабочими областями, и количество их можно увеличить до любого разумного предела. Но это относится уже к категории настроек, которым будет посвящён отдельный очерк.

Управляющая панель

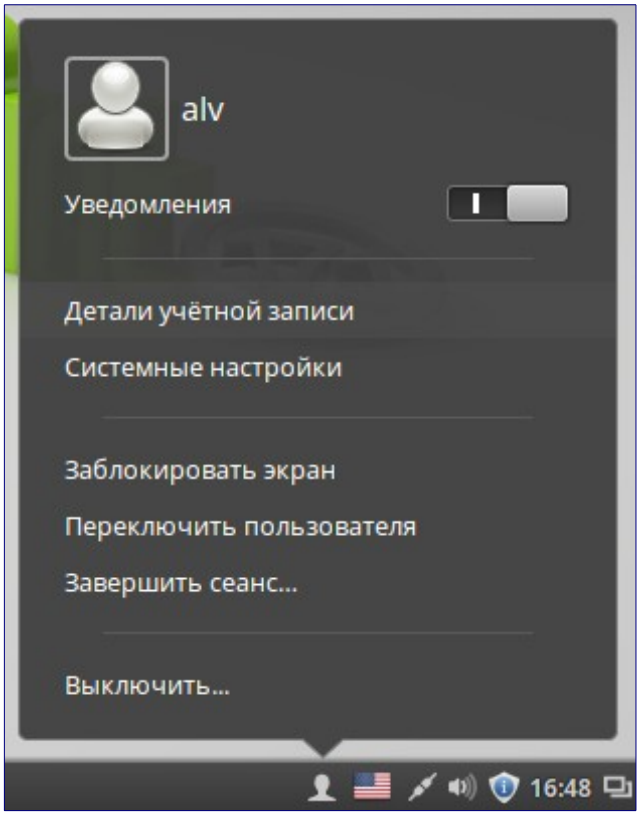
Если на рабочем столе (точнее, в рабочих его областях) происходят основные события, то управление этими событиями в значительной мере осуществляется с панели – в других средах её часто называют главной, или управляющей. Но в Cinnamon её принято называть без определений, поскольку здесь она обычно имеется в единственном экземпляре. Хотя в принципе в нём может быть включена и вторая панель, о чём речь пойдёт в очерке о настройках.

Положение панели по умолчанию – вдоль нижней части экрана, хотя её можно переместить и наверх. И разделяется она на следующие части (слева направо):

- кнопка главного меню (опять же – обычно просто Меню);
- область запуска приложений (Panel Launcher) с пиктограммами – как уже было сказано, по умолчанию их всего три;
- область открытых приложений (Window list);
- область системных сообщений (Notifications);
- системный лоток (System Tray), куда помещаются пиктограммы «перманентно работающих» приложений;
- несколько пиктограмм разного назначения – список открытых окон (Windows Quick List) и подключаемых накопителей (Removable Drives), модуль сетевых соединений, регулятор громкости, часы, индикатор раскладки клавиатуры, пиктограмма менеджера обновлений.



Кроме того, имеется «пользовательская кнопка» – она показывает сведения о текущем аккаунте, вызывает системные настройки, через неё блокируется экран и переключаются пользователи, осуществляется вход в так называемый «гостевой» сеанс, а также выполняется завершение сеанса работы, перезагрузка машины и её выключение:

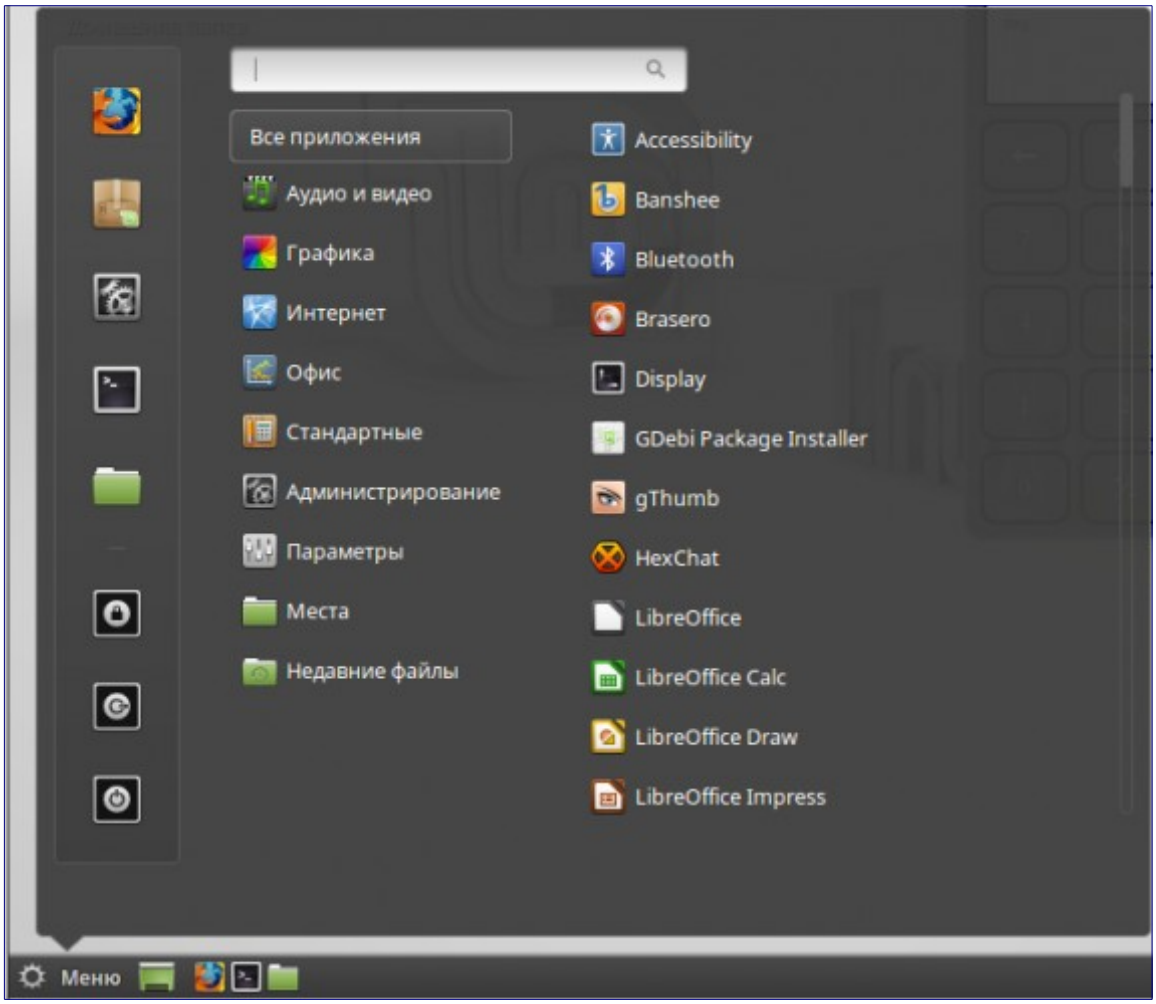


Все области и кнопки панели представляют собой так называемые апплеты – мелкие программки, не способные к самостоятельному функционированию. Некоторые из этих апплетов (Panel Launcher, Window list, System Tray) представляют собой контейнеры для помещёния в них других пиктограмм (запуска, открытых окон, и так далее), другие же существуют как бы сами по себе.

Апплеты могут добавляться на панель и удаляться с неё. Содержимое апплетов-контейнеров добавляется или автоматически (System Tray, Window list), или вручную (Panel Launcher). Удаление апплетов-контейнеров приводит к исчезновению всего их содержимого. Элементы из лаунчера можно удалять по одному, содержимое лотка и области приложений – вместе с закрытием соответствующих программ.

Главное меню

Центральным (хотя и левым крайним) элементом панели является, безусловно, главное меню, с которым и настало время ознакомиться подробнее. На первый взгляд оно ничем не отличается от обычных менюшек «пускового» типа, с их страшной многоступенчатой иерархией. Однако если вглядеться в соответствующий скриншот ещё раз – различия обнаруживаются, и весьма существенные:



Перво-наперво, обратим внимание на колонку пиктограмм вдоль левого края поля меню:

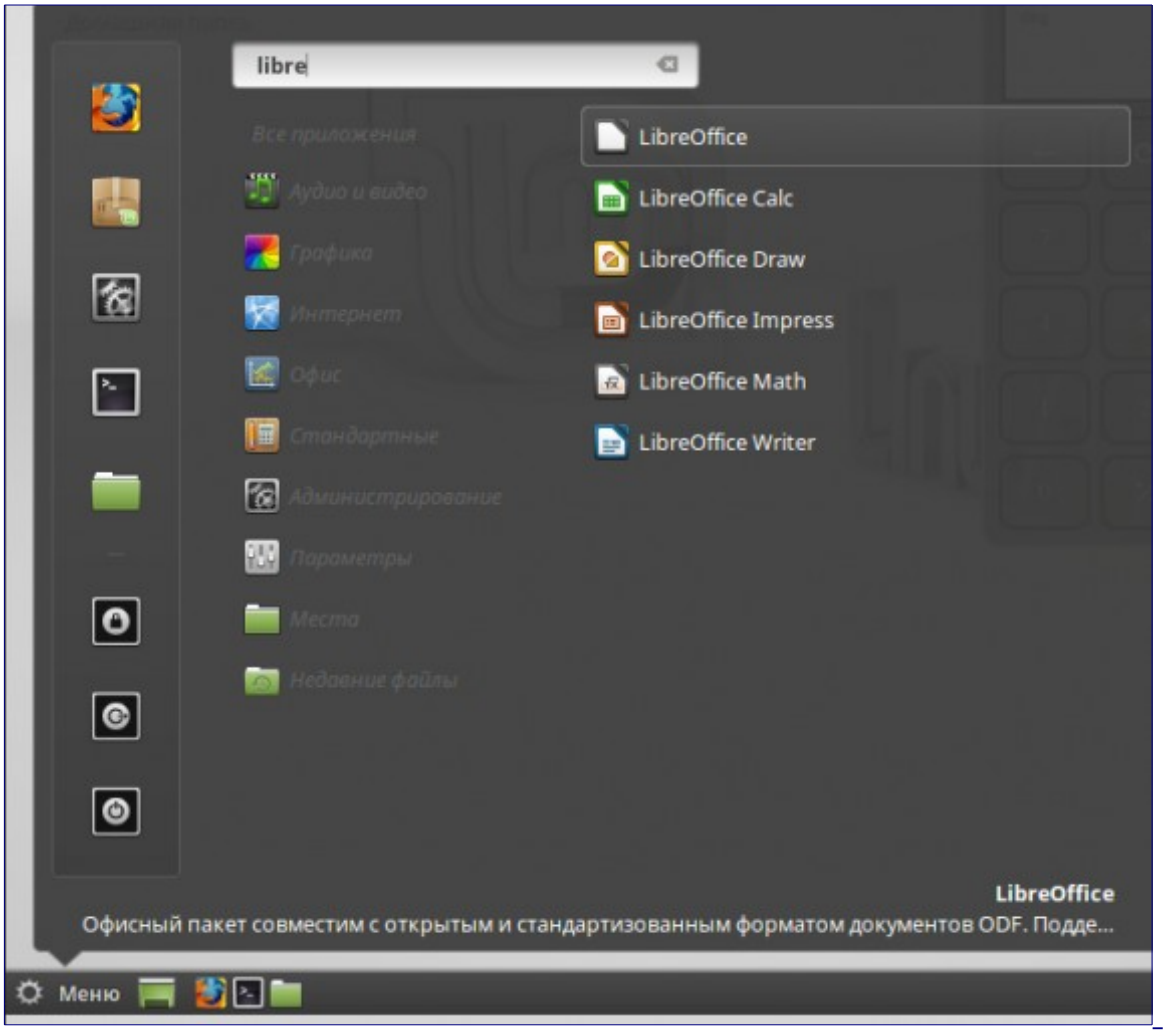


Набор их в сборке Cinnamon для Mint включает иконки для запуска:

- браузера Firefox;
- менеджера программ mintinstall;
- Центра управления (он же – Системные настройки);
- терминала;
- файлового менеджера Nemo.

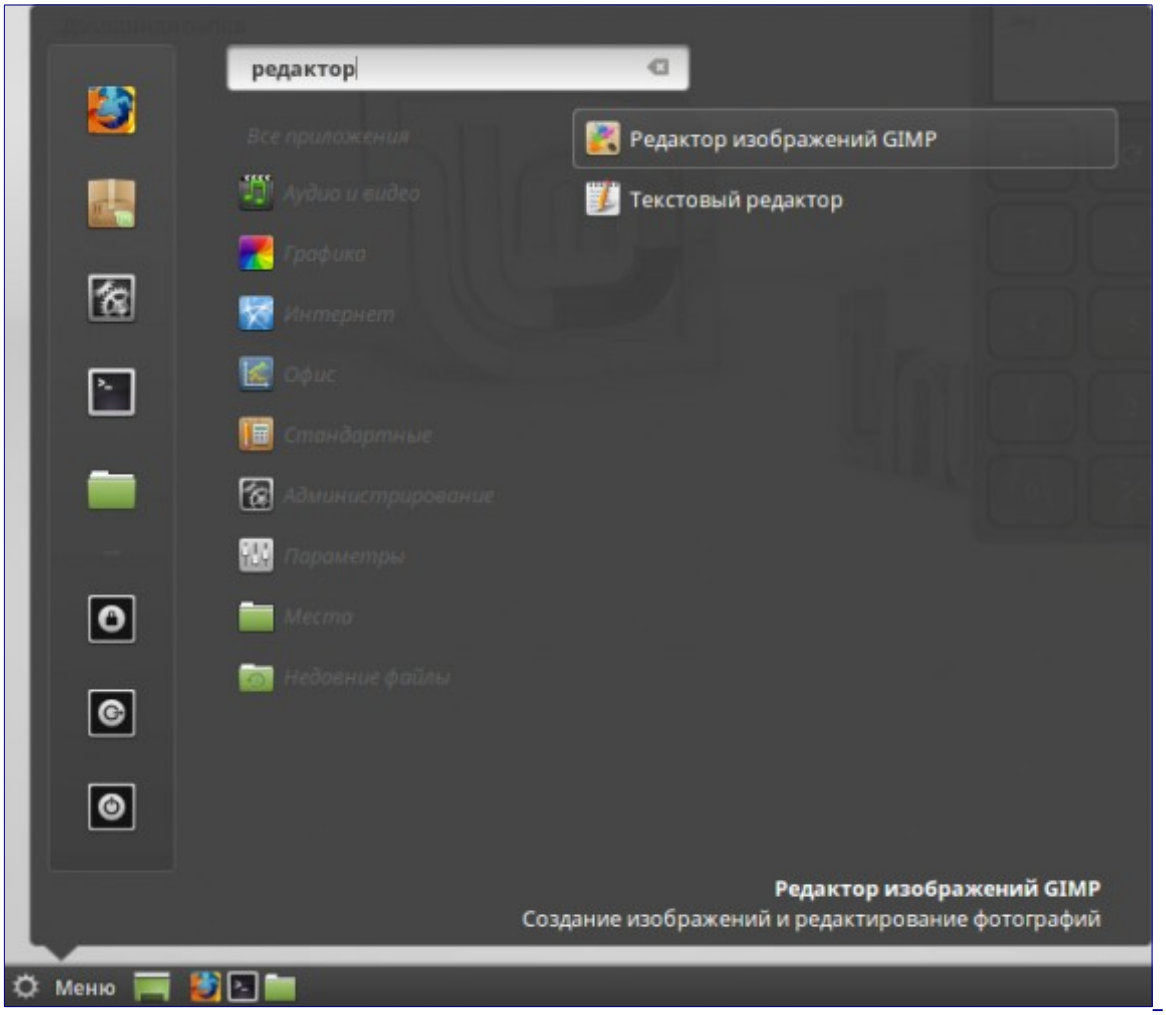
Кроме того, имеются значки блокировки экрана, выхода из сеанса и завершения работы. Конечно, к последним действиям можно получить доступ и через User Applet панели, но там его ещё надо разглядеть и до действий докопаться, а тут они всегда на виду.

Не менее примечательна поисковая строка в верхней части экрана – она предназначена для поиска приложений. Но не простого, а золотого инкрементного. То есть, введя в ней последовательность символов libre, мы получим список только тех пунктов меню, которые её содержат, то есть компонентов LibreOffice:

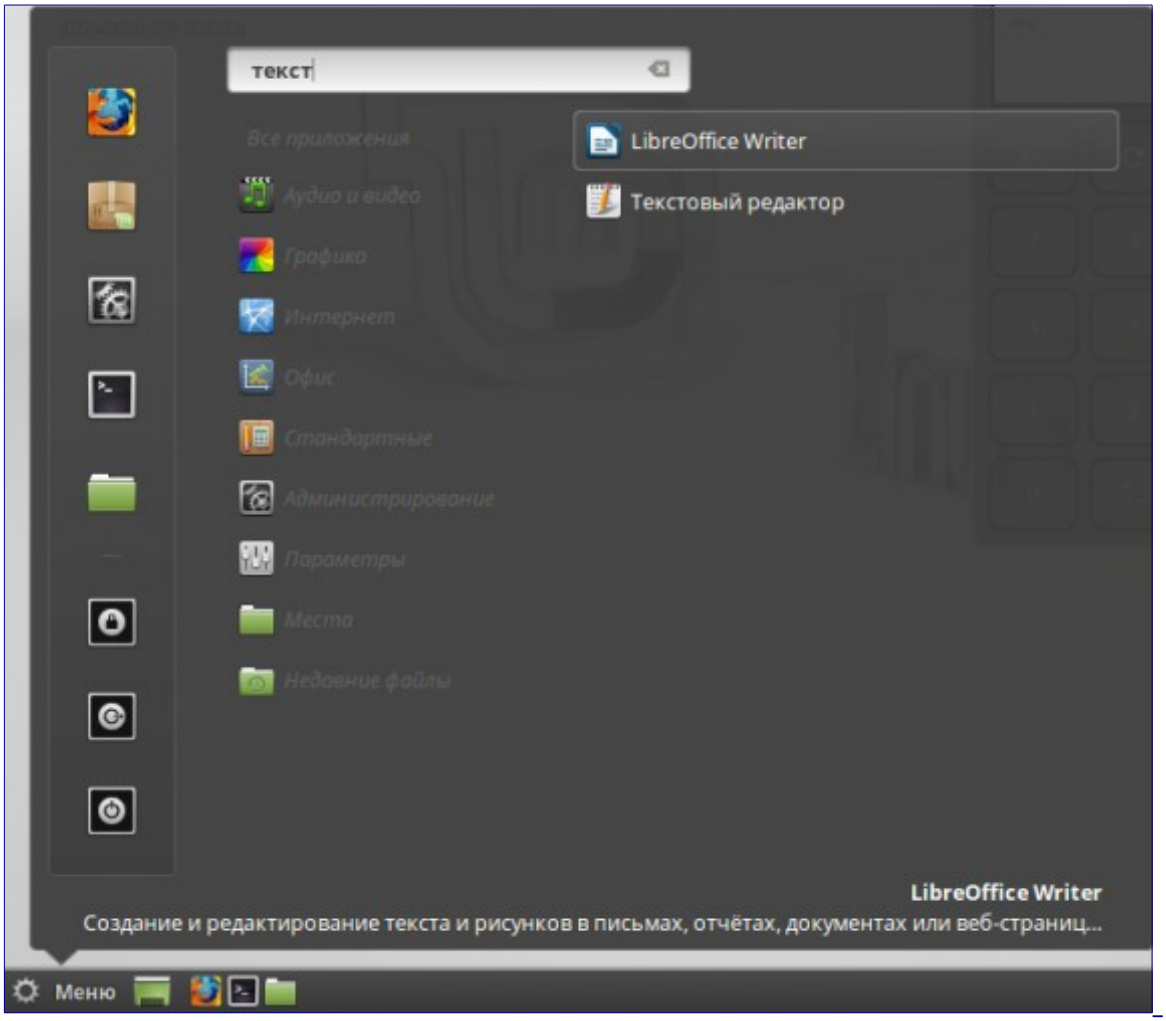


То есть строка поиска в меню Cinnamon тихо и скромно, без шума и пыли выполняет ту же функцию, что и пресловутый Dash в Unity (и его аналог в GNOME 3). Правда, с её помощью нельзя устанавливать программы или искать «парнуху» в Интернете – но не больно-то и хотелось.

А вот искать по русским словам – с некоторыми ограничениями можно. Введя, например, слово редактор, мы увидим приложения, содержащие его в своём имени:



А на слово текст ответом будет список приложений, содержащих его и в описании:



На мой взгляд, сказанного уже достаточно, чтобы проникнуться величием среды Cinnamon. Однако это ещё не всё – оно проявляется и в управлении окнами. Однако прежде чем говорить на эту тему – остановимся на вопросе, как эти окна открывать.

Вызов приложений

Открываются окна обычно вместе с запущенными в них приложениями. Способов же запуска последних в Cinnamon, как и во всех современных десктопах, несколько.

Первый, наиболее универсальный – запуск из командной строки терминала путём ввода соответствующей команды. Однако обычно полноразмерное терминальное окно использовать для этого не целесообразно, его вполне можно заменить панелью мини-терминала – она вызывается обычной для всех десктопов комбинацией клавиш Alt+2:



До некоторого времени панель мини-терминала в Cinnamon не блистала функциональностью – в ней не было ни автоподолнения, ни истории команд, имелась только возможность ввести команду руками или вставить её из

буфера. Однако в Cinnamon 2.6 все эти прелести появились. Историю команд можно, как и в шелле, просмотреть с помощью стрелок Up и Down. И автодополнение команды происходит после набора первых трёх символов её имени, в случае альтернатив выводя все доступные варианты:

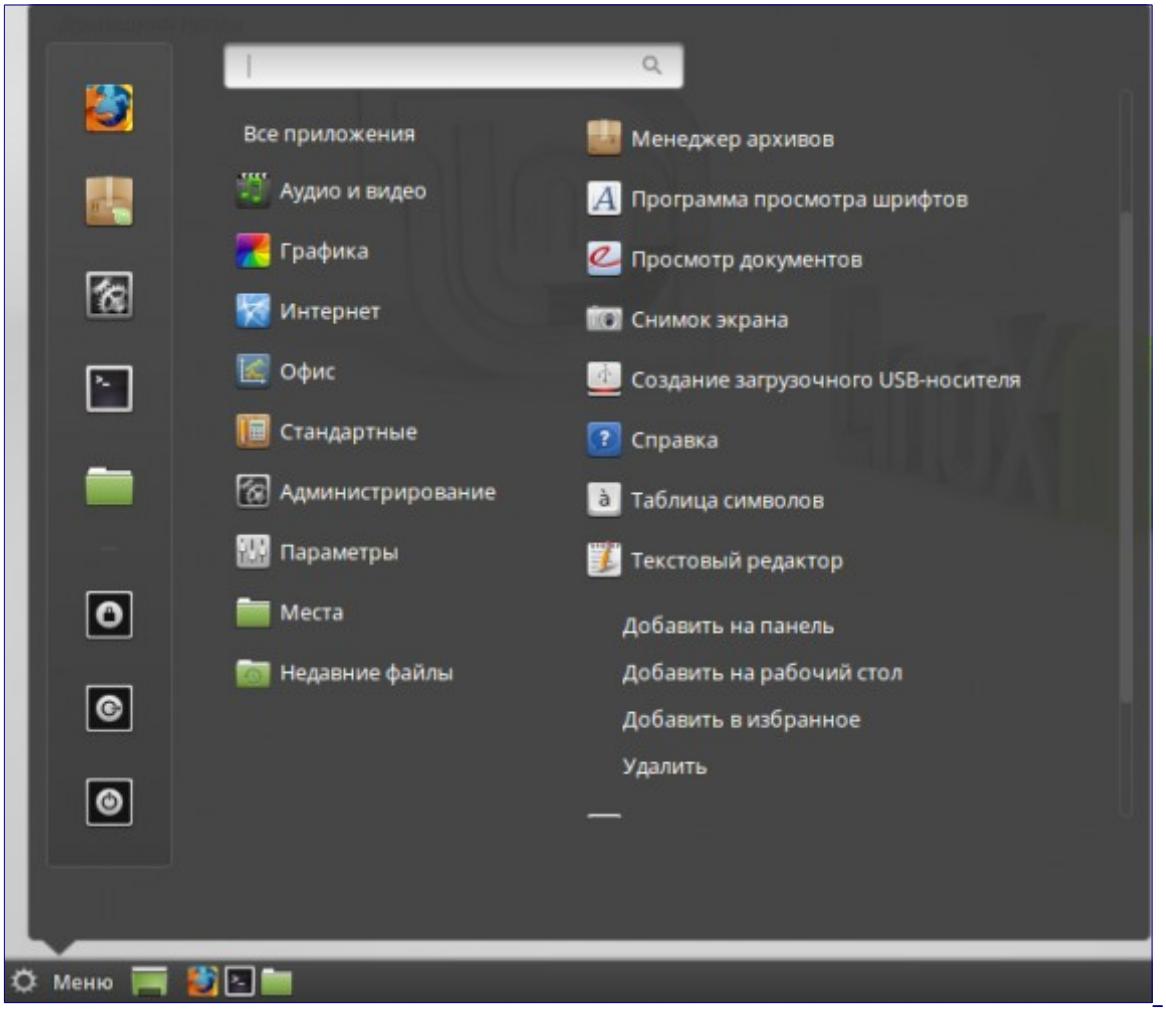


Правда, при вызове панели минитерминала блокируются как любые любые действия мышью, так и ввод с клавиатуры. Открывается она всегда по центру, и перемещёнию не поддаётся. Так что пользоваться этой панелью не всегда удобно. Однако её возможности сполна, а то и с лихвой, заменяются функциями строки инкрементного поиска в главном меню, о которой только что говорилось. Именно она становится основным инструментом запуска приложений для тех применителей, которым, как автору этих строк, проще набрать несколько символов из имени программы, нежели, подобно Баяну, «мысью рыскать» в её поисках по менюшному древу. Тем более, что привычные программы, вызываемые давно памятными командами, в меню, тем более русифицированном, могут носить самые неожиданные имена, и потому опознаваться с трудом.

Сразу по вызове главного меню мышью или хоткеем (в Cinnamon, как и в Unity, для этого по умолчанию зарезервирована левая win-клавиша, она же Super_L), строка поиска находится в фокусе ввода, и можно начинать набор имени приложения. Только нужно помнить про раскладку клавиатуры: как бы ни было настроено её наследование (о чём речь пойдёт в одном из следующих очерков), в этой строке она всегда будет наследоваться от раскладки последнего активного окна, а не от раскладки по умолчанию.

Разумеется, для запуска приложений можно пользоваться и главным меню непосредственно, тем более что некоторые из них (браузер, терминал, Центр управления, файловый менеджер), как только что говорилось, вынесены в нём на отдельную панель в виде кнопок быстрого запуска. А вот остальные программы в меню уже надо поискать – тем более что, повторяю, там они могут именоваться весьма причудливо.

Благо, пиктограммы наиболее востребованных приложений из меню можно поместить в Launcher на главной панели, на рабочий стол или в пункт Избранное. Для этого достаточно щёлкнуть правой кнопкой на имени нужной программы и из контекстного меню выбрать требуемый пункт:



Кроме того, иконки приложений можно просто перетаскивать мышью из меню в Launcher. Причём – сразу в желаемое его место, тогда как при автоматическом помещёнии иконок они попадают в его конец, и их перетасовка потребует дополнительных телодвижений.

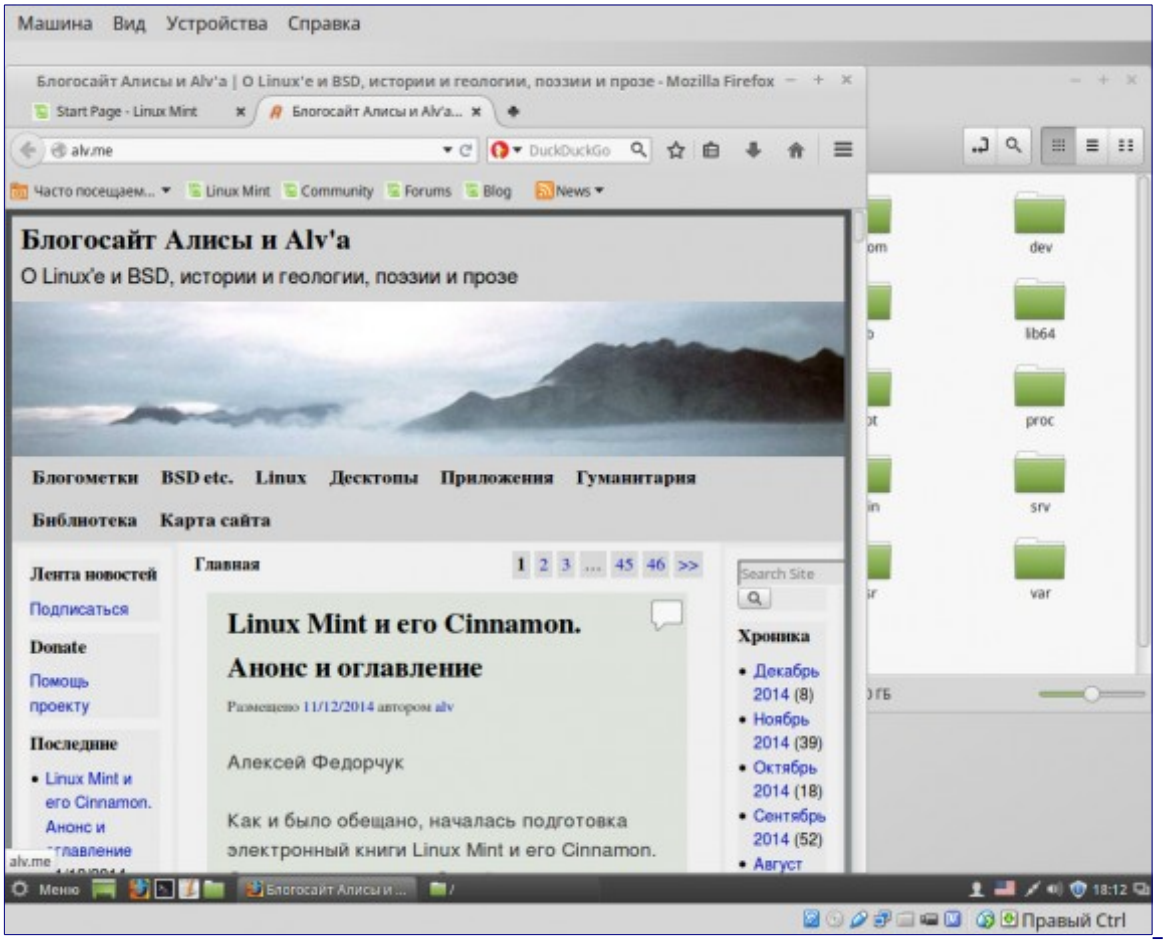
Как только что было сказано, пиктограммы запуска приложений можно поместить и на рабочий стол, и запускать их оттуда. Однако я этим способом никогда не пользуюсь — он кажется мне неудобным, и и рабочего стола у меня обычно не видать за распахнутыми окнами.

Вот теперь, разобравшись с запуском приложений и открытием вмещающих их окон, можно переходить и к управлению последними.

Управление окнами

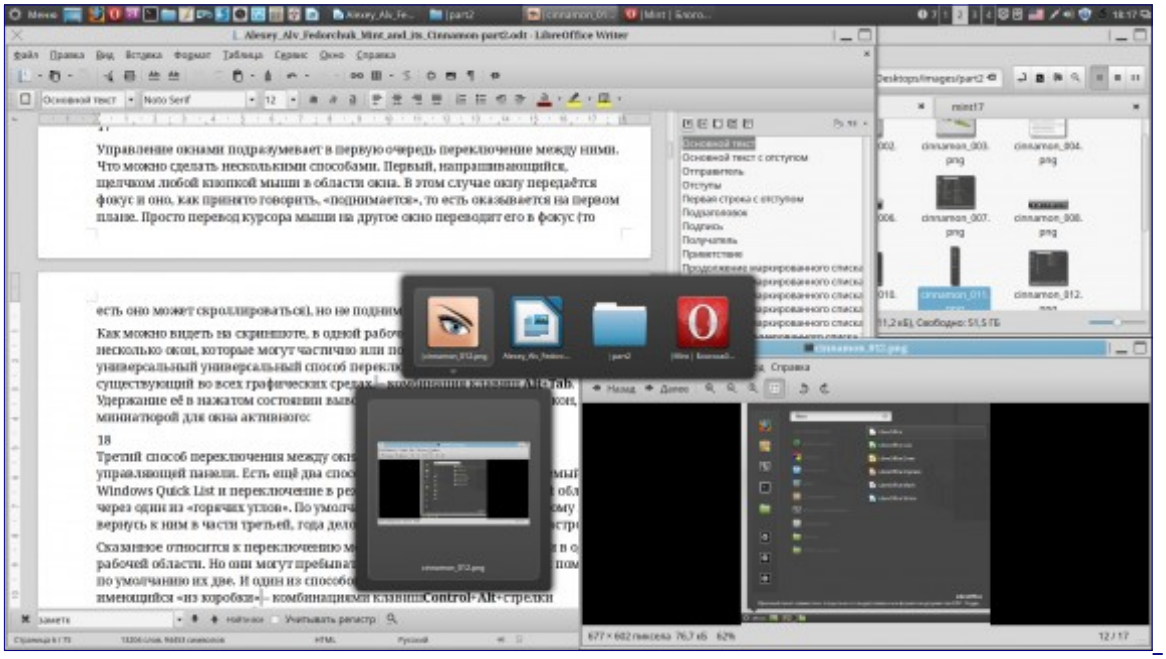
В предыдущем разделе очерка речь шла о способах запуска приложений, в этом же поговорим о способах управления приложениями, которые уже запущены. Поскольку мы (пока ещё) живём в системе, которая официально называется X Window System, то большая часть приложений запускается в её окнах. Так что в основном применителю придётся иметь дело с ними.

Вид окон с запущенными приложениями зависит от темы рабочего стола, стиля окон и других индивидуальных настроек. Но по умолчанию они выглядят примерно так:



Управление окнами подразумевает в первую очередь переключение между ними. Что можно сделать несколькими способами. Первый, напрашивающийся, щелчком любой кнопкой мыши в области окна. В этом случае окну передаётся фокус и оно, как принято говорить, «поднимается», то есть оказывается на первом плане. Просто перевод курсора мыши на другое окно переводит его в фокус (то есть оно может скроллироваться), но не поднимает.

Как можно видеть на скриншоте, в одной рабочей области может быть открыто несколько окон, которые могут частично или полностью перекрываться. И тогда универсальный универсальный способ переключения между окнами, существующий во всех графических средах, – комбинация клавиш Alt+Tab. Удержание её в нажатом состоянии выводит ленту значков открытых окон, с миниатюрой для окна активного:

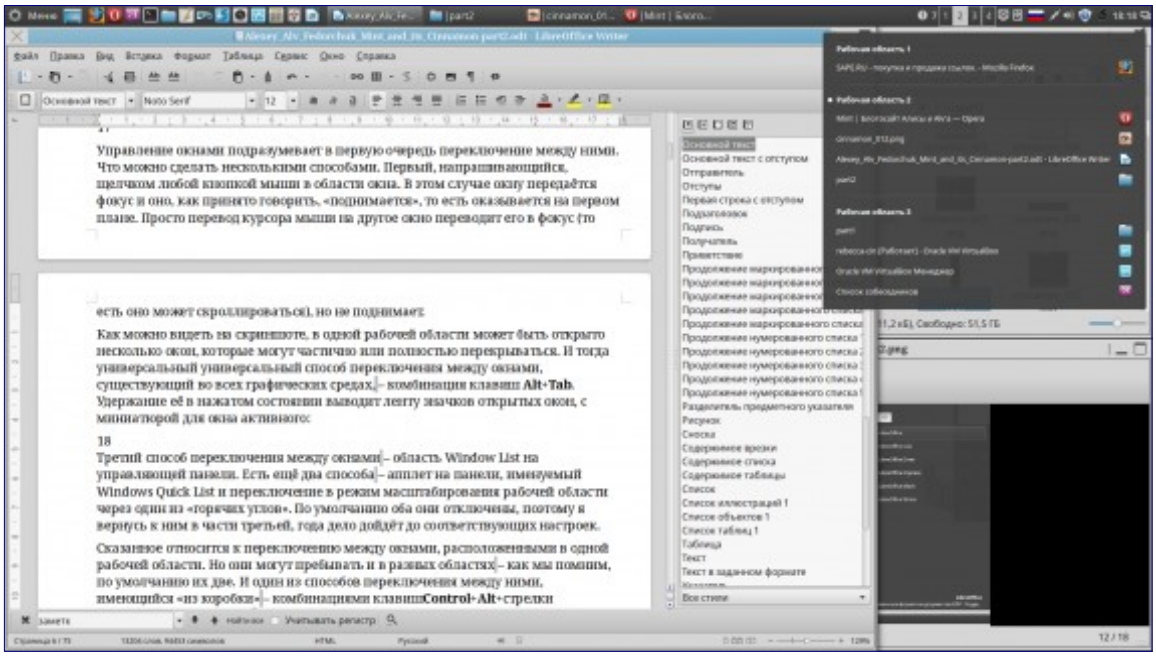


Третий способ переключения между окнами – область Window List на управляющей панели:



Есть ещё переход в режим масштабирования рабочей области, но по умолчанию этот способ отключен, поэтому я вернусь к нему чуть позже.

Сказанное относится к переключению между окнами, расположенными в одной рабочей области. Но они могут пребывать и в разных областях – как мы помним, по умолчанию их две. И в этом случае один из способов переключения между ними, имеющийся «из коробки» – апплет Windows Quick List (в последних версиях Cinnamon он помещён на панели по умолчанию):



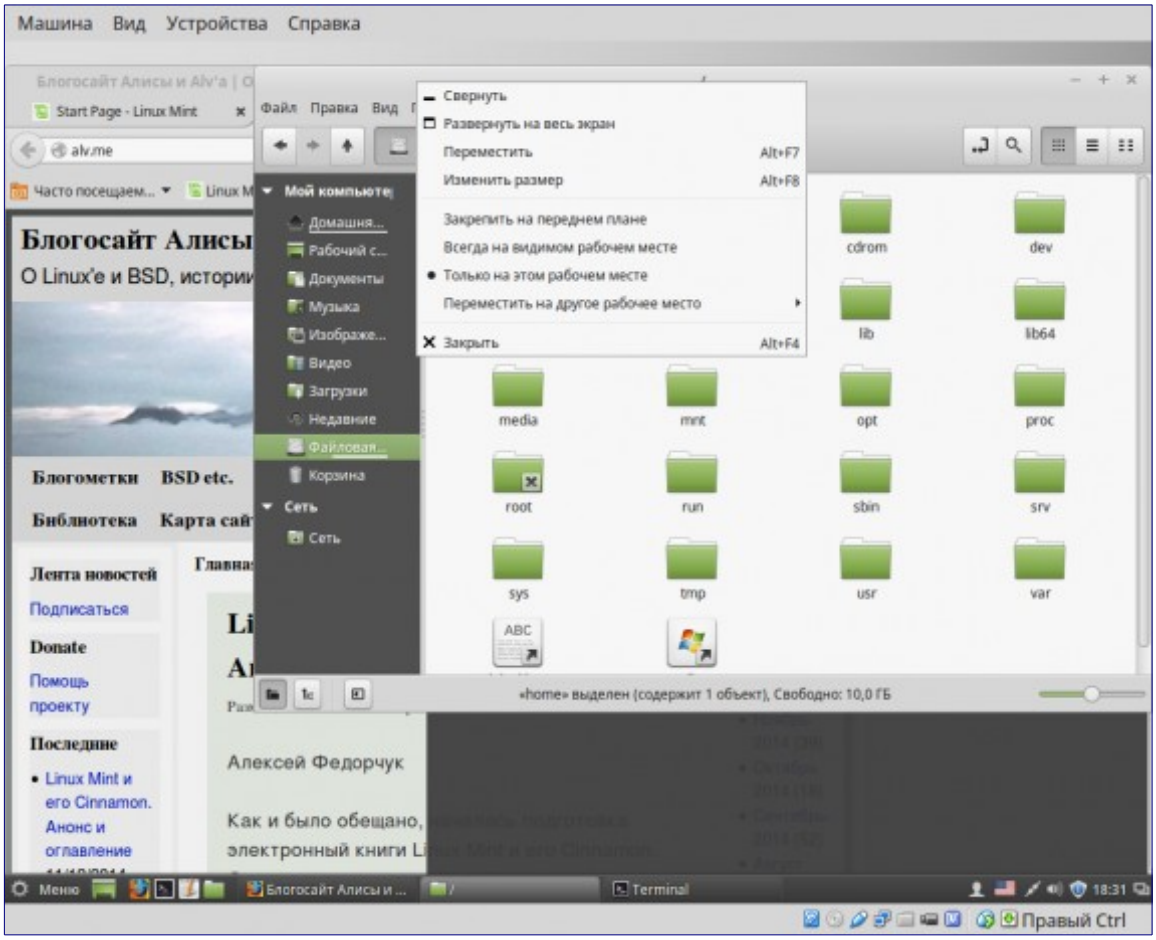
Второй же – переход в режим Expo через один из «горячих углов», о чём также будет говориться вскоре.

Теперь посмотрим, что можно делать с самими окнами. В строке заголовка

каждого окна, кроме самого заголовка, в правой его части можно видеть три управляющие кнопки (слева направо): сворачивания, максимизации/восстановления исходного размера, закрытия – назначение их очевидно.

Кое-какие манипуляции с окнами можно выполнять и кликами мыши по строке заголовка. Так, двойной щелчок левой её кнопкой вызывает максимизацию окна, повторение его – восстанавливает исходный размер. Тот же двойной клик, но уже правой кнопкой сворачивает окно на панель задач. Для средней кнопки предусмотрен только одинарный клик – он «опускает» окно на задний план.

Лишь закрыть окно нельзя кликами мыши по строке заголовка. Но это делается (если не обращаться к штатному меню запущенного в окне приложения) комбинацией клавиш Alt+F4 – подобно Alt+Tab, она также универсальна для почти всех графических сред (или вообще всех современных?). Кроме того, закрыть окно можно из его собственного меню – оно вызывается щелчком правой кнопки мыши по строке заголовка, и содержит такие пункты:



Назначение первых четырёх и последних двух пунктов абсолютно понятно. А вот о тёх «средних» пару слов сказать можно. Отметка Закрепить на переднем плане – это запрет перекрытия данного окна другими. А пункты Всегда на видимом рабочем месте и Только на этом рабочем месте (включён по умолчанию) – альтернативны: при включении первого окно будет «кочевать» вслед за перемещёниями пользователя по рабочим областям.

Тайлинг окон

Всё, что было только что сказано относительно управления окон, не

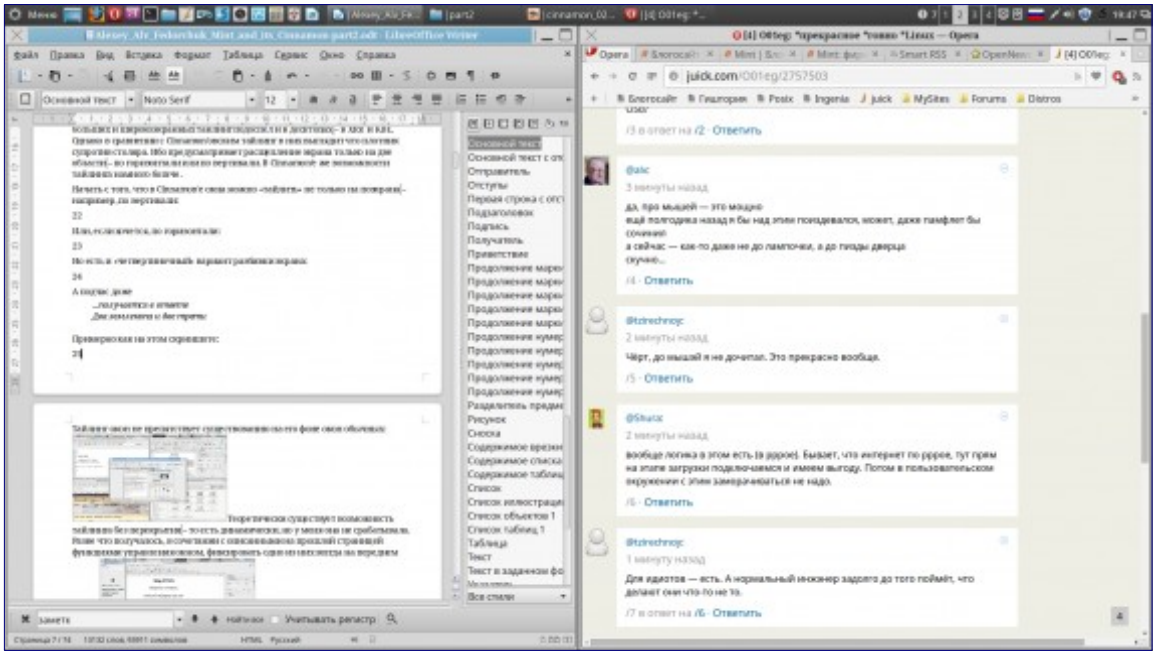
покажется чем-то необычным применителю любого современного десктопа и приверженцам подавляющего большинства оконных менеджеров. А вот сейчас речь пойдёт о вещи более неожиданной – о тайлинге окон. Это – та самая вторая особенность (после строки поиска в меню), которая столь восхитила меня в Cinnamon'е.

Для начала – пара слов о том, что такое тайлинг. Он основывается на той же идее, что и консольная утилита screen или двухпанельные файловые менеджеры – потомки командира Norton'а – расщеплении экрана на ряд независимых областей, в каждой из которых локализуется окно с запущенным в нём приложением. Это подобно покрытию пола кафелем (tiling), чем и порождена аллюзия.

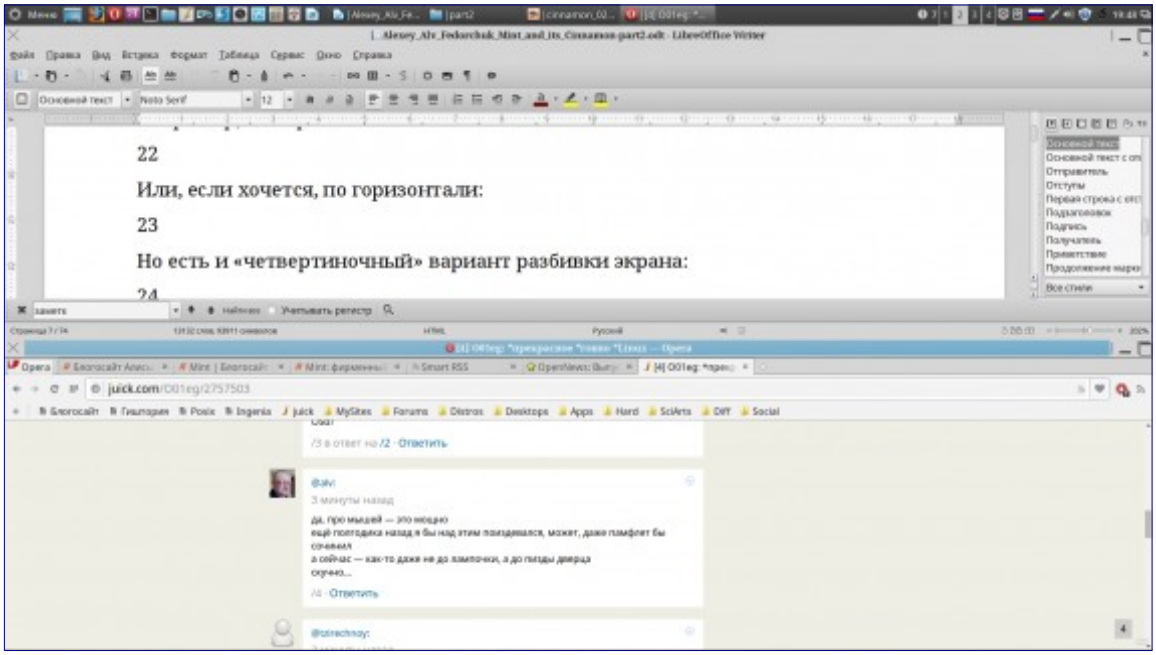
При этом понятие управления окнами как бы лишается смысла – тайлинговые системы управляют не столько окнами, сколько теми самыми областями экрана, в которых окна открываются. Области эти (в чём-то они похожи на фреймы, некогда популярные среди web-дизайнеров) могут быть статическими, с жёстко определёнными размерами, и динамическими, при котором их размеры изменяются при масштабировании окон запущенных в них приложений. В Cinnamon реализована первая модель.

Конечно, тайлингом удивить пользователей менеджеров окон типа Awesome сотоварищи не легче, чем испугать ежа голой... эээ... спиной. Однако во времена не очень больших экранов я этой идеей не проникса (парадигма «одно приложение – одна рабочая область» была мне ближе). А ко времени мониторов больших и широкоэкранных тайлинг подоспел и в десктопах – в Xfce и KDE. Однако в сравнении с Cinnamon'овским тайлинг в них выглядит что плотник супротив столяра. Ибо предусматривает расщепление экрана только на две области – по горизонтали или по вертикали. В Cinnamon'е же возможности тайлинга намного богаче.

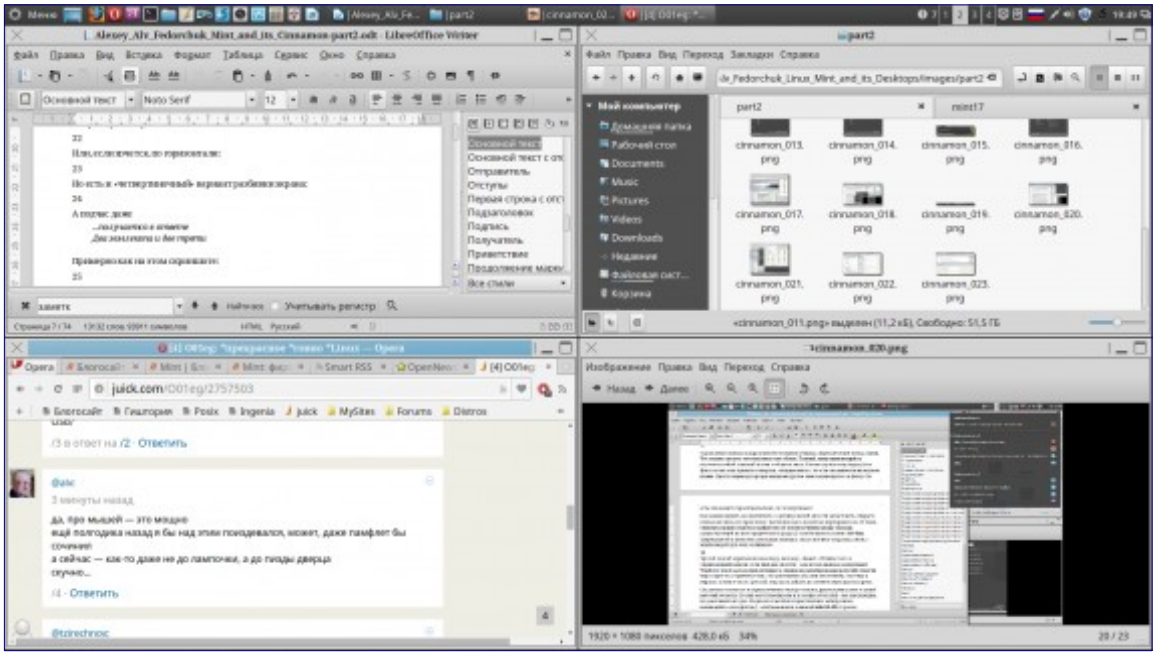
Начать с того, что в Cinnamon'е окна можно «тайлить» не только на поэкрана – например, по вертикали:



Или, если хочется, по горизонтали:



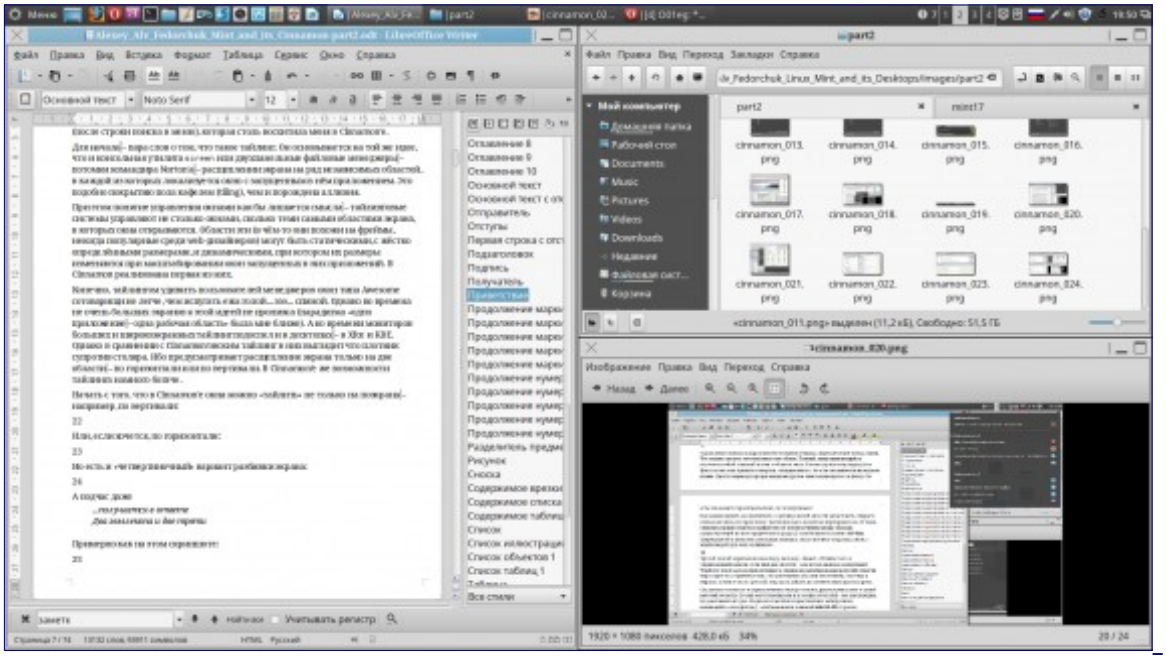
Но есть и «четвертиночный» вариант разбивки экрана:



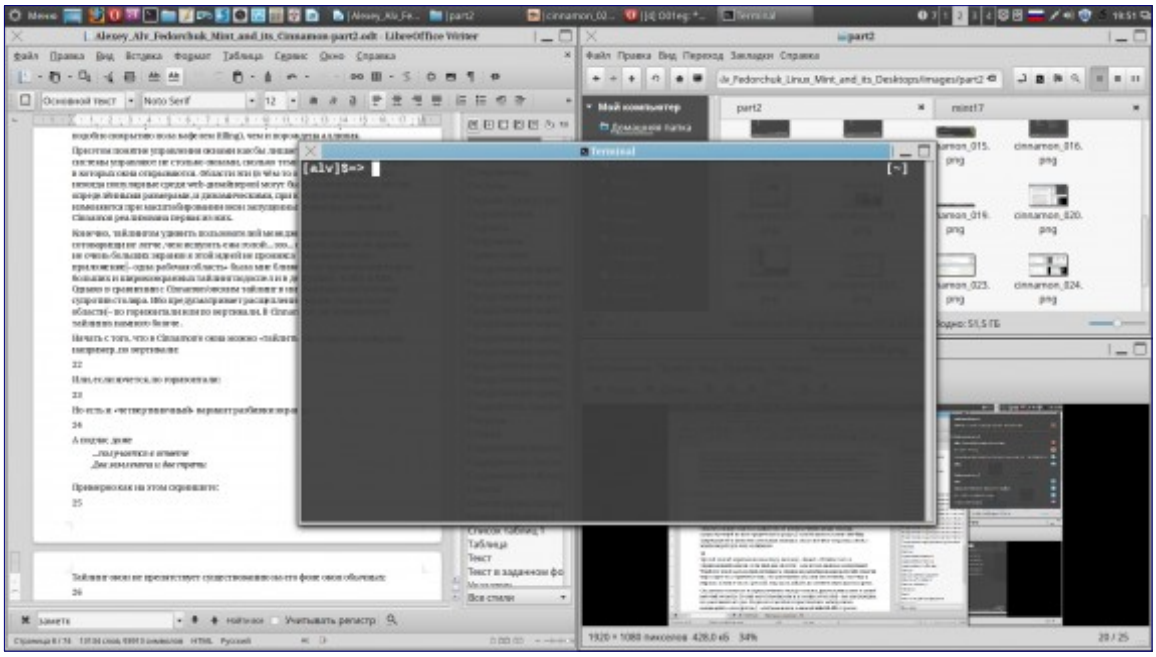
А подчас даже

...получается в ответе Два землекопа и две трети

Примерно как на этом скриншоте:



Тайлинг окон не препятствует существованию на его фоне окон обычных:

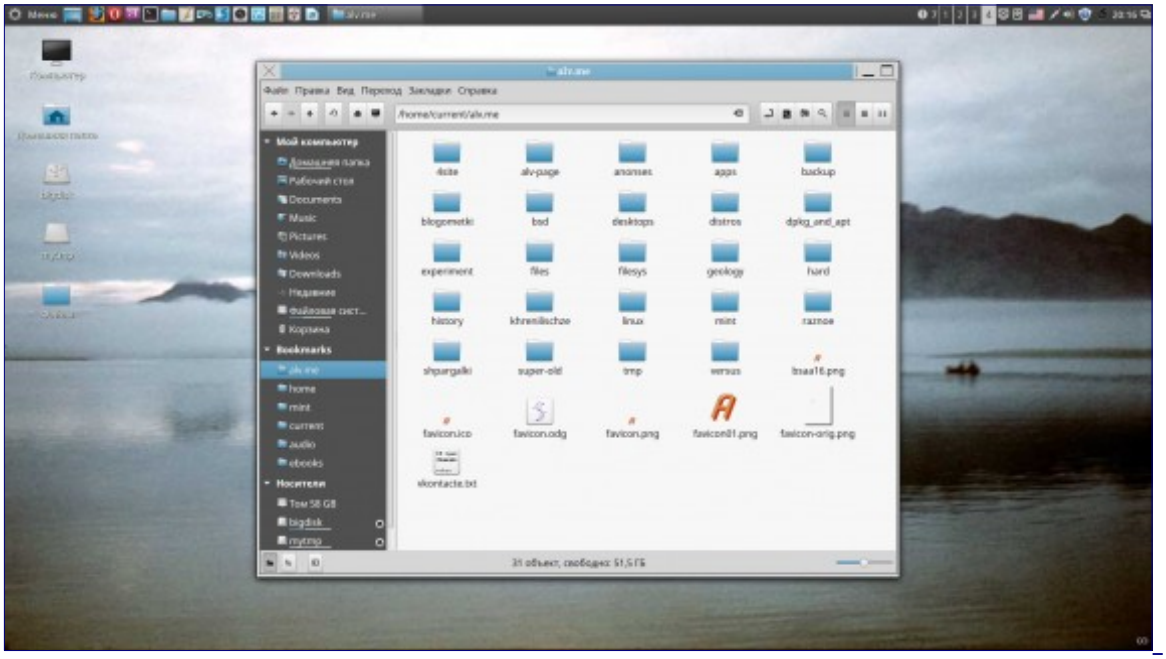


Вот только, к сожалению, средства «тайлить» окна на произвольные области экрана на предусмотрено. Однако и без этого область его использования достаточно широка, в чём мы убедимся после рассмотрения средств управления тайлингом.

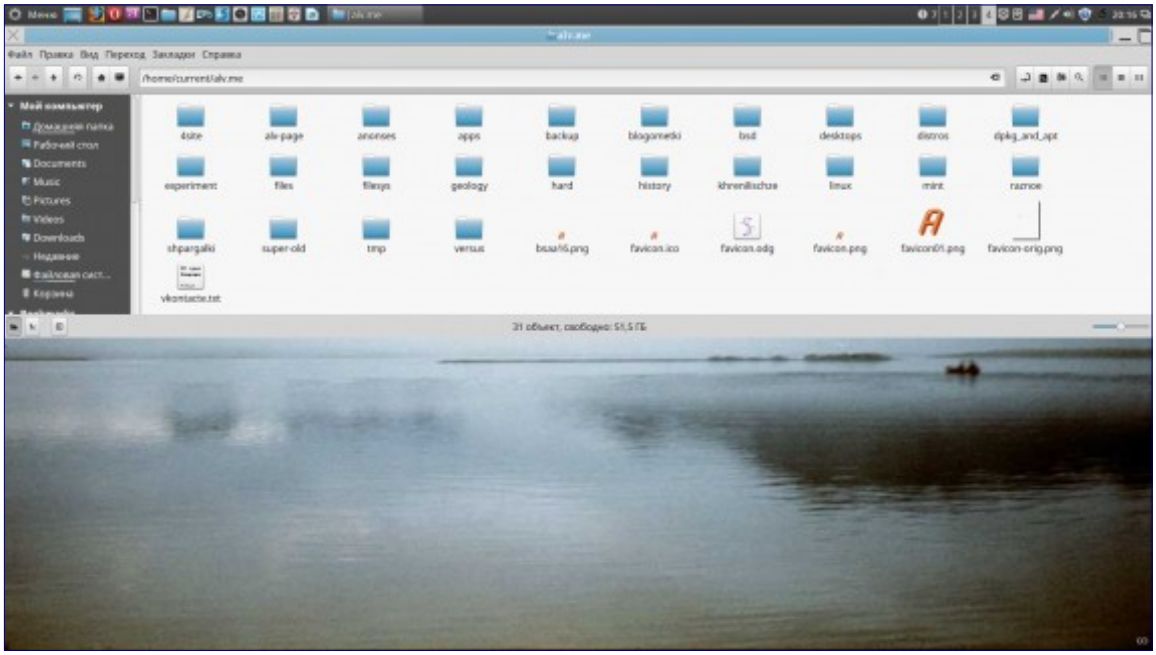
Управление тайлингом

Тайлинг окон может выполняться двумя способами – посредством мыши и с клавиатуры. Как обычно, первый – легче, то есть «ленивей», второй – быстрее и эффективней. Замечу в скобках: как обычно, это вовсе не означает оценки в терминах товарища Маяковского. Иногда «хорошо» – это лениво развалясь в кресле, елозить мышью по экрану, а иногда – напрягать пальцы рады быстроты выполнения неких действий.

Рассмотрим сначала тайлинг мышью, выполняемый над окном с положением и размерами, соответствующими умолчаниям десктопа:



При подтаскивании окна мышью к верхней границе экрана оно занимает верхнюю же его половину:



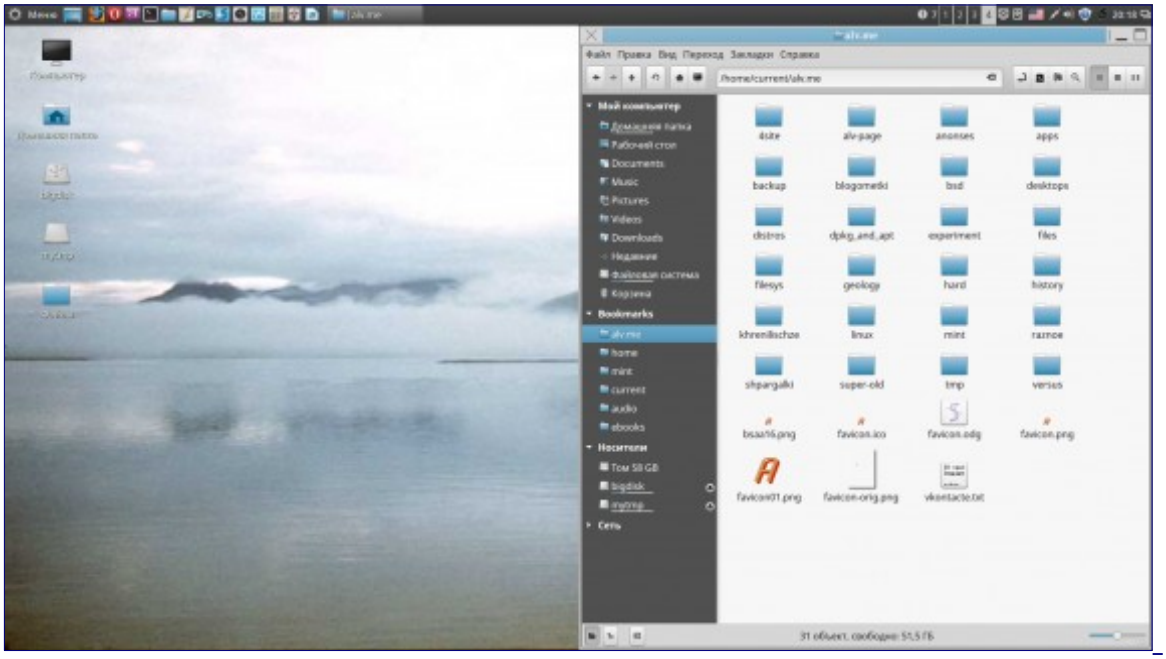
Аналогичное движение к нижней границе экрана разворачивает окно на нижнюю его половину:



Перемещёние окна к боковой стороне экрана «тайлит» его на левую



или правую половины, в зависимости от стороны «подтаскивания»:



Если передвинуть окно в любой из углов дисплея, оно займёт соответствующую его «четвертинку»:



То есть всё просто, но... Предположим, что при сочинении текста в текстовом редакторе (или word-процессоре) появилось желание параллельно бросить взгляд на картину, призванную этот текст иллюстрировать: в этом случае тайлинг посредством мыши потребует отрыва руки от клавиатуры и переноса её на спину грызуна. И вот тут-то на помощь и придёт тайлинг с клавиатуры, который осуществляется комбинацией из двух пальцев – клавиши Super (она же – левая Win) и одной из стрелок управления курсором.

Опять же подвергнем издевательствам исходное окно с умолчальными параметрами. Комбинация Super+Right развернёт её на правую половину экрана, Super+Left – вернёт в исходное состояния. Из которого комбинацией Super+Left оно развернётся на левую половину, а последующий Super+Right – возвратит взад. Из положения «половинка справа» комбинация Super+Up превратит окно в «четвертинку» в правом верхнем углу, Super+Down – «четвертинку» в правом нижнем.

Принцип, я думаю, понятен: Super плюс стрелка в любую сторону – окно на соответствующую половинку экрана, Super плюс стрелка в обратную – возврат в исходное положение, Super плюс стрелка из «половинного» состояния – перевод в состояние «четвертиночное». Запомнить это не сложно, навык до рефлекторного уровня приобретается очень быстро. А как его можно применить на практике – сейчас увидим.

Тайлинг на практике

Всё это очень блаародно – вправе сказать читатель, – но в чём преимущество тайловых окон перед обычными? Долгое время ответа на этот вопрос у меня не было, тем более, что я окнами почти не пользовался: во всё ширь каждого десктопа у меня было открыто одно приложение. Пока не опробовал тайловый режим – сначала в Xfce 4.10, где он незатейлив, а затем и в среде Cinnamon, в которой, как я пытался показать на предыдущих страницах, он куда более изощрён.

Тем не менее, объяснить преимущества тайловых окон над масштабированными довольно трудно – это надо попробовать самому и оценить. Для меня оно выразилось в возможности мгновенно перейти от сочинения текста в полноэкранном режиме к режиму параллельного просмотра текста, иллюстраций к нему, файловой иерархии и так далее. И сделать это лёгким движением даже не руки, а двух пальцев.

Причём параллельным просмотром нескольких окон можно не ограничиваться. А, например, перетаскивать мышью или просто копипастить горячими клавишами картинки из файлового менеджера или вьювера изображений в word-процессор. Аналогичным образом можно перетаскивать из теста документации в командную строку терминала примеры командных конструкций. Или, наоборот, команды или исходники сценариев – помещать в сочиняемый текст.

Конечно, всё это можно сделать и при традиционном расположении окон, но прошу поверить на слово брату незабвенного Голубкова:

Так лучше, Лёня.

В общем, обращение с содержимым окон становится похожим на работу с двухпанельными файловыми менеджерами a la Midnight Commander. Или даже с четырёхпанельными: возможно, кое-кому из читателей памятен «пай-мальчик» – Pie Commander, незаконный четырёхглазый отпрыск командира Norton'a...

Правда, в двух- и тем более в «четырёхплиточной черепице» становится мучительно больно за бесцельно расходуемое экранное пространство, занятое в каждом окне строкой меню. И возникает сожаление об отсутствии возможности встроить его в главную управляющую панель, как это делается по умолчанию в Unity. Но увы – нет в жизни совершенства, ибо нет в Cinnamon такой возможности. Иначе он безусловно достиг бы высшей степени совершенства, а его разработчики имели бы полное право впасть в нирвану.

Интерфейс Cinnamon: краткий итог

В этом очерке были описаны особенности интерфейса Cinnamon, которые представляются мне наиболее существенными. Пора подводить итоги.

В своём современном виде Cinnamon наделён всеми функциями, присущими остальным десктопам. Причём реализованными если не идеально, то близко к тому. А две из них в сочетании оказываются почти уникальными. Это – строка инкрементного поиска в меню и развитый тайлинг. Обе они представлены и в Unity, и Xfce, и в KDE. Но в первой среде пресловутый Dash как раз функционально перегружен, предусматривая поиск в Интернете не только всякой ерунды, типа программ и пакетов, но и весьма важной и разнообразной «парнухи». В Xfce строка поиска по меню как раз зарыта в дебрях её минитерминала. Ну а в KDE эта строка есть только в «меню современного вида», которое само по себе многим (в том числе и автору этих строк) представляется неудобным. Что же до тайлинга – как я уже отметил, его реализация в Cinnamon среди всех десктопов превзойдённа.

При этом всё сказанное в этой части касалось особенностей Cinnamon'а по умолчанию. Но особенно ярко они заиграют после соответствующих настроек, которым будут посвящены следующие очерки.

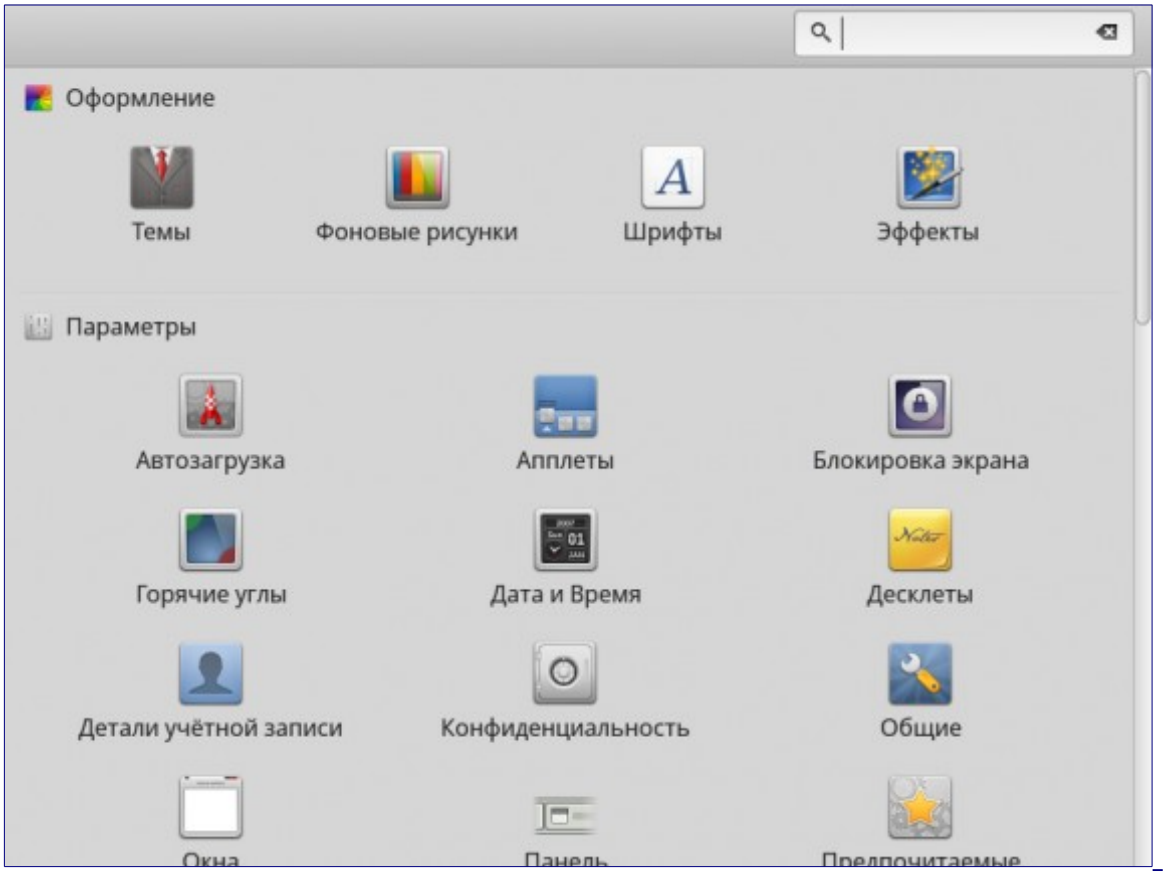
Cinnamon и системные настройки

Следующая серия очерков посвящена описанию способов, как штатных, так и не очень, которые изменяют те умолчания Mint и Cinnamon, что были рассмотрены в очерке предыдущем.

Системные настройки: вступление

Подавляющее большинство настроек в Cinnamon выполняется из панели его Системных настроек (cinnamon-settings), именуемых также Центром управления. С ними тесно интегрированы фирменные утилиты Mint, имеющие отношение к конфигурированию системы. Однако они будут предеметом отдельного очерка.

Пиктограмма запуска Системных настроек занимает почётное место в левой колонке главного меню управляющей панели. Через один-два клика мышью до них можно добраться и из контекстных меню как самой панели, так и рабочего стола. И при первом запуске выглядит он так:



Ранее Системные настройки имели два режима — нормальный и расширенный, и реликты такого положения можно найти в сетевых материалах. Однако, начиная с Cinnamon версии 2.2, эта дискриминация ликвидирована, и расширенный режим стал нормой жизни применителя.

Если пролистать окно Системных настроек до конца, можно увидеть, что отдельные её модули группируются в четыре секции:

- Оформление
- Параметры
- Оборудование
- Администрирование

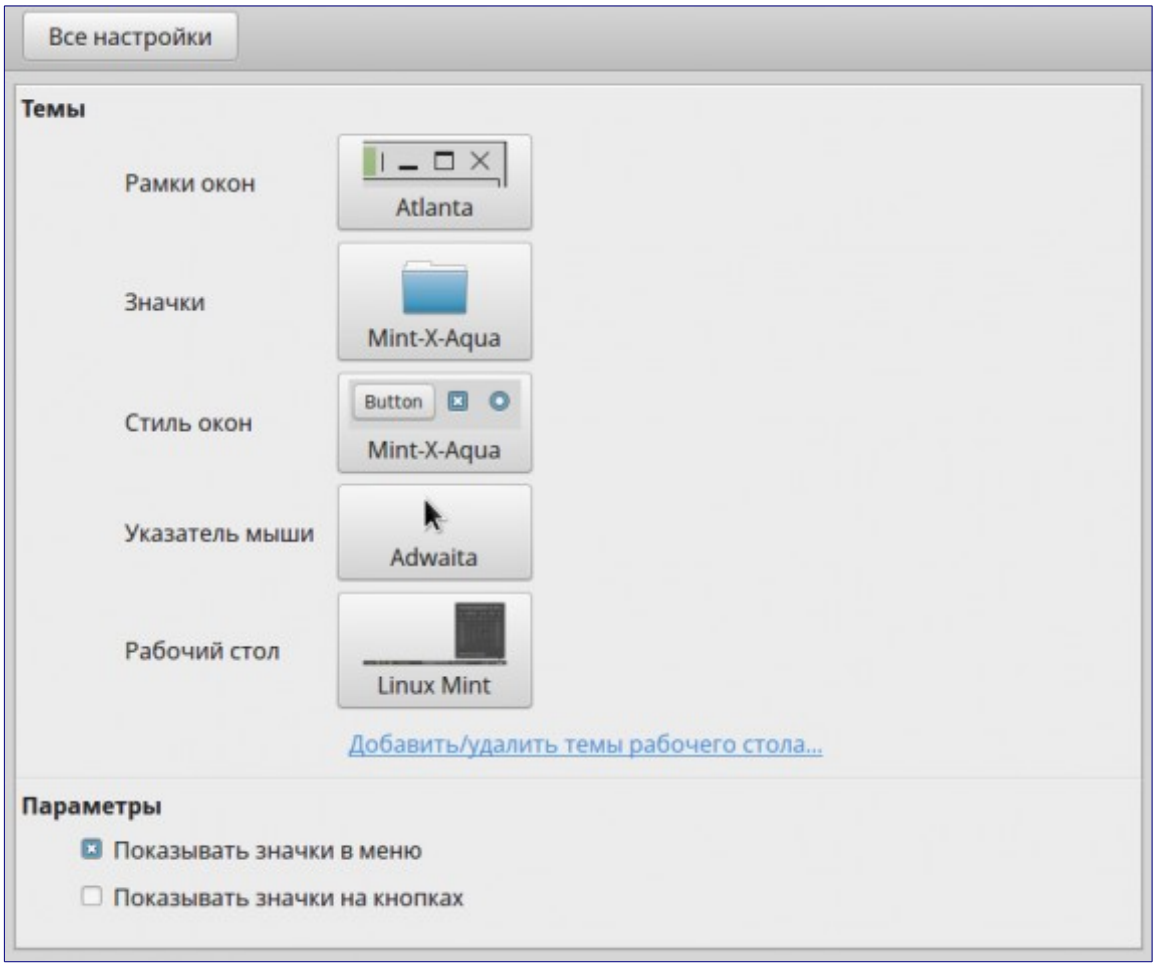
Далее я последовательно рассмотрю возможности каждого модуля примерно в том порядке, в котором они следуют в русскоязычном варианте Cinnamon. Отклоняясь от него, когда когда того потребует логика.

Секция Оформление

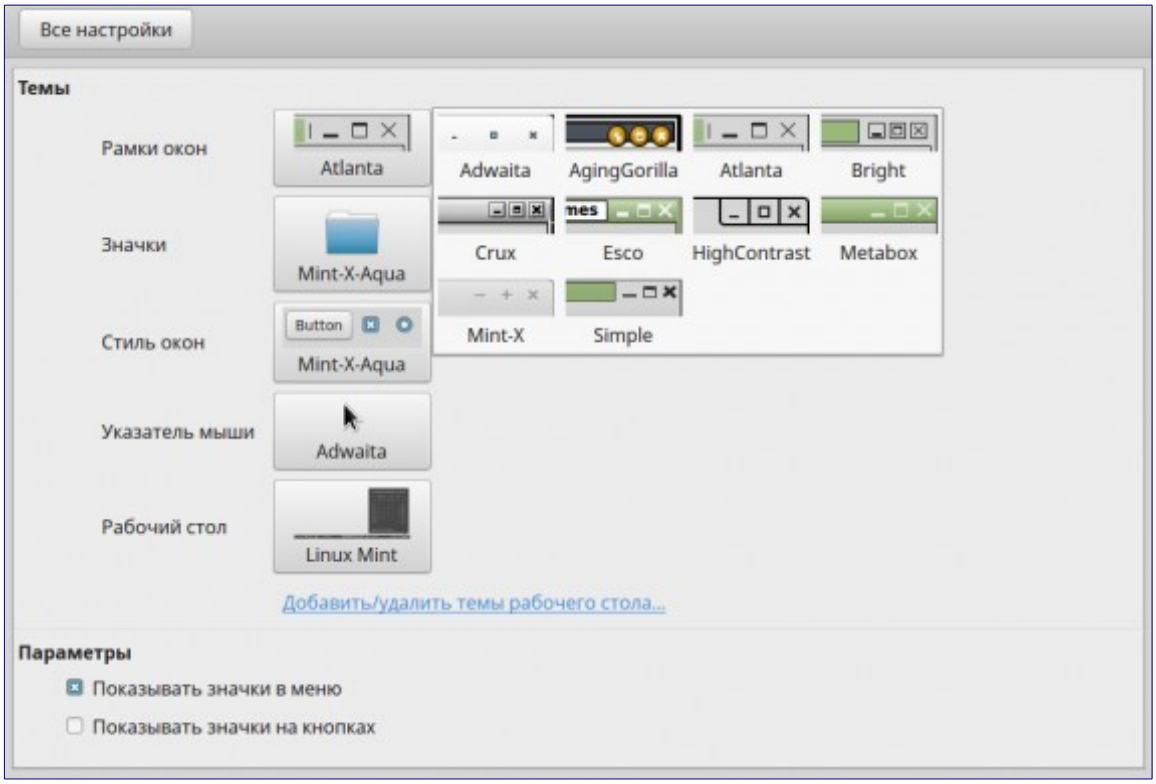
Секция Оформление включает в себя модули:

- Темы;
- Фоновые рисунки;
- Шрифты;
- Эффекты.

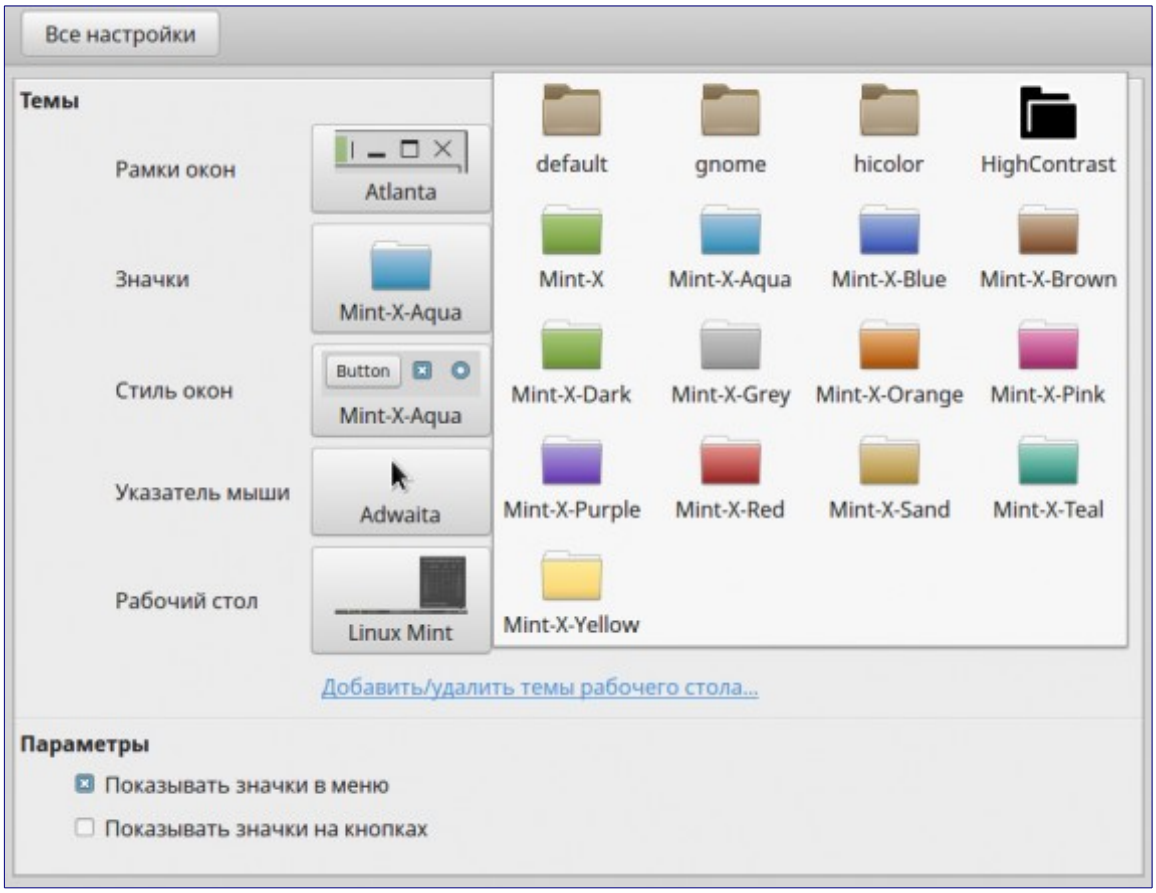
Модуль Темы выглядит таким образом:



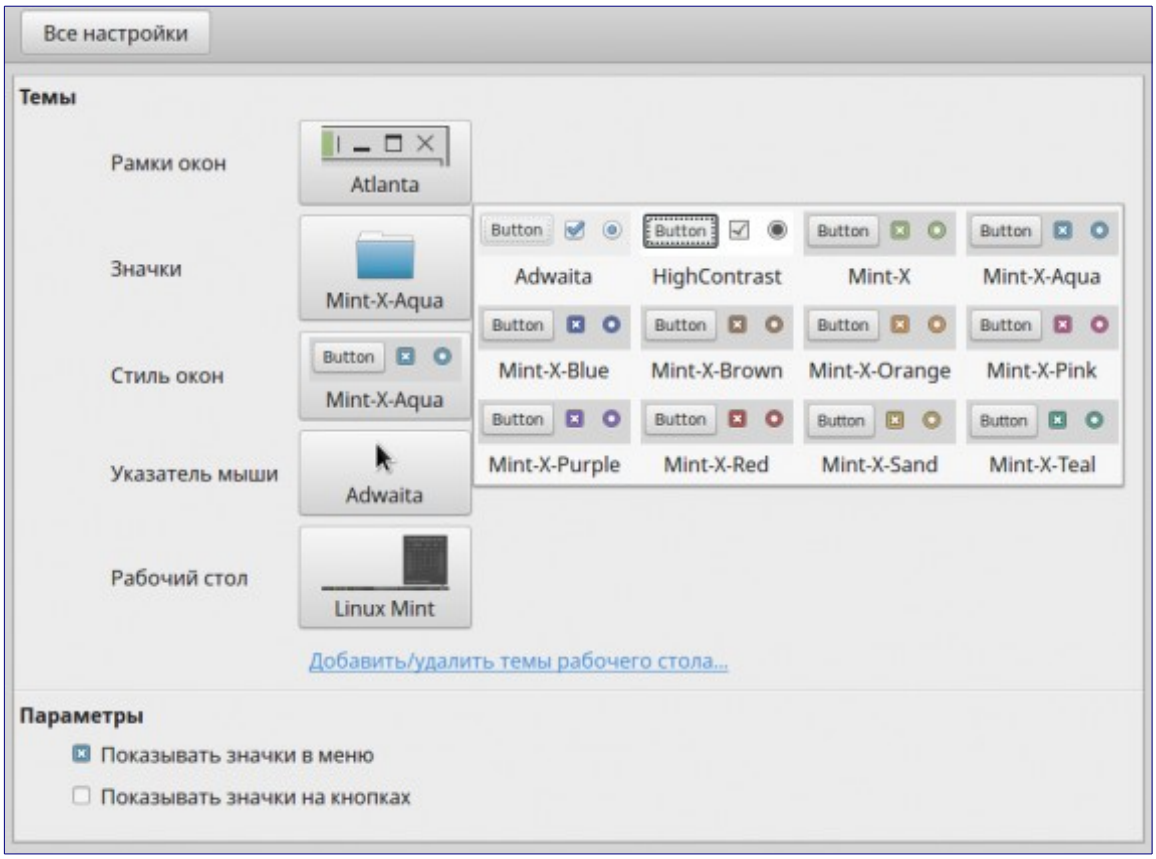
Организация его подчёркивает, что стилевое оформление отдельных элементов интерфейса полностью независимо друг от друга и от темы рабочего стола. Это можно проиллюстрировать серией скриншотов для стилей рамок окон:



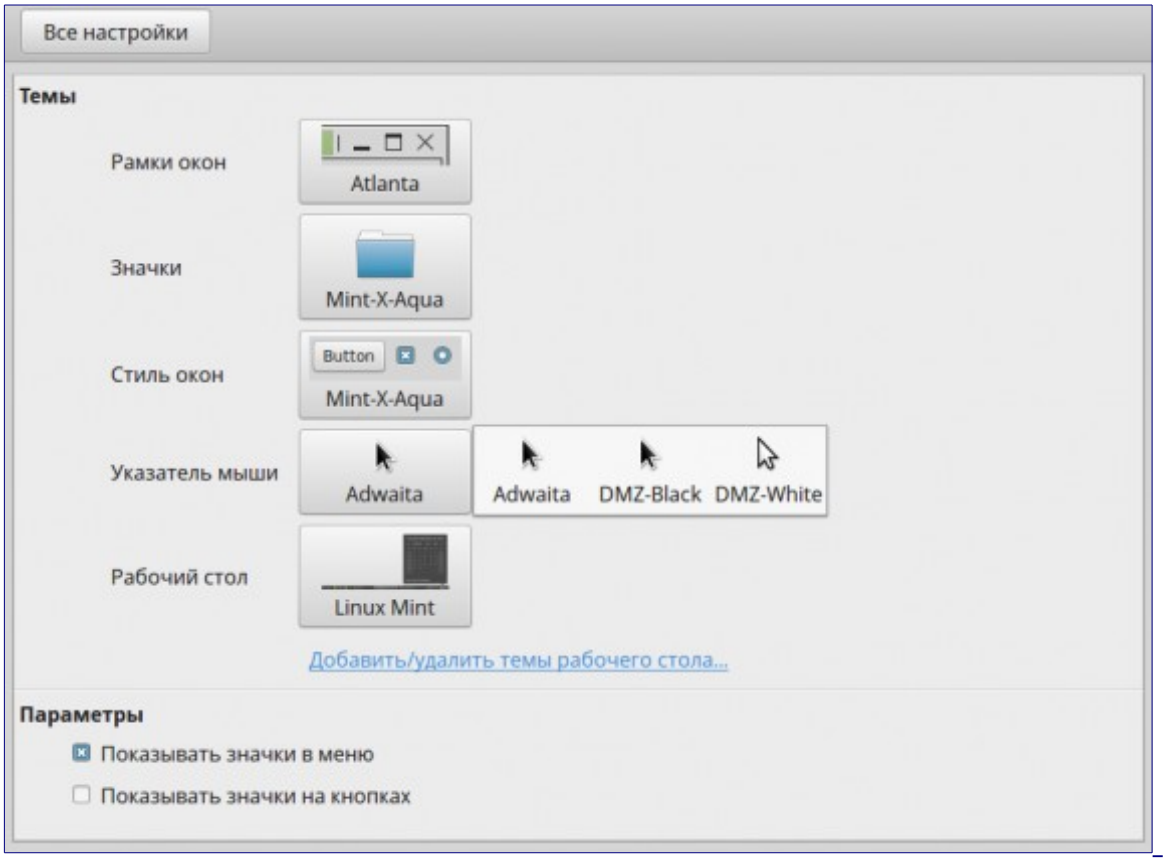
Пиктограмм:



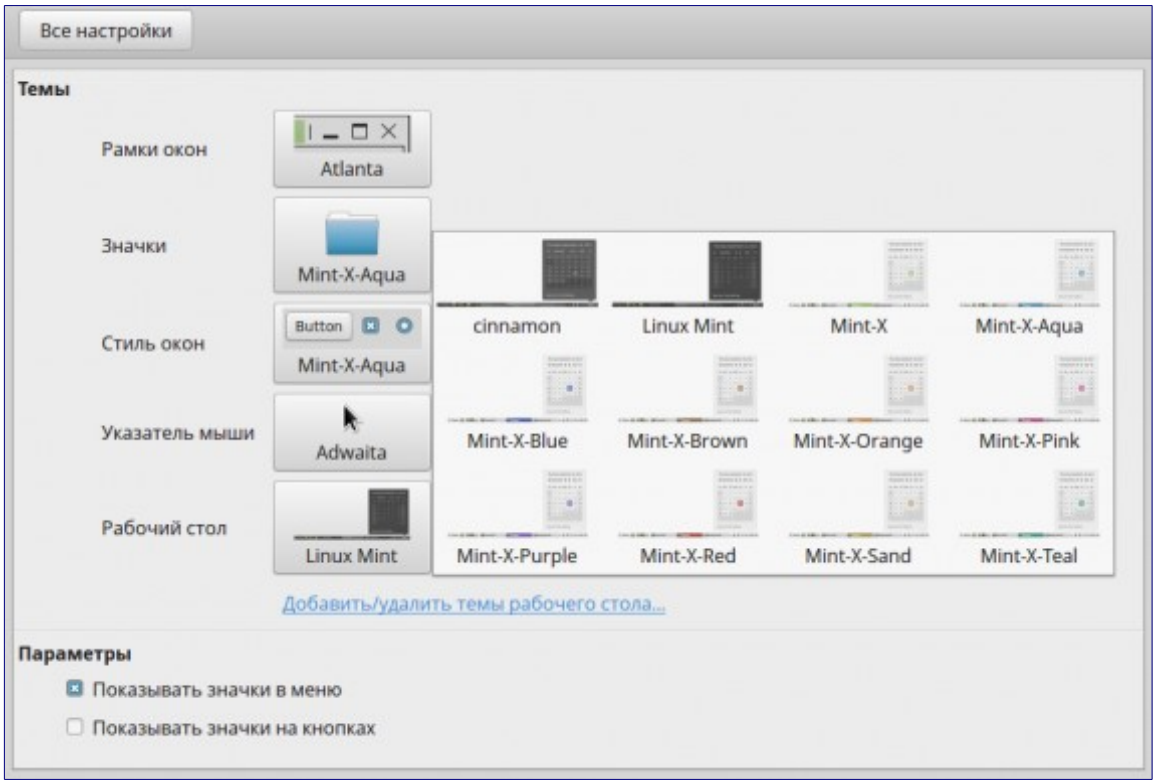
Управляющих кнопок окна:



Указателей мыши:



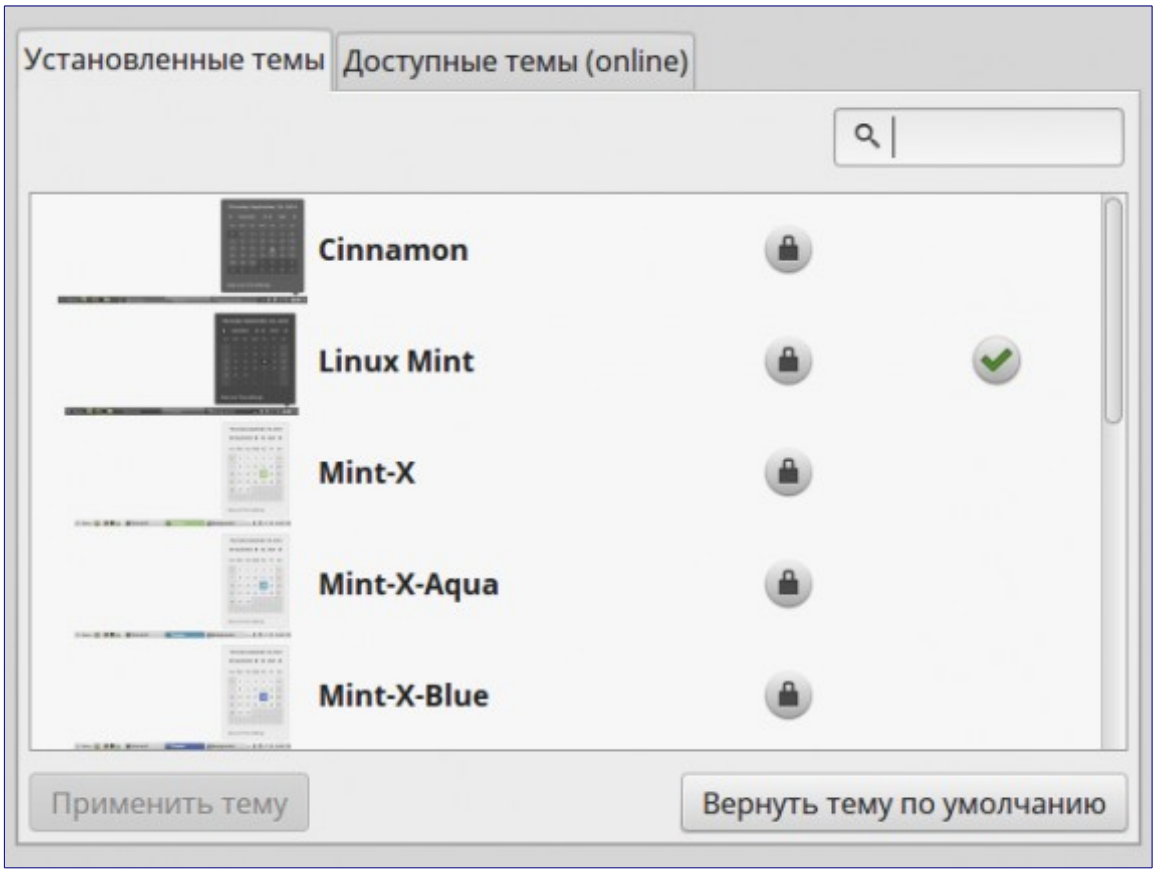
И, наконец, для собственно тем рабочего стола:



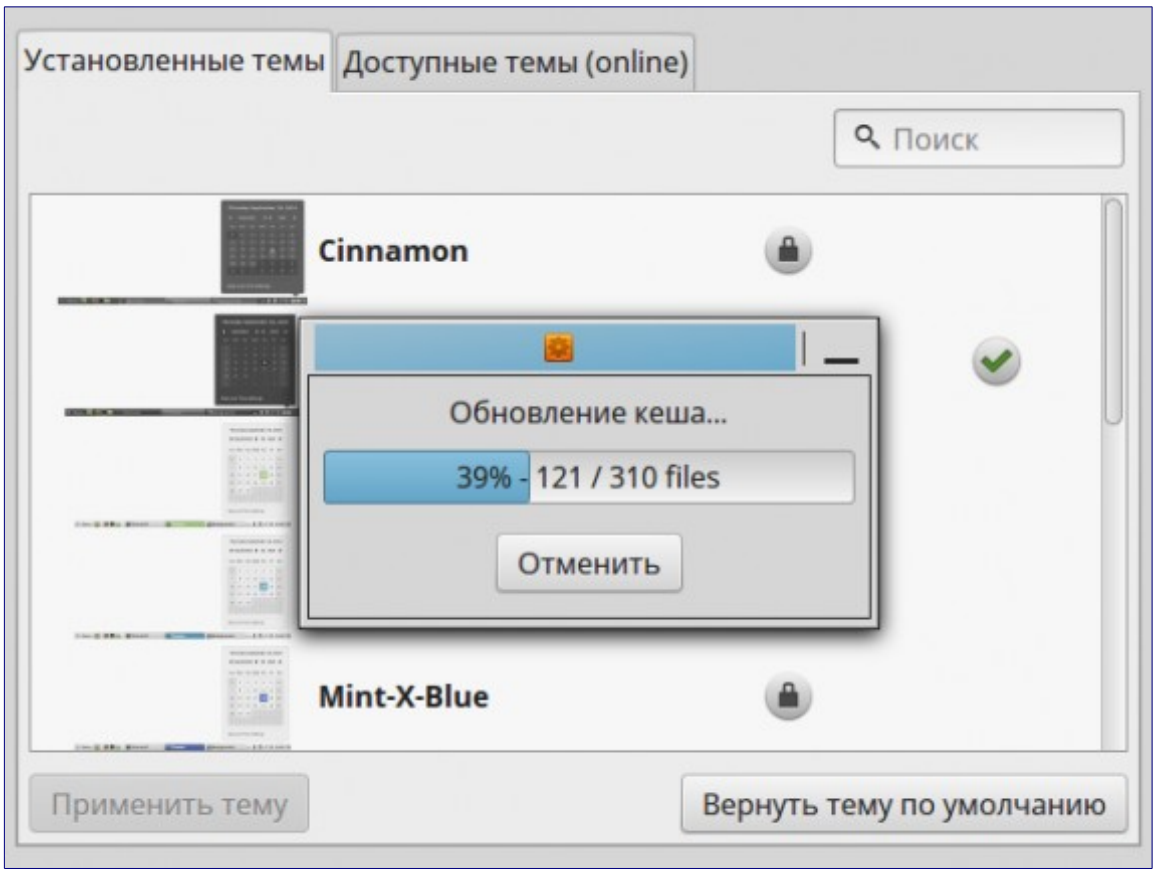
Обращает на себя внимание изобилие расцветок умолчальной темы Mint-X, с одной стороны, и пиктограмм — с другой. Это позволяют комбинировать элементы оформления интерфейса в очень широких пределах, достигая максимального визуального эффекта. Что, между прочим, сколько бы ни иронизировали на эту тему, имеет практическое значение, особенно для людей с плохим зрением или нарушением цветовосприятия.

С помощью кнопки Добавить/удалить темы рабочего стола умолчальную

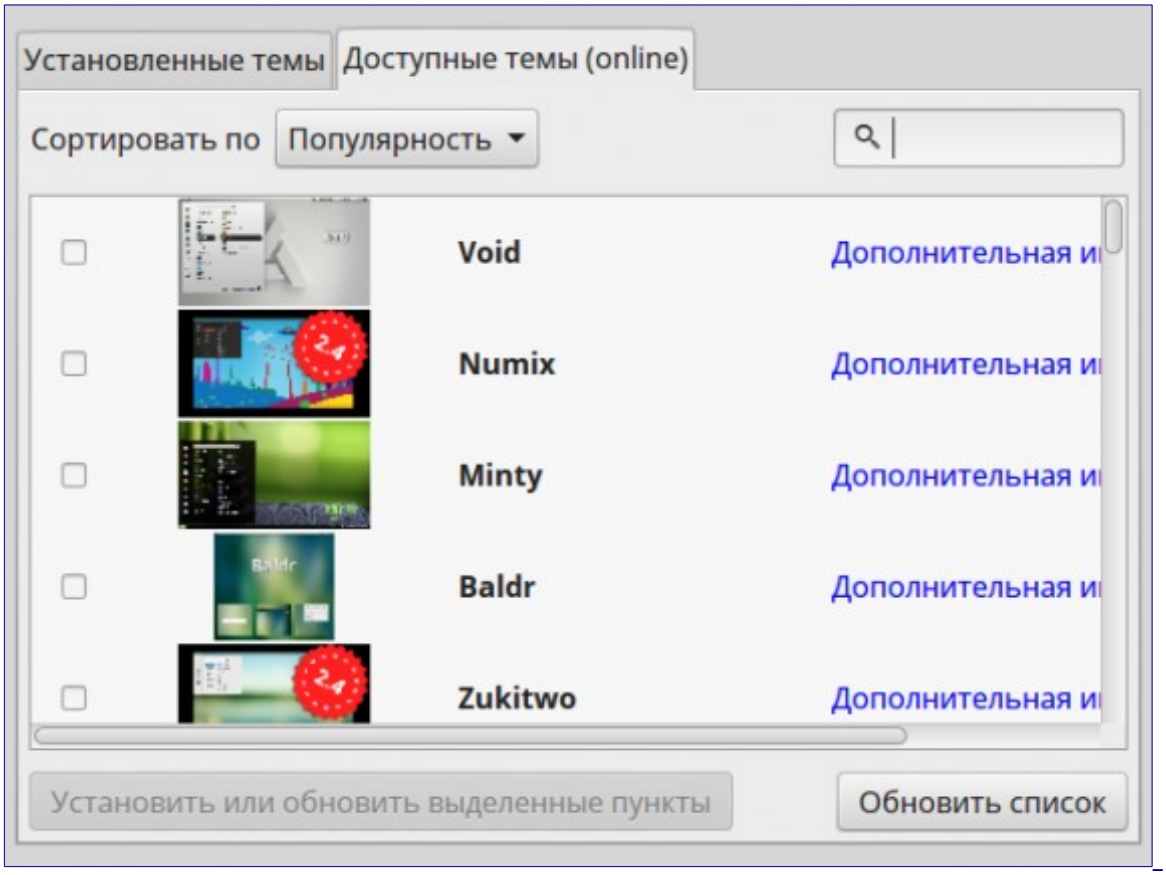
тему Mint-X можно заменить на одну из предустановленных:



Можно также обратиться к коллекции тем, доступных на [сайте специального субпроекта](#) — это сначала потребует обновления кеша тем, что может занять немало времени:

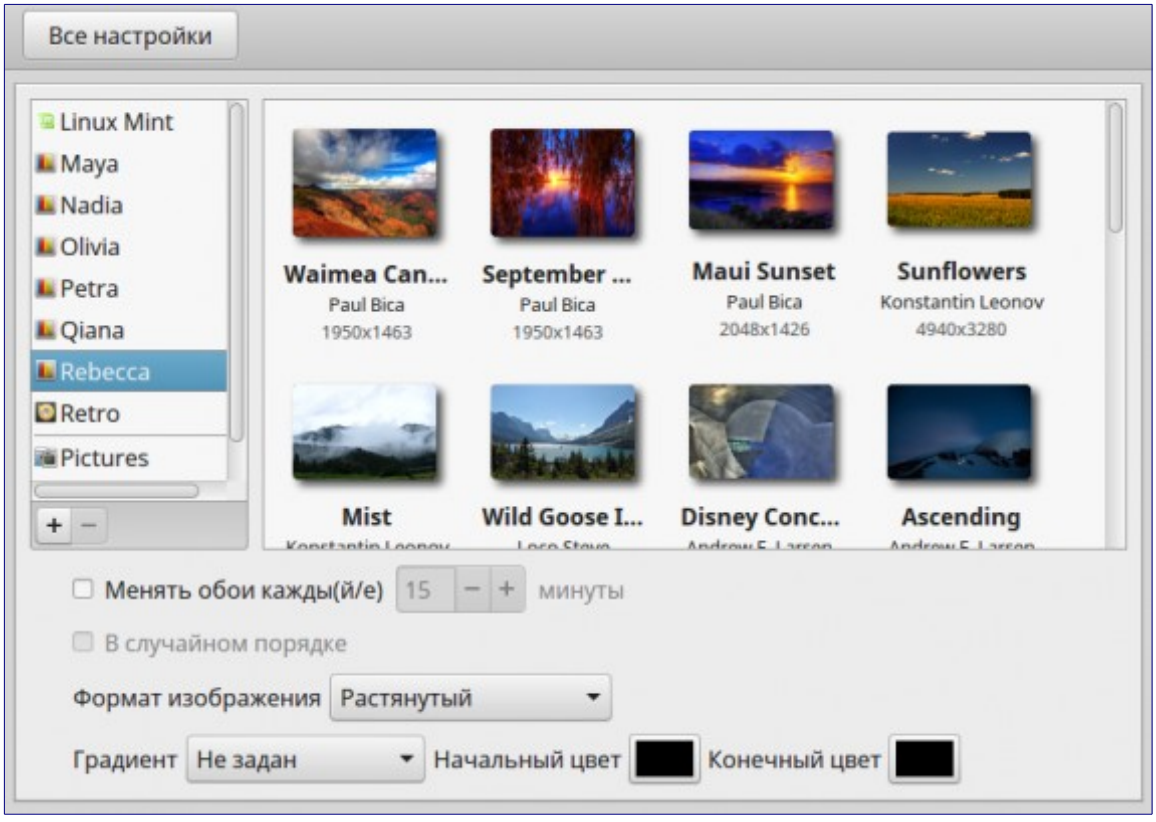


Но будет вознаграждено обильным уловом:

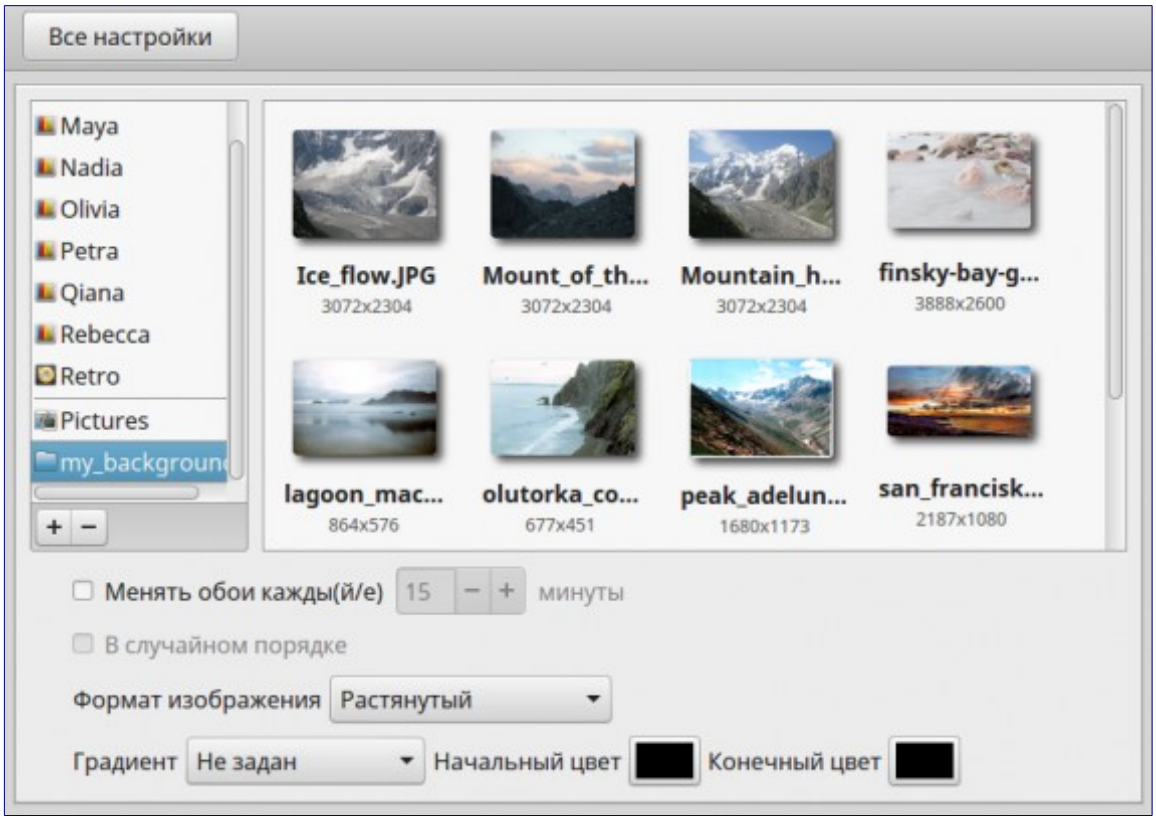


Правда, перепробовав немало тем, я в конечном счёте вернулся к умолчальной Mint-X. И даже отказался от её модификации — этому занятию ранее я тоже отдал свою дань, и потому опишу в конце данного очерка.

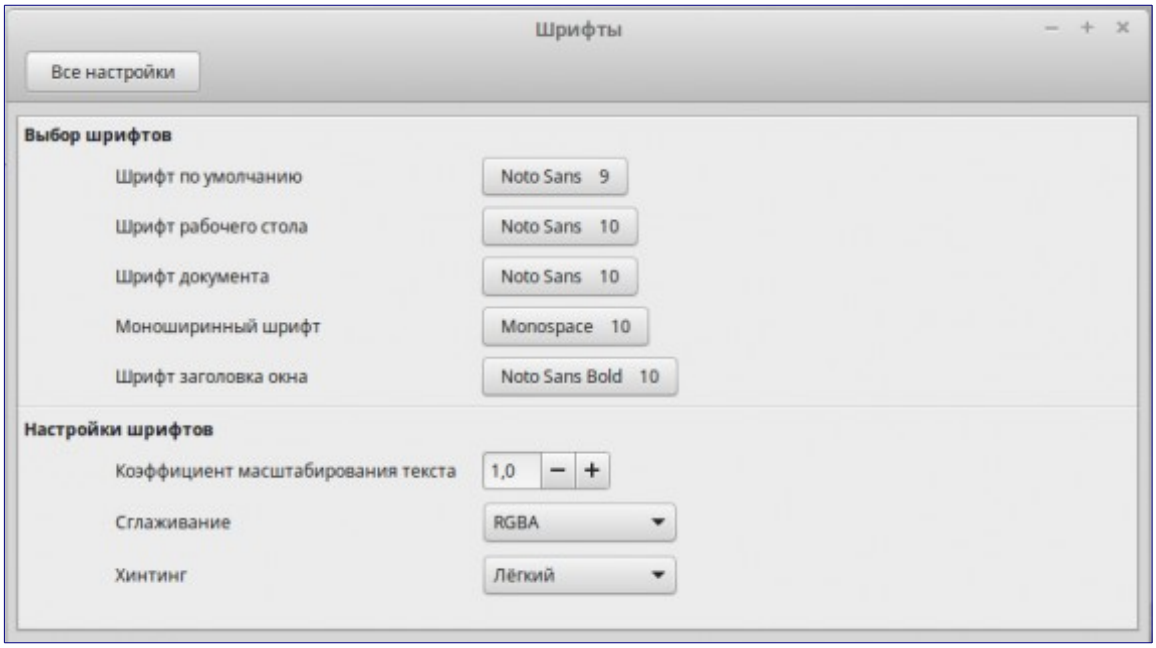
Модуль Фоновые рисунки в нынешней Cinnamon — это не просто банальные обои. Нет, конечно, их можно использовать и в этом качестве. Но самый цимес модуля — организация слайд-шоу из нескольких предустановленных наборов картинок, носящих имена былых и нынешних релизов Mint:



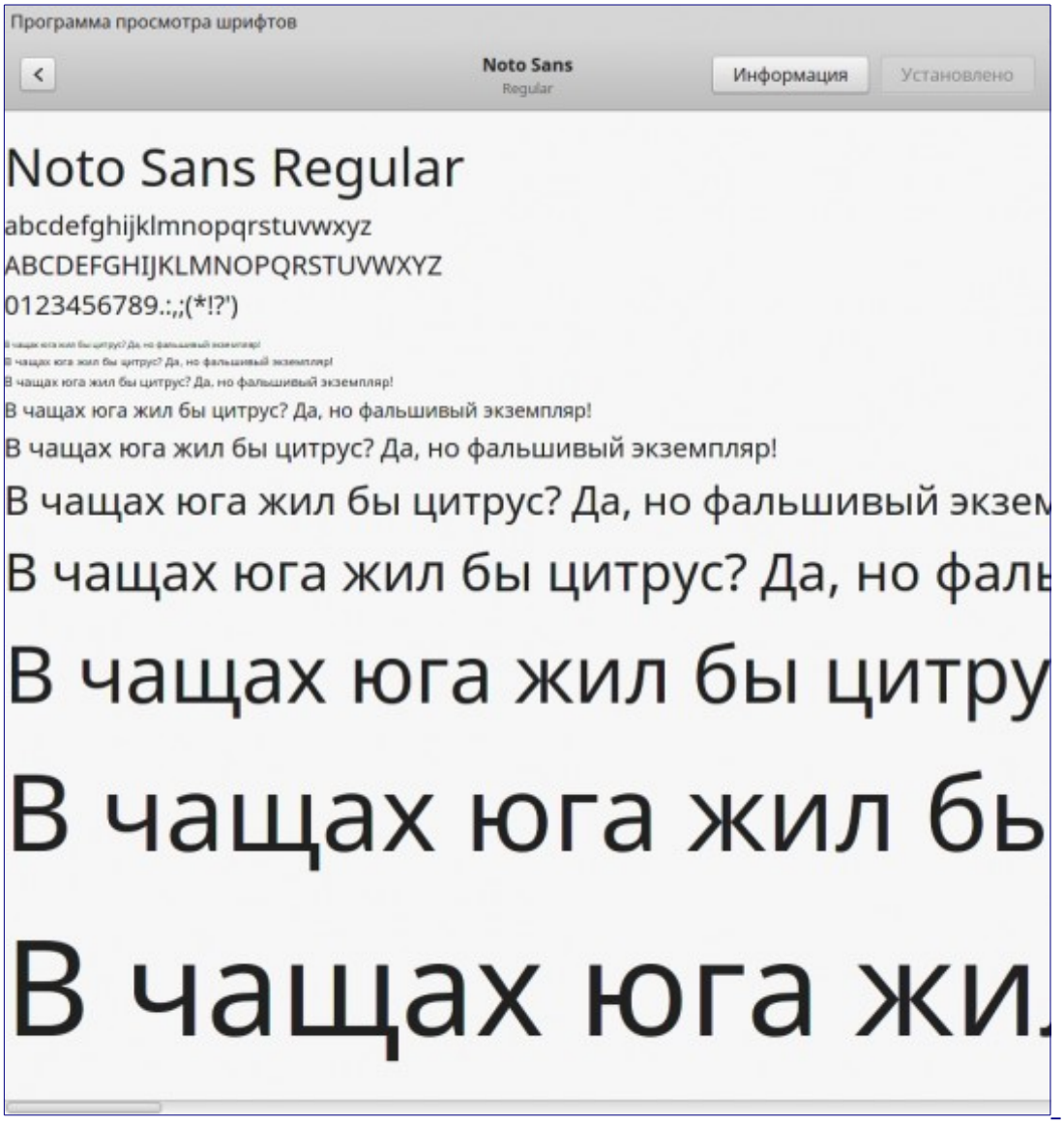
И не только из них — в качестве такого набора можно легко подключить каталог с собственными изображениями, на следующем скриншоте таковым выступает my_backgrounds):



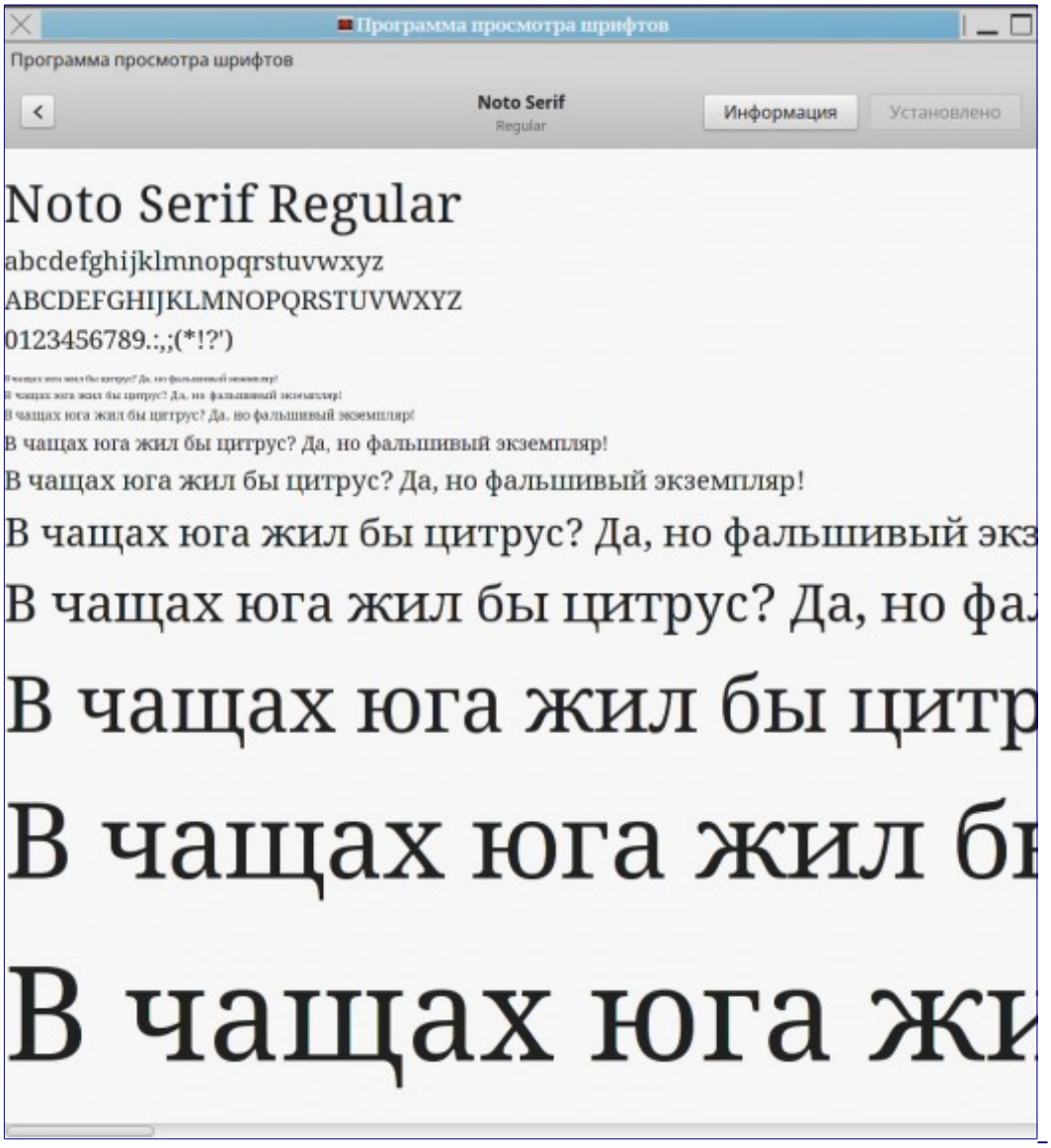
Модуль Шрифты позволяет определить гарнитуры, шрифтоначертания и кегли для элементов интерфейса, документов и терминальных окон. По умолчанию в качестве интерфейсных (пропорциональных) шрифтов в Cinnamon, начиная с версии 2.4, используется гарнитура Noto Sans собственной выделки:



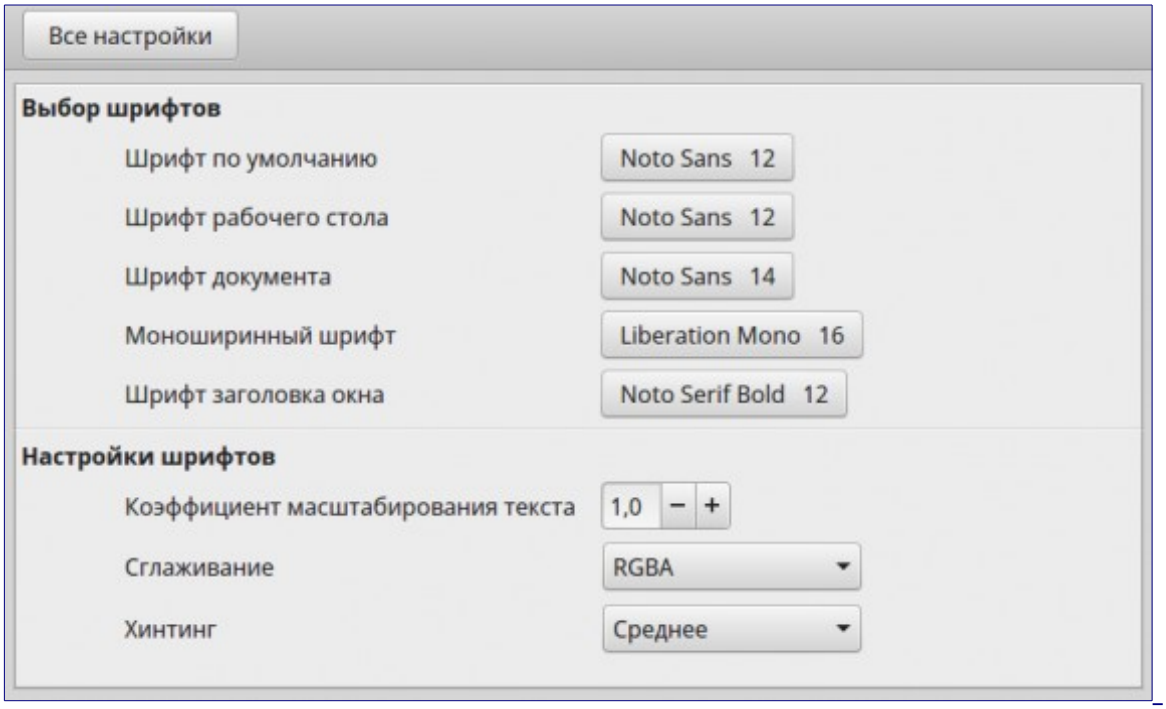
Мне эта гарнитура понравилась до чрезвычайности:



Особенно с учётом того, что она не одинока — в дополнение к ней имеется и гарнитура с отсечками, как нетрудно догадаться, именуемая Noto Serif:

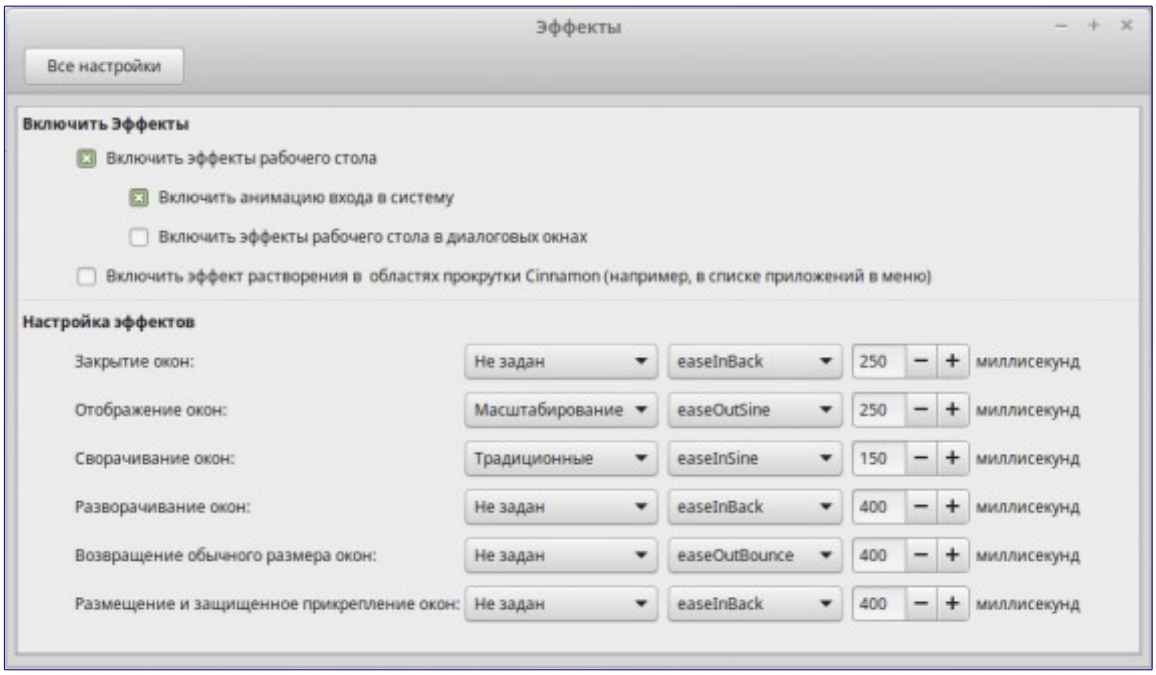


То есть семейство гарнитур Noto оказывается почти самодостаточным не только для интерфейса среды и её приложений, но и для оформления документов. Почти — потому что в нём явно не хватает какой-либо моноширинной гарнитуры для использования в текстовых редакторах и терминальных окнах. Впрочем, я это восполняю последнее время за счёт моноширинного представителя семейства Liberation — Liberation Mono. В результате чего мои шрифтовые настройки выглядят следующим образом:

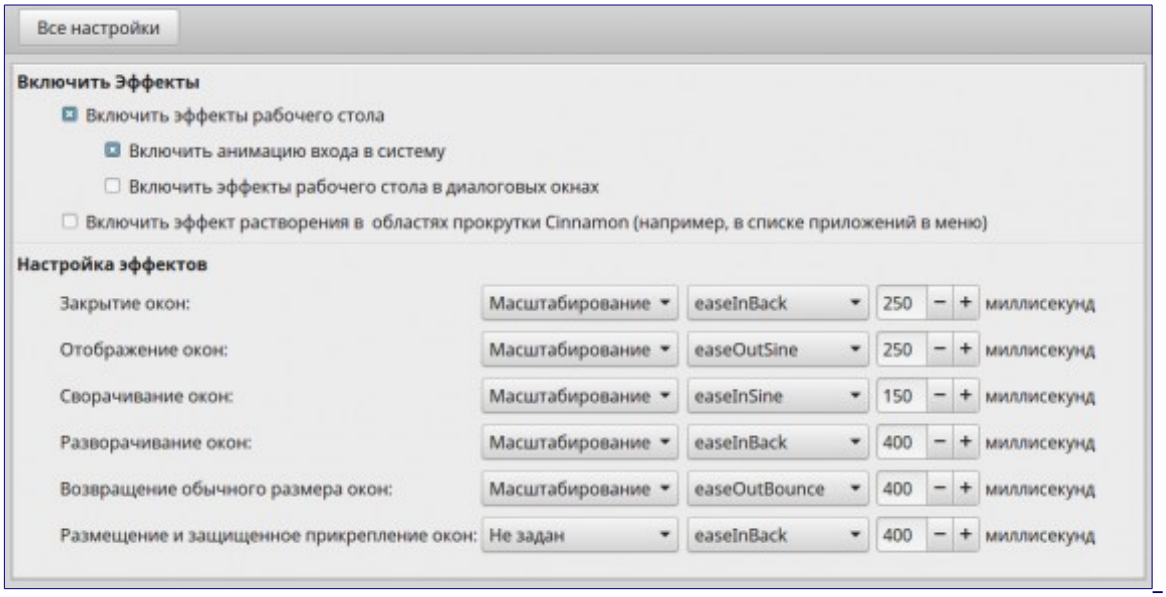


Тут нужно оговориться, что настройка шрифтов главного меню управляющей панели, подписей и всплывающих подсказок на ней возможна только путём прямого редактирования стилевого файла выбранной темы рабочего стола, о чём будет говориться в следующем разделе этого очерка.

А пока — про Эффекты. По умолчанию этот модуль выглядит таким образом:



До недавнего времени тут я просто снимал галочку с бокса Включить эффекты рабочего стола и больше на эту тему не думал. Предоставляя разбираться с «красивостями» тем, кто таким образом охмуряет молоденьких вендузятнец. Однако в Cinnamon эффекты мне неожиданно понравились своей плавностью и ненавязчивостью. Так что я затратил некоторое время на приведение к такому виду:



На этом в данный момент я с оформлением закончил. Однако, как и обещал, под занавес расскажу о том, как редактировать темы рабочего стола. Хотя, как уже было сказано, в конце концов я вернулся к умолчальной Mint-X, но как опыт это было интересно.

Cinnamon и собственные темы

В отличие от большинства остальных очерков, в этом разделе я описываю не актуальную версию Cinnamon 2.4, а её предшественниц 2.0 и 2.2. Где ни одна тема, даже самая красивая, не подходила мне целиком и полностью. В текущем же релизе неожиданно оказалось, что тема по умолчанию, Mint-X, устраивает меня во всех отношениях. Тем не менее, я решил включить описание моих тогдашних развлечений в эту книгу — вдруг когда возникнет желание заняться этим делом снова?

Как только что было сказано, настройка шрифтов Cinnamon действует на все элементы его интерфейса, кроме панели и главного меню. Причём следствие, проведённое в то время, показало, что в панели и меню шрифты зависят от темы оформления: при смене её гарнитура в этих элементах интерфейса менялась, хотя кегль оставался если не неизменным, то обычно маленьким и трудно различимым.

После отправки дела на доследование оказалось, что так оно и есть: кегли шрифтов для меню и разных элементов панели (а в ряде случаев – даже и гарнитуры) жёстко прописывались в CSS-файле всех тем, которые я просмотрел. А поскольку все они были изготовлены зоркими соколами, кегли эти везде были очень маленькими, от 7 до максимум 11 пунктов.

Обсуждать вопрос о том, насколько это идеологически правильно, здесь не буду. Конечно, на мой взгляд, неправильно абсолютно – ибо противоречит идее сквозных настроек десктопа, последовательно проводимой в KDE и Xfce. Однако идеология – идеологией, а практика – практикой: поскольку это оказался чуть ли не единственный недостаток Cinnamon'а, его следовало по возможности искоренить, а не рассуждать на тему

Что делать, блин?
И кто, блин, виноват?!

Что делать – было ясно: редактировать тему, наиболее близкую по всем

остальным показателям. А как делать – в принципе стало ясно из прочтения материала Клемента Лефевра, к переводу или оригиналу которого и отсылаю заинтересованного читателя. Здесь же лишь кратко опишу последовательность собственных действий.

В качестве подопытного кролика я выбрал тему Void, все относящиеся к ней файлы имели место быть у меня в каталоге ~/.themes/Void/cinnamon. В том числе и cinnamon.css, который я отредактировал самым простым способом: без лицемерия явным образом указал гарнитуру:

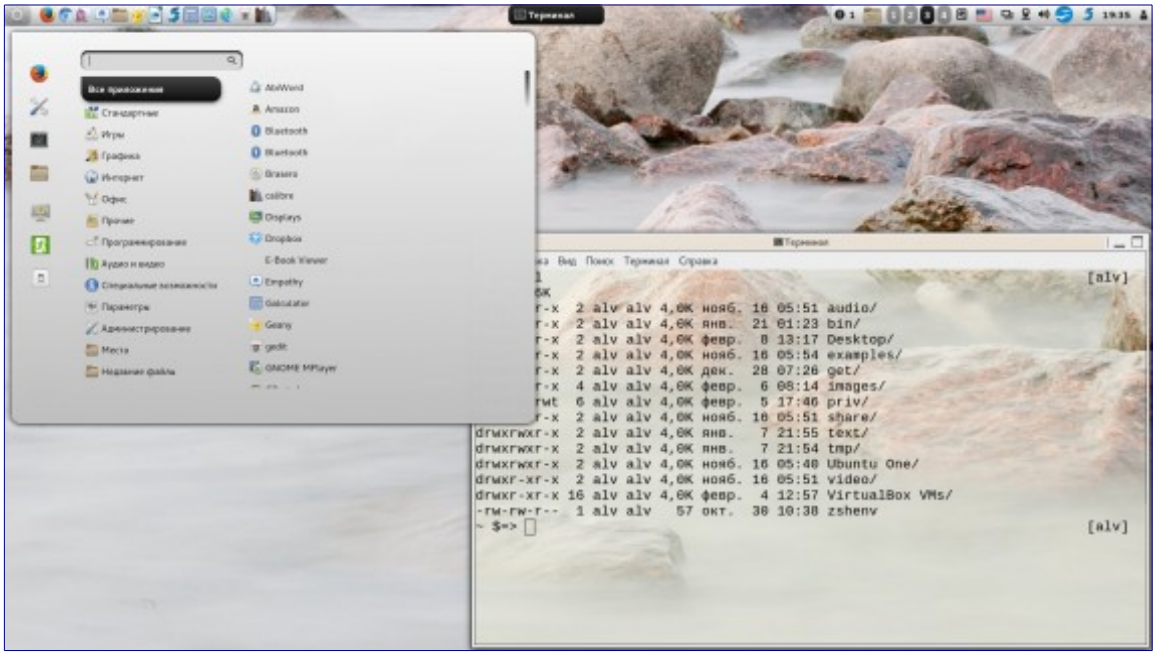
```
stage {

font-family: "Cantarell";

}
```

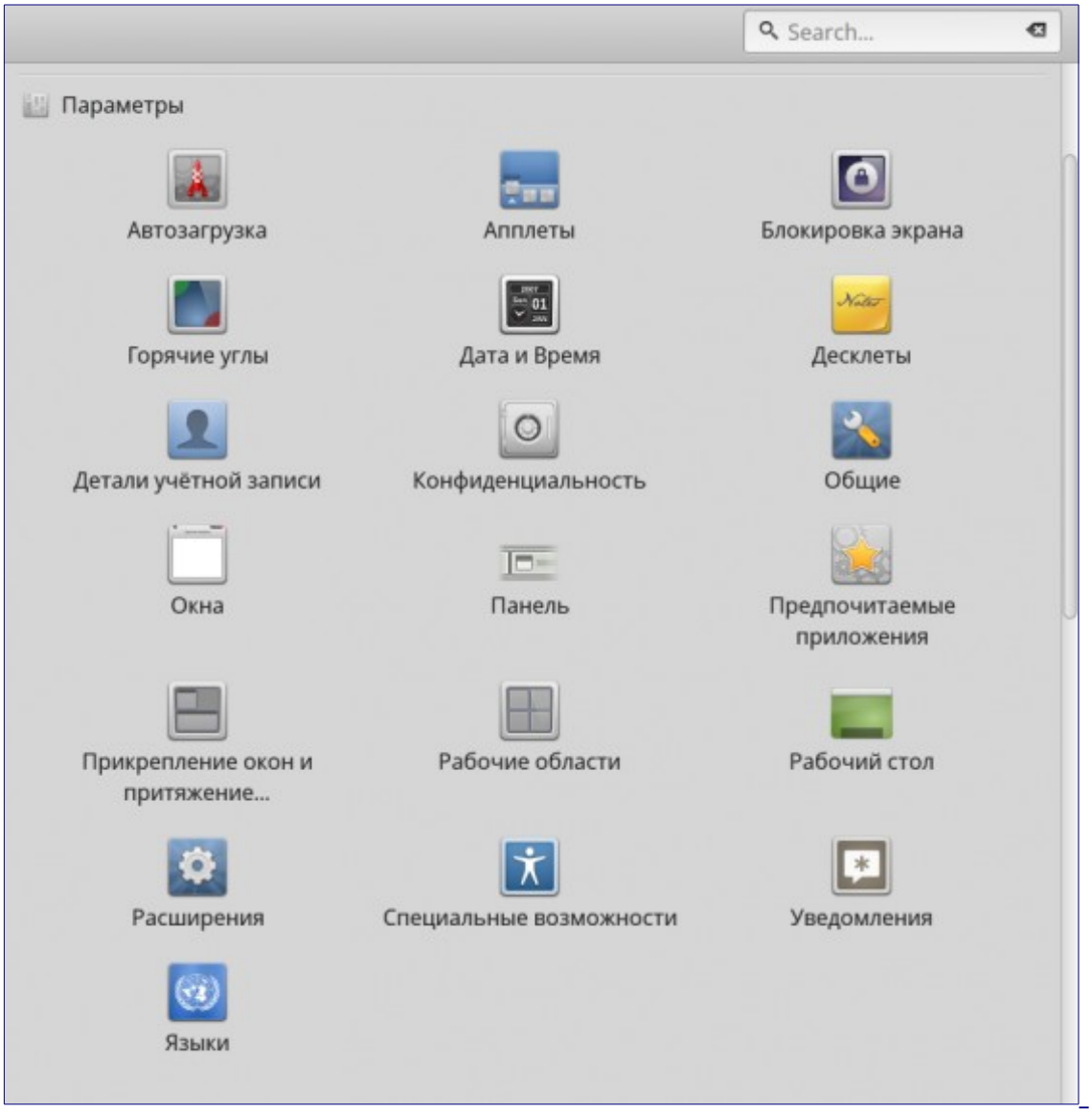
Затем просто добавил один-два пункта к кеглям шрифтов всех интерфейсных элементов. А заодно из темы Canelita потырил пиктограмму для кнопки главного меню – умолчально-зелёная в мою цветовую гамму вписывалась плохо.

И вот что получилось в итоге:



Секция Параметры

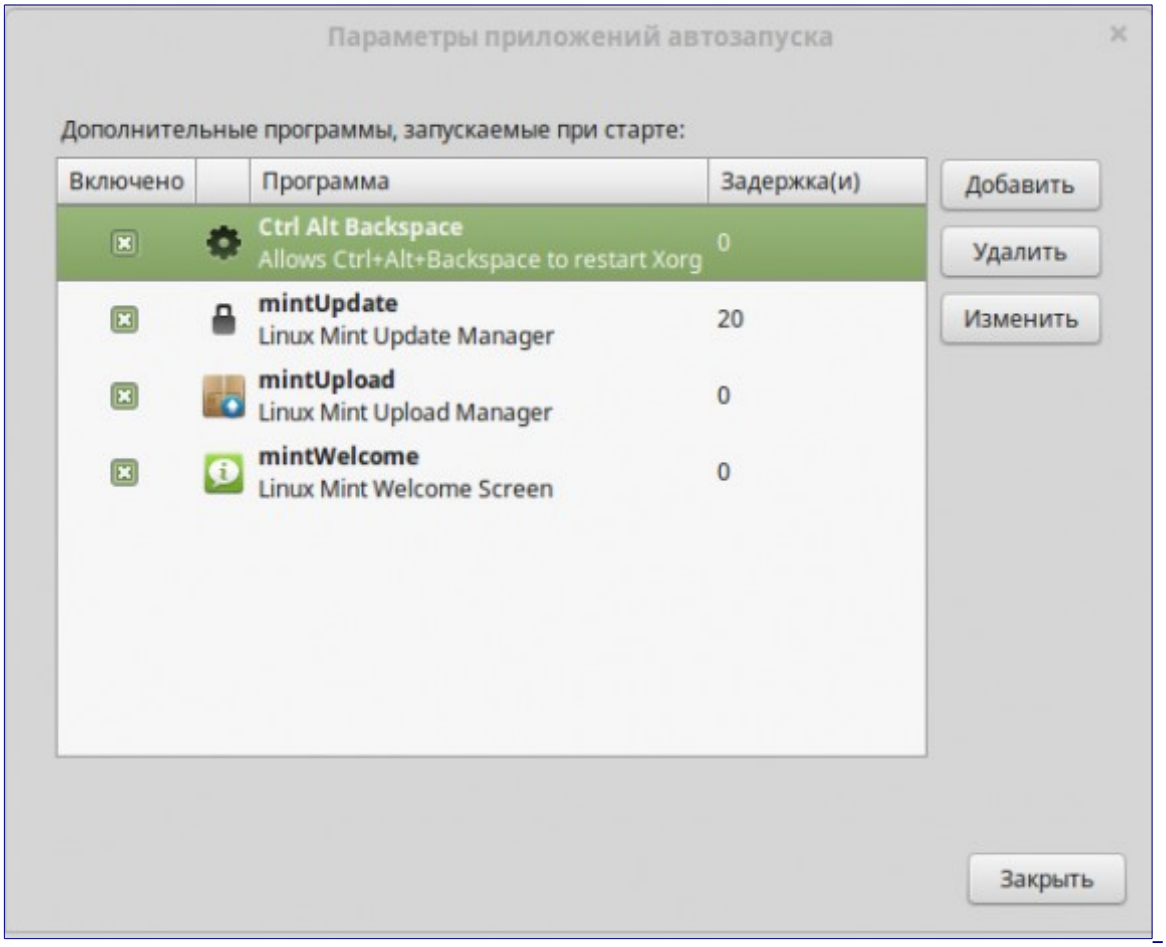
Если модули секции Оформление отвечают за внешний вид среды Cinnamon, то в секции Параметры собраны модули, определяющие её поведение. Это — самая обширная секция среди Системных настроек, и в полностью обозримом виде она выглядит так:



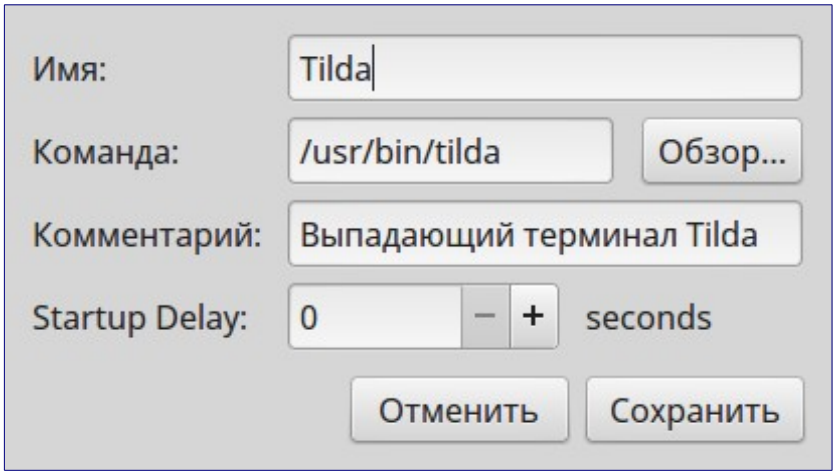
Далее в этом очерке модули секции будут рассмотрены не в том порядке, в котором они приведены на скриншоте, то есть алфавитном. А в порядке, диктуемом логикой. Хотя начну рассмотрение я таки по алфавиту.

Автозапуск

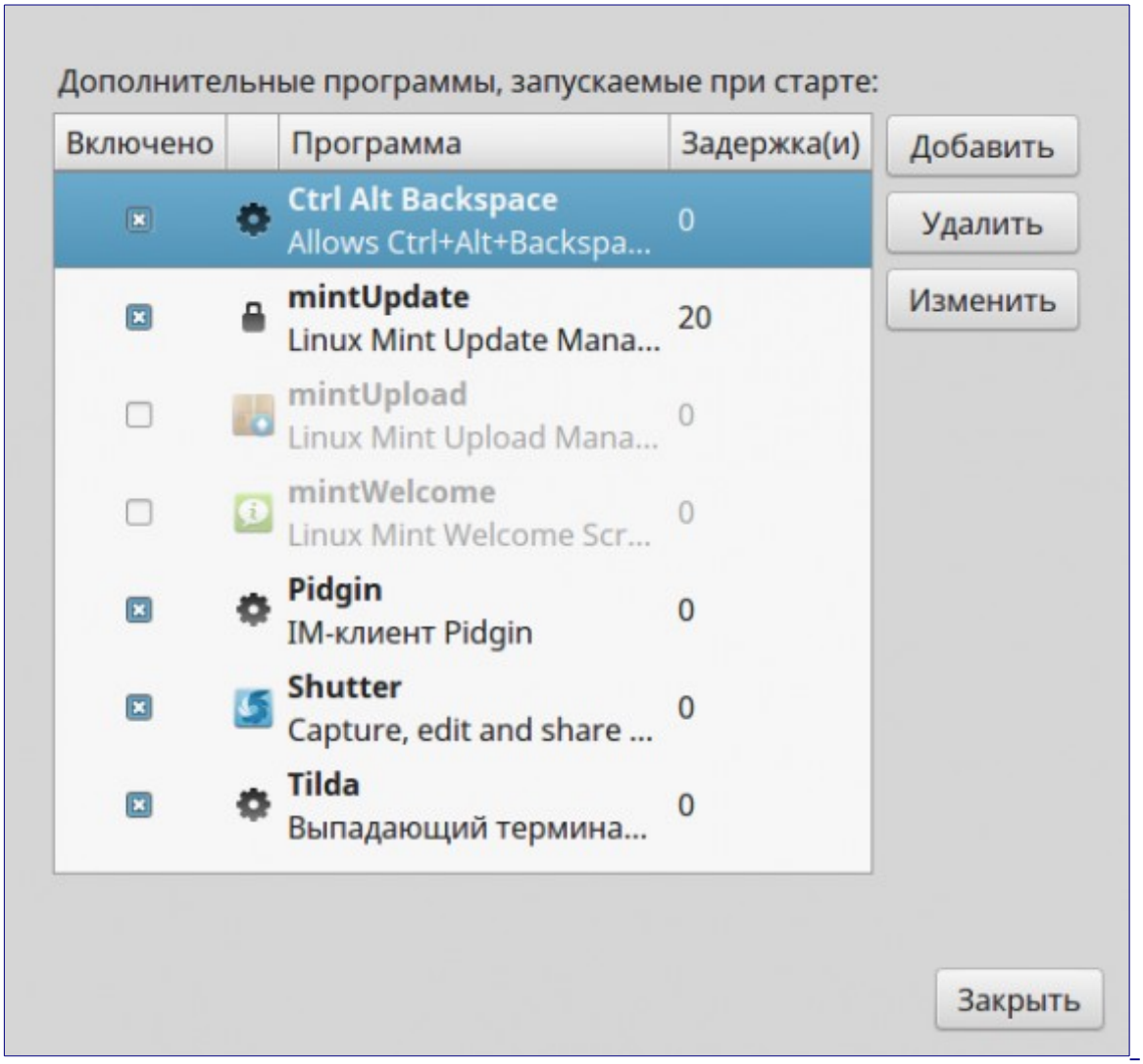
Модуль Автозапуск предназначен для управления программами, автоматически загружаемыми при старте сеанса Cinnamon, как системными, так и сугубо прикладными. По умолчанию список их выглядит так:



Способы коррекции списка очевидны: снятие «птицы» с соответствующего бокса отключает загрузку данной программы (например, mintwelcome, выводящей приветственную панель при запуске сеанса), кнопка Удалить исключает её из списка вообще, а с помощью кнопки Добавить список пополняется программами по желанию применителя. Для чего в появившейся панели надо заполнить поля Имя (желательно соответствующее имени программы), Команда (имя запускающего файла, при необходимости — с указанием пути) и, по желанию, Комментарий (в свободной форме). Например, для выпадающего терминала Tilda (использование которого без автозапуска лишено смысла) это выглядит так:



В результате таких действий у меня панель автозапуска выглядит таким образом:



По умолчанию в панели автозапуска отображаются далеко не все автоматически запускаемые приложения — в ней мы не увидим всякого рода системных служб. Чтобы сделать их видимыми, нужно отредактировать соответствующие конфиги в каталоге /etc/xdg/autostart/, имеющие вид *.desktop. Каждый из них включает в себя параметр NoDisplay, отвечающий за вывод на экран, и значение его по умолчанию true. Которое достаточно заменить на false, чтобы увидеть весь стартовый букет служб и приложений. Сделать это в один присест можно при помощи утилиты sed, запущенной с правами администратора:

```
$ sudo sed -i 's/NoDisplay=true/NoDisplay=false/' /etc/xdg/autostart/*
```

Годится для этого и любой развитый текстовый редактор, позволяющий выполнять поиск и замену в серии файлов (Geany, Komodo Edit).

В предыдущих версиях Cinnamon в панели автозапуска имелся боксик Автоматически запоминать запущенные приложения при выходе из сеанса. Правда, работал он из рук вон плохо, и потому в текущей версии был в Системных настройках ликвидирован как класс. Но возможность включить сохранение сеансов сохранилась в Редакторе dconf, о котором будет говориться своевременно.

Апплеты, десклеты и расширения

Апплеты уже мельком упоминались в очерке об интерфейсе Cinnamon, а вот про десклеты и расширения речи ещё не было. Так что для начала скажу пару

слов о том, что это такое.

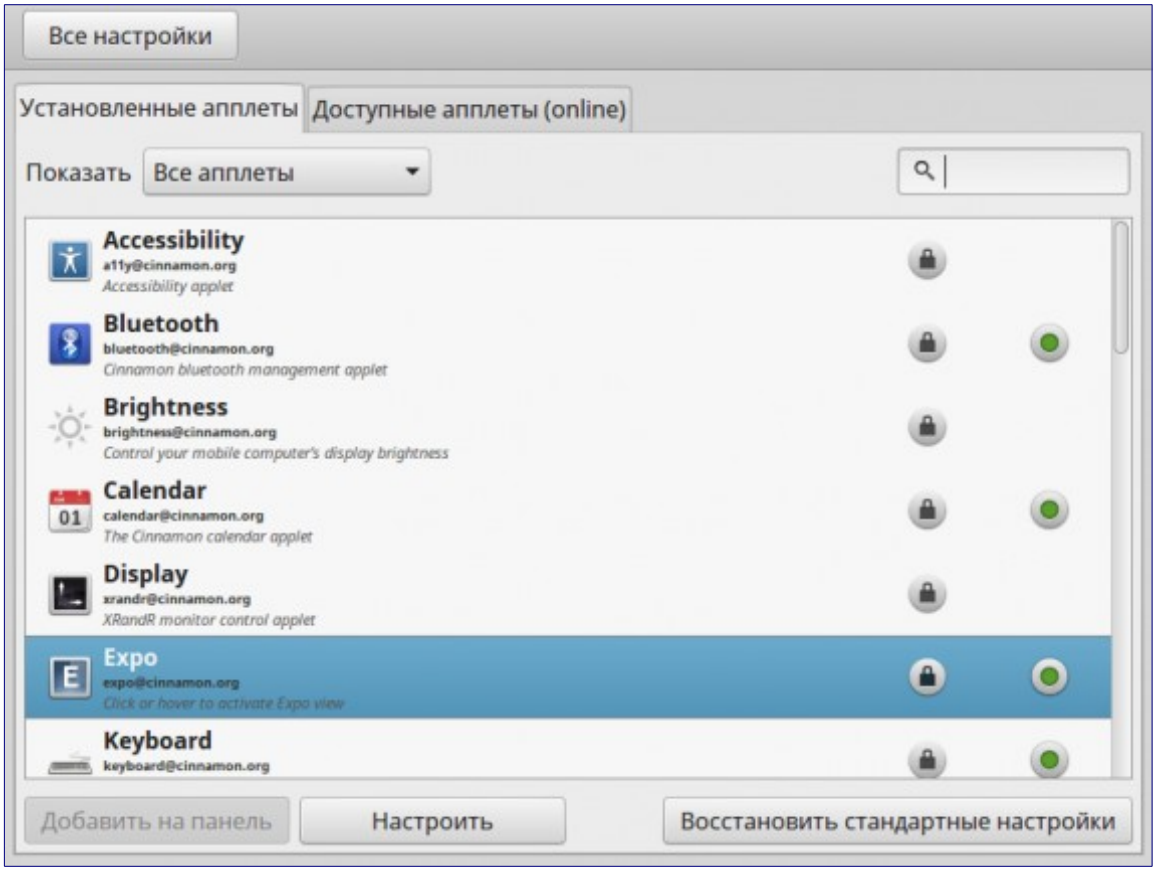
Название апплеты является уменьшительно-ласкательной формой английского application, то есть по русски — программульки, или маааленькие программы. Однако главное не в их размере, а в том, что они работают только внутри других, «полноценных», программ, и не способны к самостоятельному существованию. В нашем случае апплетами являются пиктограммы управляющей панели Cinnamon'а, которые вне её не то что функционировать — жить не могут.

Десклеты — это название принято в Cinnamon'е для элементов, которые в других рабочих средах называют виджетами (то есть «штуковинами»). Как и апплеты, самостоятельно роли они не играют. Но, в отличие от тех, встраиваются на рабочий стол (откуда, видимо, и название).

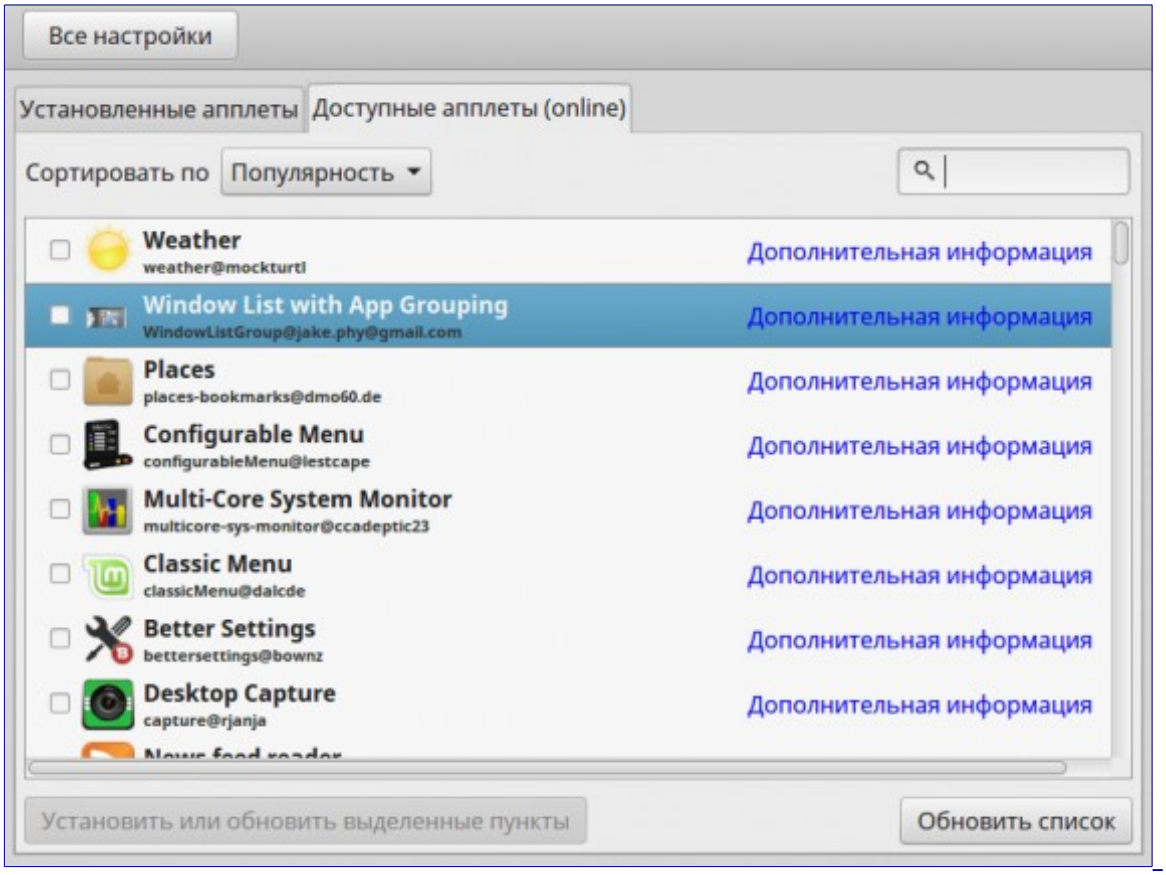
Что же до расширений (extensions), их назначение понятно из названия: подобно плагинам и прочим add-on'ам, они добавляют к базовой функциональности десктопа дополнительные возможности (например, добавление второй нижней панели плюс к главной управляющей).

А теперь посмотрим, как эти самые апплеты, десклеты и расширения настраиваются. Начав по алфавиту, с первых.

Собственно, настройка апплетов сводится к двум моментам. Первый — это добавление на панель уже установленных апплетов или, напротив, удаление с неё добавленных:



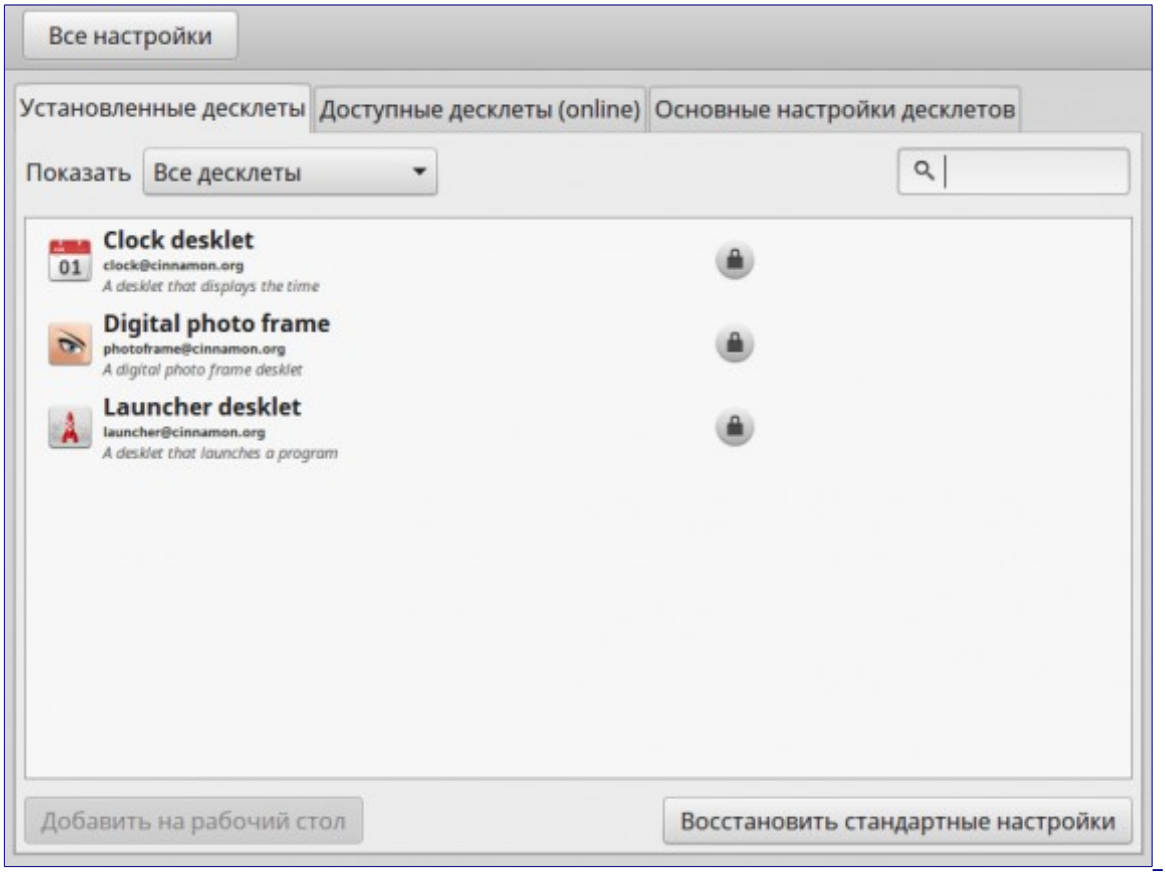
Второй момент — загрузка списка апплетов не установленных, выбор из них нужных и установка последних:



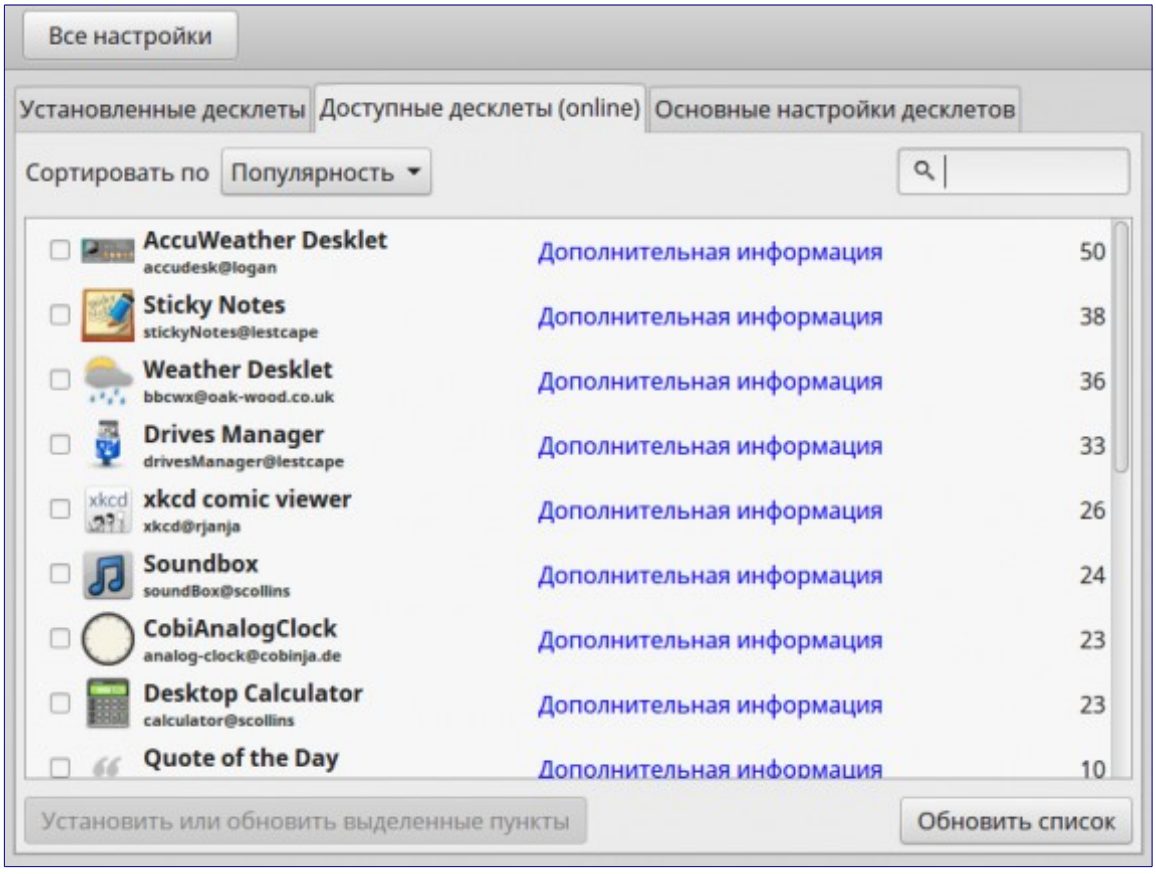
Я думаю, что обе задачи заинтересованный читатель сможет решить без труда. Как и вопрос с определением нужности или ненужности конкретных апплетов. Попрошу только обратить внимание на апплеты Workspace switcher и Expo в списке предустановленных, но не используемых — они скоро потребуются нам в одном из ближайших разделов этого очерка.

А при установке подгружаемых апплетов надо учитывать, что при переходе от Cinnamon версии 2.2 к 2.4 произошла смена API. И, по сообщениям в Сети, не все из сторонних апплетов, сочинявшихся ещё для прежних версий, обязаны корректно работать в последнем релизе. Впрочем, со временем эта проблема теряет актуальность.

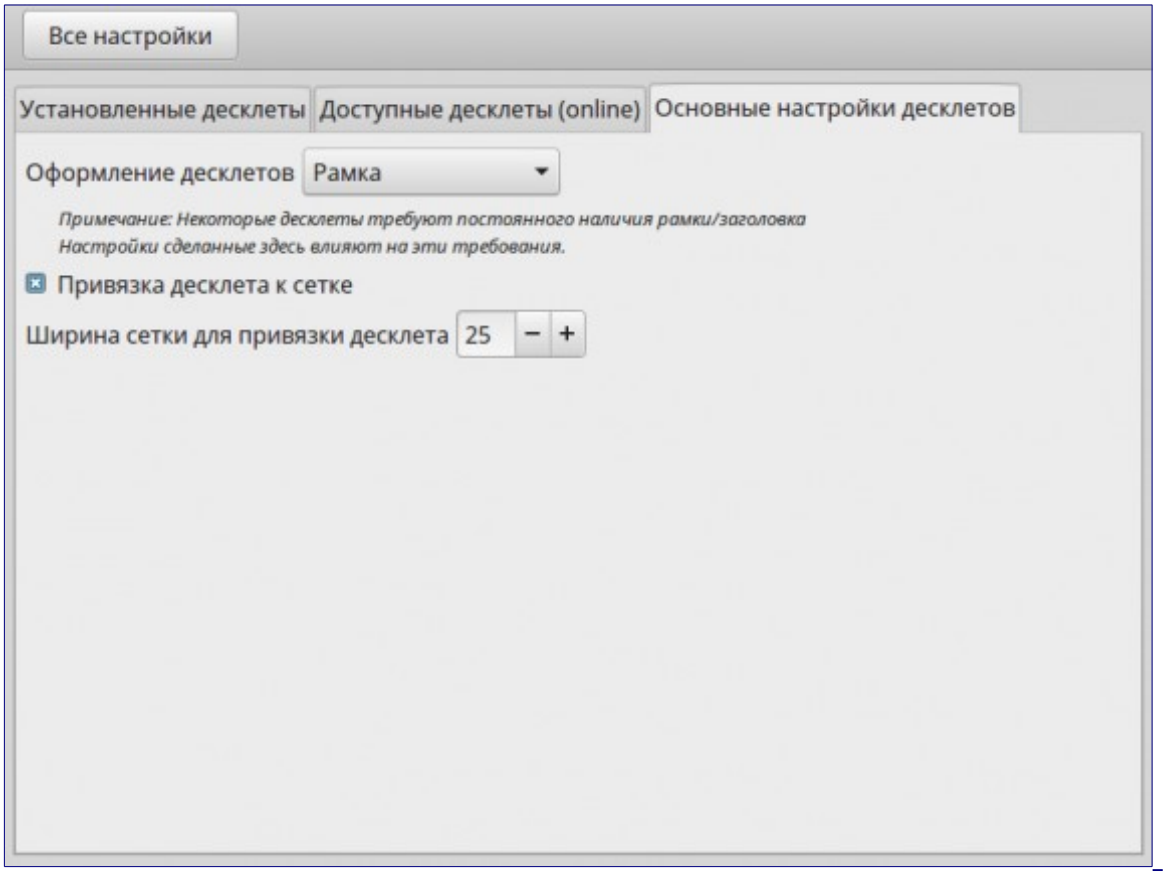
С десклетами дело обстоит сходным во всех отношениях образом. Во-первых, их также можно выбрать из числа предустановленных, каковых всего три:



Во-вторых, можно просмотреть список десклетов, доступных в Сети, и включить их в список установленных:



В-третьих, можно настроить оформление десклетов (в рамке, с заголовком или без ничего) и их размещёние (привязка к сетке с заданным шагом).:



Для подключения десклетов из списка установленных их остаётся только добавить на рабочий стол — подобно пиктограммам рабочего стола, они появятся во всех рабочих пространствах:



Вопрос, нужны ли народу десклеты, остаётся спорным. С одной стороны,

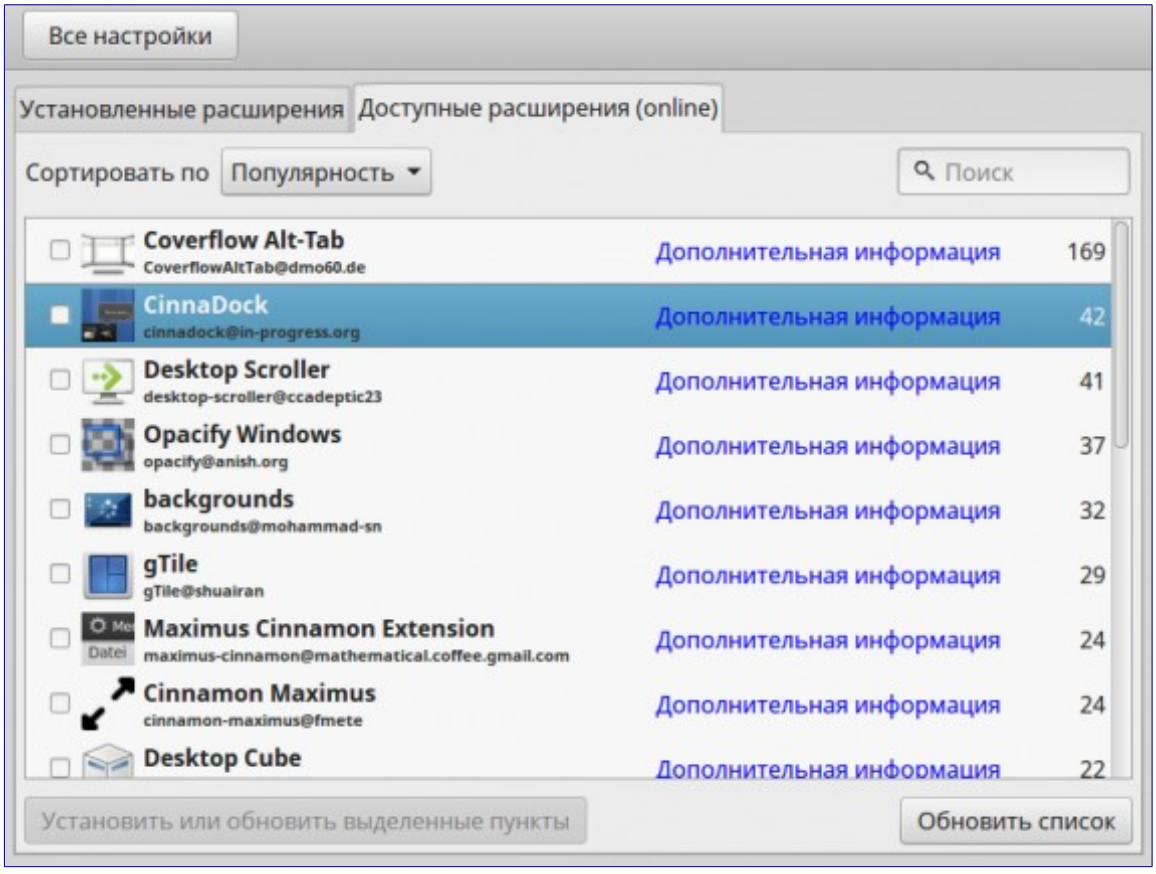
болтающиеся на экране дополнительные часы, калькулятор, информатор о занятости дисковых разделов или сводка погоды (а никакого иного полезного функционала я среди них не обнаружил) не особенно и видны в рабочем режиме. Но с другой — и не мешают, а при случае могут и пригодиться.

Однако тут надо учитывать два момента. Первый — к десклетам относится всё сказанное об апплетах относительно совместимости их с текущей версией Cinnamon.

Второй момент — более существенный. Как показала практика, даже десклеты, заведомо предназначенные для соответствующей версии Cinnamon, могут работать некорректно и даже приводить всю среду в полностью неработоспособное состояние.

Правда, лечится это достаточно просто — нажатием кнопки Восстановить стандартные настройки, если не помогло — ручной очисткой каталога ~/.local/share/cinnamon/desklets, но всё равно радости мало. Учитывая сомнительную пользу даже от тех десклетов, которые кажутся подозрительными на полезность.

С расширениями ситуация ещё менее однозначна. В предустановленном виде их нет ни одного, да и список доступных не так велик:



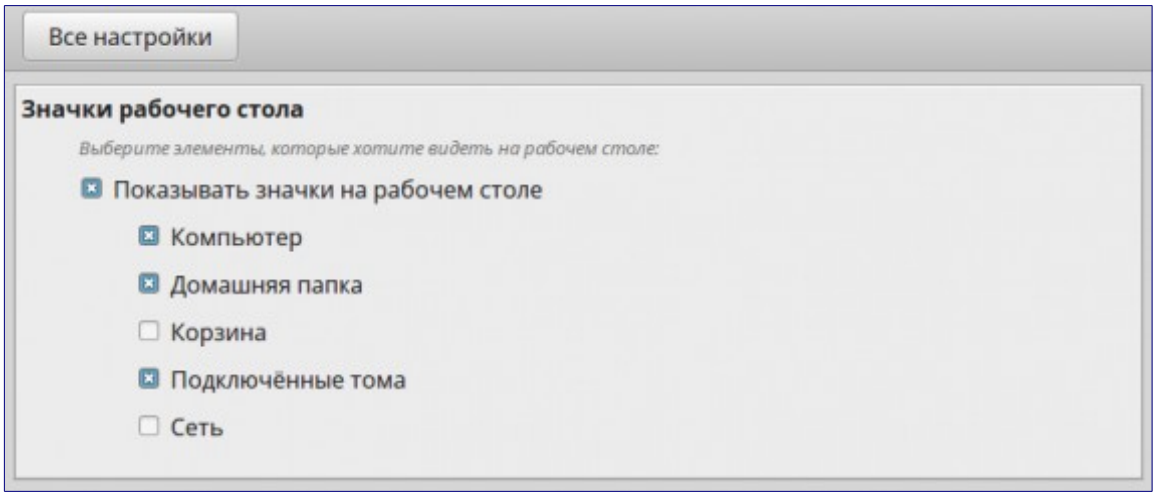
И среди существующих расширений вызывающих подозрения в своей полезности оказалось не мало — например, CinnaDock, похожий, судя по описанию, на Cairo, упомянутый выше 2 Bottom Panels или трёхмерный переключатель задач — 3D App Switcher. Но слова относительно совместимости с Cinnamon 2.4 относятся к расширениям ещё больше, чем к десклетам. В частности, ни одно из заинтересовавших меня в текущей версии этой среды даже не устанавливалось, не говоря уже об активизации. Я понимаю, что со временем это всё устаканится — но вот тогда и вернусь к

этому вопросу.

Панель и рабочий стол

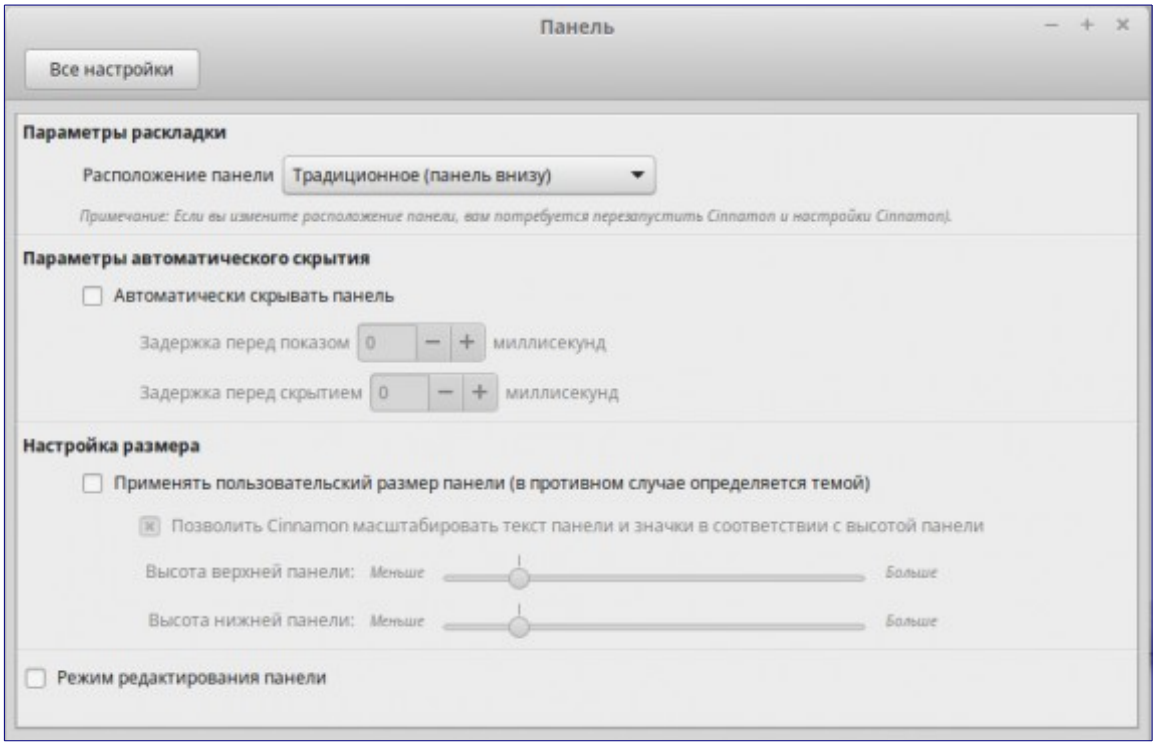
После рассмотрения апплетов, десклетов и расширений логично будет перейти к конфигурированию элементов, их вмещающих — управляющей Панели и Рабочего стола.

Правда, про настройки рабочего стола сказать особо нечего. Здесь можно только отметить, какие пиктограммы из заданного списка следует на него выводить:



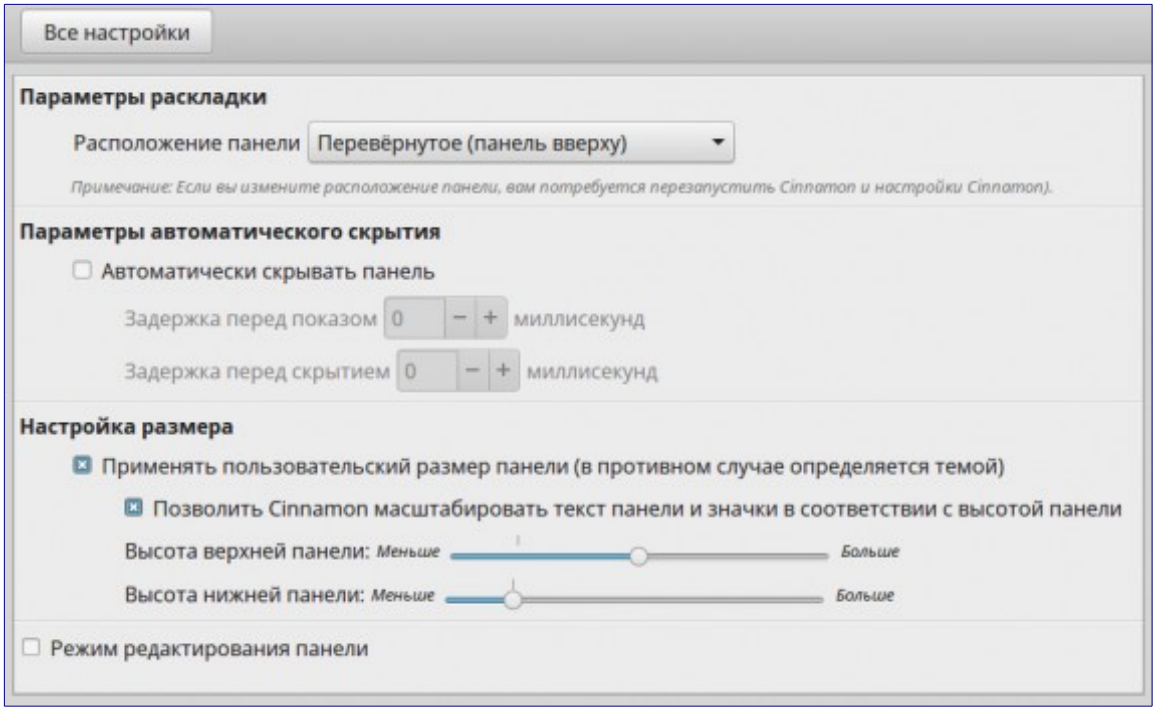
Впрочем, модуль всё равно полезный, так как позволяет запретить вывод пиктограмм на рабочий стол вообще, что я обычно и проделываю: в редкие мгновения, когда я вижу рабочий стол вообще, предпочитаю любоваться картинкой на нём, а не пялить глаза в какие-то пиктограммы.

В модуле Панель по умолчанию всё достаточно стандартно — расположение её «традиционное», то есть внизу экрана:



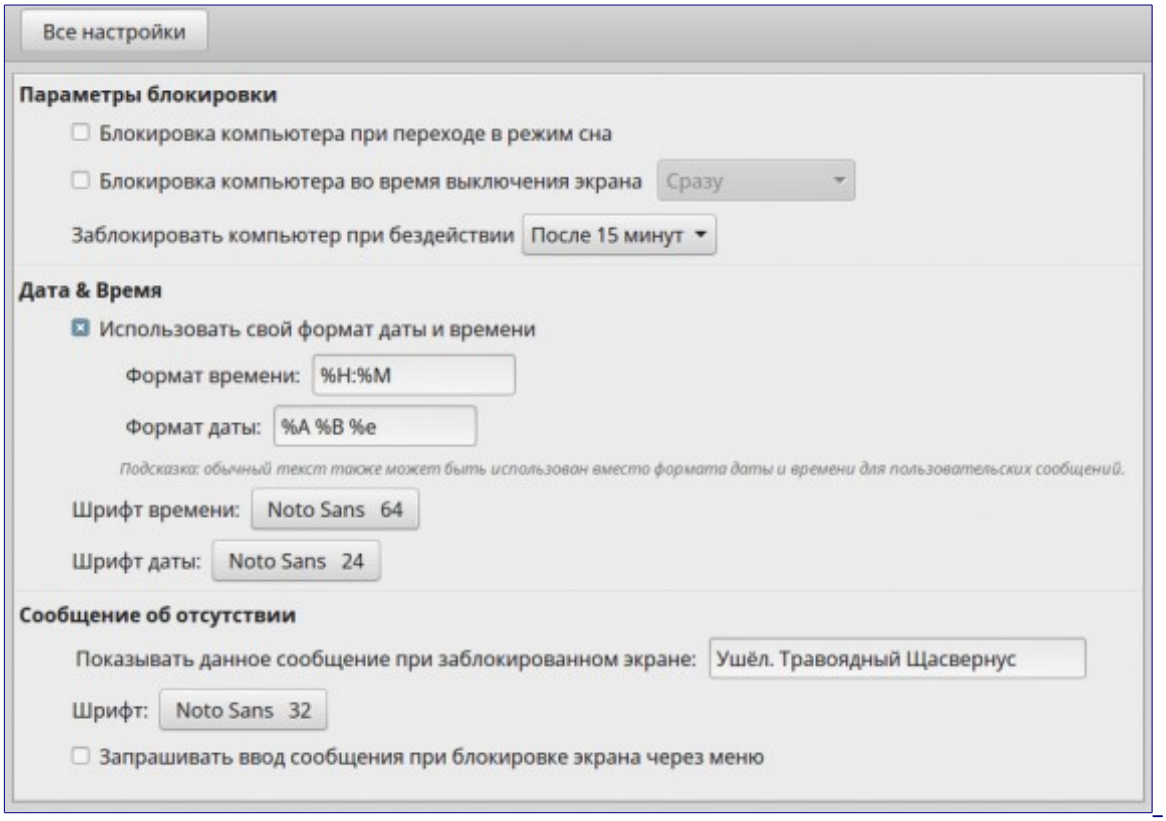
Однако есть немало возможностей для видоизменения. Так, панель может

располагаться не только внизу, но и вверху экрана. Кроме того, их может быть две — и вверху, и внизу (как было по умолчанию в GNOME 2, и как ныне принято в его форке — MATE). Самое же главное — можно задавать произвольную высоту панели, с масштабированием текста и пиктограмм на ней:



Относительно автоматического скрытия всё понятно без комментариев — для кого-то это полезно, для иного же (например, для меня) — не удобно. А вот включение режима редактирования панели позволяет перетасовывать отдельные её элементы, в частности, пиктограммы в Launcher'е и в трее. Правда, при этом отключается запуск с помощью кнопок приложений на ней — то есть свои рабочие функции она выполнять перестаёт. Так что включать этот режим следует только на время — действительно при необходимости что-то перетащить.

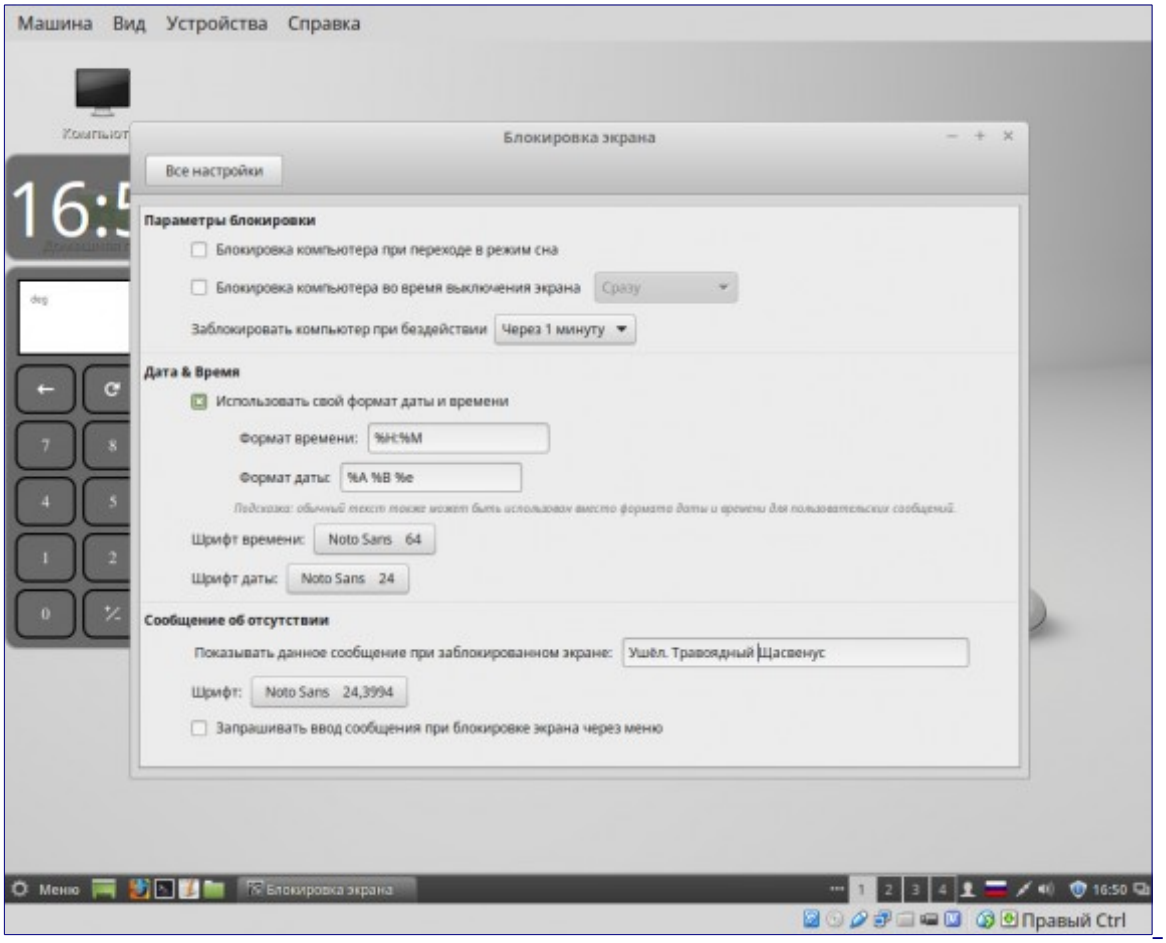
Здесь же уместно сказать и о модуле Блокировка экрана. Сама по себе блокировка с параметрами по умолчанию — штука, меня страшно раздражающая. Особенно когда для разблокирования требуется ввод пароля. Так что первым делом я её отключаю напрочь — даже на ноутбуке. Оставляю только включение, через разумный промежуток времени, скринсейвера (который здесь, впрочем, тоже называется блокировкой):



А вот сам скринсейвер — предмет гордости разработчиков, и гордости законной. Потому что в нём можно задать вывод времени и даты в период блокировки, причём в собственном формате. Кроме того, вместо времени и/или даты можно указать любой произвольный текст, например

Руки прочь от моего компа

Впрочем, текст можно указать и в специально предназначенном для этого поле, например, подобно Кристоферу Робину, вот такой:

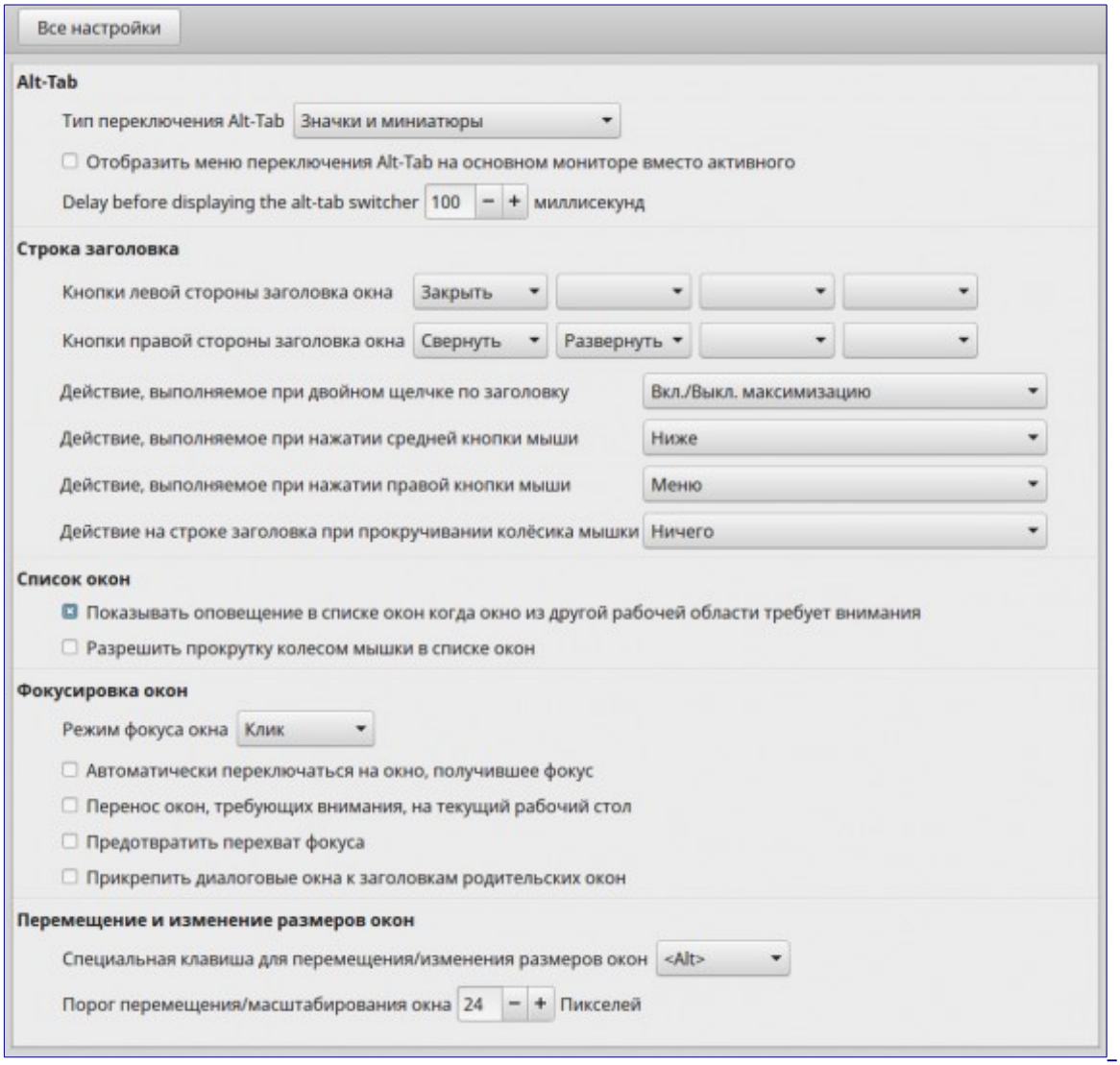


Как можно видеть на скриншоте, предшествовавшем этому, настройке поддаются и шрифты вывода времени, даты и текстового сообщения.

Окна и их тайлинг

В очерке об интерфейсе Cinnamon говорилось о средствах управления окнами по умолчанию. Настало время посмотреть, как эти умолчания можно изменить. Разумеется, делается это через пункт Окна. Здесь можно настроить стандартные действия при щелчках мышью на строке заголовка и режим передачи фокуса окну, если не устраивают умолчания — и думаю, очевидным способом. Меня умолчания устраивают, так что вдаваться в подробности не буду.

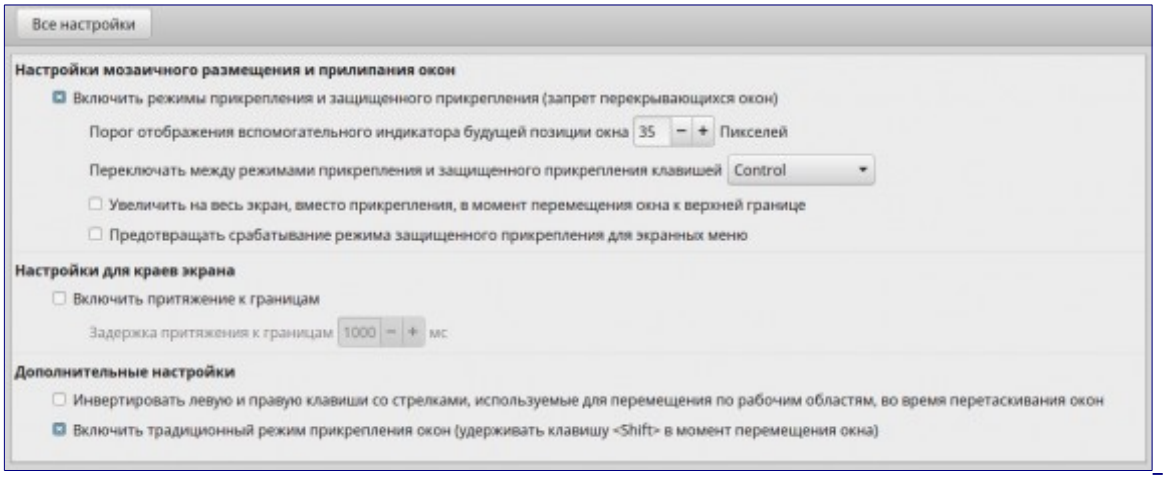
Самое же интересное здесь — это примирение «правосторонников»» и «левосторонников», то есть приверженцев расположения кнопок управления окном с той или другой стороны строки заголовка:



Как известно, в своё время революционная идея «левого уклона» кнопок в Ubuntu вызвала массу нападок со стороны «правых уклонистов» — твёрдых ~~искровцев~~ GNOME'вцев. В Cinnamon принято компромиссное решение — любые кнопки можно поместить как с левой стороны титульной строки, так и с правой.

Что, впрочем, тоже не ново, и испокон времён внедрено в оконных менеджерах KDE и Xfce. И чем я с давних пор пользуюсь — при общем «правом уклоне» делаю для кнопки закрытия окна «отмашку влево»: за долгие годы работы проверено, что это сильно снижает вероятность нажать её случайно, что обычно весьма не желательно.

В очерке об интерфейсе я много места посвятил описанию тайлинга окон — как одной из особенностей, делающих Cinnamon «лучшим из десктопов». Настраивается же тайлинг в модуле Прикрепление окон и притяжение... Вот только настраивать здесь особенно нечего: для приобщения ко всем прелестям тайлинга достаточно отметить боксик Включить режим прикрепления — а это и так сделано по умолчанию:



Что же до возможности изменить переключатель между режимами простого и защищённого прикрепления — мне она оказалась без надобности, умолчальный Control справляется с этой задачей не хуже других.

Смысл включения притяжения к границам (так называемый Flip) — вовсе не в каком-то притяжении, а, как мне объяснили резонные люди, в самом обычном переключении на соседнюю рабочую область при подведении курсора мыши к границе текущей. Это меня всегда раздражало во всех графических средах — и эту опцию я всегда отключаю.

На счёт инверсии клавиш со стрелками ничего сказать не могу, ибо не ощутил необходимости. А вот режим, названный традиционным, оказался не вредным. Только означает он, вопреки тому, как можно понять написанное, следующее: при его включении, если вам не хочется, чтобы какое-либо окно «превращалось в черепицу» при перемещении его к краю экрана, следует удерживать клавишу Shift.

Рабочие области и Горячие углы

Говоря в «интерфейсном» очерке о переключении между окнами, я вскользь упомянул о переключении в режим Expo, при котором на экран выводятся все наличные рабочие области. А уже в этом очерке просил обратить внимание на одноимённый апплет. Настало время поговорить обо всех этих материях подробнее.

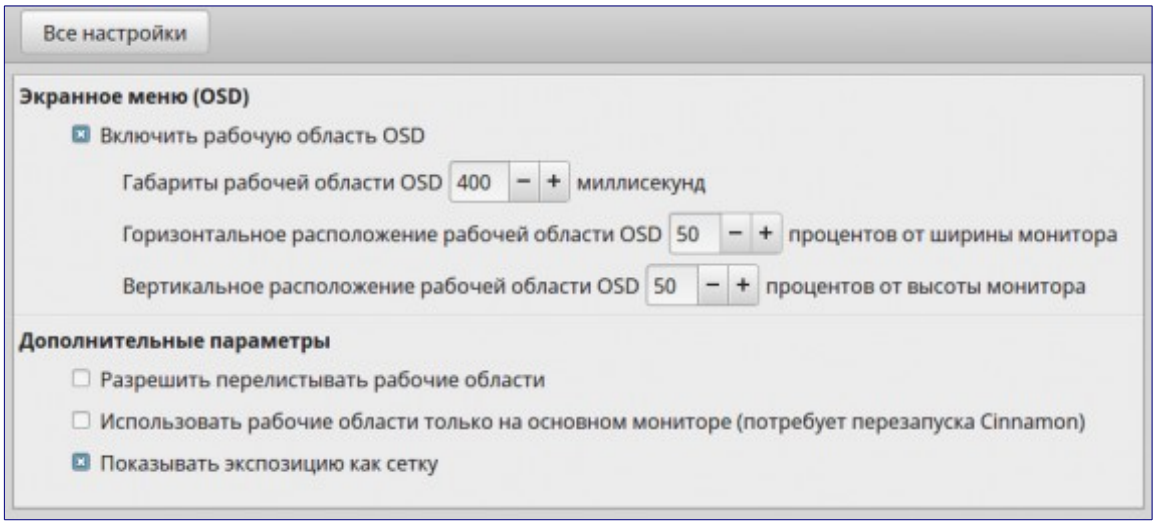
В том же «интерфейсном» очерке говорилось, что рабочих областей в Cinnamon по умолчанию две, и способа изменить это число на поверхности не видно, а переключение между имеющимися возможно только по комбинации клавиш Control+Alt с Right или Left. И всё это правда, чистая правда, но далеко не вся правда.

Так, найти альтернативный способ переключения между рабочими областями не так уж и сложно даже при слабом знании английского. Это — упомянутый ранее апплет Workspace switcher: будучи включённым, он выводит на панель обычный переключатель рабочих столов, привычный всем пользователям интегрированных сред и многих оконных менеджеров:



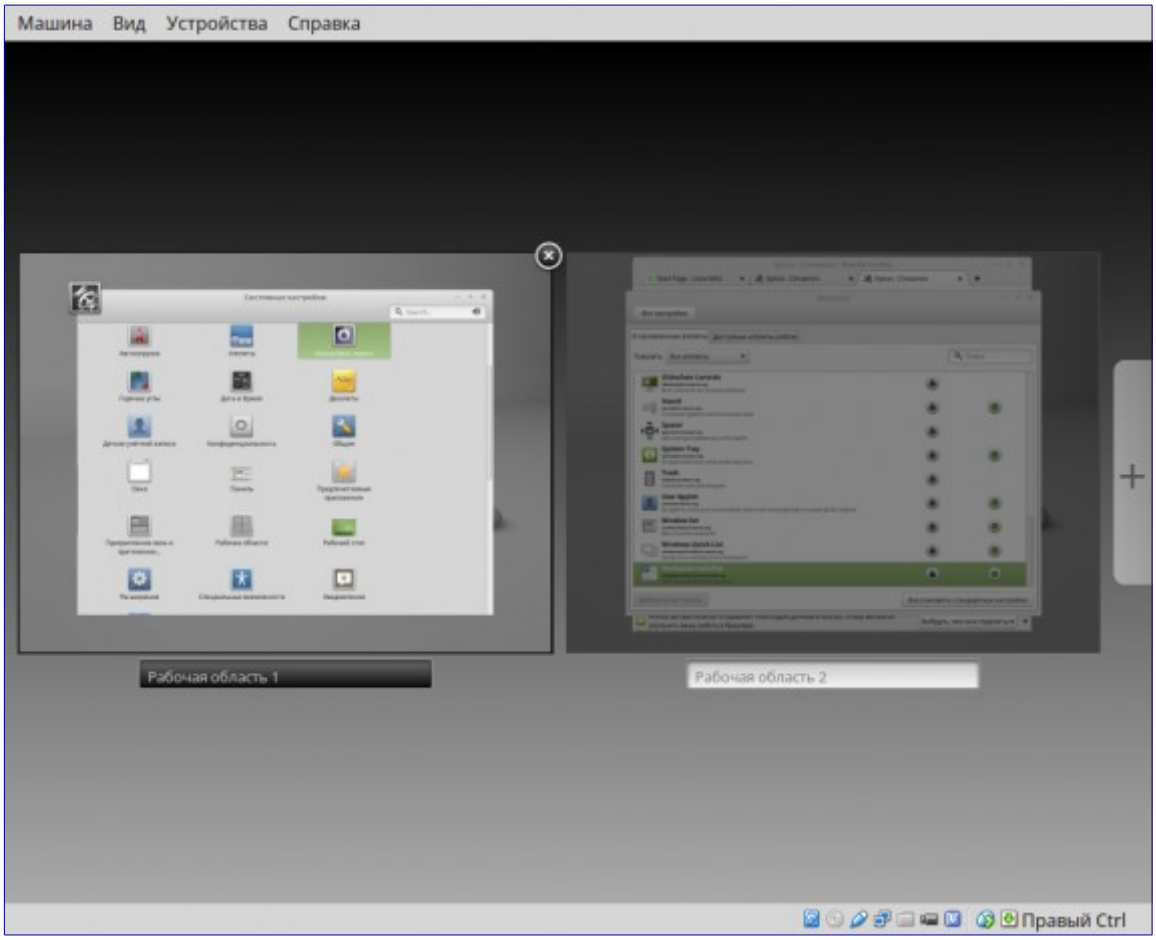
Далее, логично было бы ожидать, что количество рабочих областей можно задать в модуле, который называется Рабочие области. Однако, открыв его,

мы не увидим там и намёка на эту опцию:

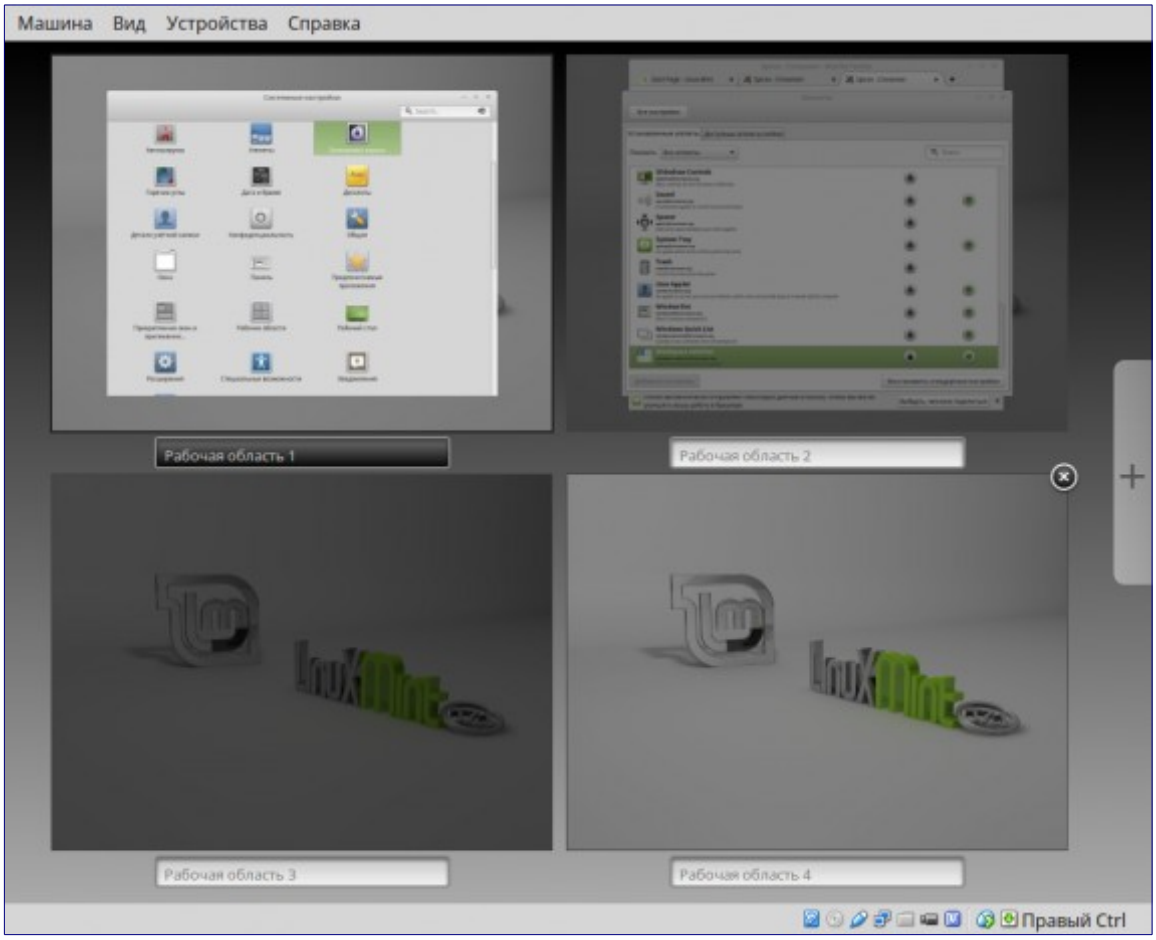


А смысл всех остальных опций этого модуля остаётся не очень понятным — так что рассмотрение его немного отложим.

Потому что настало время вспомнить об апплете Expo. Ибо он служит для переключения Cinnamon в одноимённый режим — режим вывода на экран всех наличных рабочих областей одновременно:

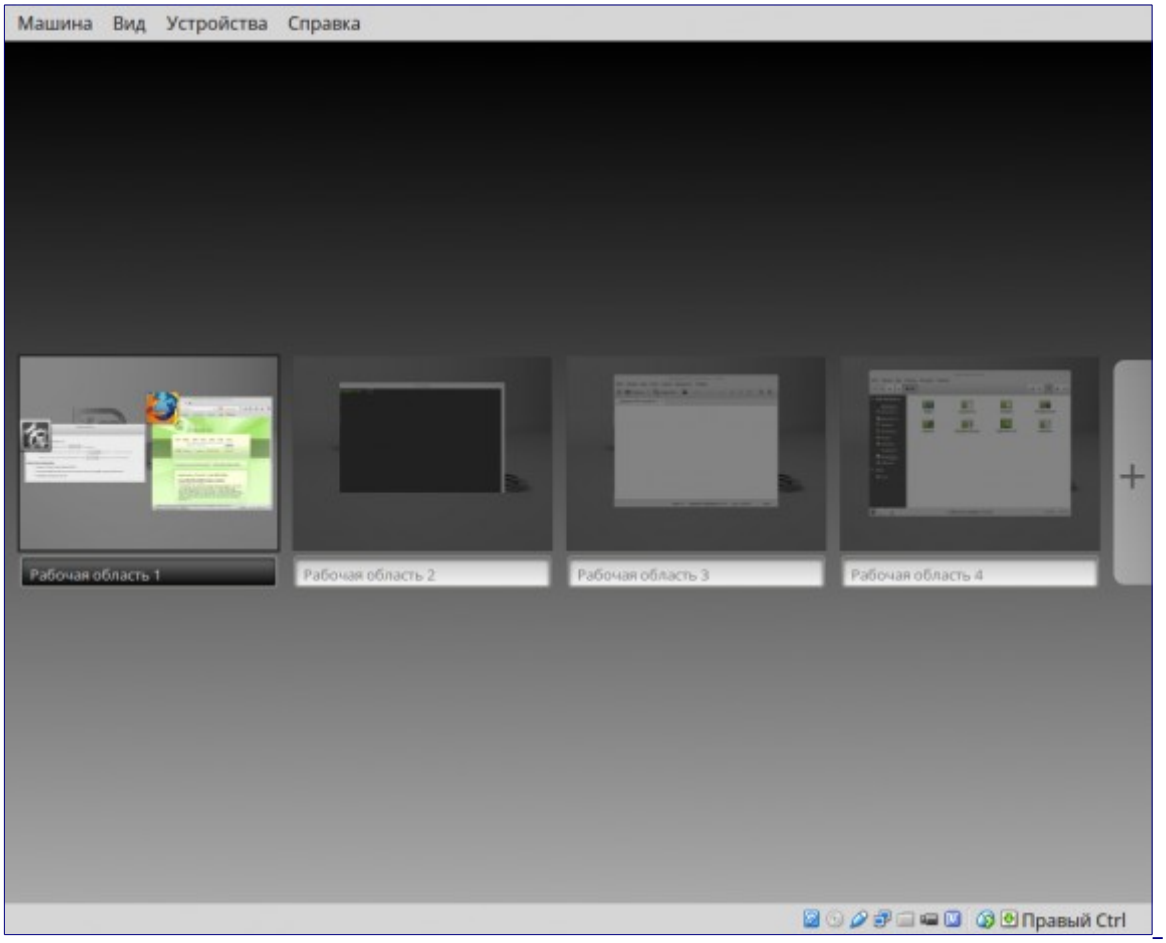


Очевидно, что это ещё один способ переключения между рабочими областями — для чего достаточно кликнуть мышью на нужной. Но главное — это единственный способ увеличить их число, для чего служит большой жирный плюс с правой стороны экрана. А уменьшить количество рабочих областей можно с помощью крестика, появляющегося в правом верхней углу любой области при наведении на неё курсора:

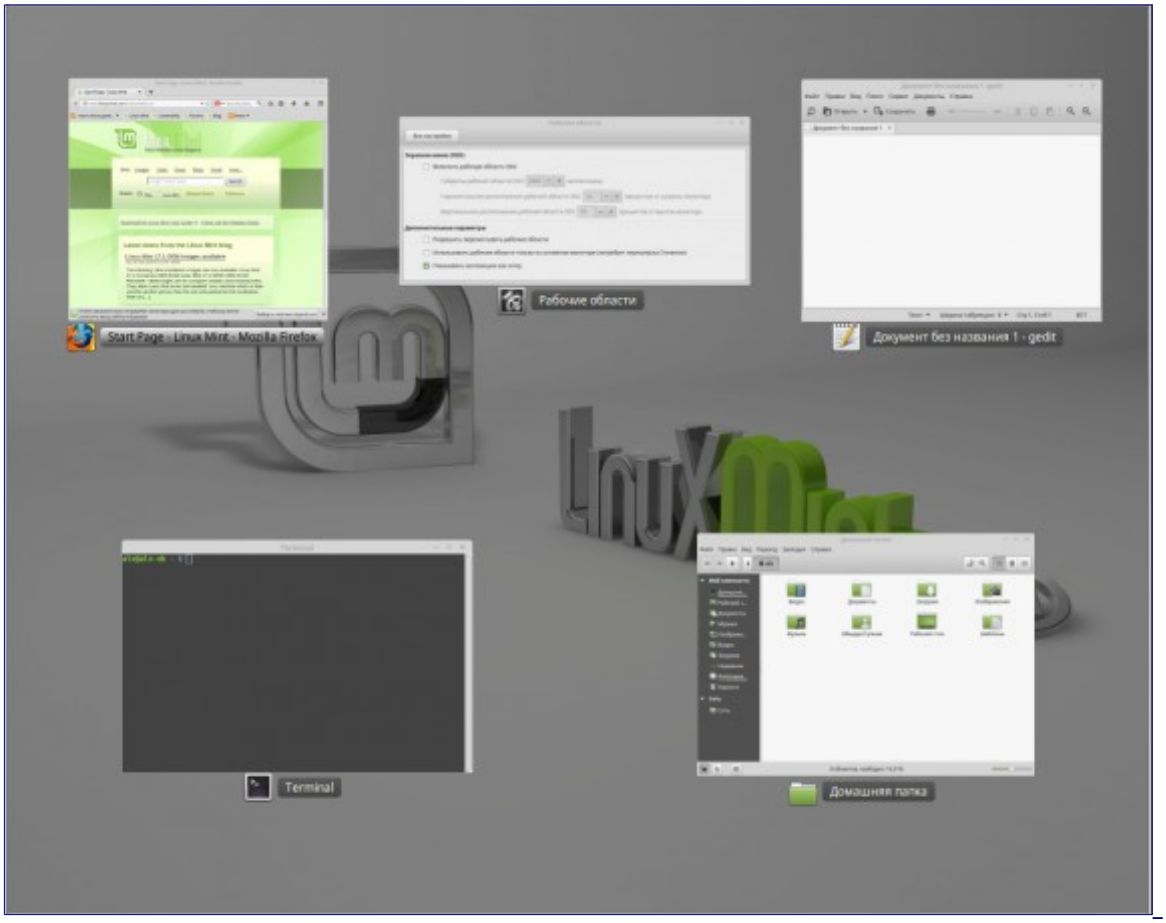


Вот теперь, просветлев относительно режима экспонирования, можно вернуться к модулю Рабочие области, дабы осознать смысл его опций. И действительно, можно предполагать, что загадочная опция Гарабиты, которые почему-то измеряются в миллисекундах — это время задержки переключения в режим экспонирования, горизонтальное и вертикальное расположение в процентах — это размеры рабочих областей при представлении в этом режиме, перелистывание — возможность перемещёния между ними, прокручивая колёсико мыши. И радоваться своей солдатской смекалке.

Радость эта омрачается одним: изменение любой опции не влечёт за собой никаких последствий от слова абсолютно. И в результате оказывается, что единственный рабочий пункт этого модуля — Показывать экспозицию как сетку. По умолчанию он включён. Снятие же с него отметки приводит к тому, что рабочие области в Expo-режиме выстраиваются в одну линию:

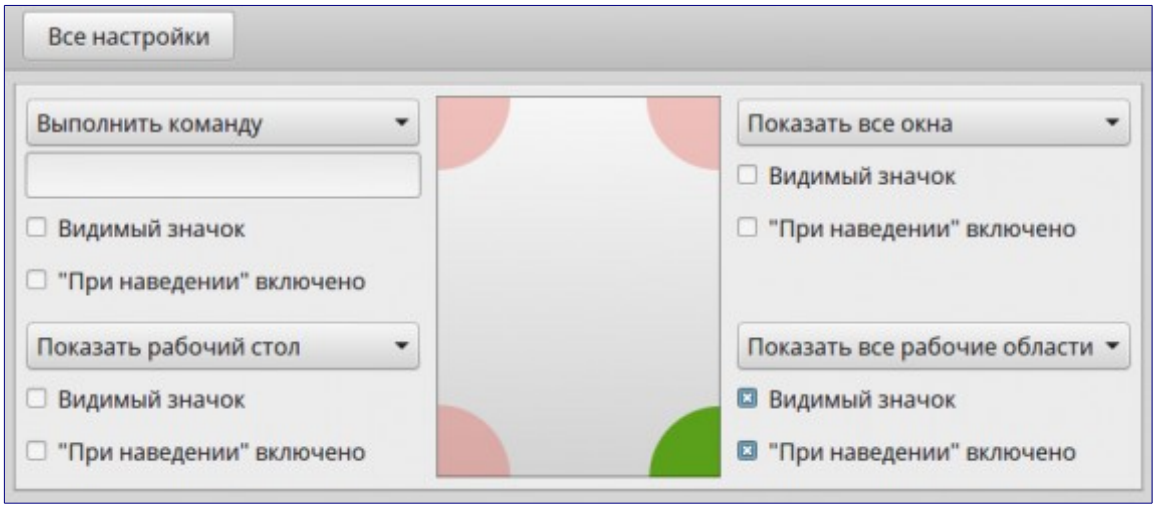


Кстати, апплет Expo — не единственный способ переключения в одноимённый режим. Согласитесь, что было бы странно, если такая важная функция выполнялась с помощью внешнего апплета, да ещё и не включаемого по умолчанию. Главный, встроенный, способ перехода в режим экспонирования рабочих областей — комбинация клавиш Control+Alt+Up. Задействована и противоположная комбинация — Control+Alt+Down: Она переводит Cinnamon в режим, который можно назвать экспонированием окон (или режимом масштабирования): одновременный вывод всех окон всех открытых приложений текущей рабочей области, в том числе и свёрнутых:

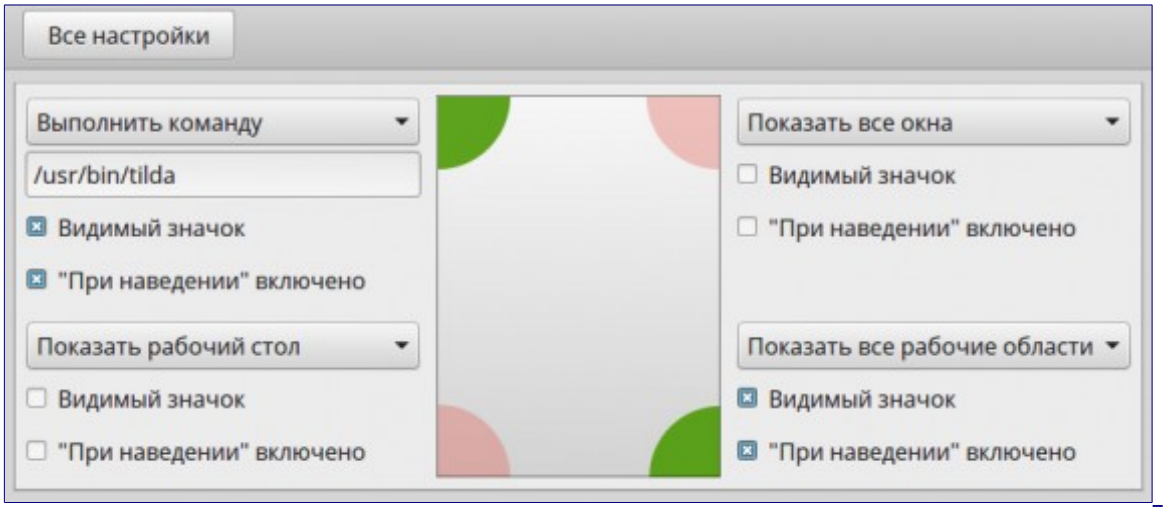


Однако и это ещё не всё: существует ещё один способ переключения в оба режима экспонирования. По умолчанию он отключён, а за его настройку его отвечает модуль Горячие углы. Он обеспечивает привязку к любому из углов экрана одного из трёх действий: переключения в режим экспонирования рабочих областей, в режим масштабирования окон или очистку рабочего стола (то есть сворачивания всех окон). Действия эти происходят при подведении курсора мыши к соответствующему углу. Повторное перемещёние курсора в тот же угол возвращает Cinnamon в нормальный рабочий режим.

На следующем скриншоте в качестве «горячего» выступает правый нижний угол — к нему привязано экспонирование рабочих областей:



Кроме того, потенциально через «горячий угол» можно выполнить произвольную команду. Так, на следующем скриншоте приведена попытка настроить правый верхний угол на вызов выпадающего терминала Tilda:

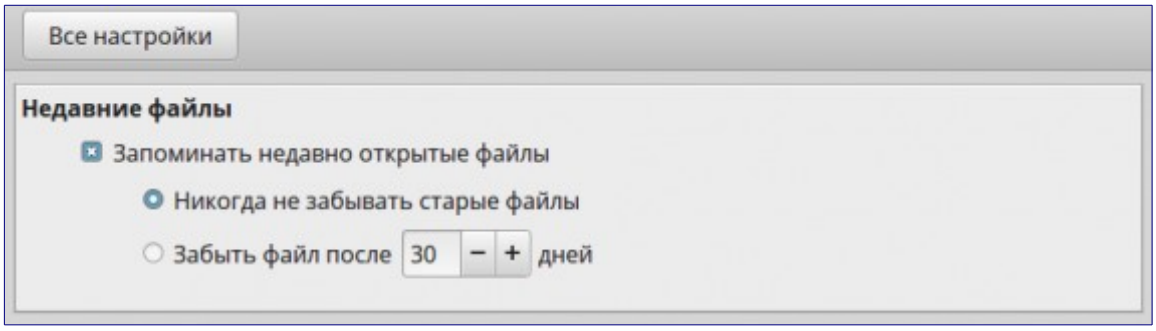


Однако практика показала, что это неудобно, в том числе и потому, что при каждом наведении на «горячий угол» вызывался новый экземпляр терминала, который для начала желал, чтобы его настроили. А никакого другого применения этой фиче я не придумал. Может, у кого из читателей фантазия окажется богаче?

Конфиденциальность, Общие, Уведомления

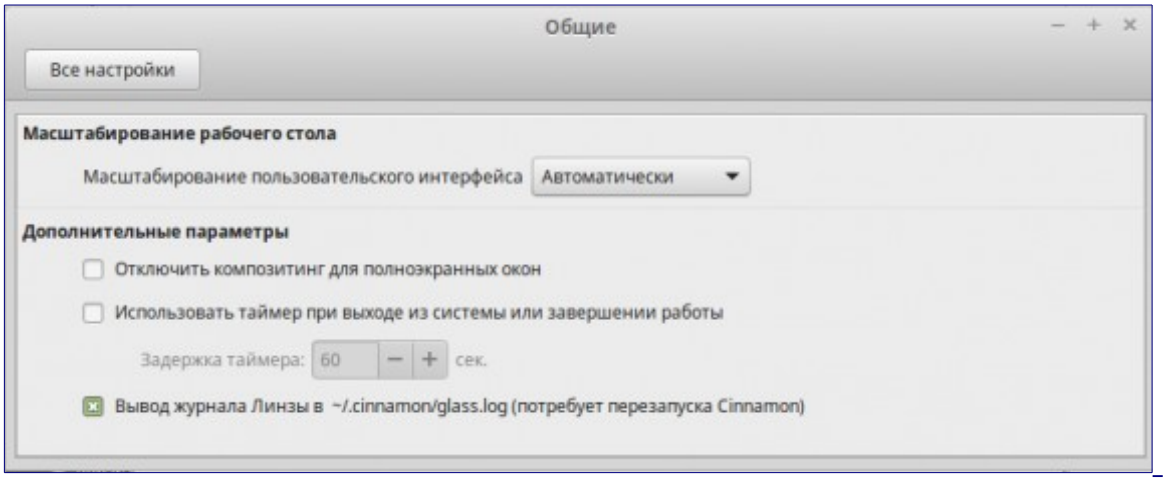
Общего между этими тремя модулями, пожалуй, только то, что они были переработаны при переходе на Cinnamon версии 2.4.

Вся настройка конфиденциальности сводится к решению вопроса, запоминать ли открываемые файлы, и если запоминать — навсегда или на какой-то определённый срок:

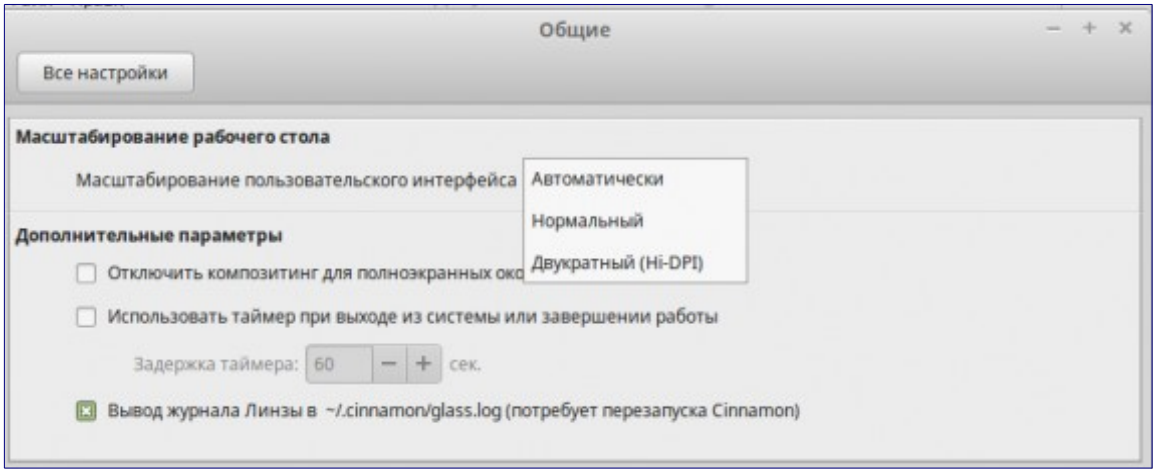


Я остановился на варианте «вечного хранения».

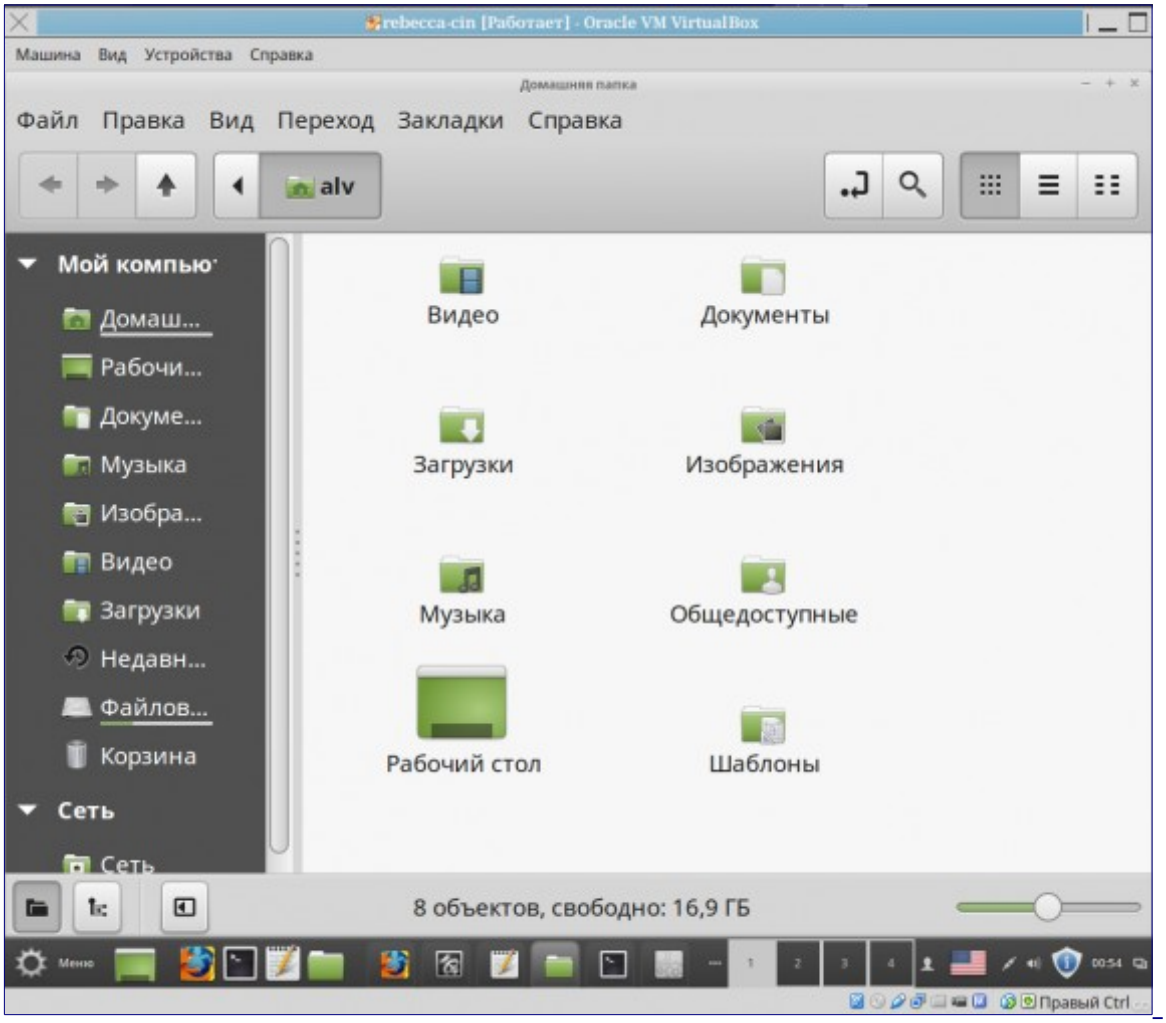
В модуле Общие обнаружилась интересная функция — возможность масштабирования рабочего стола. Для этого надо значение по умолчанию, Автоматически



заменить на Двукратный (HiDPI):

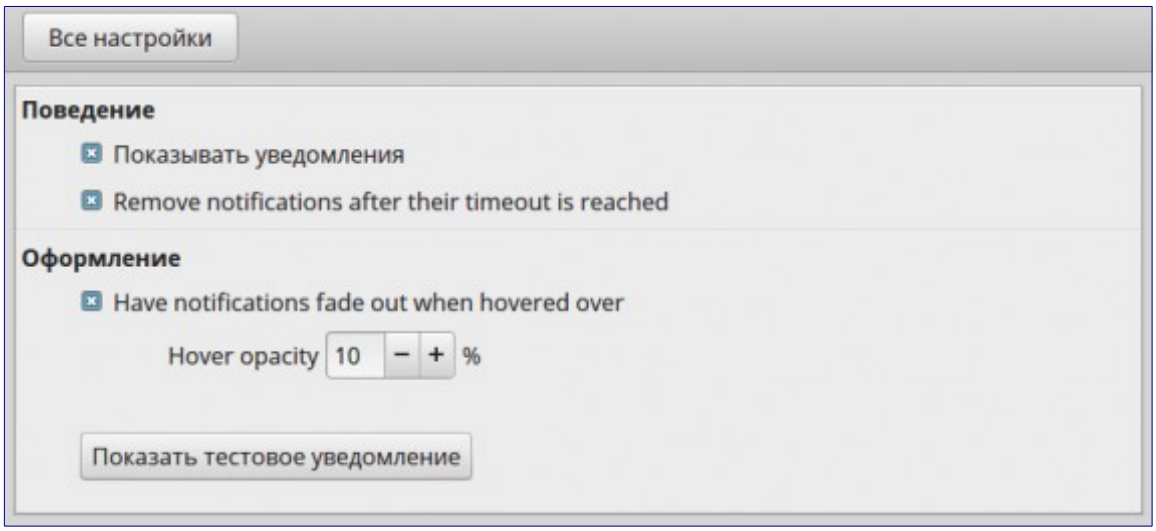


Правда, после этого рабочий стол приобретает устрашающий вид даже для моего зрения:

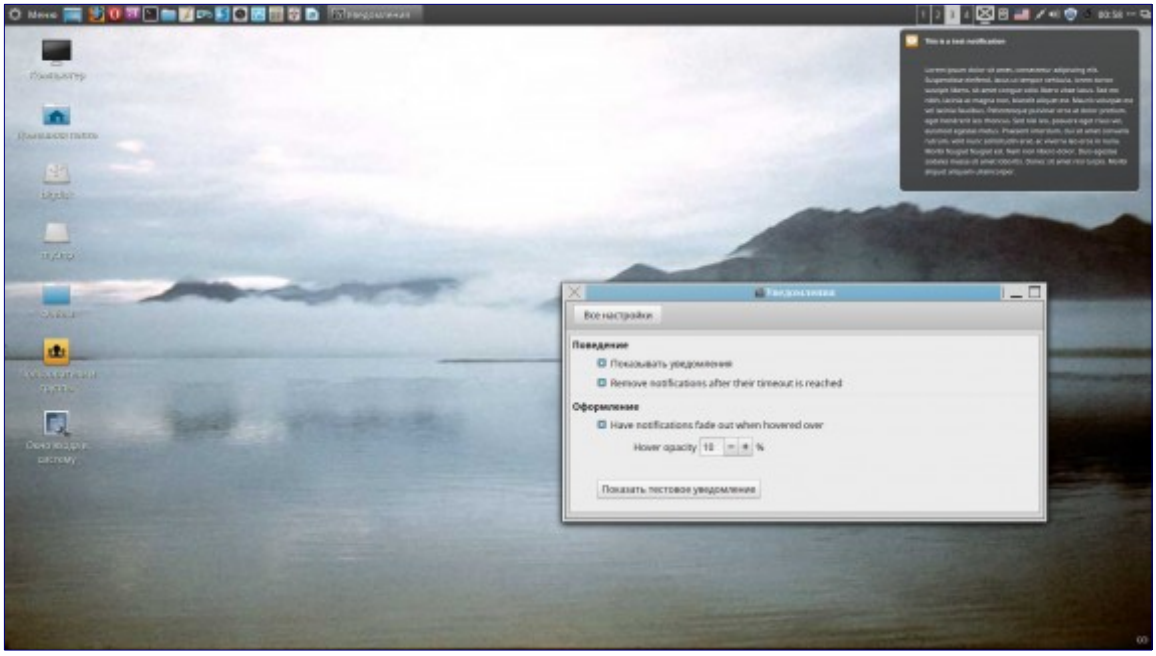


Но возможно, что в некоторых случаях это будет менее плохо, чем не видеть никакого вида вообще. Ну а пока я поспешил вернуться к виду нормальному (он оказался идентичным автоматическому).

В модуле Уведомления можно, во-первых, отказаться от их вывода (по умолчанию он, разумеется, включён). Они часто раздражают — например, сообщения IM-клиента о том, что гражданин Имя Рек вошёл в сеть, а гражданин Некто из неё, напротив, вышел. Однако совсем отключать уведомления неправильно, потому что временами они несут разумное, хотя обычно как раз не доброе. А вот удалять сообщения по истечении тайм-аута — как раз разумно, что и проделано на скриншоте:



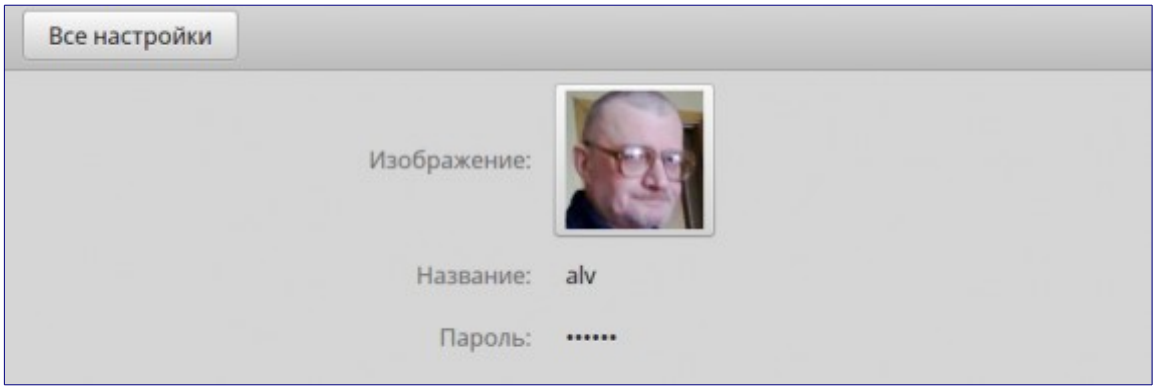
Кроме того, здесь же можно отрегулировать прозрачность уведомления, и немедленно протестировать это дело нажатием на соответствующую кнопку:



Я уменьшил прозрачность практически до минимума, так как иначе ничего не видел. А ведь уведомления — для того, чтобы их читать, не так ли?

Детали учётной записи

Не смотря на название модуля, Детали учётной записи, никаких особых деталей там нет:

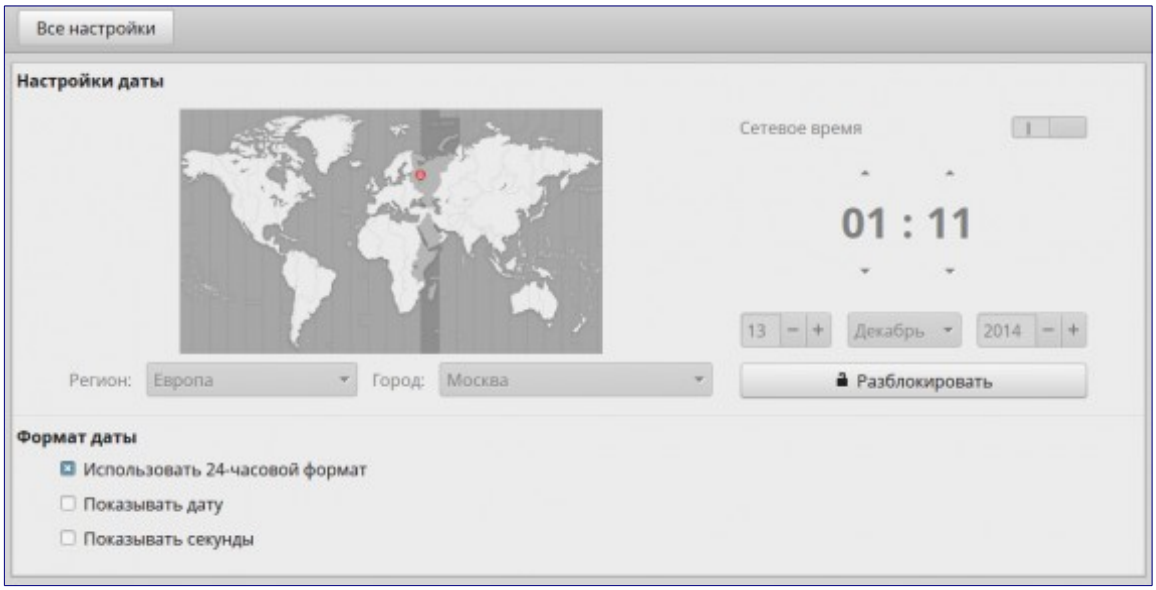


А есть возможность поменять пароль (к чему в некоторых случаях время от времени прибегать приходится) и изменить логин. Хотя вот последнего как раз делать не следует: численный идентификатор пользователя (так называемый UID) при этом не меняется, так что, скажем, на правах доступа к файлам это сказаться не должно. Но в каких-то случаях программы могут обращаться не к UID'у, а именно к названию аккаунта, и ничего, кроме путаницы, это не даст. Зато можно приклеить собственную аватарку, как это видно на скриншоте.

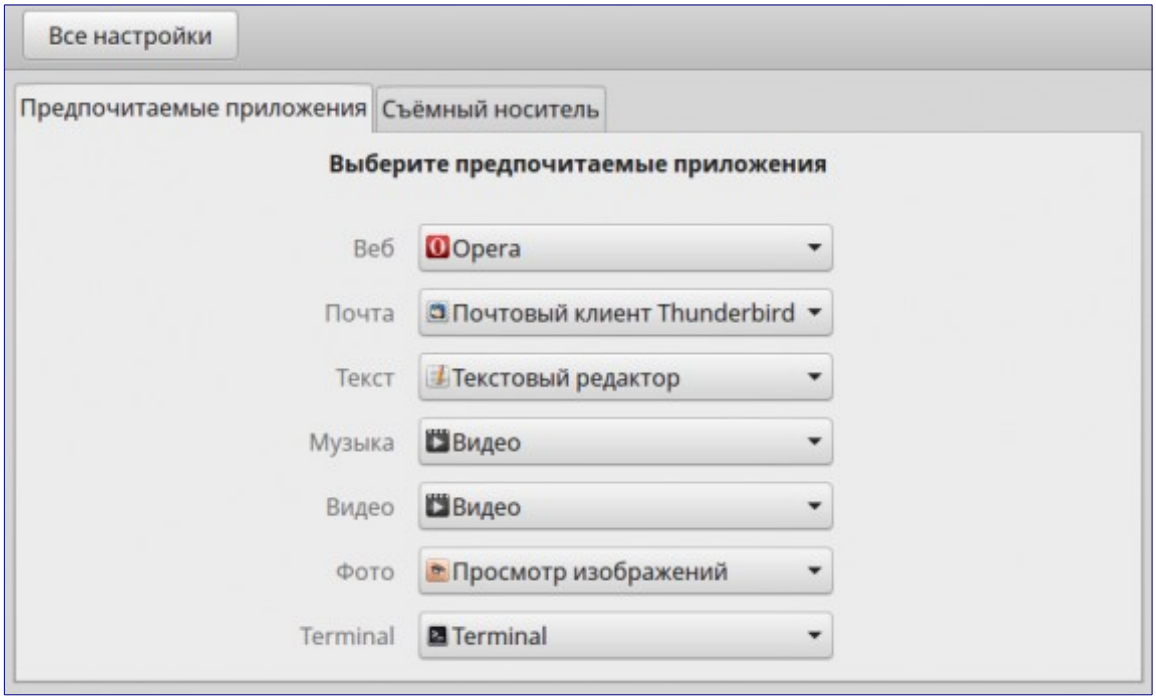
Прочие параметры

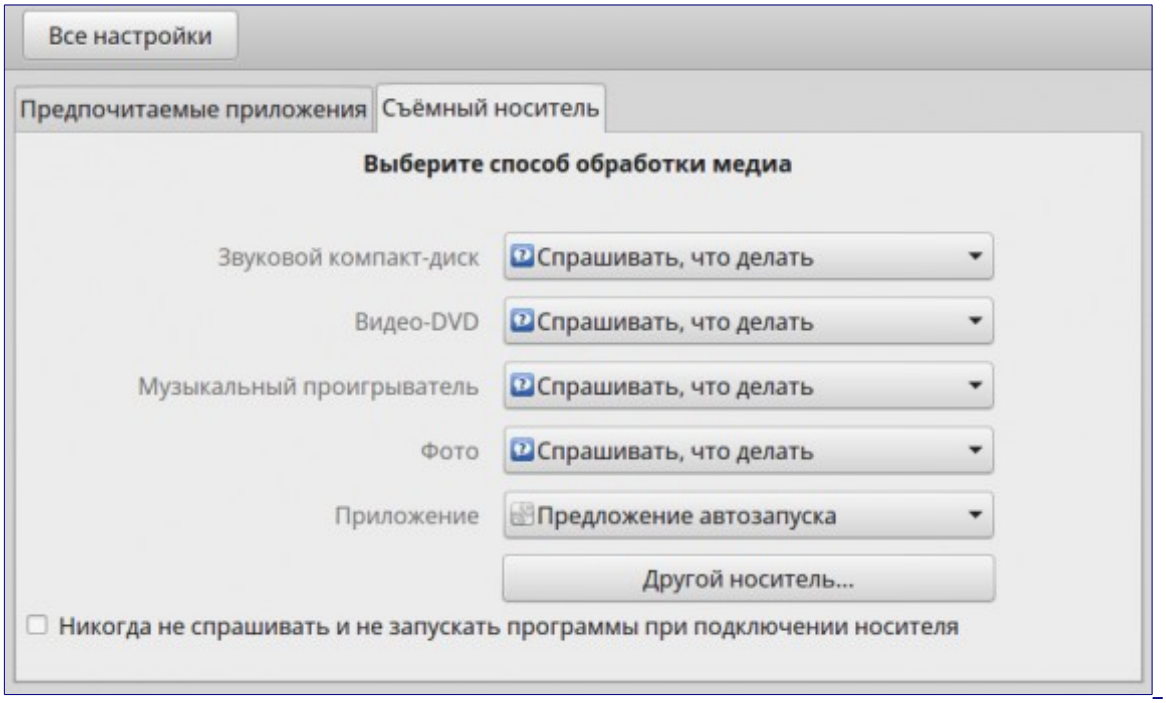
По остальным пунктам секции Параметры я позволю себе пробежаться галопом — они или не очень существенны (с моей точки зрения), или тривиальны. Неохваченными у нас остались (слева направо и сверху вниз):

Дата и время, смысл которого очевиден:

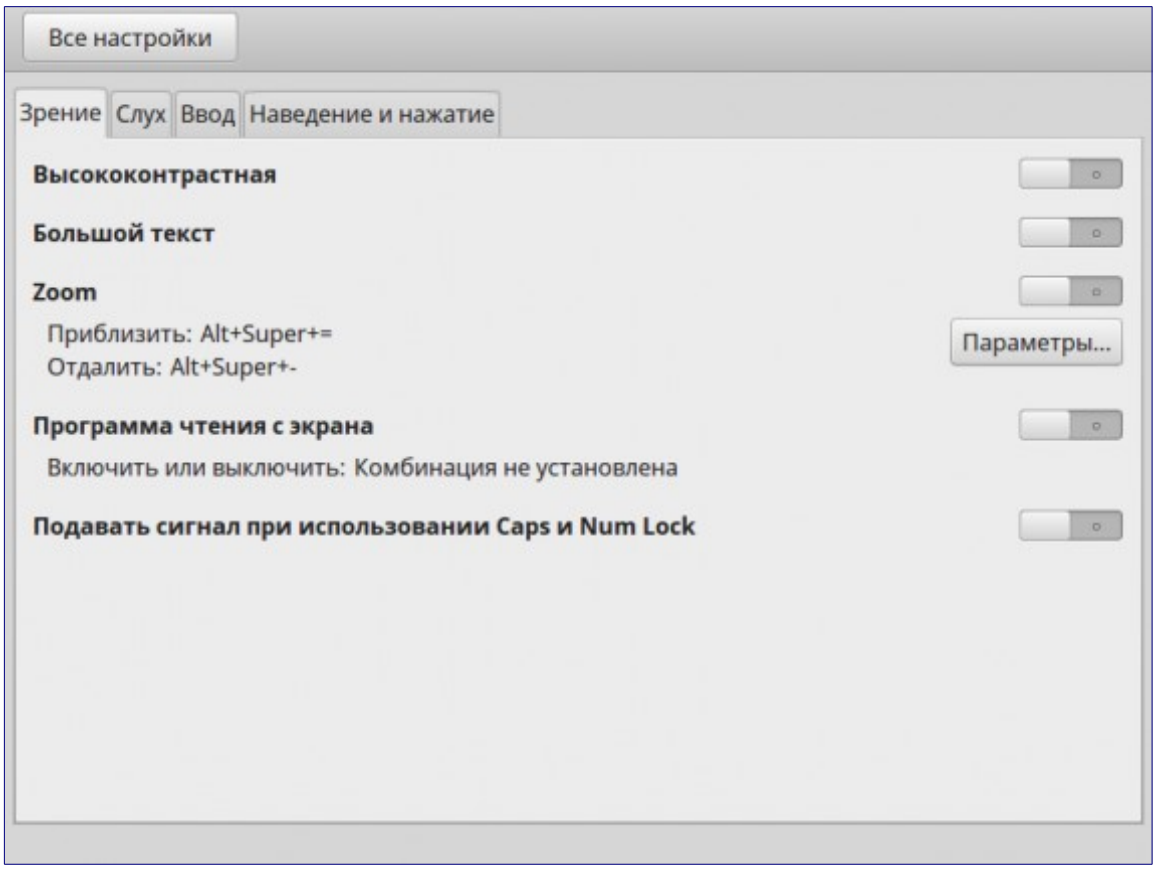


Приложения и съёмные носители — аналогично:





Ну а Специальные возможности — они и есть специальные:

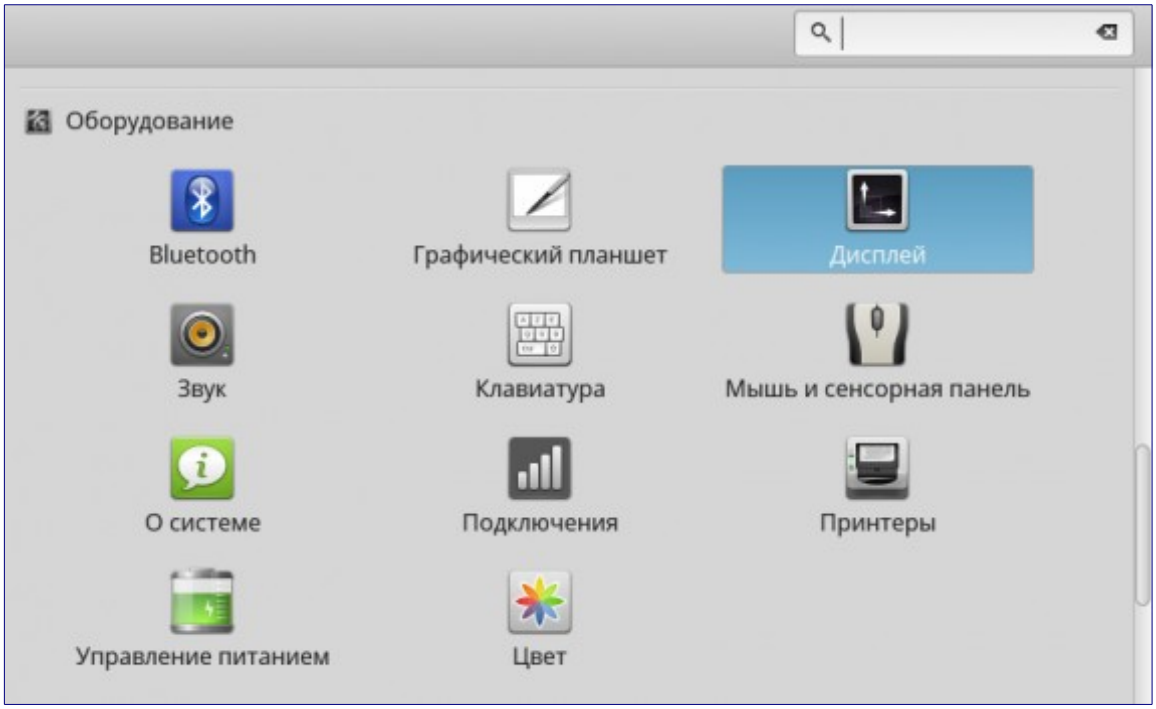


Не смотря на то, что они предусмотрены во всех рабочих средах и вроде бы предназначены как раз для таких как я (в частности, плохо видящих), прибегнуть к ним мне не приходилось: они либо не нужны, либо бесполезны.

Я умышленно не сказал ничего о модуле Языки. Потому что на самом деле это не модуль Системных настроек Cinnamon, а одна из фирменных утилит Mint, в ряду которых и будет рассмотрена в одном из последующих очерков.

Секция Оборудование

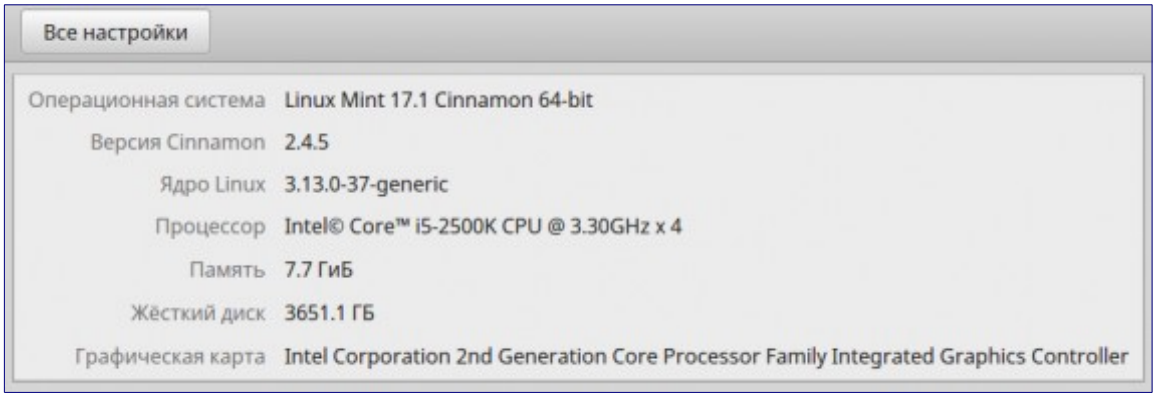
В секции Оборудование собраны модули, имеющие, как ни странно, отношение именно к оборудованию. И обозреть их все единым взглядом можно здесь:



А поскольку внутренней логикой они не особо между собой связаны (за одним исключением), и к тому же кое-какого оборудования у меня просто нет, то рассматривать эти модули я буду в произвольном порядке, руководствуясь собственными соображениями о их важности. Начав, однако, с самого общего — О системе.

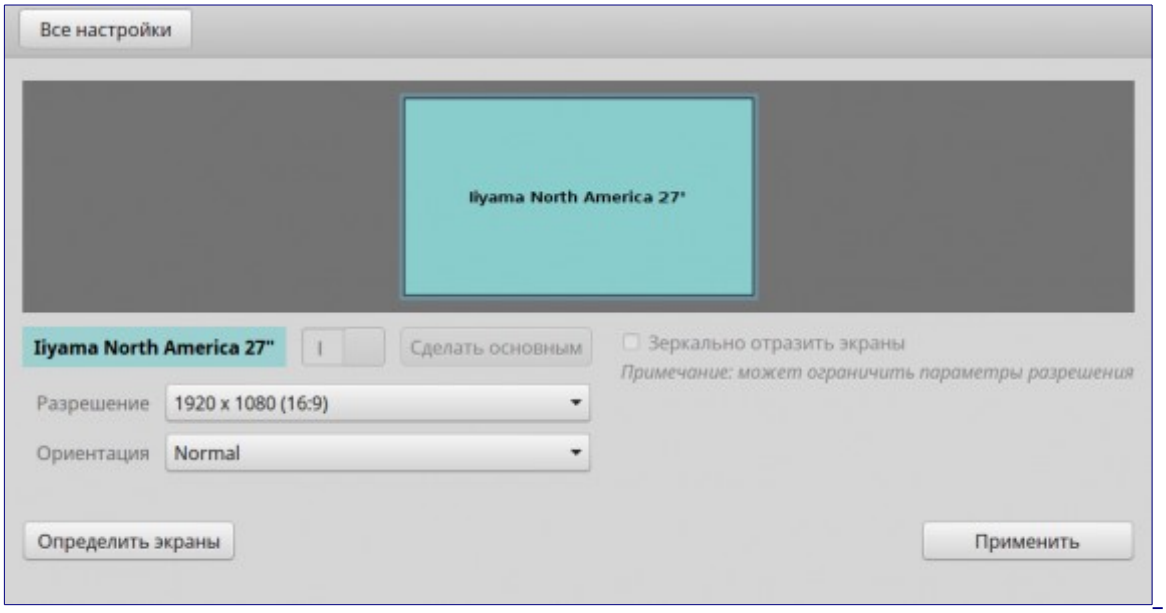
О системе и дисплее

Модуль О системе не предусматривает никаких настроек, а просто выводит сведения о ней, родимой, в простой и наглядной форме:



Думаю, тому, кто знает такие слова, как операционная система или процессор, не составит очень большого труда догадаться о том, что означают и соответствующие им значения.

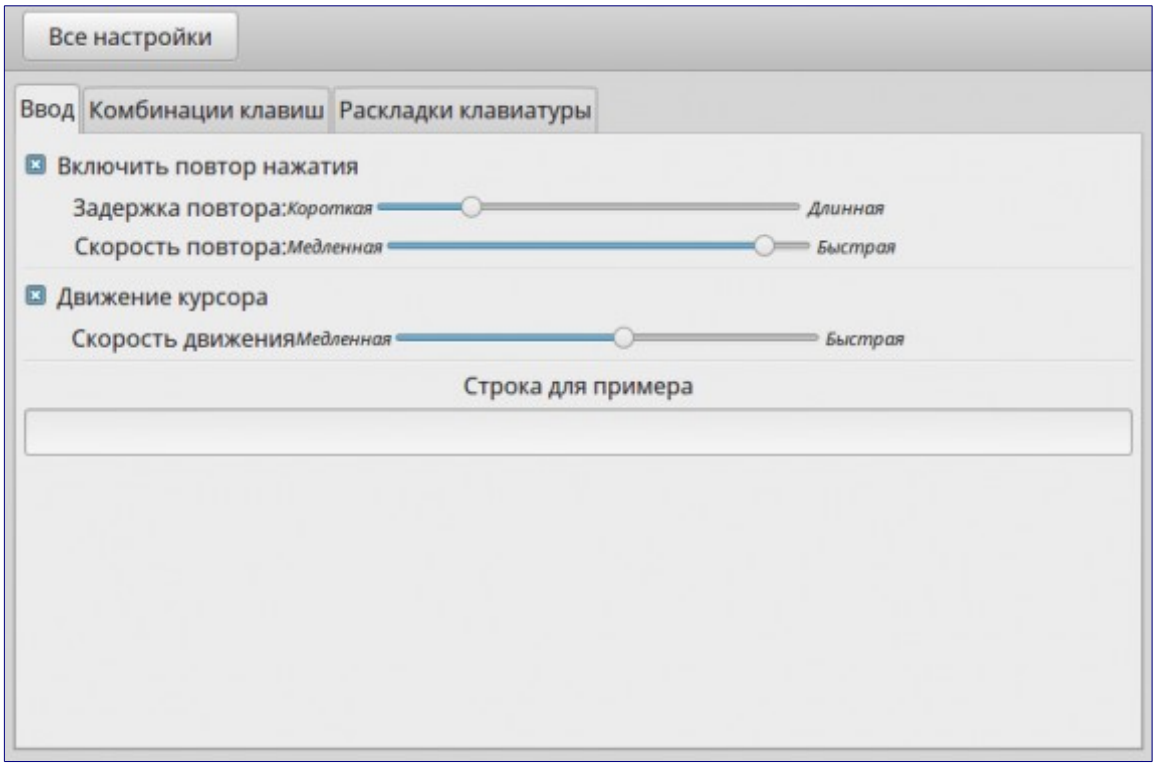
Модуль Дисплей я приплюсовал сюда же, так как в случае одномониторной конфигурации, да ещё и не с поворачивающимся экраном, он тоже просто выводит самые общие сведения о мониторе и его разрешении:



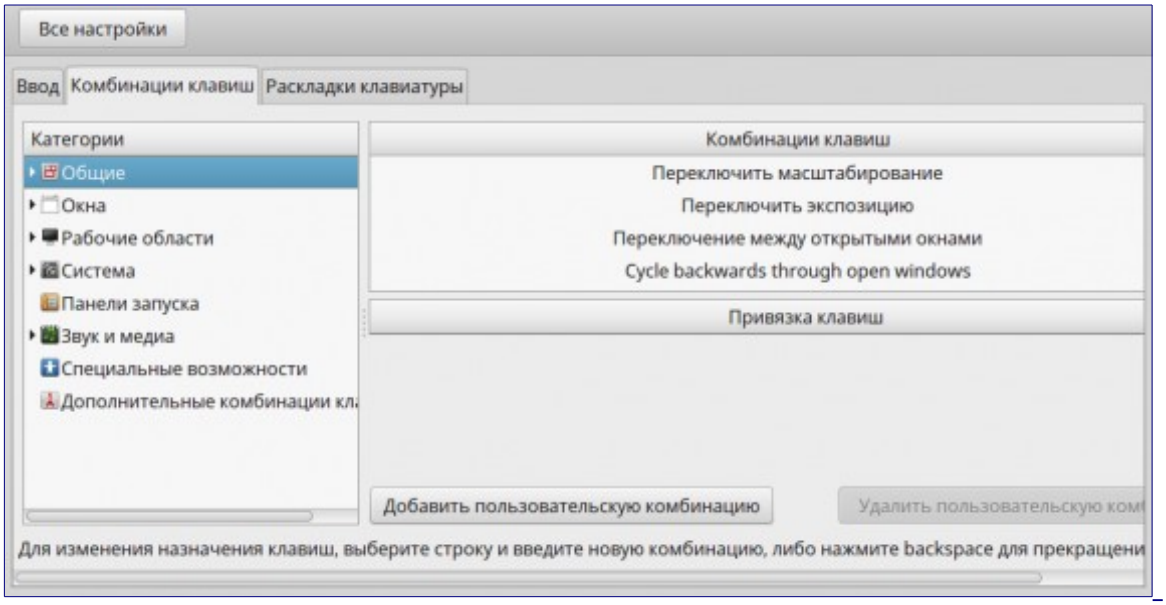
Во всех же прочих пунктах секции предполагается совершение некоторых настроечных действий. Из них для нас важнейшим является настройка клавиатуры — с неё-то и начнём.

Клавиатура

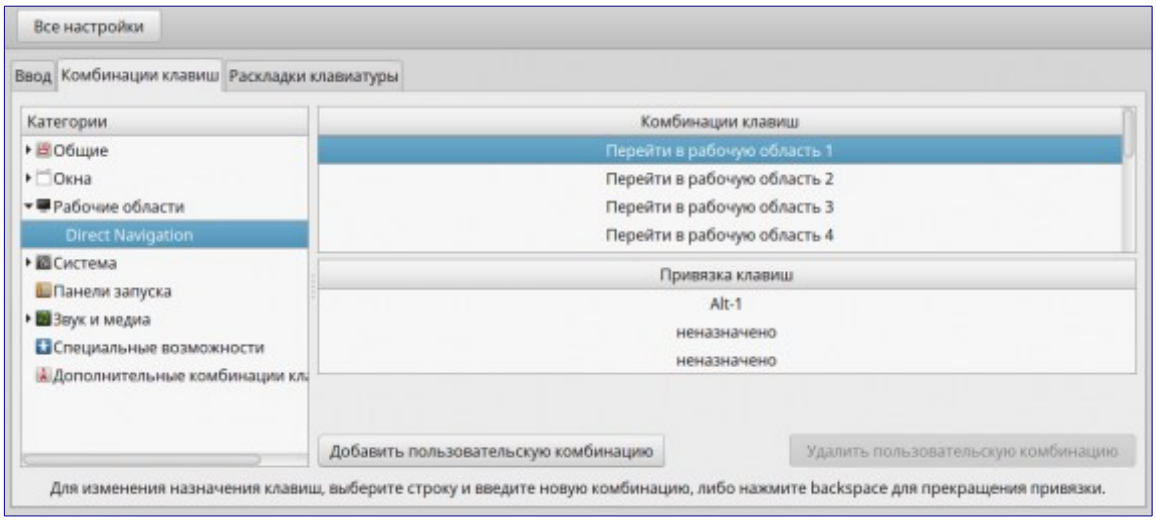
В модуле Клавиатура для начала можно настроить поведение при нажатии клавиш и реакцию курсора:



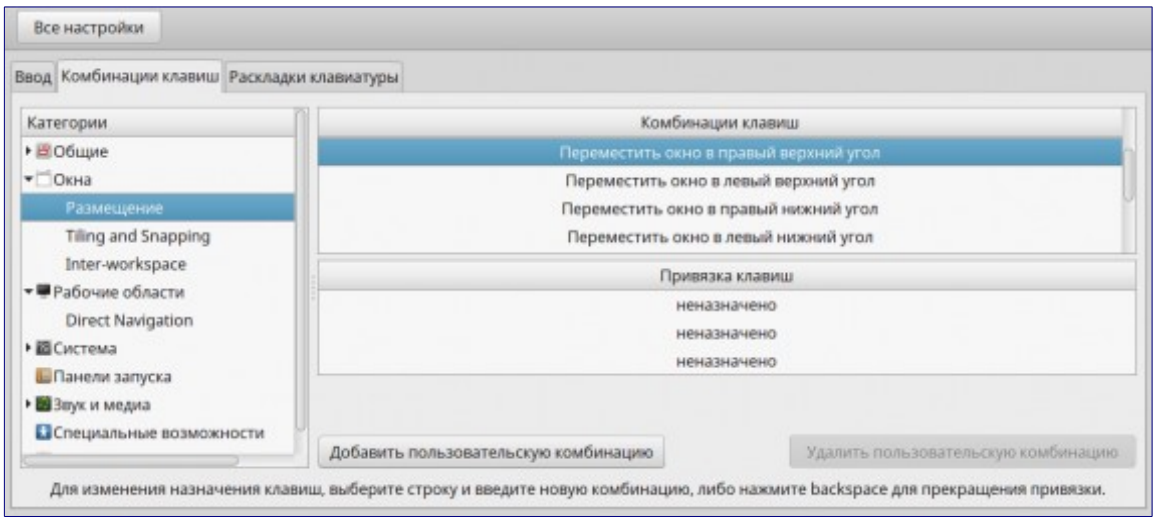
А во второй вкладке, Комбинации клавиш, определить хоткеи на очень многие случаи жизни:



В частности, здесь я в обязательном порядке устанавливаю клавишные комбинации для переключения между рабочими областями — Alt+1, Alt+2 и так далее (пункт Рабочие области):

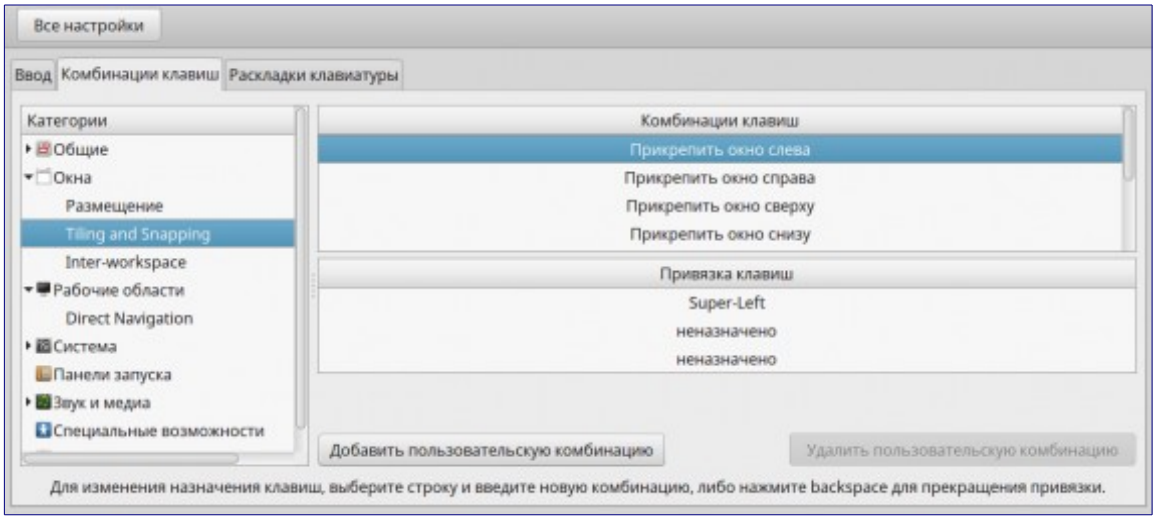


В некоторых случаях может быть полезным назначение хоткеев для перемещёния окон в подпункте Размещёние пункта Окна — в правый верхний угол, левый верхний угол, и так далее (по умолчанию не включены):

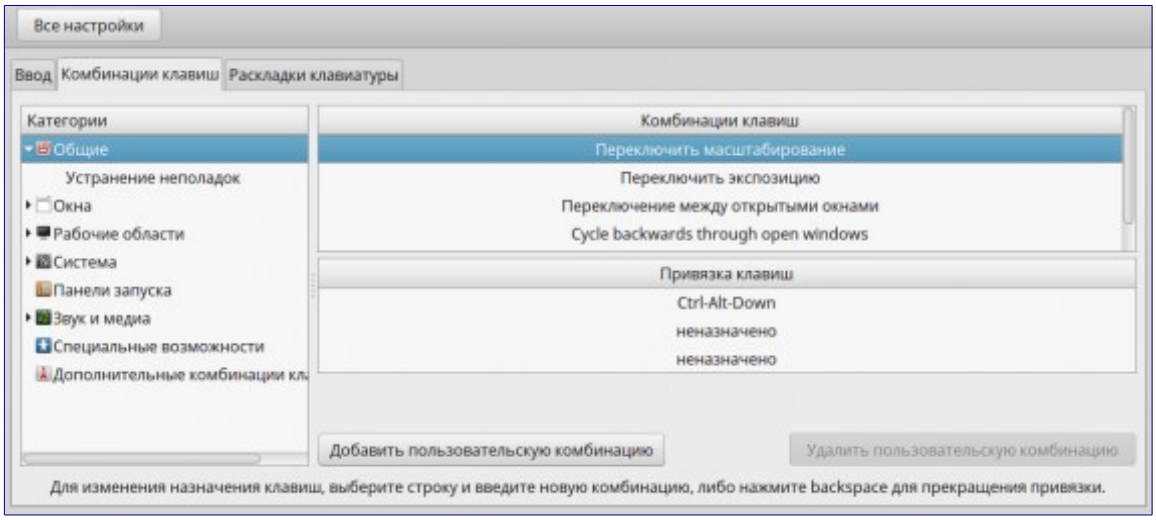


Такие перемещёния не следует путать с управлением тайлингом — размер

окна при этом не меняется. А собственно настройки управления тайлингом выполняются в том же пункте Окна — соответствующий подпункт так и называется, Tiling and Snapping. Именно здесь можно изменить умолчальные комбинации типа Super плюс стрелки, описанные в предыдущем очерке:



В пункте Общие (он идёт первым в списке) можно перенастроить переходы в режим Expo и масштабирования:



Раньше здесь же можно было лишить клавишу Super_L (она же — левая win-клавиша) её сакрального значения — вызова меню. В версии 2.4 эта возможность пропала, но кто знает? Может, появится снова. Ведь Cinnamon — не Unity, где всё на супер-клавишу завязано, так что нет смысла трястись над этой священной коровой.

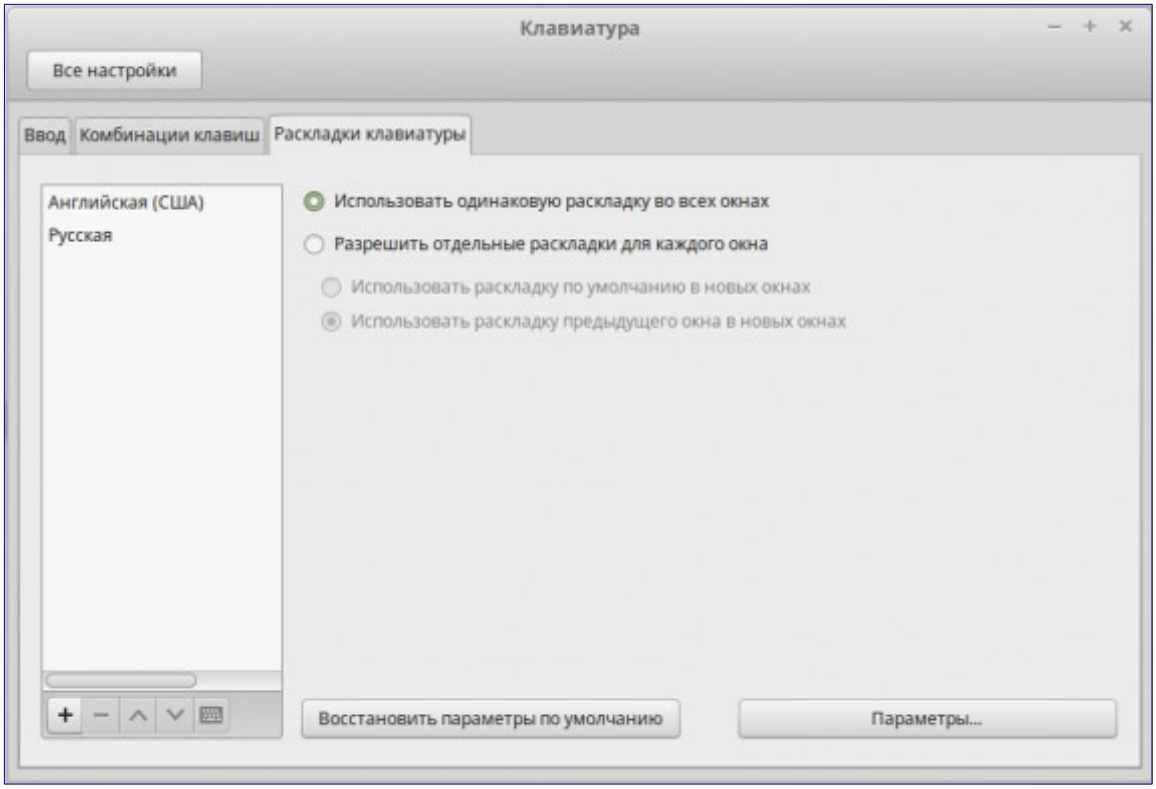
В мои цели не входит описывать все действия, для которых можно определить комбинации. Но думаю, что и сказанного достаточно для иллюстрации факта: перед любителями оперировать хоткеями открывается очень широкое поле деятельности.

Вкладка Раскладки клавиатуры очень важна для для многих применителей-текстовиков, и потому её рассмотрение выделяется в особое производство.

Раскладки и переключатели

Настройка раскладок клавиатуры и переключателей между ними может быть не актуальной для тех применителей, которых устраивают умолчания

инсталлятора — вариант winkeys для русской раскладки и комбинация Alt+Shift в качестве переключателя раскладок:

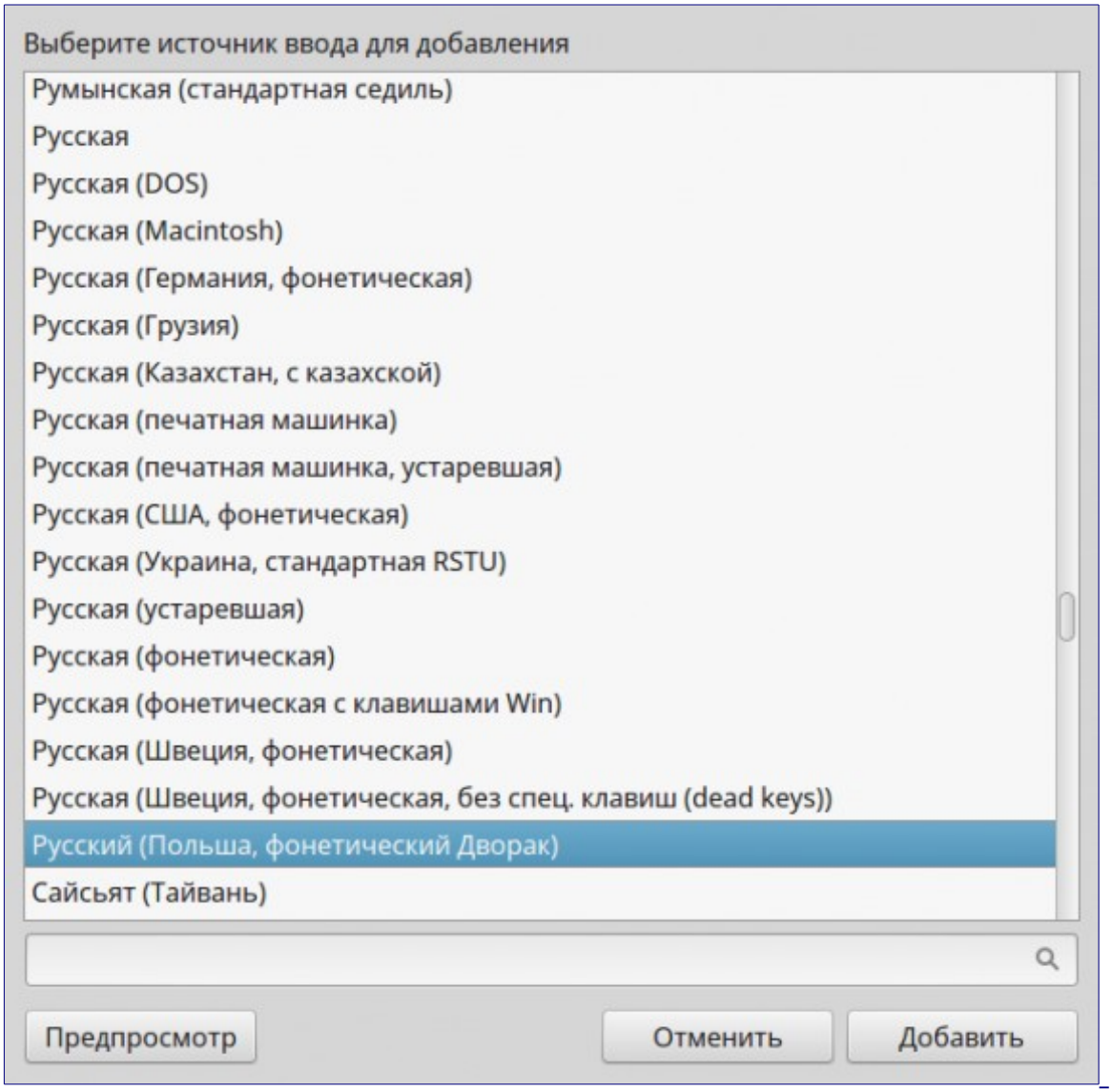


Но для требовательных применителей-текстовиков здесь есть все возможности не изменять своим привычкам, а заодно и приобрести новые, полезные.

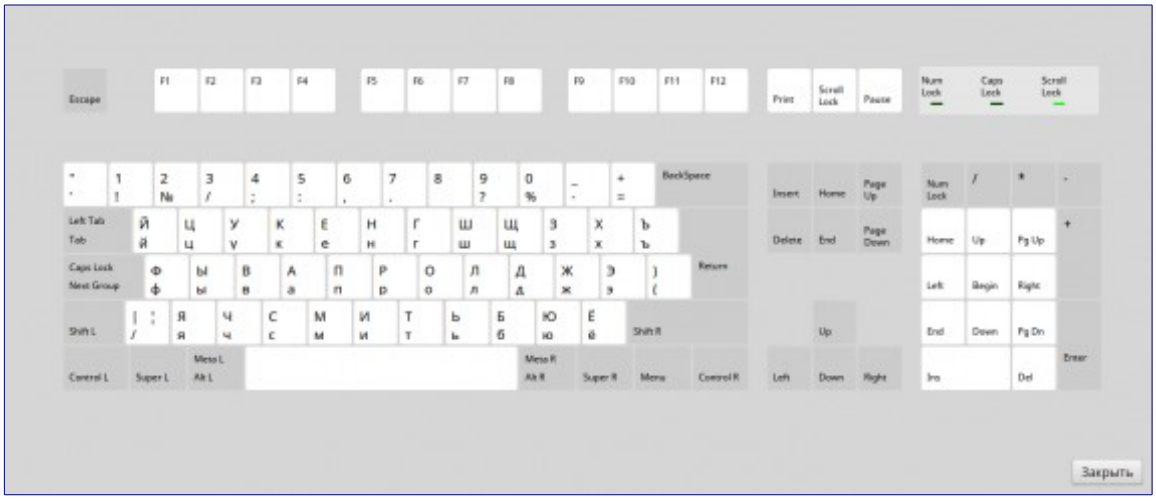
Перво-наперво следует определиться, использовать ли одинаковую раскладку во всех окнах, или в каждом — свою, что делается отметкой одной из соответствующих радиокнопок (см. предыдущий скриншот). А во втором случае — решить, будет ли раскладка нового окна наследоваться от таковой окна предыдущего, или же от умолчальной раскладки, той, что стоит в списке первой (на самом деле это тоже раскладка окна — так называемого корневого окна Иксов).

До некоторого времени наследование раскладки в Cinnamonm работало произвольным образом. В версии 2.4 это изжито, и при включении последней опции раскладка в новом окне действительно будет умолчальной. За единственным, отмеченным ранее, исключением: в строке инкременетного поиска главного меню среды раскладка почему-то всегда наследуется от текущей.

Далее переходим к выбору варианта русской раскладки. Список их включает все обычные варианты, немало экзотических и несколько национальных:



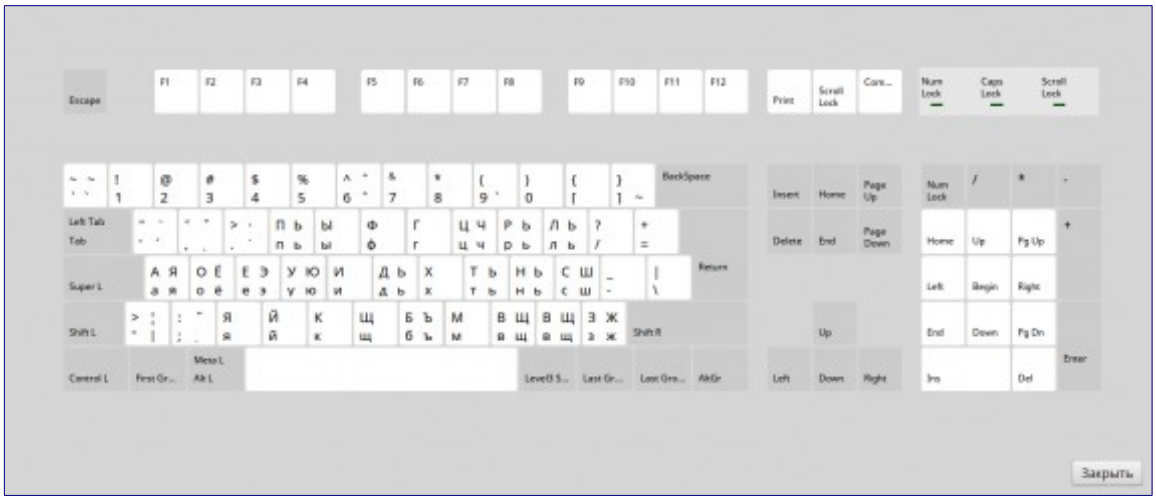
В случае сомнений раскладку можно, с помощью кнопки Предпросмотр, поглядеть на экране:



Я, ввиду большого стажа ремингтониста (это — мужской вариант професси машинистки), предпочитаю вариант Typewriter Legacy, обзываемый по русски печатная машинка, устаревшая. Вдаваться в обоснования этого выбора и тем более агитировать за него здесь неуместно. Заинтересованных отсылаю к специальному материалу.

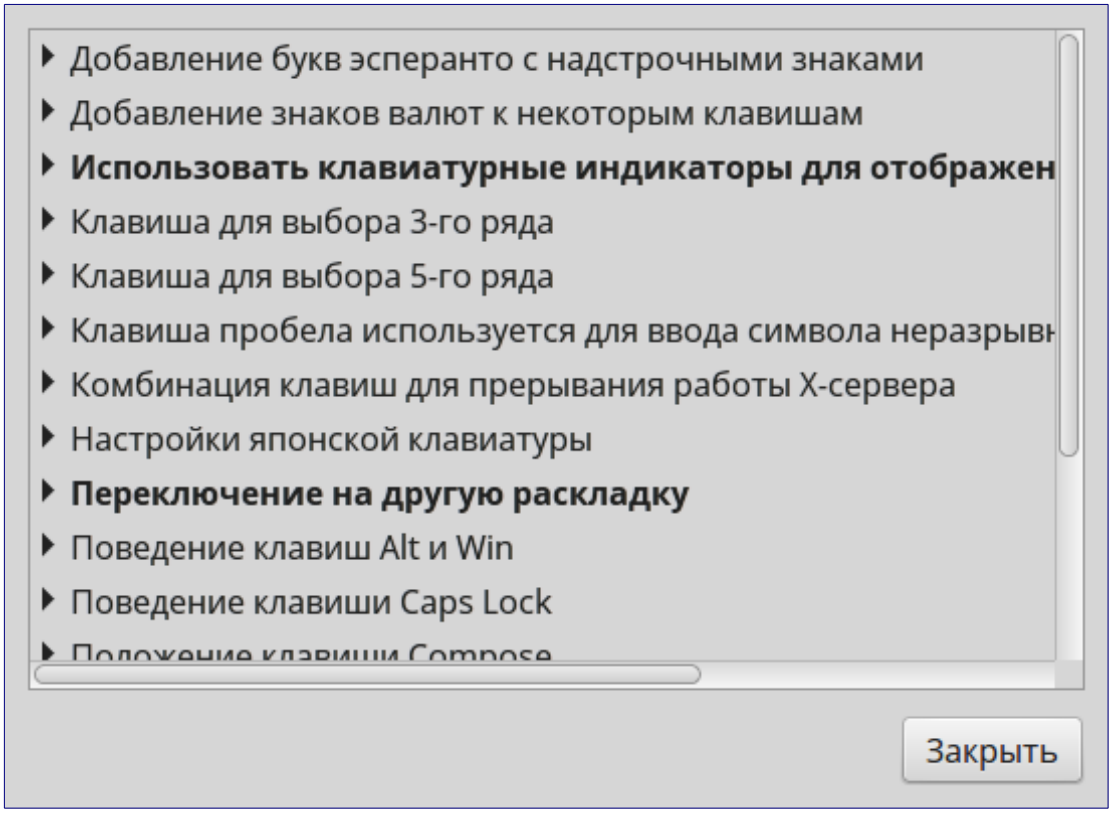
Из прочих вариантов интересен вот этот: Русский (Польша, фонетический

Дворак). Чего там осталось от раскладки профессора Дворака — разве что то, что это и близко не QWERTY. Но расположение алфавитно-цифровых символов и знаков препинания на ней выглядит вполне разумным:

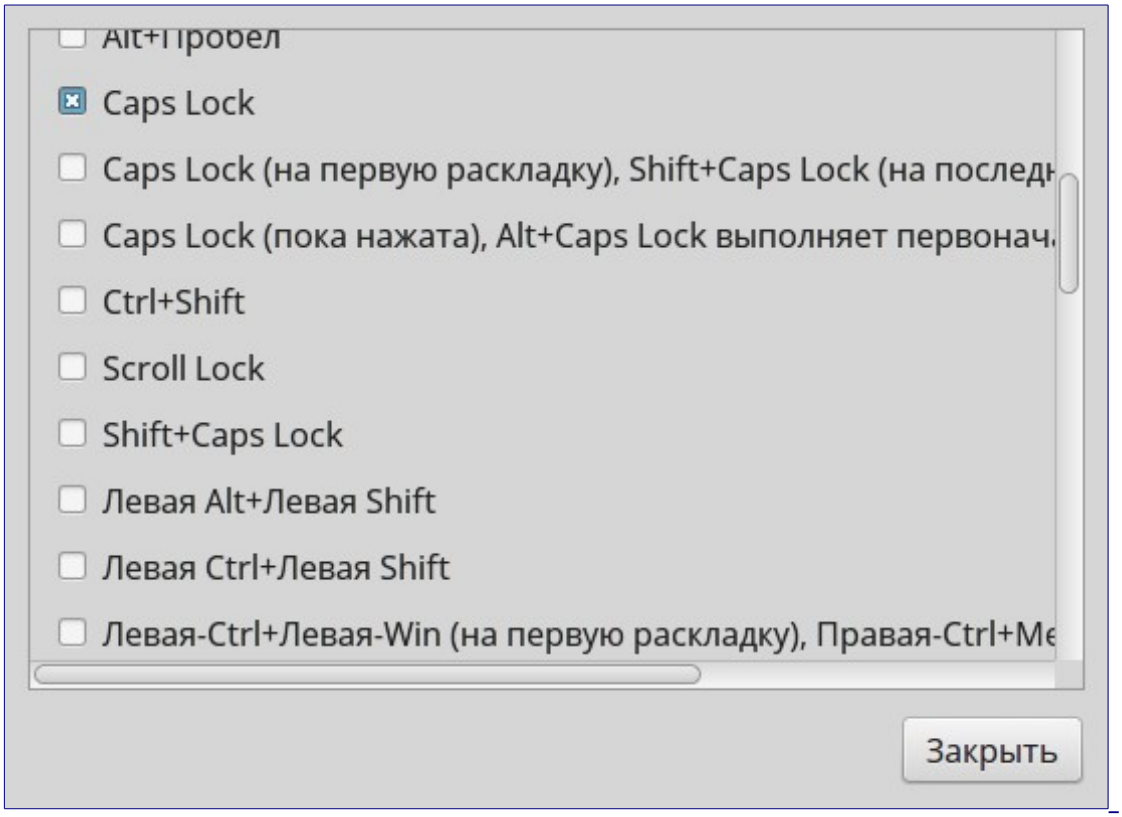


Правда, есть большие сомнения в том, что эта разумность стоит полного переучивания.

Нет напряга и с переключателями раскладок — они входят в число параметров, вызываемых, как ни странно, кнопкой Параметры:



Легко догадаться, что переключение настраивается в пункте Переключение на другую раскладку, и в нём можно найти переключатель на любой вкус:



Имеются и немодальные переключатели — это своего рода заменители корректоров ввода при «неправильной» раскладке, примером которых в Linux'е выступает программа X Neural Switcher (точнее, в Иксах, ибо работает также и в любых BSD-системах). Ибо из применителей может сказать, положа руку на сердце и поклявшись на своём Священном Писании, что он никогда, никогда, никогда... не забывал переключать раскладку клавиатуры с латиницы на кириллицу (или наоборот)?

Достоинства и недостатки XNeur обсуждались многократно, поэтому повторю только главный из последних: эта утилита принадлежит к тем, которые полагают себя умнее своих создателей (и, тем более, применителей), и потому часто автоматика его срабатывает прозвольным образом. Конечно, он имеет и «ручной» режим работы, но его использрвание очень узко: если вовремя заметить, что в «не той» раскладке набрано одно слово или его кусок, обычно проще и быстрей тут же перенабрать его, нежели выделять кусок текста и заказывать для него перекодирование.

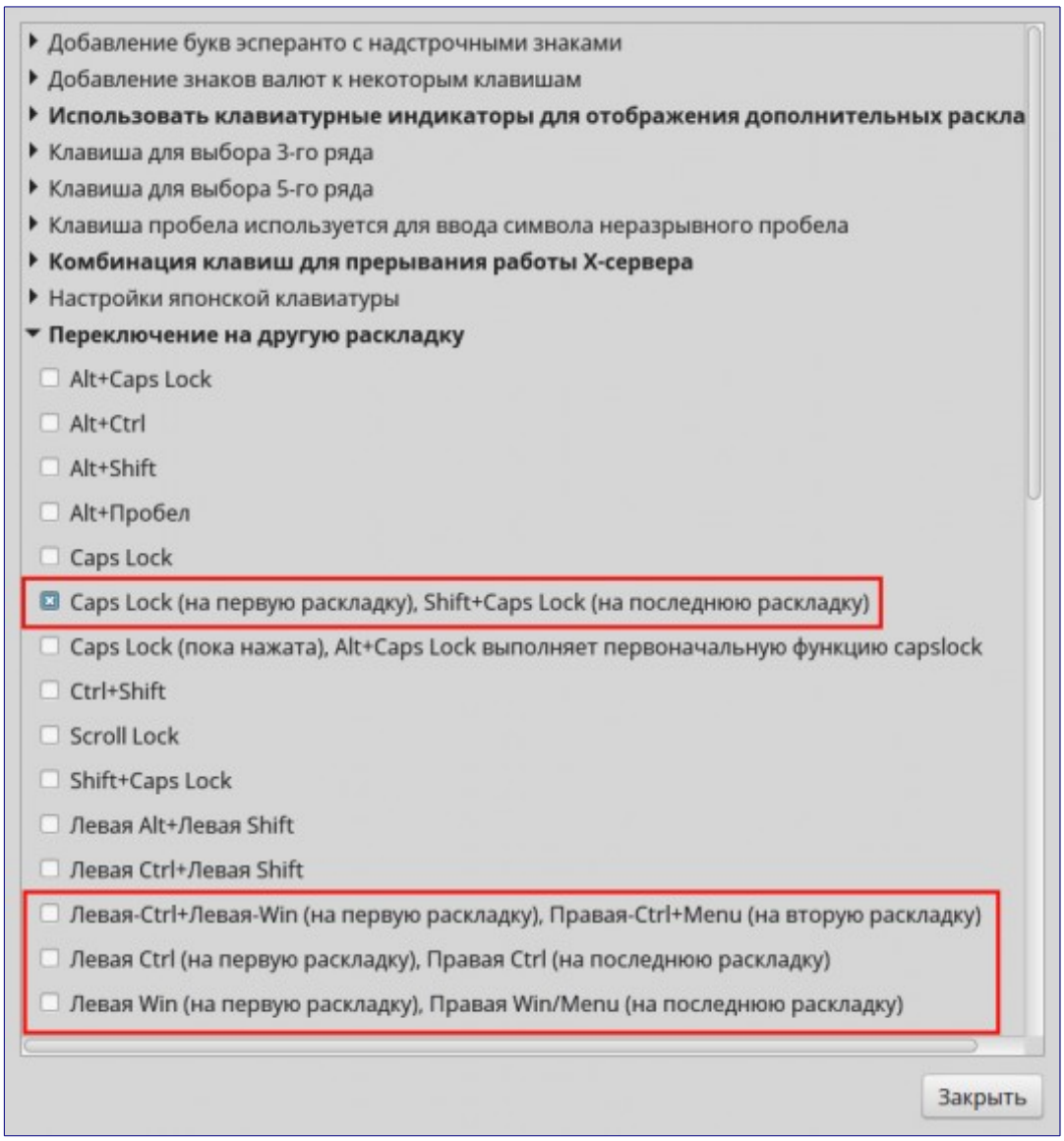
Тем более, что «проблему забывчивости» применительно к переключению раскладок можно попытаться решить другим способом — я бы назвал его «кембриджским». Как известно, в Оксфордском университете, воспитывавшем английских джентльменов, учили мыть руки после туалета. А в более прагматичном Кембридже, давшем миру немало естествоиспытателей, учили не справлять малую нужду на руки. Первому алгоритму следует программа XNeur, второй же можно реализовать, сведя к минимуму вероятность забывчивости при наборе. Чему очень поспособствуют те самые немодальные (или нециклические) переключатели раскладок.

Суть немодальных переключателей в том, что они ничего не переключают, а включают. То есть одна определённая клавиша (или их комбинация) всегда включает английскую раскладку, а другая делает то же самое для раскладки русской. И в использовании их есть только одна проблема — привыкание. То есть нужно отучиться смотреть на индикаторы раскладки. Нужно забыть о том:

- какая раскладка является текущей;
- какая раскладка является умолчальной;
- от кого наследуется раскладка нового окна — от корневого окна (то есть повторяет умолчальную) или от окна текущего.

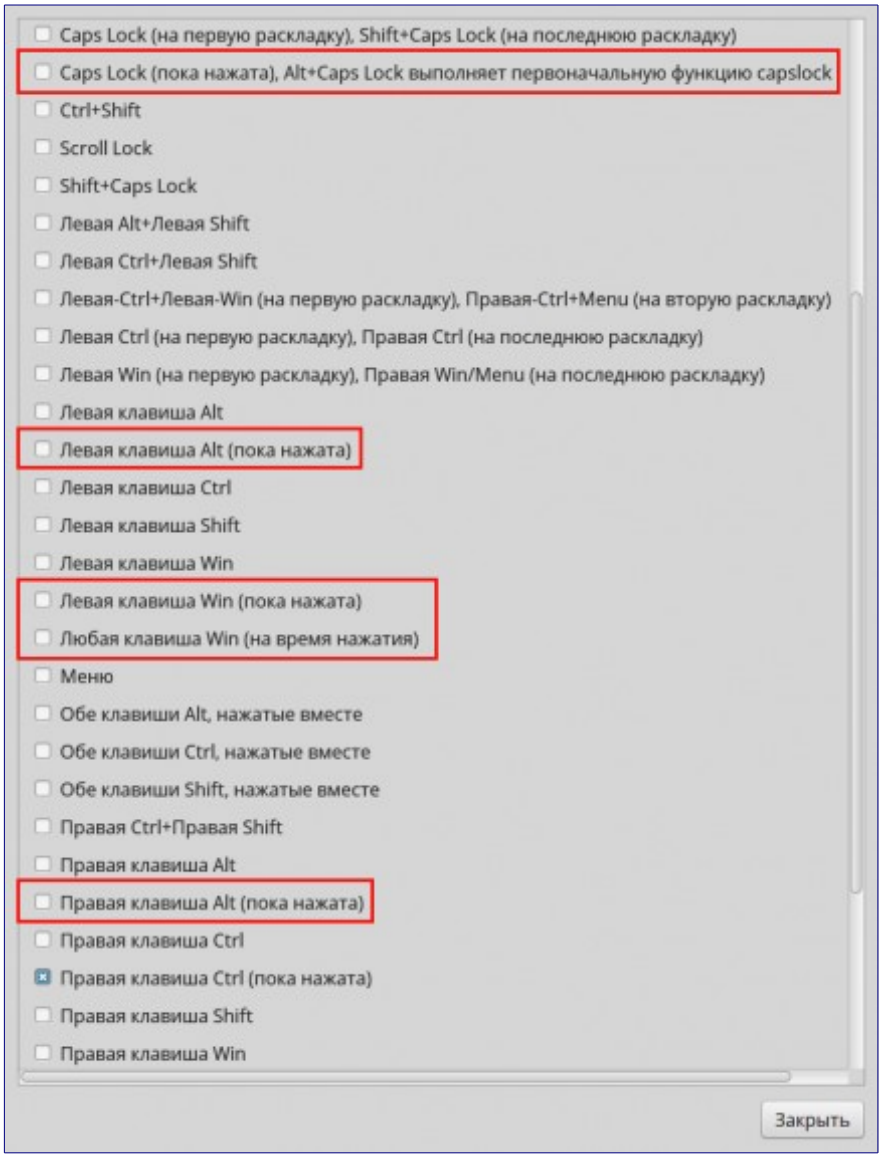
А помнить нужно только одно: перед вводом любого кириллического текста нажать, скажем, комбинацию Shift+CapsLock, а переходя к вводу латиницы — клавишу CapsLock. Подобно тому, как при вводе прописной буквы мы автоматически нажимаем Shift, не задумываясь особо о причинах этого.

Немодальных переключателей предлагается довольно много:



Далее, чтобы побороть забывчивость при переключении раскладок, надо переключать их как можно реже. И на сей предмет придуманы временные переключатели, действующие, пока нажата определённая клавиша — в частности, они совершенно незаменимы при вводе типографских символов с использованием клавиши Compose, о чём я скажу чуть позже. Причём они не исключают использования любых постоянных переключателей, как модальных, так и немодальных.

Традиционно в качестве временного переключателя используется правая клавиша Control, но и тут выбор достаточно велик:



А вообще все возможные переключатели раскладок, модальные, немодальные и временные, можно посмотреть в файле /usr/share/X11/xkb/rules/evdev.lst — в секции ! option, где они перечислены в строках, начинающихся с grp:

```
! option

 grp                Switching to another layout

 grp:switch         Right Alt (while pressed)

 grp:lswitch        Left Alt (while pressed)

 grp:lwin_switch    Left Win (while pressed)

 grp:rwin_switch    Right Win (while pressed)

 grp:win_switch     Any Win key (while pressed)

 grp:caps_switch    Caps Lock (while pressed), Alt+Caps Lock does the original capslock action

 grp:rctrl_switch   Right Ctrl (while pressed)
```

grp:toggle Right Alt

grp:lalt_toggle Left Alt

grp:caps_toggle Caps Lock

grp:shift_caps_toggle Shift+Caps Lock

grp:shift_caps_switch Caps Lock (to first layout), Shift+Caps Lock (to last layout)

grp:win_menu_switch Left Win (to first layout), Right Win/Menu (to last layout)

grp:lctrl_rctrl_switch Left Ctrl (to first layout), Right Ctrl (to last layout)

grp:alt_caps_toggle Alt+Caps Lock

grp:shifts_toggle Both Shift keys together

grp:alts_toggle Both Alt keys together

grp:ctrls_toggle Both Ctrl keys together

grp:ctrl_shift_toggle Ctrl+Shift

grp:lctrl_lshift_toggle Left Ctrl+Left Shift

grp:rctrl_rshift_toggle Right Ctrl+Right Shift

grp:ctrl_alt_toggle Alt+Ctrl

grp:alt_shift_toggle Alt+Shift

grp:lalt_lshift_toggle Left Alt+Left Shift

grp:alt_space_toggle Alt+Space

grp:menu_toggle Menu

grp:lwin_toggle Left Win

grp:rwin_toggle Right Win

grp:lshift_toggle Left Shift

grp:rshift_toggle Right Shift

grp:lctrl_toggle Left Ctrl

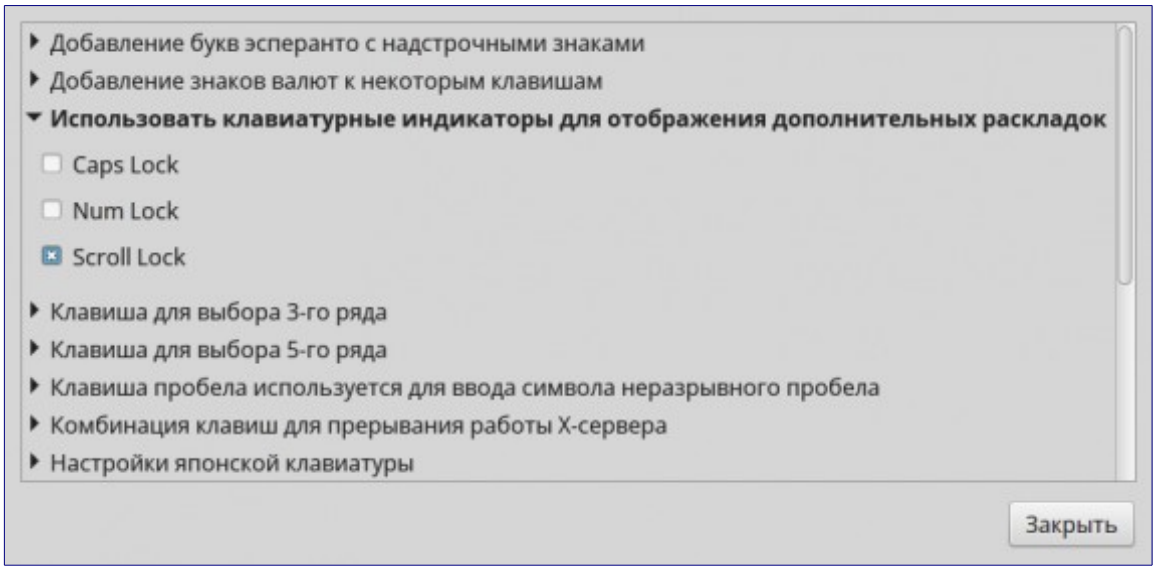
grp:rctrl_toggle Right Ctrl

grp:sclk_toggle Scroll Lock

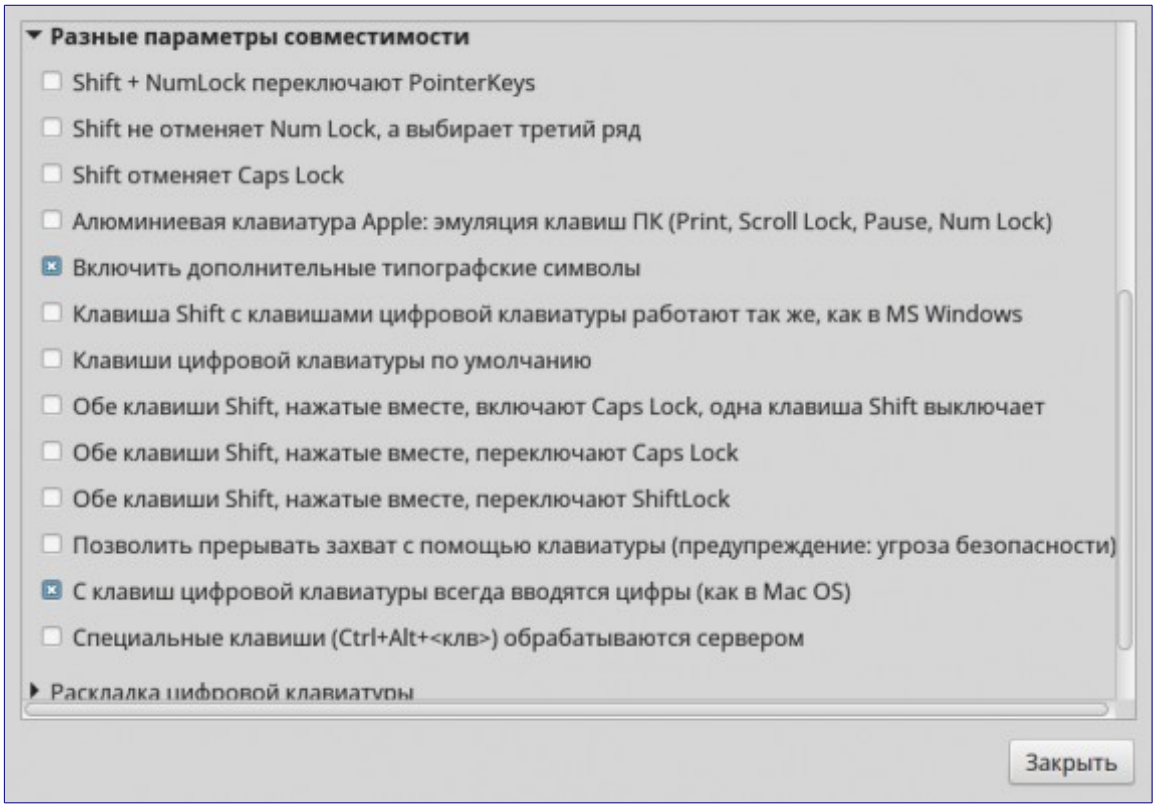
grp:lctrl_lwin_rctrl_menu LeftCtrl+LeftWin (to first layout), RightCtrl+Menu (to second layout)

Руководствуясь этим списком, переключатели раскладок (в числе прочих параметров) можно изменить и через Редактор dconf, о котором со временем пойдёт речь.

Кроме того, настройкой переключателей параметры клавиатуры не исчерпываются. Например, на десктопах резонно Использовать клавиатурные индикаторы для отображения дополнительных раскладок:



В ряде случаев полезно в Разных параметрах совместимости установить опцию С клавиш цифровой клавиатуры всегда вводятся цифры — вне зависимости от настроек BIOS и общесистемных:

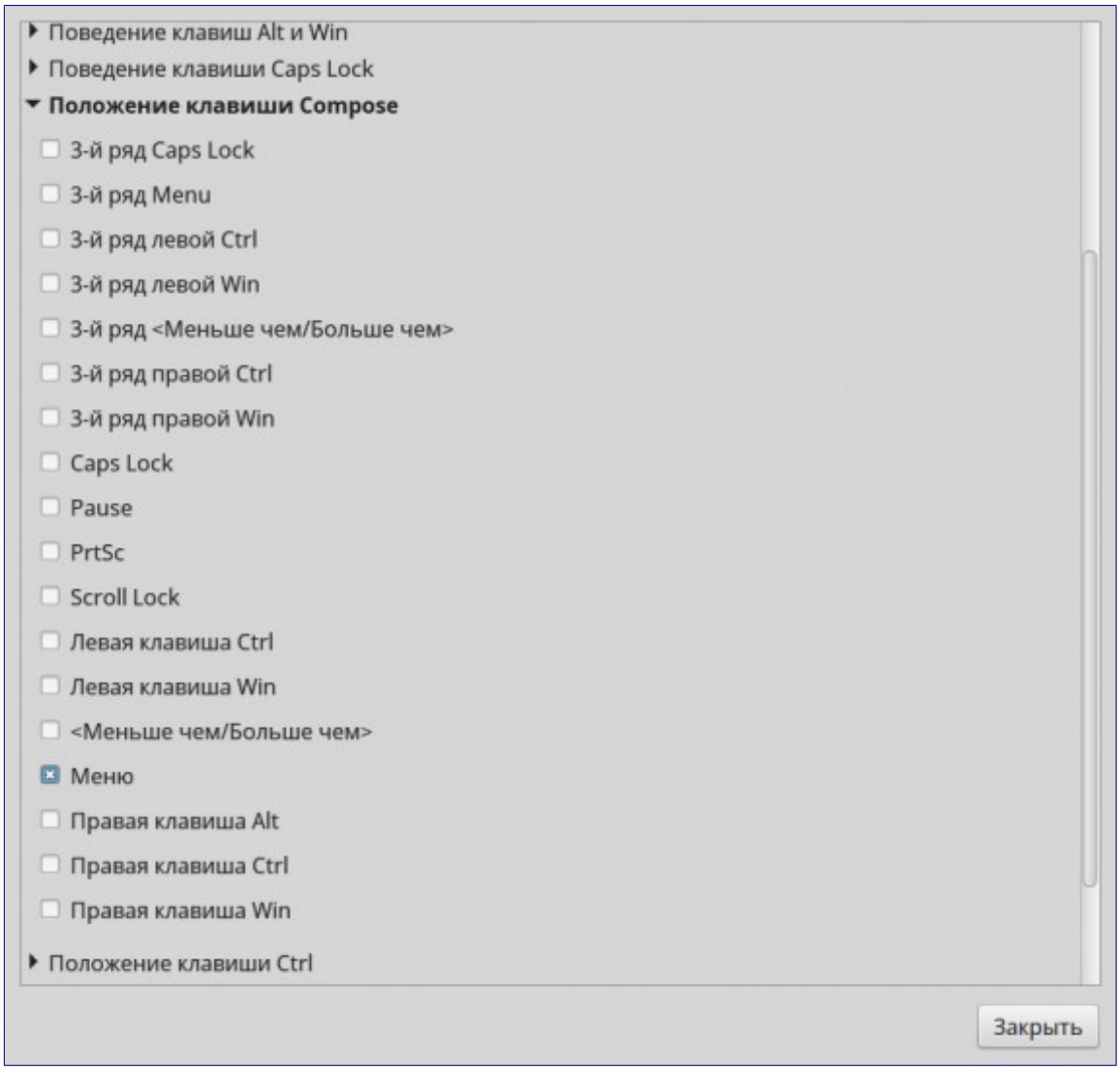


И, наконец, сакраментальный вопрос о вводе типографских символов. Хотя уже давно, как заметил Bregо,

...данная задача решается не просто и даже не очень просто, а примитивно.

А именно — так. Для начала поддержку ввода типографских символов следует включить — это делается в тех же Разных параметрах совместимости (см. предыдущий скриншот).

Далее нужно определить Положение клавиши Compose — она служит для переключения в режим ввода единичного типографского символа. Список вариантов тут более чем обширен:



И последнее, но самое главное: затвердить, как символ веры (своей, разумеется) список кейбиндов для типографских символов. Если и не полный, который можно найти в файле /usr/share/X11/locale/en_US.UTF-8/Compose, то хотя бы суперминималистический, образуемый с помощью Compose плюс:

--. — en dash

--- — em dash

spb spb nbsp неразрывный пробел

<< « открывающая кавычка >> » закрывающая кавычка

.. … многоточие

oo ° градус

+- ± плюс-минус

oc © копирайт

^2 ² в квадрате

12 ½ одна вторая

Набор любой степени — Compose ^ #

где # — нужная степень

Набор (почти) любой дроби — Compose x y

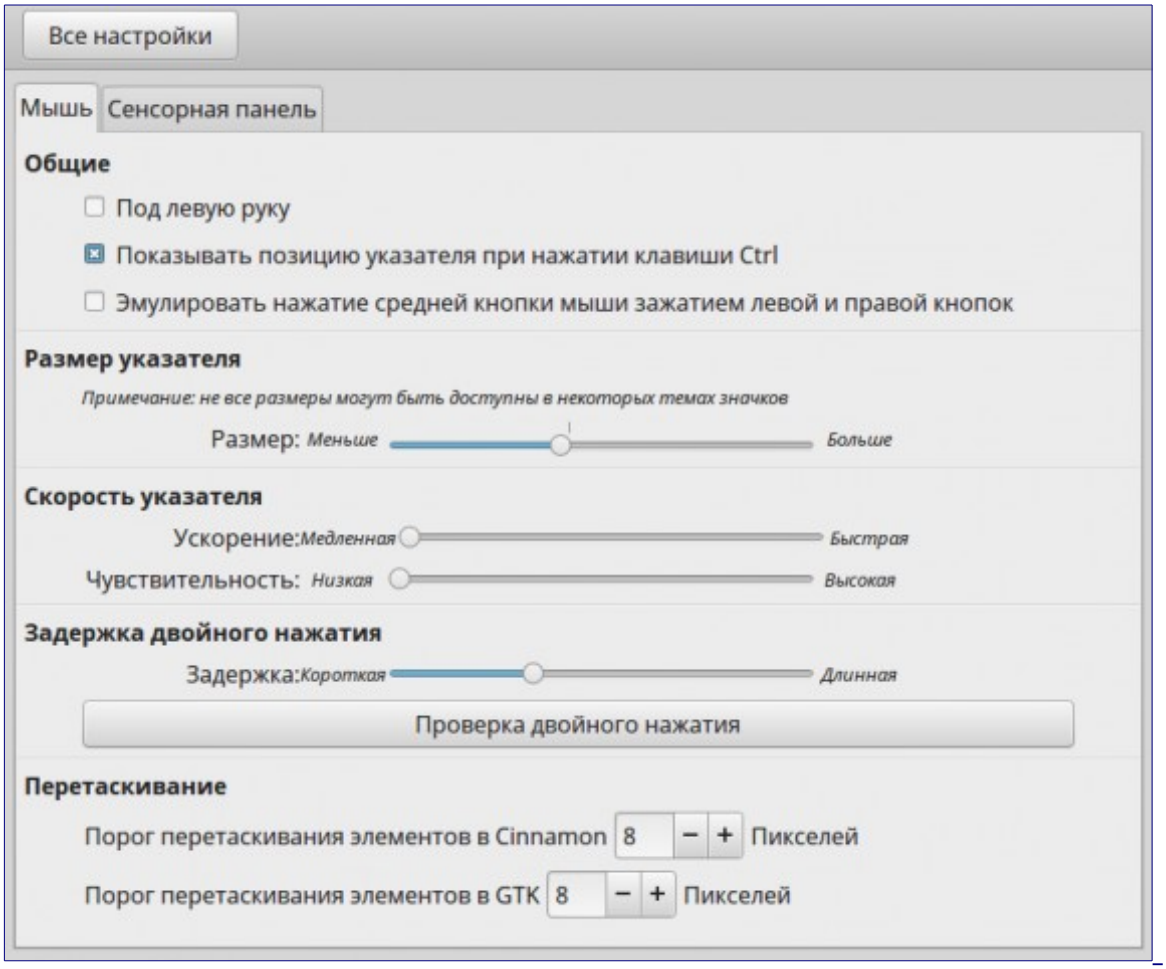
где x — числитель, y — знаменатель

Правда, правила для набора дробей имеют исключения. Так, в моей системе не выводятся все дроби, содержащие цифру 7, что в числителе, что в знаменателе. За единственным исключением — дробь ⅞ почему-то пропечатывалась. Не печатаются также и дроби с цифрой 9, в том числе и бессмысленные, тип 3/9 и 9/3. Причём девятка не проходит ни в каких сочетаниях, и ни в какой позиции, без всяких, в отличие от семёрки, исключений.

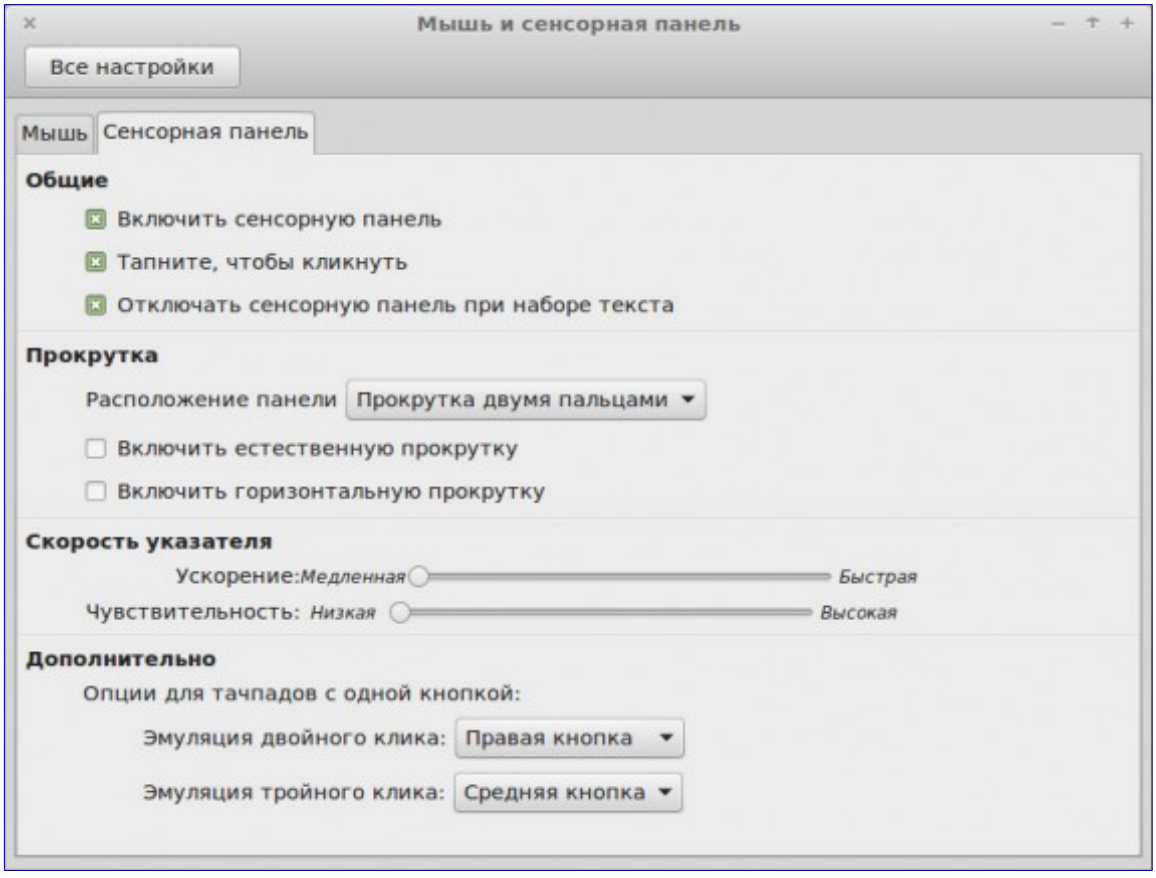
Да, ещё самое распоследнее: следует помнить, что ввод типографики после Compose работает только при латинской раскладке клавиатуры. И вот тут-то самое время вспомнить о временных, или «удержальных», переключателях — при некотором навыке они здорово упрощают ввод типографики.

Мышь и сенсорная панель

О настройках мыши мало что можно сказать. Разве что рассмотреть скриншот и решить, что следует здесь поменять — и следует ли что-то менять вообще. Для меня оказалась полезной опция Показывать позицию указателя при нажатии клавиши Ctrl — способствует отысканию курсора, если он затерялся в океане безбрежного широкоформатного монитора:

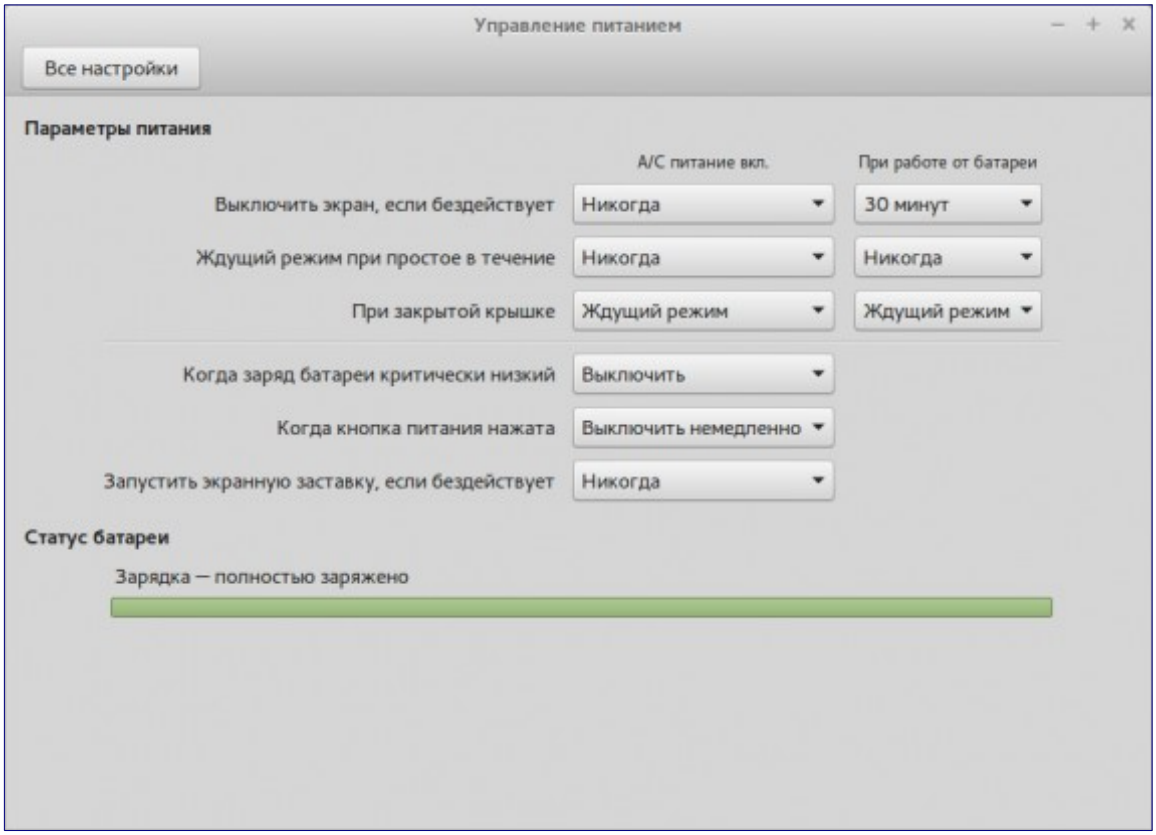


А вот настройка сенсорной панели — очень существенный момент при работе на ноутбуке. Хотя на своей Ноутбучке я ограничился переключением режима прокрутки — водить по ней двумя пальцами мне удобней, чем елозить одним по краю:



Но, как хвастаются разработчики (и как видно на скриншоте), в Cinnamon 2.4 появилась поддержка «однокнопочных» тачпадов — от Macbook'ов этой модой заразились и производители некоторых «обычных» ноутбуков. Действительно, опции, имеющие к этому отношение, присутствуют. Но что они делают и как работают — проверить не смог, ибо моя Ноутбучка имеет две натуральные, традиционно ориентированные, кнопки.

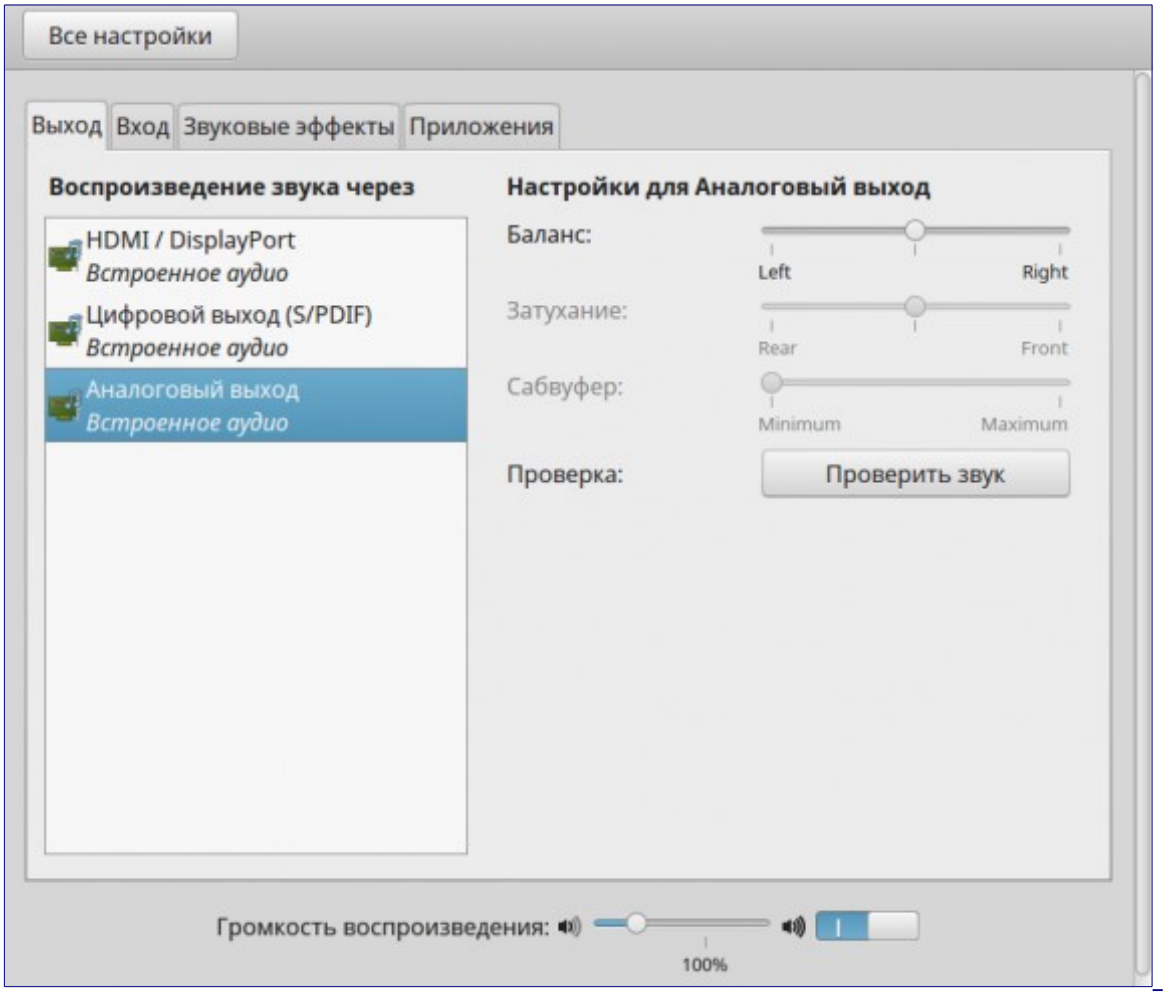
Раз уж речь зашла о ноутбуках, добавлю тут же пару слов про модуль Управление питанием. Я этим вопросом особенно не заморачиваюсь, ограничиваясь отключением всего того энергосбережения, которое можно отключить:



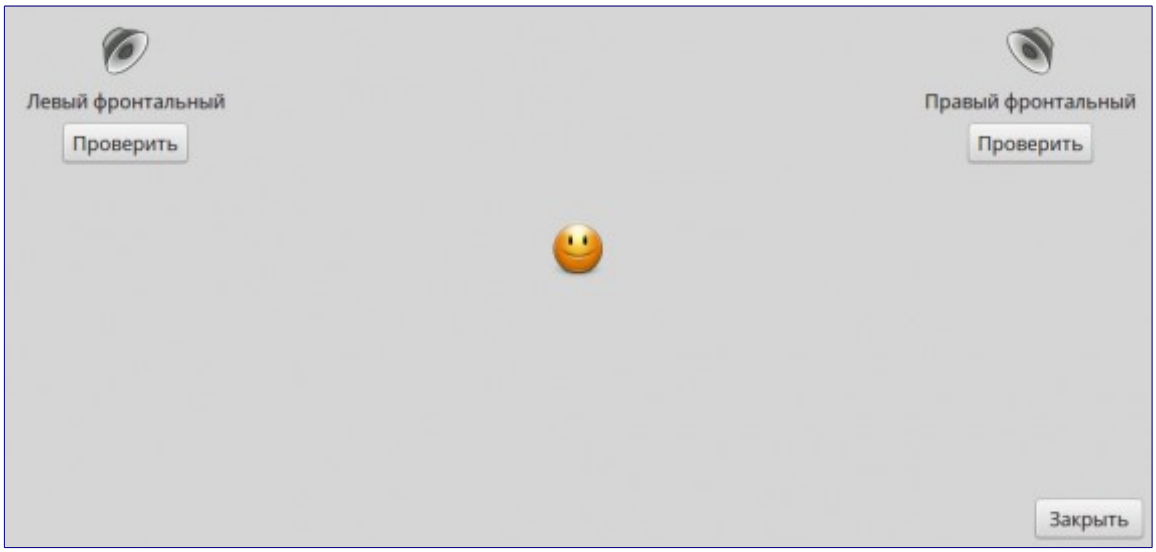
А с прочими опциями предлагаю разбираться заинтересованным лицам. Как и с модулем Bluetooth, поскольку соответствующие устройства я не использую.

Звук

С настройками звука дело теоретически обстоит так: запустив соответствующий модуль, можно для начала определить устройство воспроизведения оного. У меня таковым по умолчанию выставляется HDMI, по которому подключён монитор со встроенными колонками. Ни малейшего звука при этом не воспроизводится — он начинает звучать только при переключении на аналоговый выход (S/PDIF у меня нет):



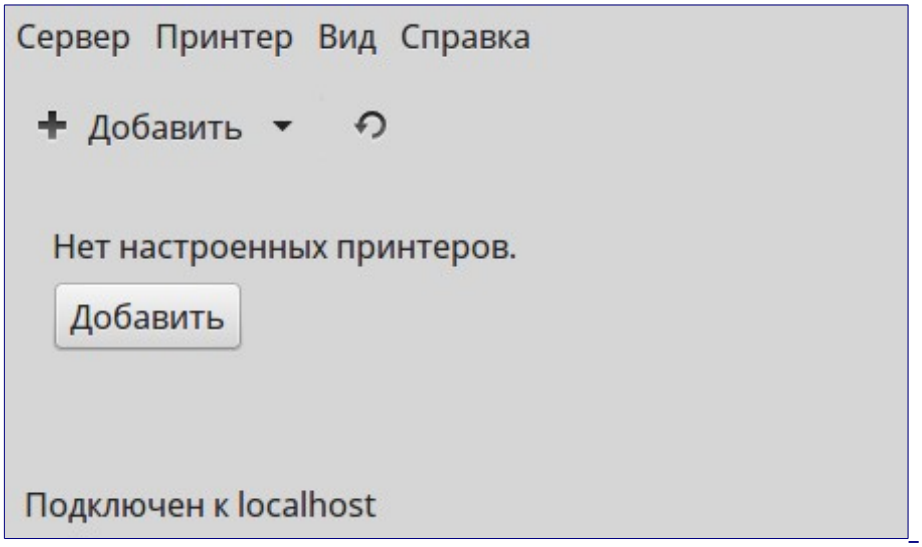
После чего остаётся проверить звучание — с неизменно превосходным результатом:



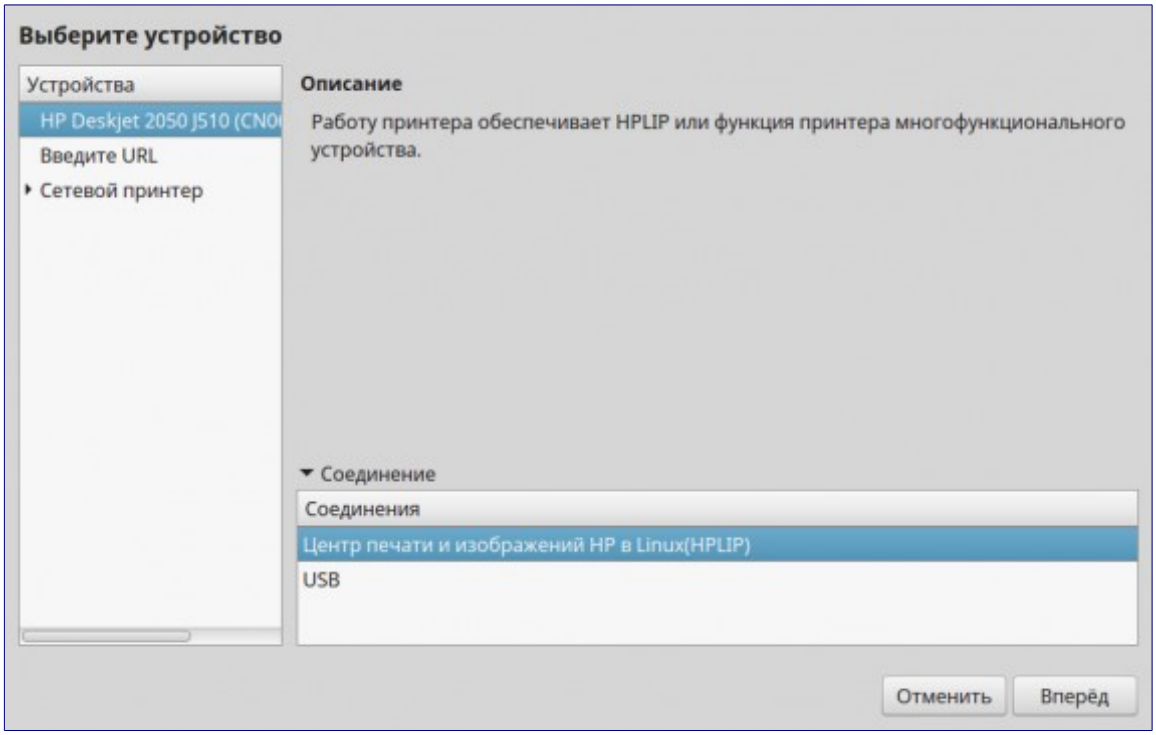
И слушать музыку или смотреть кино со звуковым сопровождением — штатными ли средствами Mint (Banshee и Totem, соответственно) или через более привычный мне Mplayer.

Принтеры и цвет

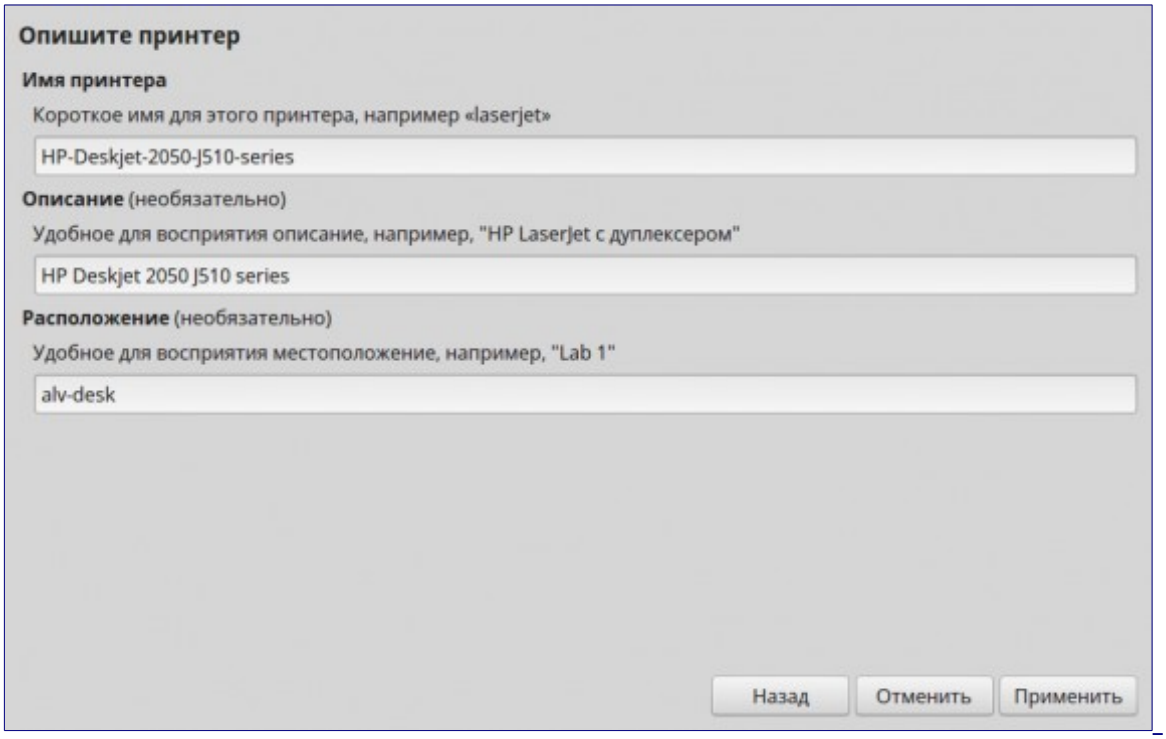
В секции Оборудование остались неохваченными вниманием два модуля — и описание обоих вполне уместится на одну страницу. Обратившись у пункту Принтеры, я увидел, что у меня нет настроенных принтеров:



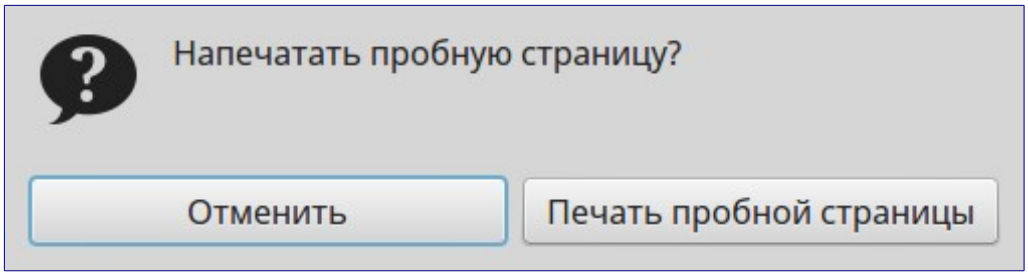
Поскольку физически у меня имелось МФУ DeskJet 2050, я нажал кнопку Добавить — и увидел, что такое действительно имеет место быть:



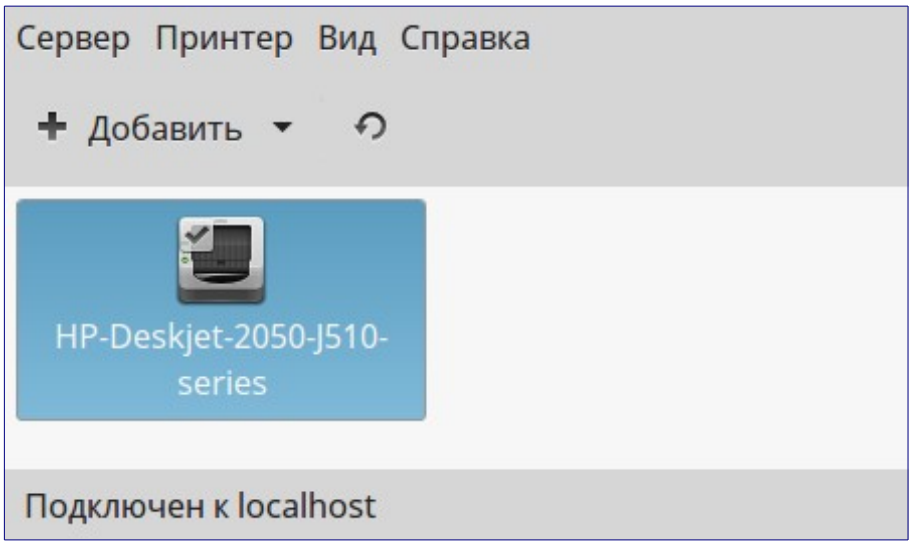
Оставалось его настроить — для чего была нажата кнопка Вперёд, после чего было выведено некое умолчальное описание принтера:



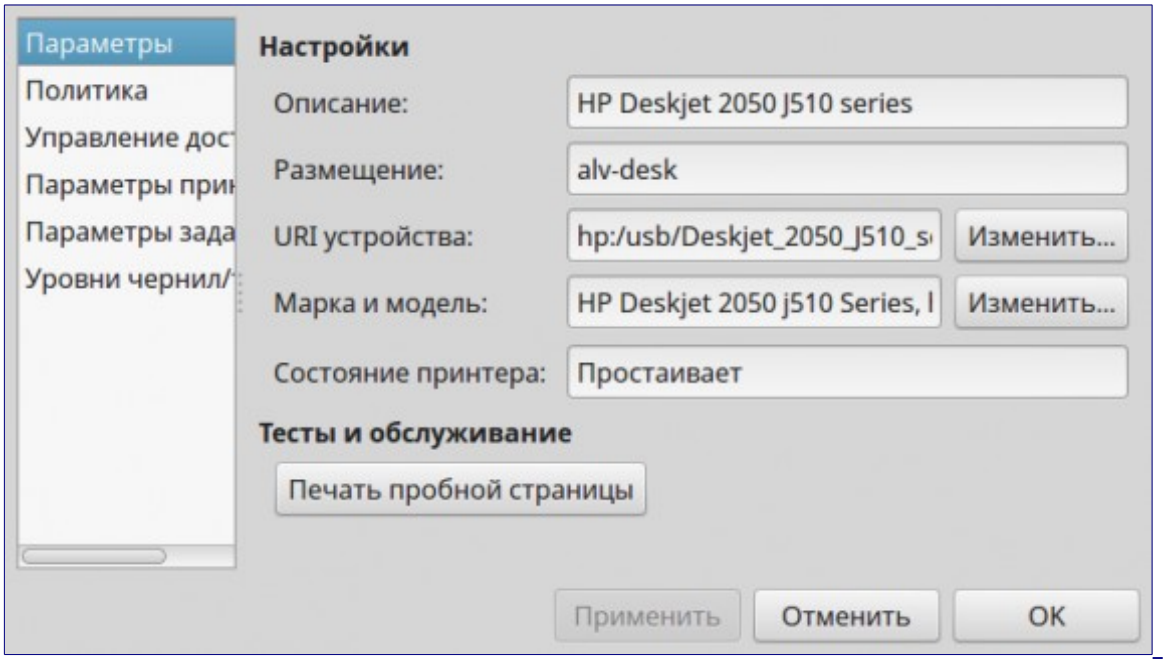
Его можно подкорректировать, но я этого делать не стал. А нажал кнопку Применить, через несколько секунд, ушедших на поиск драйверов, получил предложение напечатать пробную страницу:



Вслед за чем оказалось, что принтер волшебным образом у меня появился:



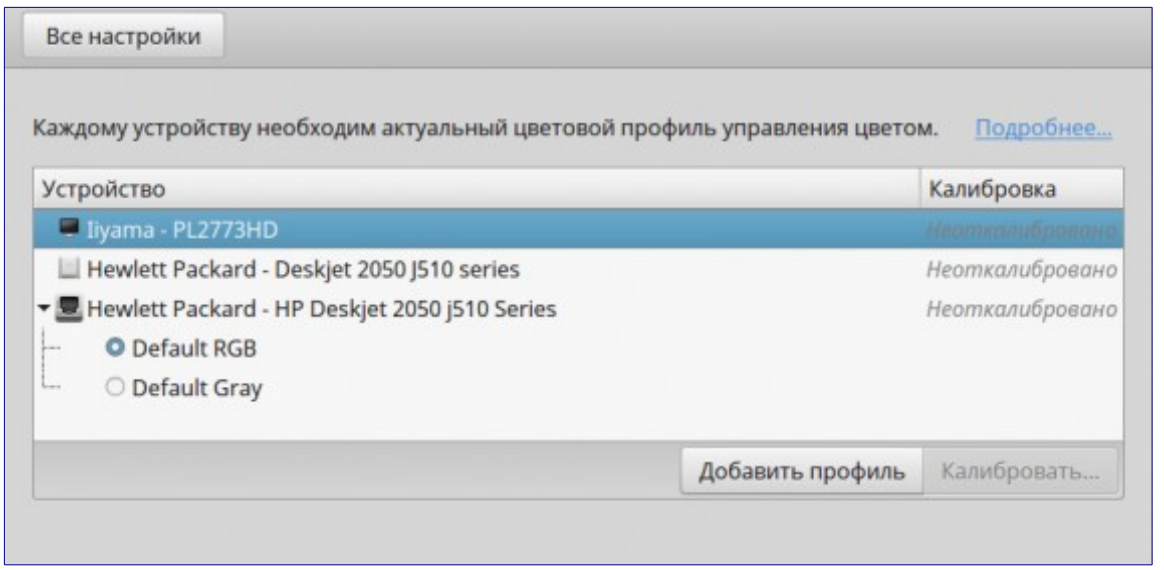
И свойства его оказались таковы:



Надо сказать, что всё это потребовалось только потому, что при инсталляции Mint 17.1 Rebecca с нуля я не включил принтер. Иначе всё было бы автоматически установлено посредством HPLIP (HP Linux Imaging and Printing). Именно так было у меня при установке Mint 17 Qiana — ни малейших усилий по настройке принтера я тогда не прикладывал. Но, когда он мне понадобился впервые (а принтер мне требуется максимум раз в квартал), обнаружил, что он есть и готов к работе.

Кстати, в обоих случаях моё МФУ, в соответствие со своим многофункциональным титулом, способно и сканировать — причём опять-таки без всяких настроечных действий. Что, впрочем, к нынешней теме не относится. Да и ставить это надо в заслугу не Cinnamon, и даже не Mint, а фирме HP и разработчикам системы HPLIP. Вот уже в который раз и на котором дистрибутиве я убеждаюсь, что HPLIP полностью избавил применителя от забот о настройке печати и сканирования. Правда, только счастливого обладателя продукции HP.

И, наконец, последний модуль секции носит имя Цвет, а суть его заключается в калибровке цветов монитора и принтера. Поскольку я в этом ничего не понимаю и ни малейшей потребности понимать не испытываю, ограничусь скриншотом:



А уж кому нужно или интересно — делайте с ним что хотите.

Секция Администрирование

Наступает волнующий момент — среди штатных настроек Cinnamon'а осталась одна секция --Администрирование, а в ней — четыре модуля:

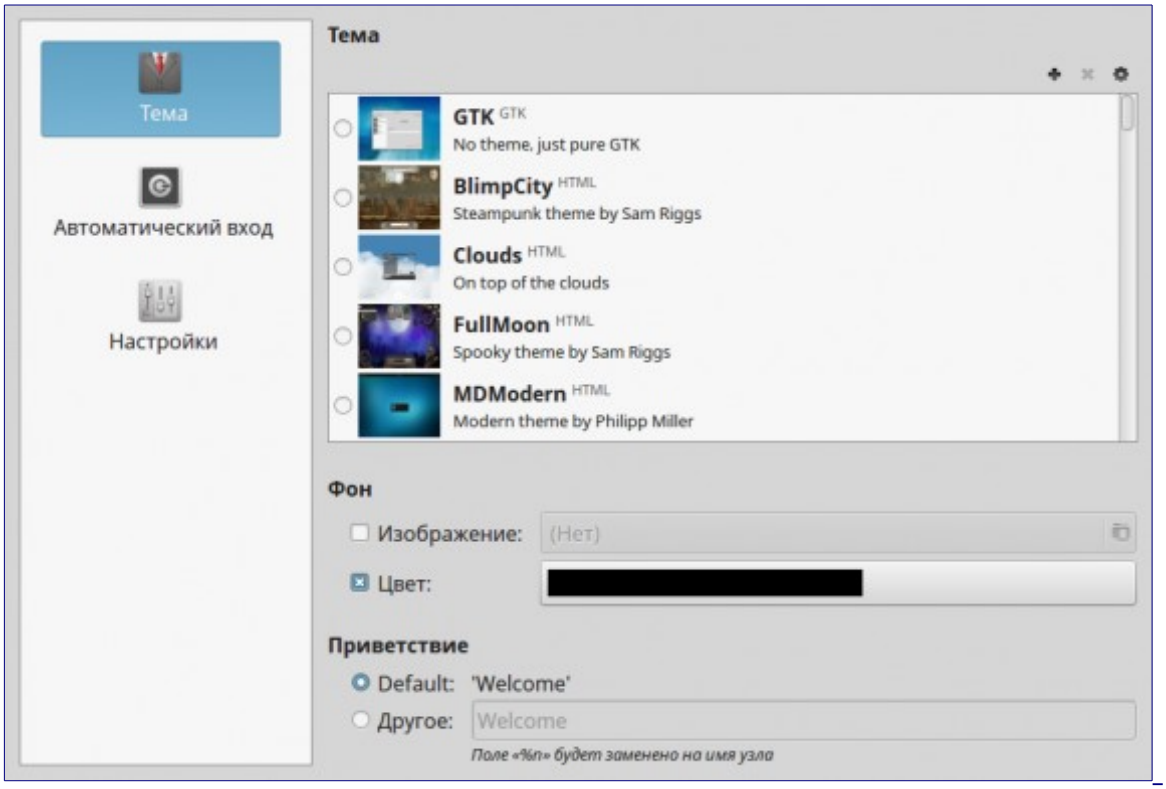
- Источники приложений;
- Менеджер драйверов;
- Окно входа в систему;
- Пользователи и группы.

Первые два модуля прямого отношения к Cinnamon не имеют, а принадлежат к кругу фирменных утилит Mint, которые будут предметом соответствующего очерка. Так что в очерке этом будут рассмотрены два последних.

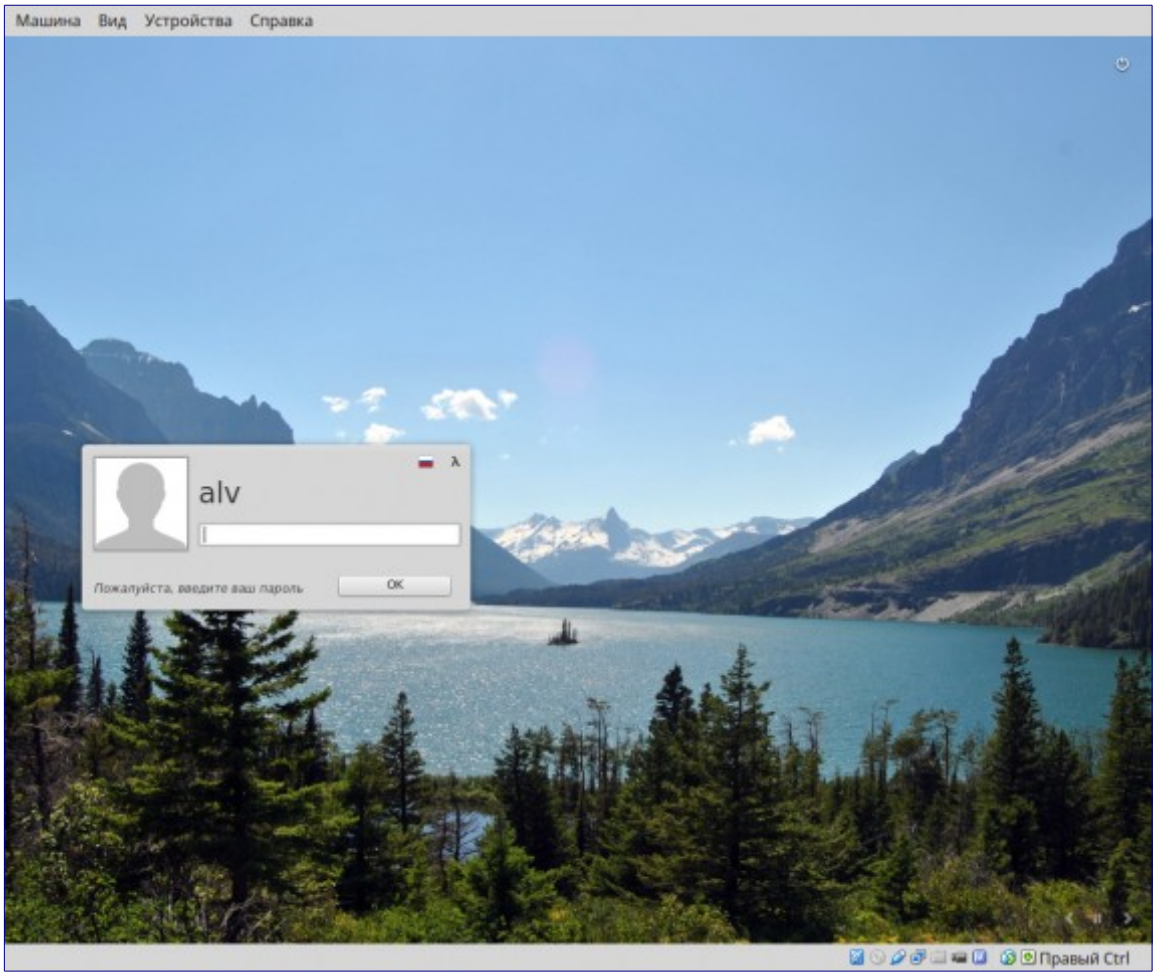
Окно входа в систему

Строго говоря этот модуль тоже не часть среды Cinnamon, а является инструментом настройки дисплейного менеджера MDM (изначально аббревиатура Mint Display Manager, ныне превратившаяся в рекурсивное MDM Display Manager), обеспечивающего во всех редакциях дистрибутива Mint авторизацию в системе. Однако он очень тесно интегрирован в Системные настройки, в том числе и в отношении внешнего вида, и потому его целесообразно рассмотреть здесь.

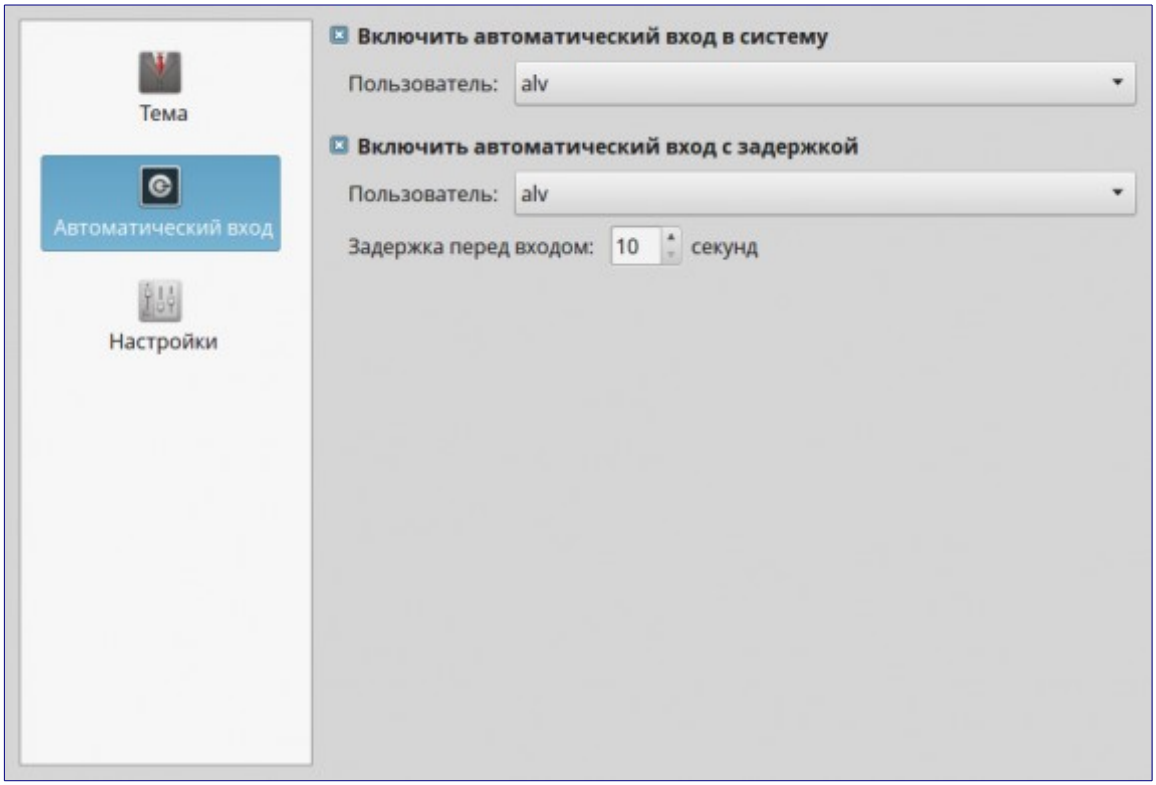
Поскольку авторизация в системе выходит за пределы компетенции отдельного пользователя, при запуске этого модуля (из CLI его можно запустить командой sudo mdmsetup) для начала запрашивается пароль, после ввода которого появляется такая панель:



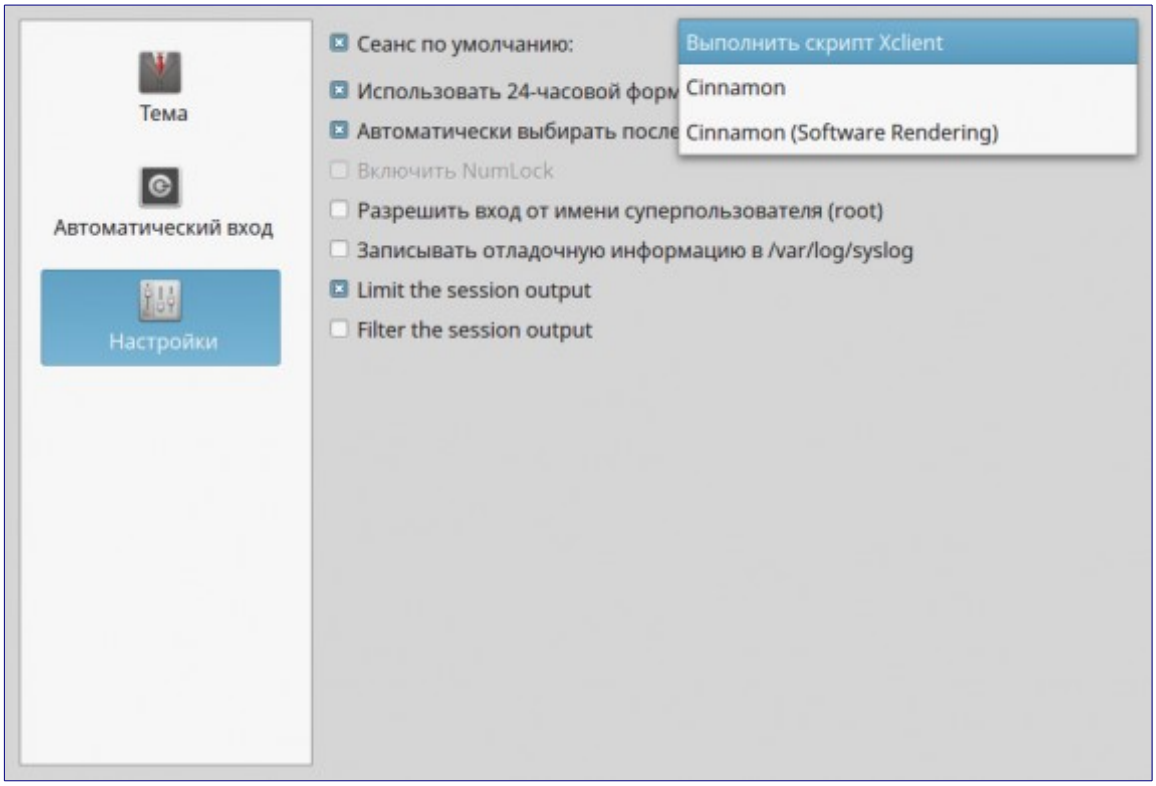
С пунктом Тема всё понятно — это выбор заставки, на фоне которой выводится окно авторизации, а также, при желании, определение собственного приветствия:



Далее можно задать автоматический вход в систему для определённого пользователя, и установить задержку перед входом, позволяющую выбрать сеанс:



Сеанс по умолчанию задаётся в пункте Настройки:

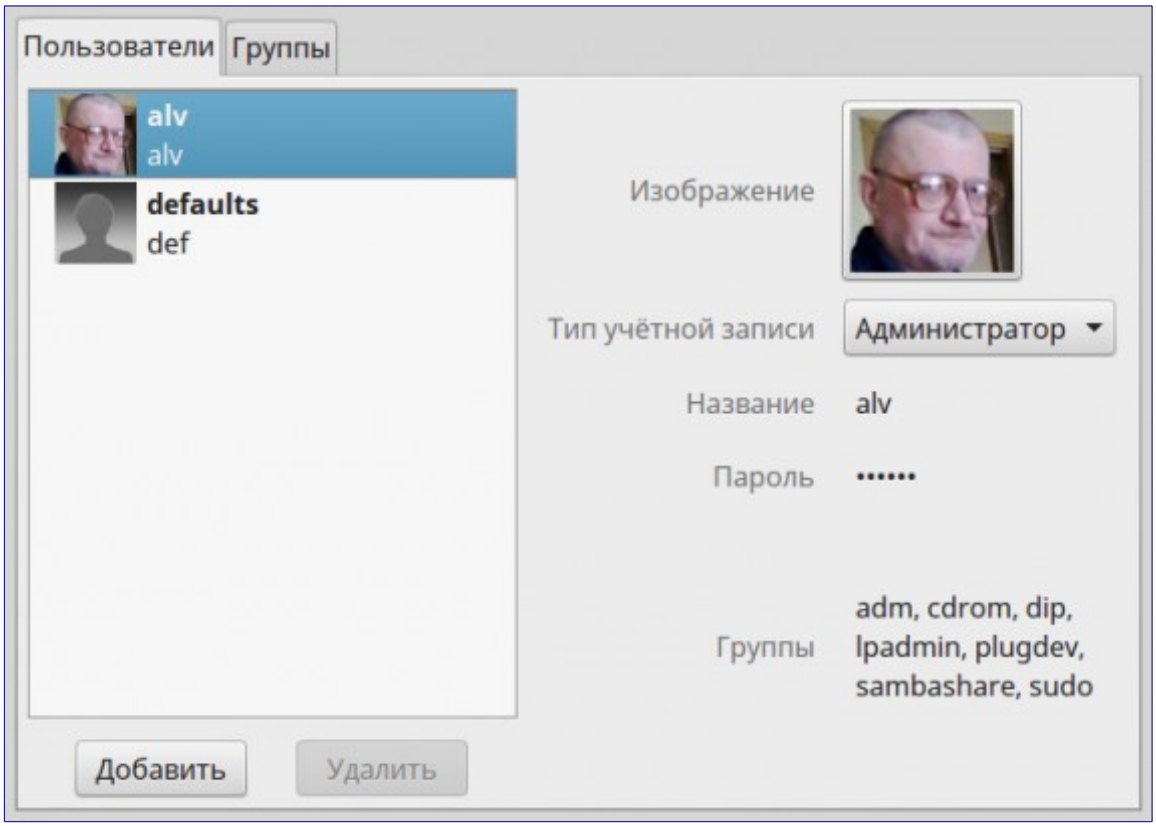


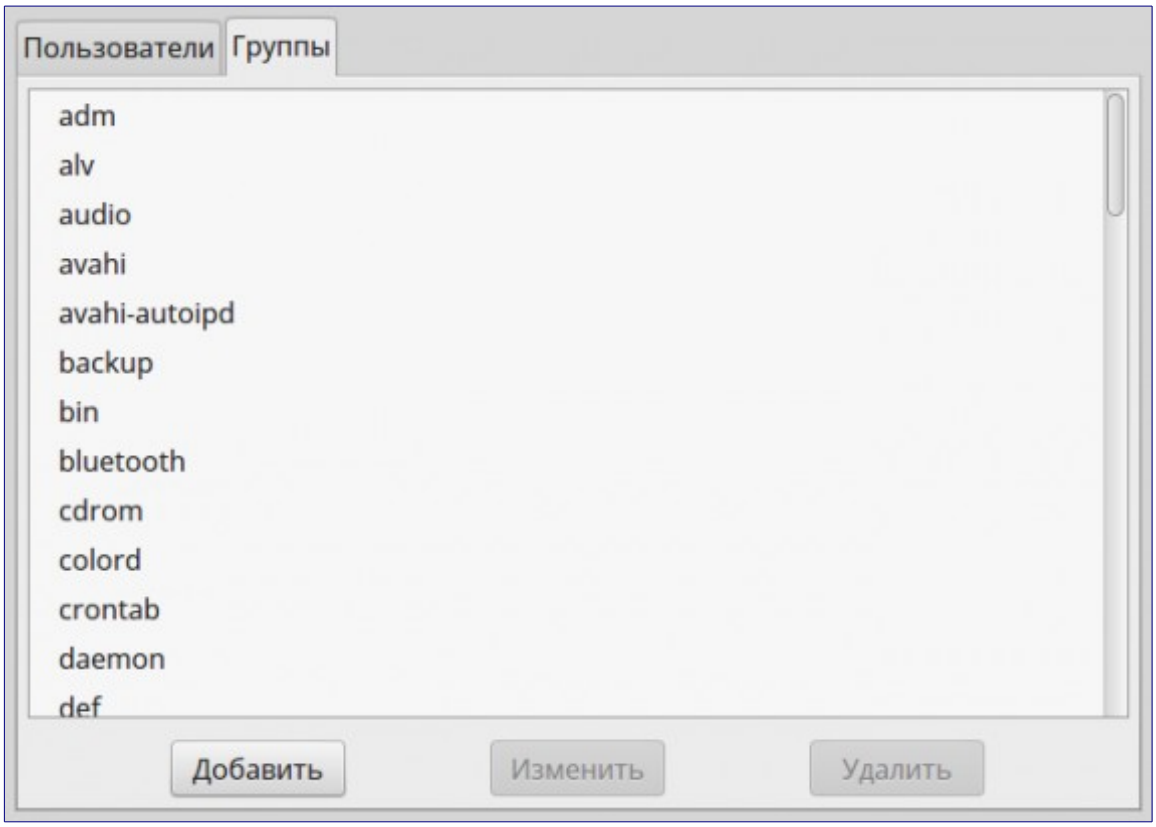
Здесь же есть вожделенная для начинающих линуксоидов опция — разрешение авторизоваться в иксовом сеансе в качестве суперпользователя. Что, конечно, круто, но делать не рекомендуется, за исключением единичных ну очень специальных случаев.

Пользователи и группы

А вот модуль Пользователи и группы — родной для Cinnamon, из CLI его можно запустить командой cinnamon-settings-users. Очевидно, что и здесь потребуется пароль, ввод которого даст доступ к святая святых — списку

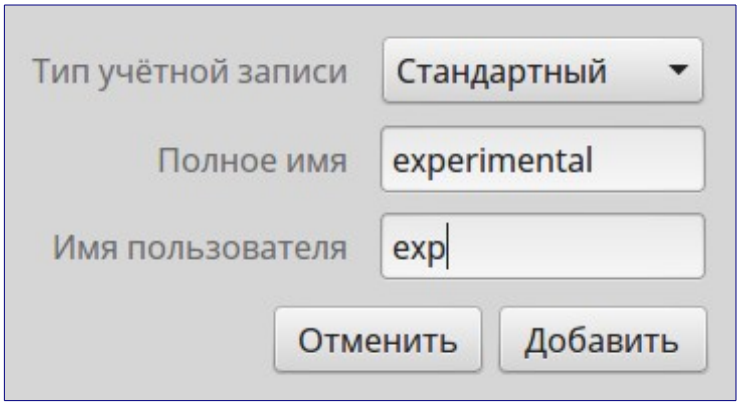
пользователей и групп:



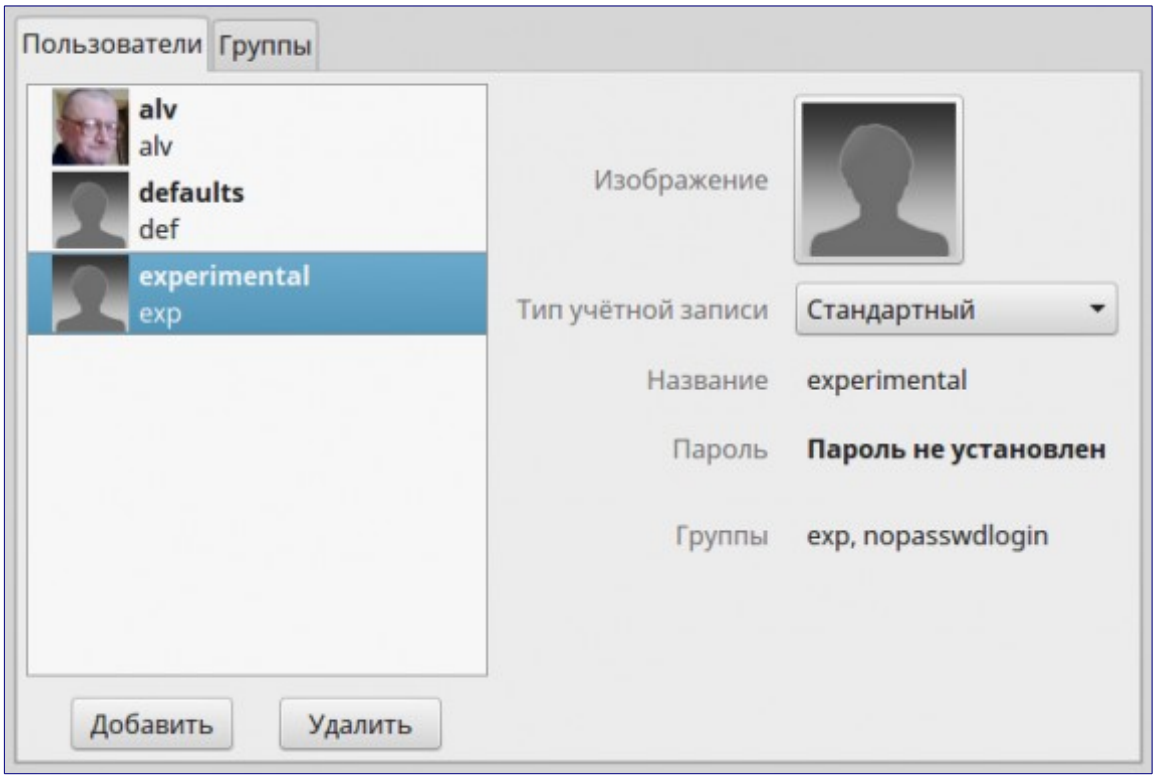


От описания возможных тут действий в каждом аккаунте позволю себе воздержаться — они почти очевидны из скриншота и становятся более чем очевидными после пары щелчков мышью. А вот о том, как создаётся новый пользовательский аккаунт, скажу чуть подробнее.

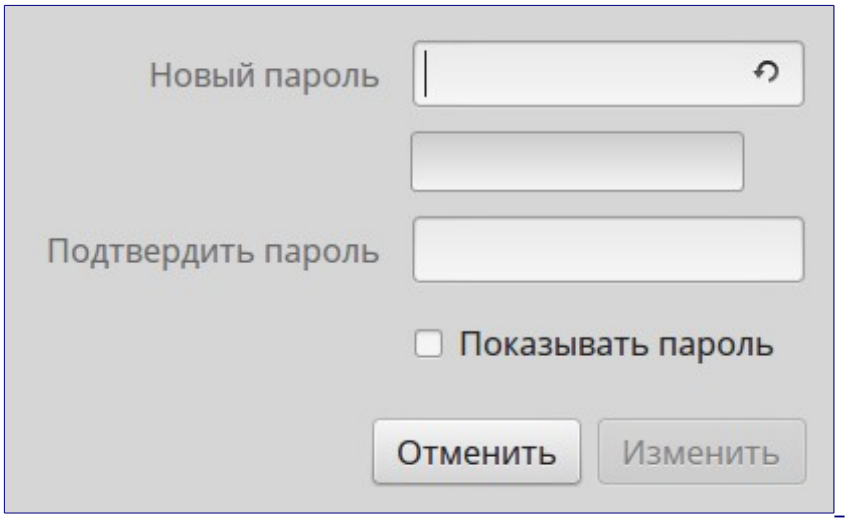
Собственно, для создания аккаунта достаточно заполнить простую форму, выбрав в ней тип учётной записи:



Тут возможно два варианта — Администратор и Стандартный. Второй выводится по умолчанию — и пусть таковым остаётся (почему — скажу чуть позже). Так что теперь достаточно нажать кнопку Добавить, чтобы новый пользователь появился в списке оных:



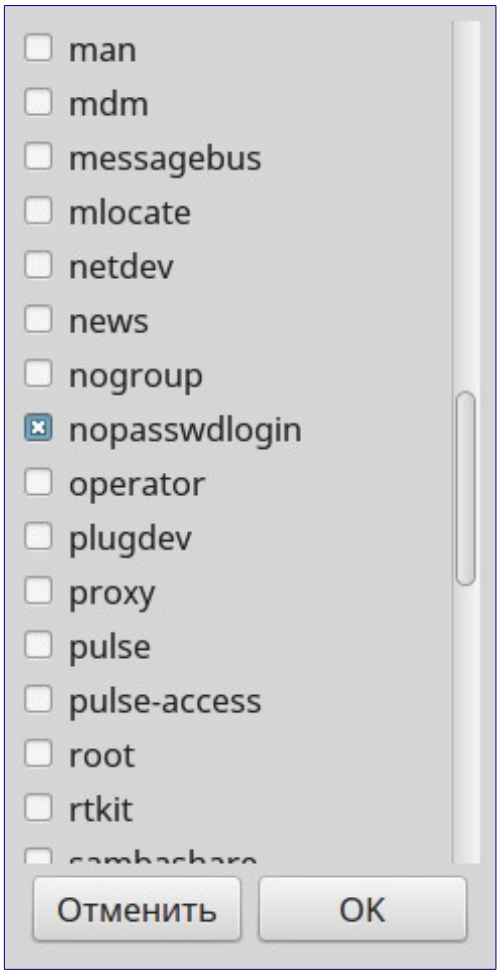
Обращаю внимание, что пароль в ходе создания нового аккаунта не запрашивается. Его можно установить здесь же, щёлкнув на поле Пароль:



Впрочем, я по ряду причин предпочитаю делать это из CLI командой

```
$ sudo passwd username
```

Только после этого надо не забыть исключить пользователя из группы nopasswdlogin, членство в которой даёт возможность беспарольного входа в систему, что не есть правильно. Для этого достаточно щёлкнуть мышью на поле Группы и снять отметку с соответствующего боксика:



Здесь, во-первых, надо подчеркнуть, что беспарольный вход в систему — это совсем не то же самое, что автоматический вход, о котором говорилось выше: во-втором случае пароль пользователя существует, просто его не нужно вводить в окошке MDM (это, в соответствие с названием, делается автоматически, за кадром). А вот при авторизации в консоли этому самому «автоматическому» пользователю пароль вводить придётся. Как и при запуске программ, требующих прав администратора. И, чтобы ни говорили записные параноики, с точки зрения безопасности на локальной машине, находящейся в индивидуальном пользовании, автоматический вход ничем не отличается от парольного.

Во-вторых, поясняю, в чём отличие административного типа учётной записи от стандартного: только пользователь с аккаунтом первого типа имеет возможность получить доступ к административным привилегиям с помощью команды sudo. Административный статус автоматически присваивается тому пользователю, чей аккаунт был создан при инсталляции системы. Так что для всех остальных аккаунтов достаточно статуса стандартного. Ибо зачем нам два генеральных секретаря? — резонно говорил незабвенный Леонид Ильич (правда, не в жизни, а в анекдоте)

И в-третьих, может возникнуть вопрос, зачем на индивидуальной машине несколько аккаунтов? Причины могут быть разные, скажу только за себя. Мой первый и главный аккаунт, alv, предназначен для меня, любимого, когда он занят выполнением своей непосредственной работы. Например, сочинением этой книжки.

Никаких потенциально опасных действий я под основным аккаунтом не делаю (или стараюсь не делать). Для всякого рода экспериментов предназначен аккаунт exp, пользователь которого не имеет доступа к правам администратора и потому напортачить на системном уровне не может — в его власти только развалить собственные пользовательские настройки. Поскольку такое рано или поздно случается — на помощь приходит аккаунт def, в котором сохраняются все настройки по умолчанию. И из которого настройки несчастного exp можно восстановить до исходного состояния простым копированием конфигов.

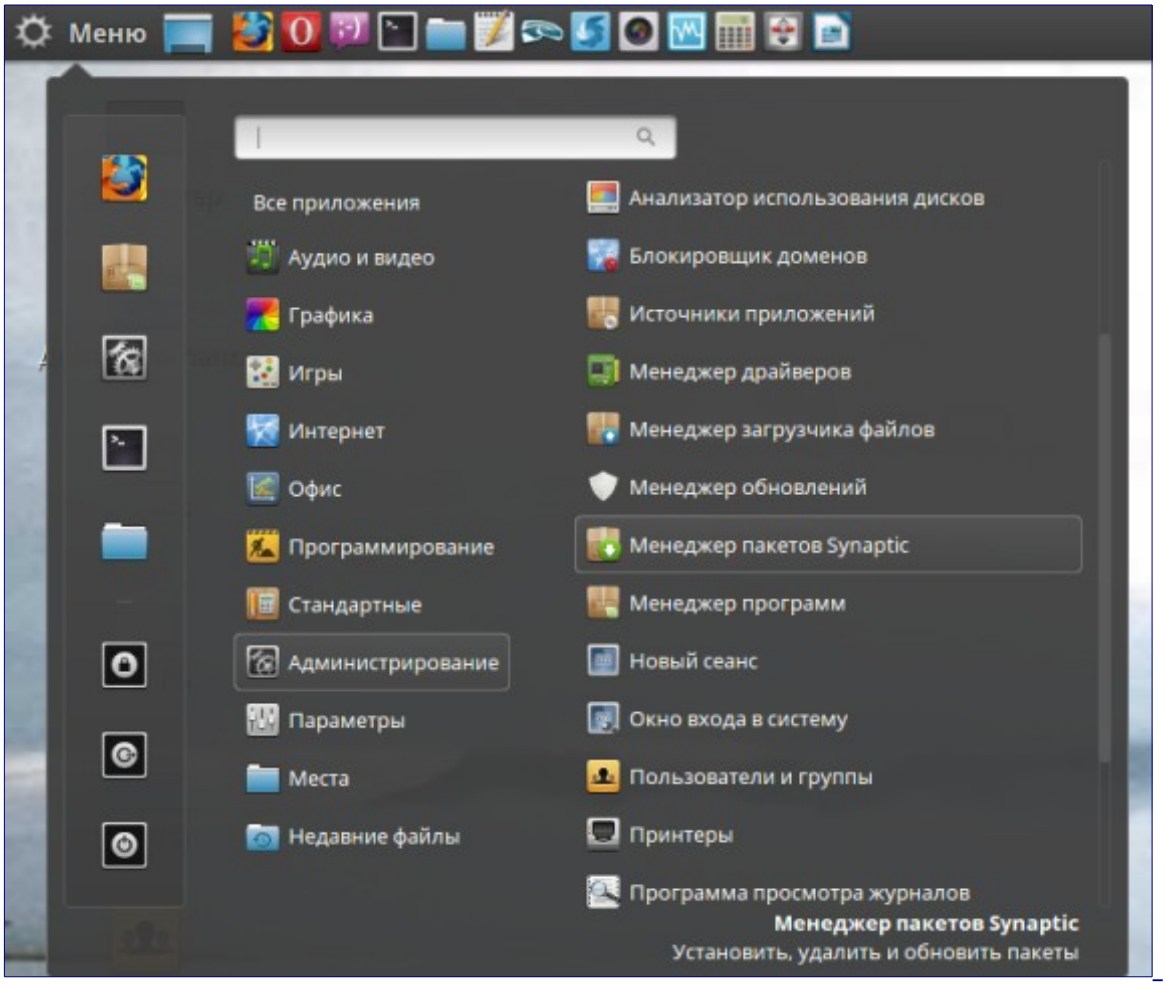
На этом разговор о настройках Mint совместно с Cinnamon можно считать законченным. Следующий очерк имеет отношение только к Mint, так как посвящён «фирменному» иснтсрументарию этого дистрибутива — и не только имеющему отношение к настройкам.

Mint: фирменный инструментарий

Редкий дистрибутив из числа тех, что носят это гордое имя по праву, не обзаводится более или менее полным набором системного инструментария, специфичного только для него (в дальнейшем я буду называть такие инструменты фирменными). Не исключение в этом отношении и Mint — он имеет набор фирменных инструментов для решения весьма широкого круга задач, от управления программным обеспечением до настройки WiFi и блокировки доменов. Они и будут предметом данного очерка.

Вступление

Как уже было сказано, фирменный инструментарий дистрибутива Mint охватывает весьма широкий круг задач и потому представлен большим количеством отдельных утилит, которые обнаруживаются в любой из его редакций, располагаясь в каталоге /usr/bin. Полный их список включает более 20 исполняемых файлов вида mint*. Большинству из них соответствует пункт в разделе Администрирование главного меню Cinnamon:



Фирменные инструменты Mint будут рассмотрены на примере среды Cinnamon. Однако десктопная специфика касается только способа их запуска из главного меню. Сами же инструменты идентичны во всех редакциях дистрибутива Mint.

Рассмотрение фирменного инструментария целесообразно начать с менеджера программ, как наиболее востребованного его компонента.

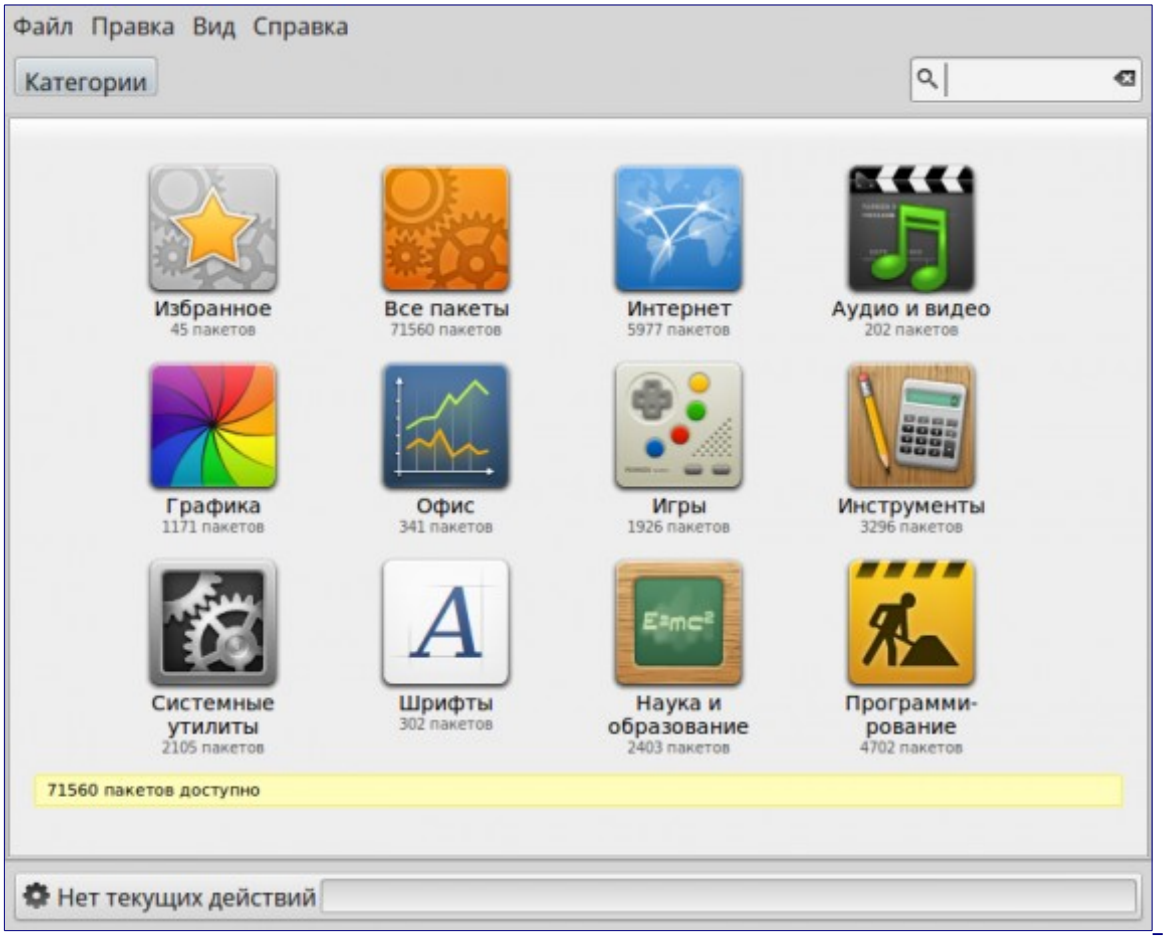
Менеджер программ mintinstall

Менеджер программ mintinstall занимает центральное положение в наборе фирменного инструментария дистрибутива Mint. Он принадлежит к классу самых «высокоуровневых» инструментов для управления пакетами, которые можно назвать интегрированными центрами приложений, о чём подробнее будет сказано в одном из последующих очерков.

Как только что было сказано, mintinstall можно запустить одноимённой командой из терминального окна или строки минитерминала. А можно обратиться к главному меню Cinnamon, где он обнаруживается в разделе Администрирование. Однако залезать в него не обязательно — пиктограмма запуска Менеджера приложений вынесена в левую колонку быстрого запуска второй сверху:

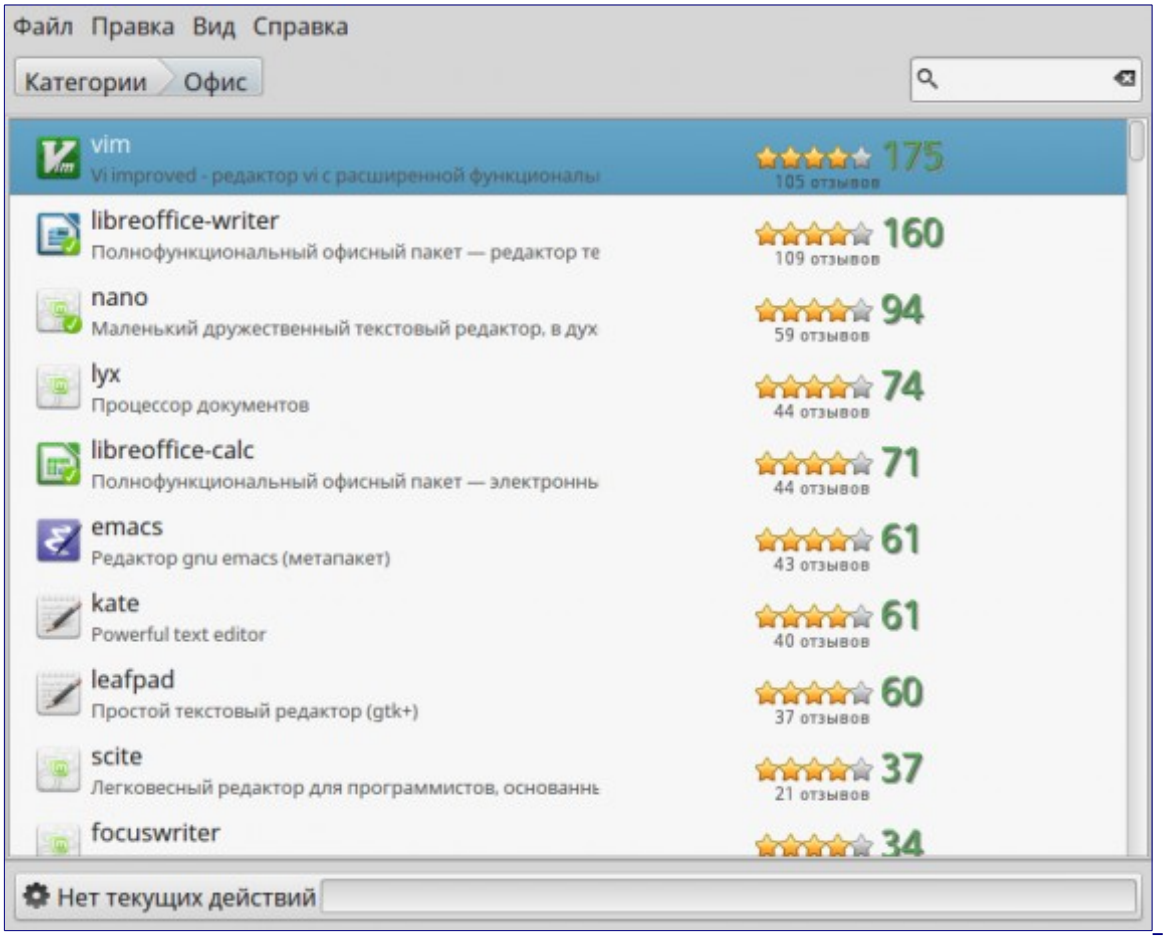


Будучи запущенным тем или иным способом, mintinstall для начала запрашивает пароль пользователя, после чего предстаёт перед его взором в следующем виде:



Первое, что обращает на себя внимание — минималистичный дизайн: никаких баннеров, новинок, рекомендаций. Только поле поискового запроса, пиктограммы категорий софта и строка состояния текущих действий (сразу после запуска, разумеется, пустая). И потому mintinstall не вызывает визуального отторжения.

В обращении mintinstall столь же прост, как и внешне. А обращение с ним, разумеется, начинается с поиска нужного пакета. Сделать это можно, во-первых, просматривая категории, например Офис:

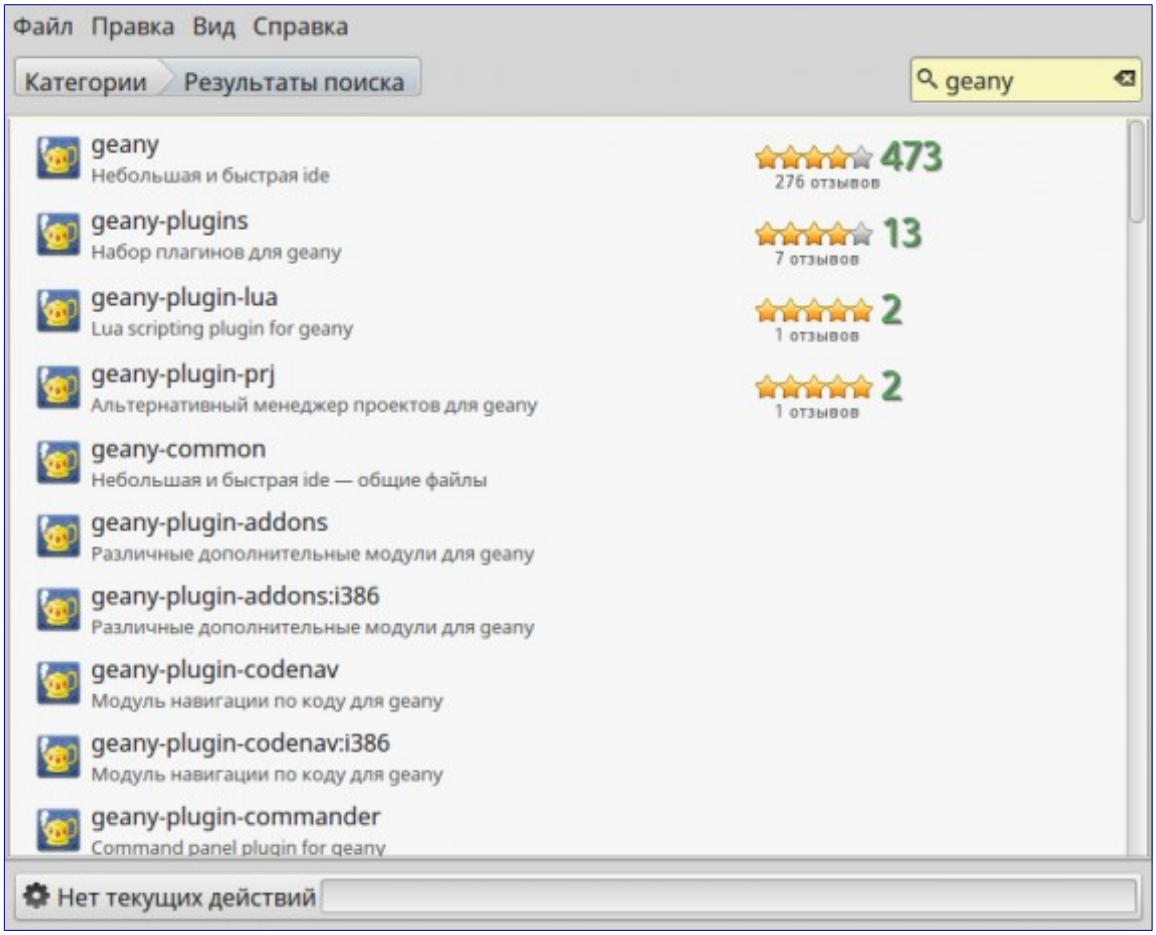


Однако это не самый простой путь. Во-первых, просматривать списки категорий (а они включают в себя от пары сотен до 4-5 тысяч позиций) — не самое весёлое занятие. Во-вторых, оно осложняется ещё и тем, что пакеты в этих списках отсортированы не по алфавиту, а по количеству полученных отзывов. В-третьих, критерии отнесения пакета к той или иной категории не всегда понятны. Так, категория Офис включает в себя не только собственно офисные пакеты, но и, скажем, текстовые редакторы, в том числе и такие, которые обычно относятся к классу системных приложений, например, vim и nano, и даже к инструментам программирования, вроде текстового редактора Geany, в некоторых кругах именуемого интегрированной средой разработки (IDE).

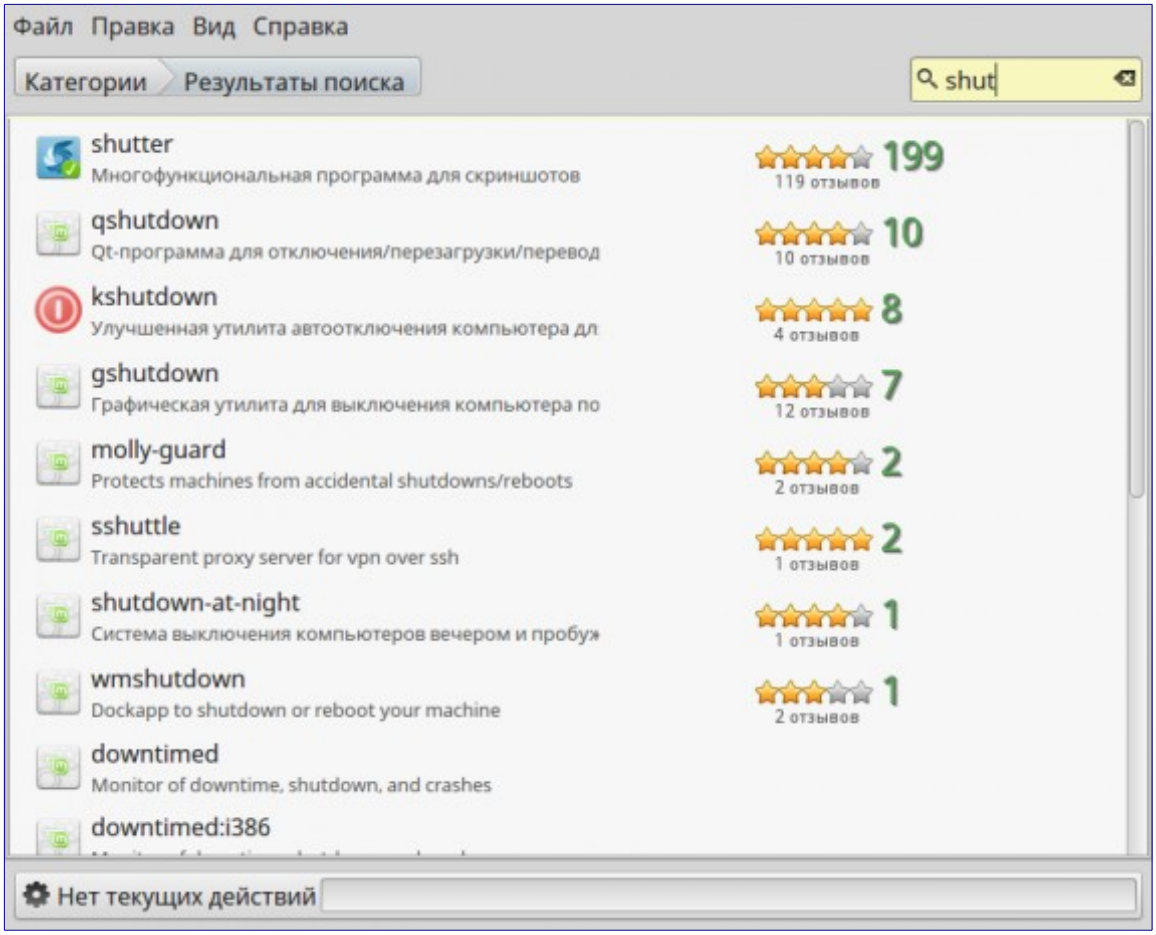
Впрочем, Geany мы не увидим ни в категории Офис , ни в категории Программирование. Ибо, и это в четвёртых, есть ещё и категория Избранное, куда попадают пакеты с наибольшим количеством отзывов в своих «законных» категориях. Именно к этой категории и удостоился чести быть причисленным Geany, имеющий на момент сочинения этих строк 473 отзыва:



Поэтому, если известно имя пакета (или хотя бы фрагмент имени), проще воспользоваться полем поискового запроса. Таким образом пакет Geany находится мгновенно — а если после его нахождения нажать кнопку Показать все результаты, то будет выведен и список всех его плагинов:

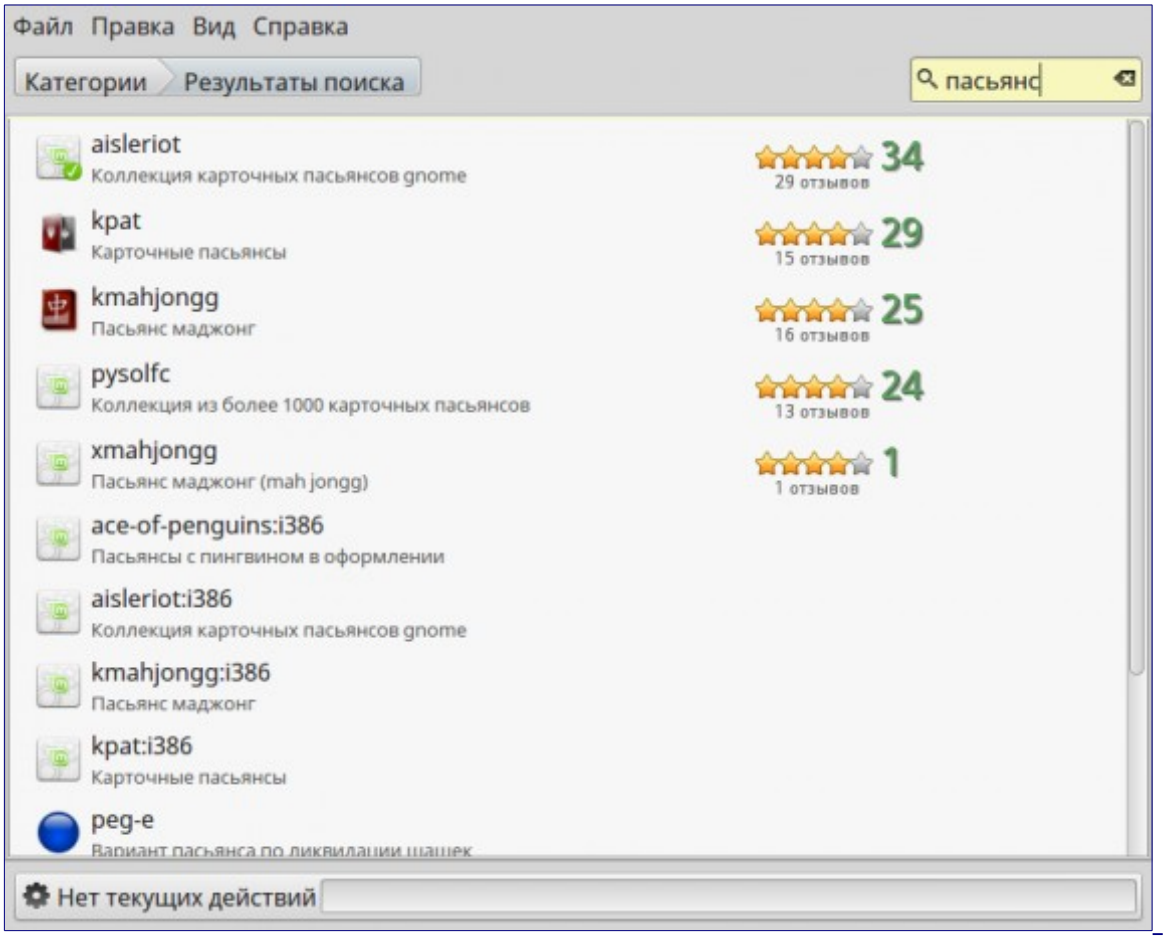


По умолчанию поиск в mintinstall простой, но его легко сделать инкрементным (как — скажу чуть позже). И тогда с каждым набранным символом список соответствий сокращается. Например, при поиске пакета Shutter, предназначенного для изготовления скриншотов (иллюстрации ко всем заметкам сделаны именно им), это выглядит так:

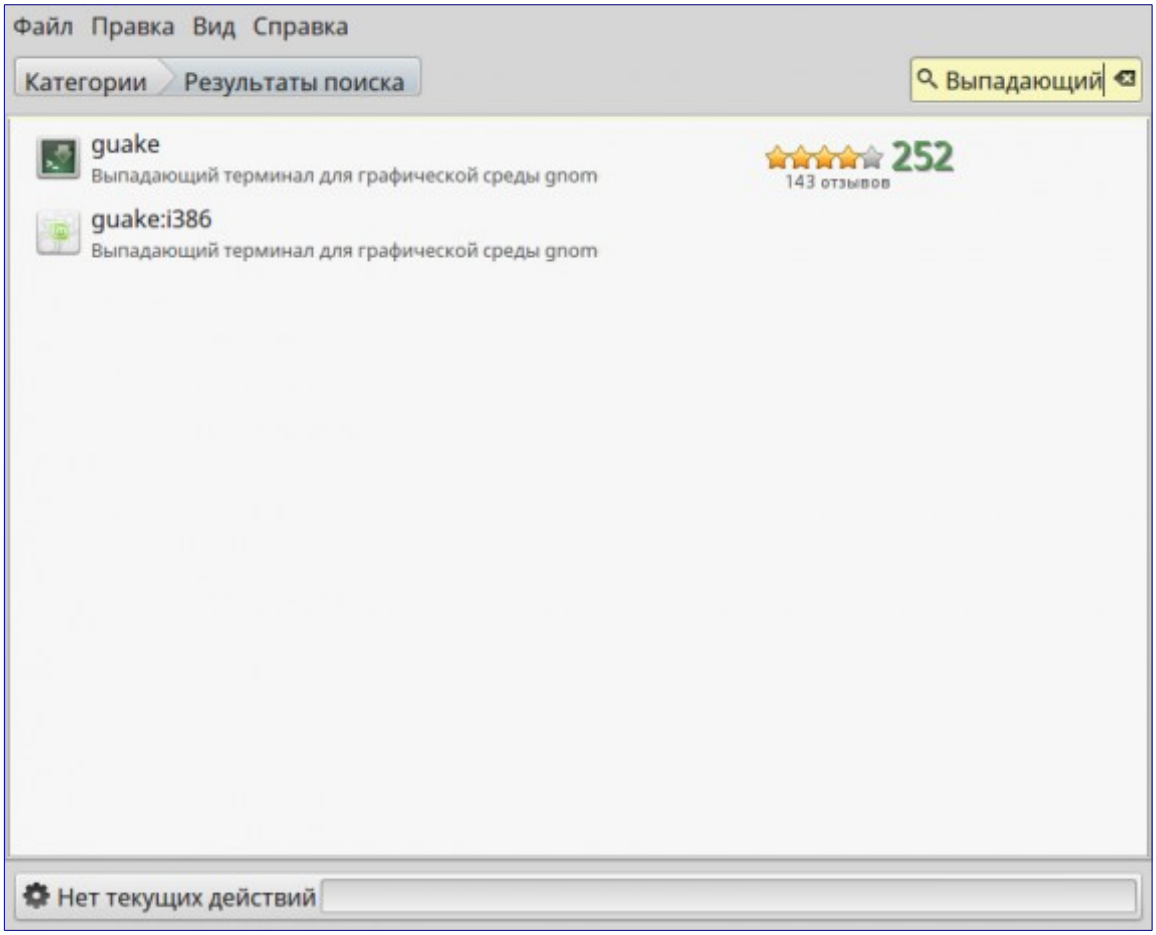


Нужно только учитывать, что порядок вывода пакетов — не по соответствию имени введённым в поле поиска символам, а опять же по количеству отзывов.

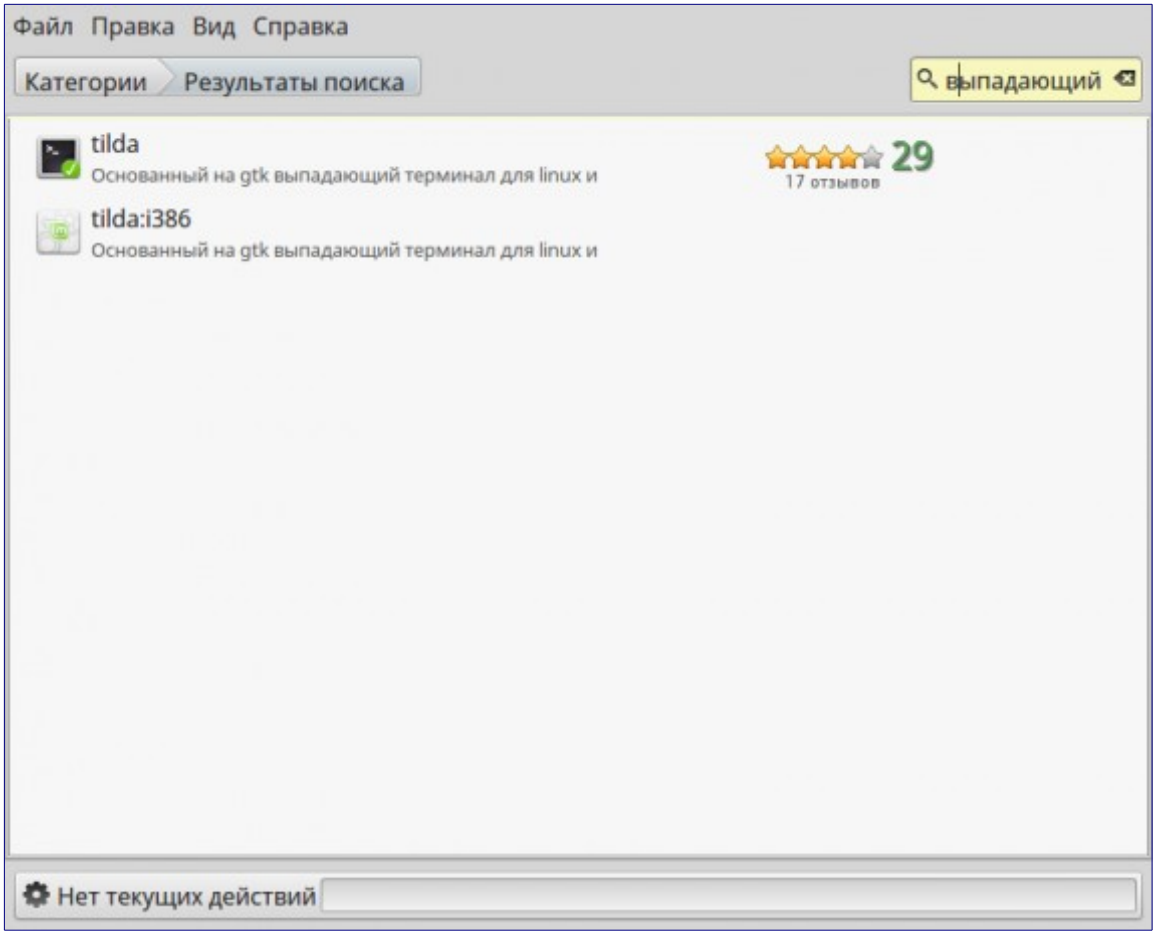
Впрочем, по умолчанию поиск осуществляется по соответствию не только имени пакета, но также и кратким описаниям, которые могут быть даже на русском языке. Это можно видеть на примере поиска пакета aisleriot, представляющего собой коллекцию пасьянсов:



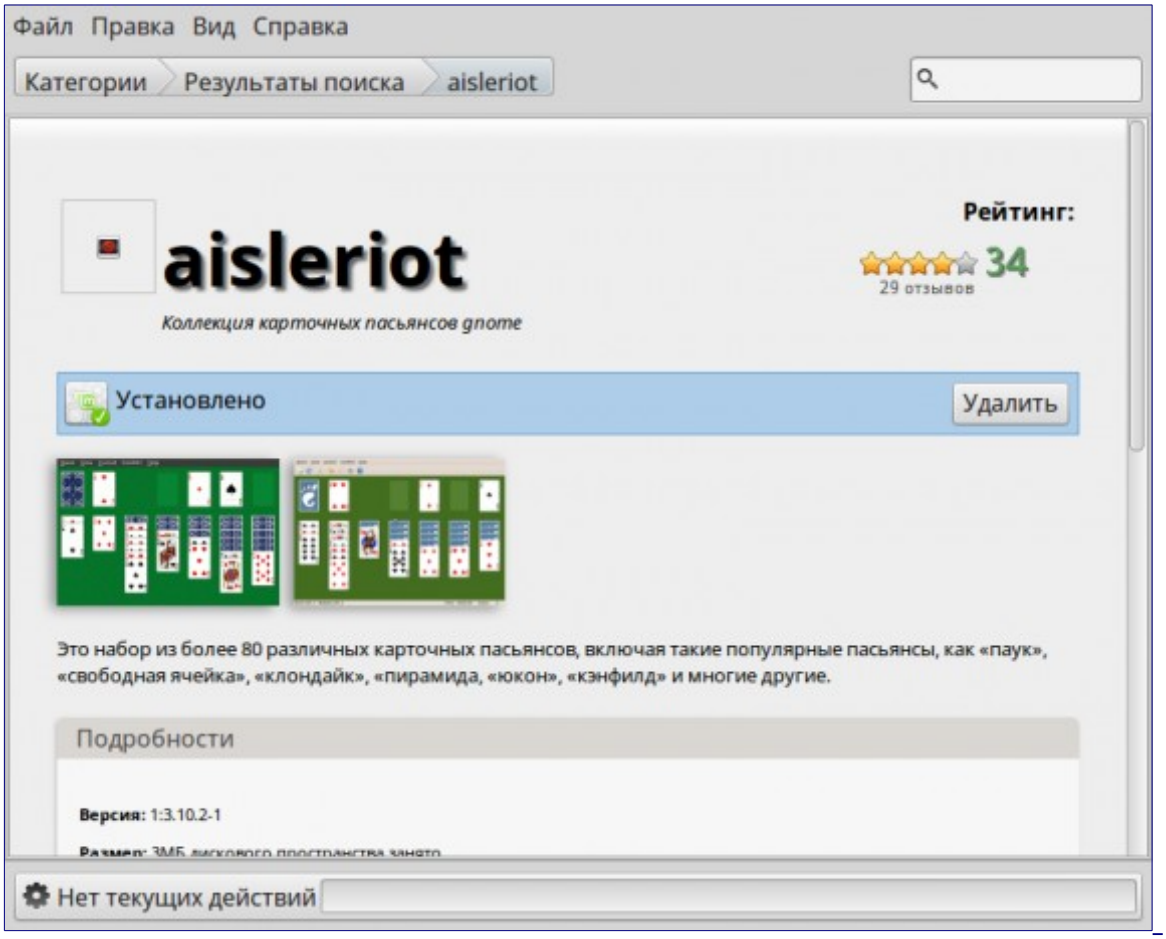
Следует помнить, что поиск — регистро-зависимый. Это можно продемонстрировать на примере поиска пакетов выпадающих терминалов. Если в поле поиска ввести слово Выпадающий, мы увидим пакет выпадающего терминала Guake:



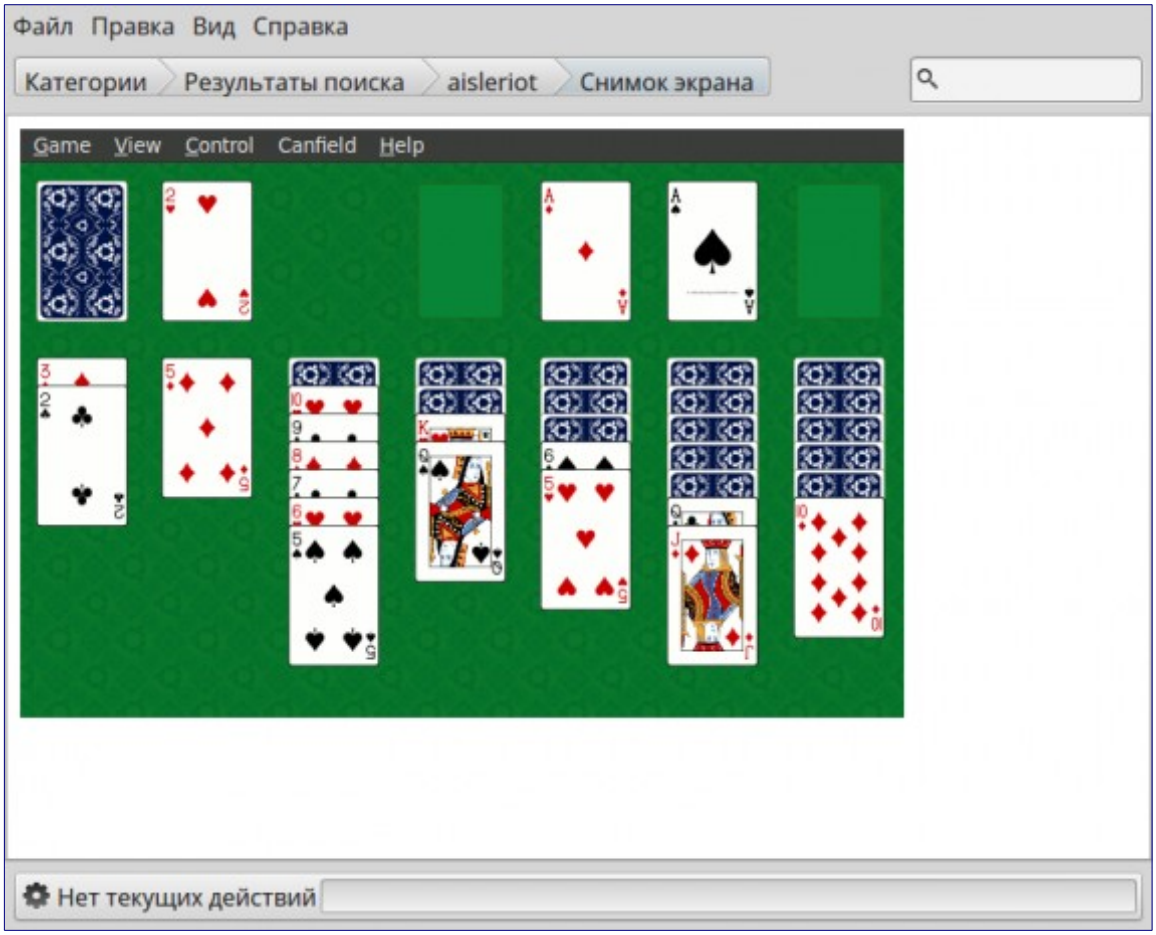
А по ключевому слову выпадающий обнаружится совсем другой выпадающий терминал, Tilda:



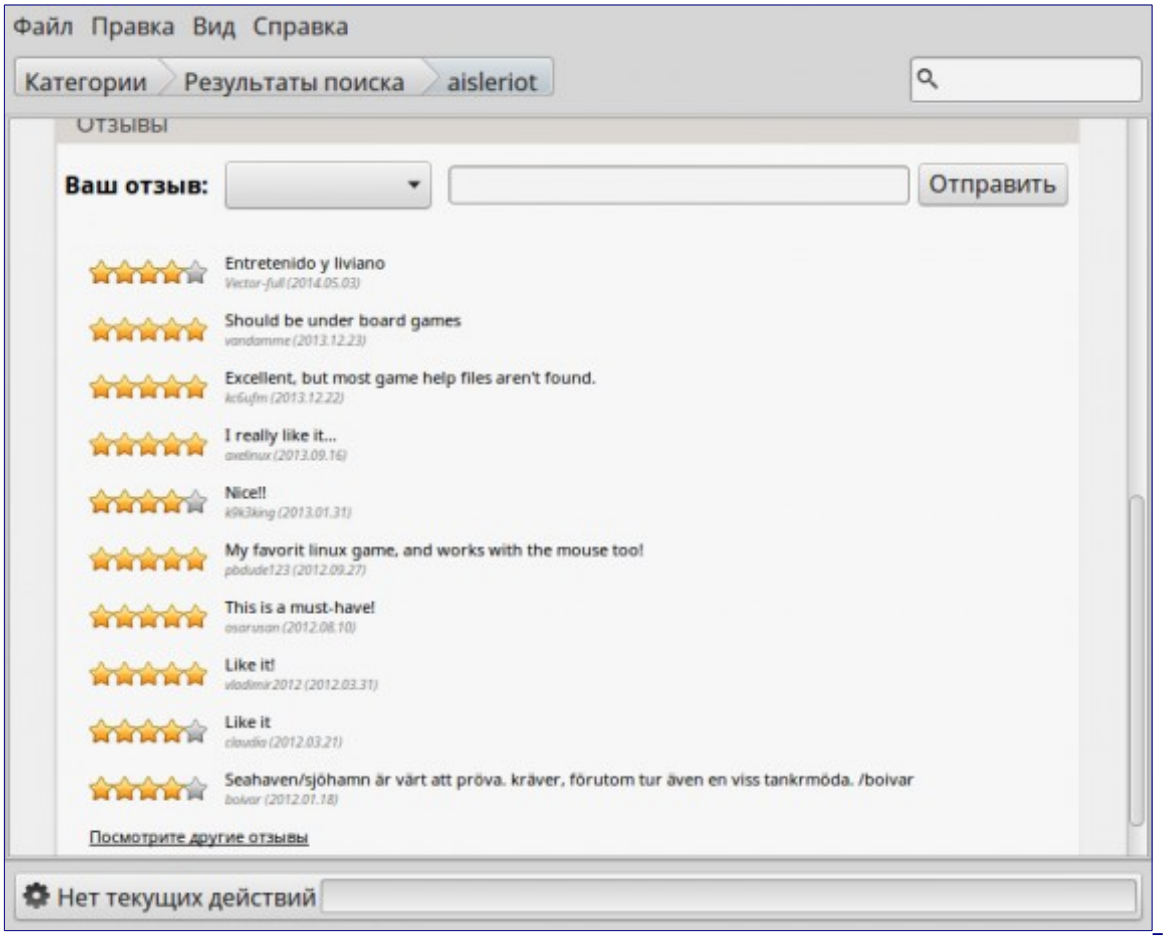
Если дважды кликнуть на строке с именем найденного пакета, появится страница с его описанием. Нередко оно будет на русском языке, и может содержать картинки:



Картинки кликабельны, так что их можно вывести «крупным планом»:



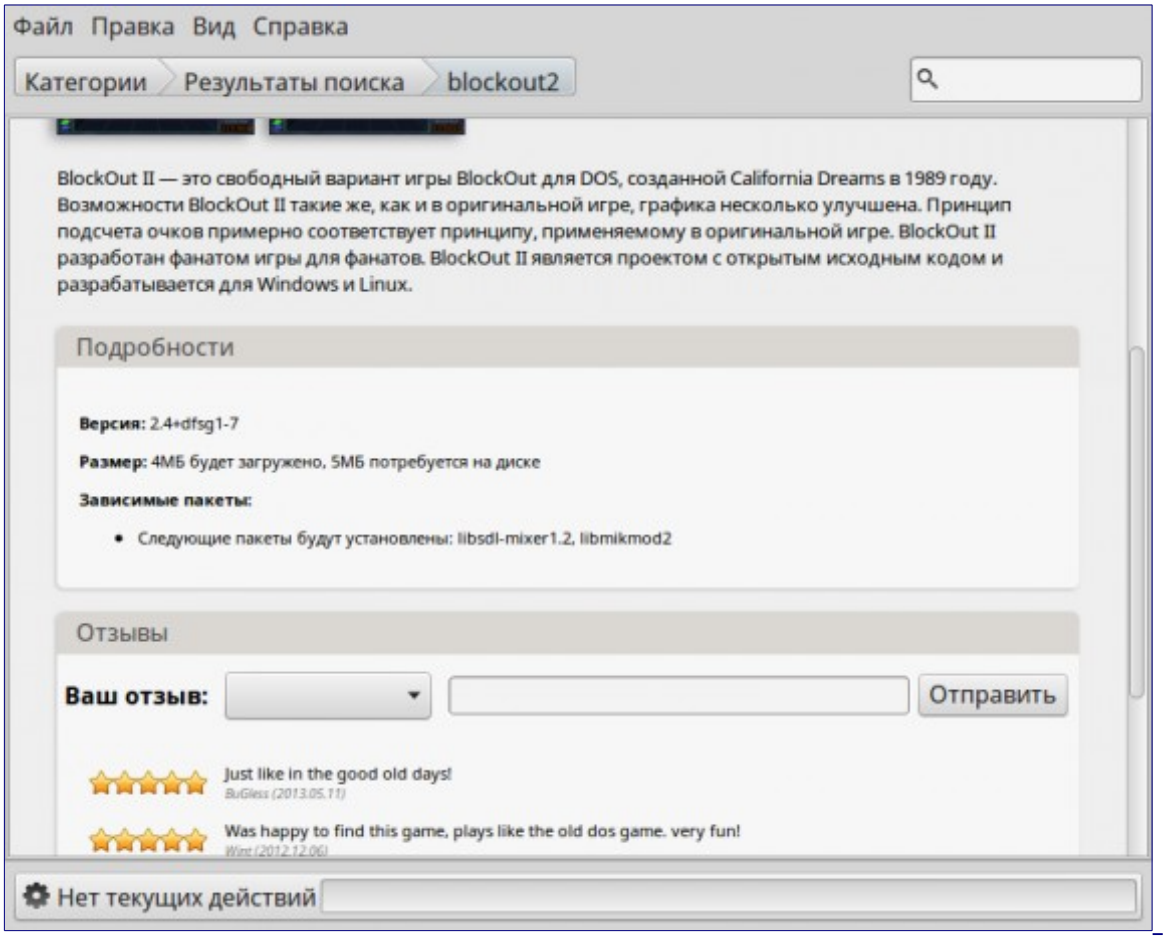
Здесь же можно прочитать и отзывы о пакете, если таковые имеются:



А можно также и оставить свой отзыв. Правда, для этого надо предварительно зарегистрироваться в сообществе пользователей (как — расскажу позже).

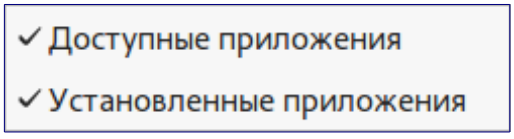
Определившись, путём чтения описания и, возможно, отзывов (хотя они очень редко несут какую-либо информацию кроме эмоций), с нужностью найденного пакета, его остаётся только установить. Для чего требуется нажать соответствующую экранную кнопку — и процесс начнётся без единого вопроса.

Столь же молчаливо будут установлены и все необходимые зависимости, поэтому с их списком лучше ознакомиться заранее. Например, для игры blockout2 он выглядит так:

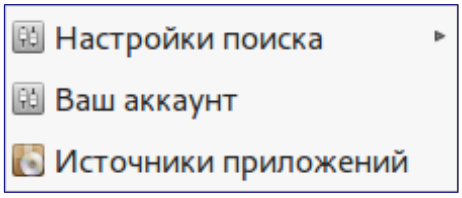


После установки пакета на его странице появляется кнопка Удалить очевидного назначения, которое также претворяется в жизнь без всяких вопросов. Нужно только учитывать, что пакеты, установленные как зависимости удаляемого, удалены не будут, их придётся вычищать или по списку по списку из раздела Подробности, или другими средствами. Первый вариант — рискованный, так как при этом можно затронуть зависимости других пакетов. Второй же лежит вне темы данного очерка — о нём будет говориться в своё время. Так что для удаления пакетов Менеджером программ лучше просто не пользоваться, за исключением очень простых или хорошо известных применителю случаев.

Кроме строки поиска, mintinstall имеет ещё и меню. Где в пункте Вид определяется, выводить ли Доступные приложения, Установленные приложения, или, как по умолчанию, те и другие:

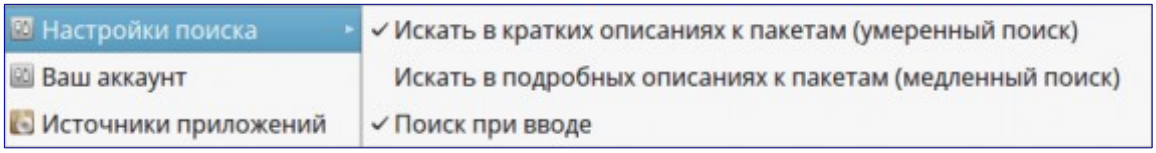


В пункте Правка — три подпункта: Настройки поиска, Доступ к аккаунту в сообществе и Источники приложений:



В первом из них, во-первых, определяется, искать ли только в кратких

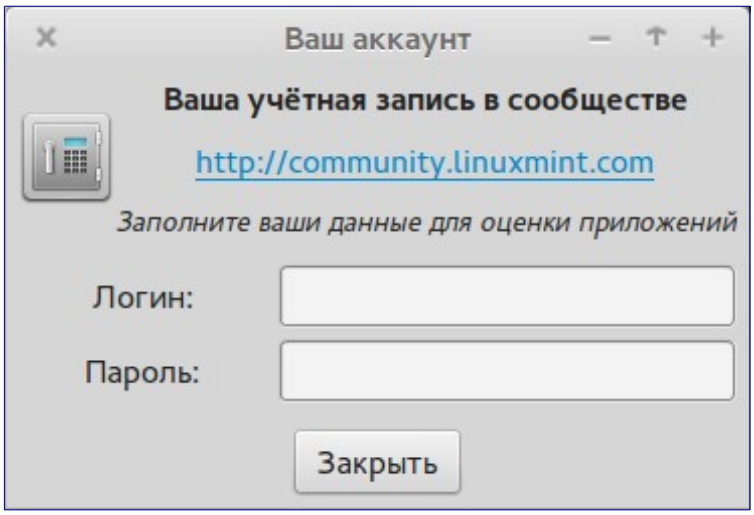
описаниях пакетов (отмечено по умолчанию) или также в подробных, а во-вторых — включить инкрементный поиск (Поиск при вводе):



Про аккаунт в сообществе Linuxmint я расскажу в маленькой интермедии. А пункт Источники приложений вызывает отдельную утилиту software-sources, которая будет рассмотрена последующем за ней очерке.

Интермедия об аккаунте в сообществе

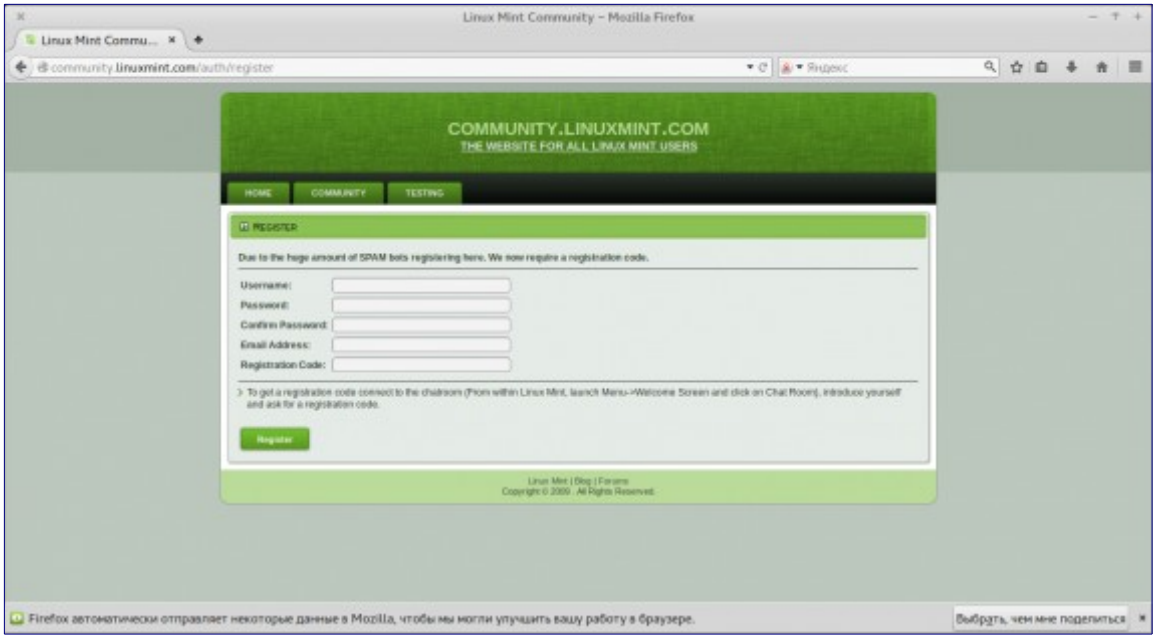
Как мы только что видели, для того, чтобы оставить отзыв о пакете в Менеджере программ, необходимо открыть аккаунт в сообществе Linux Mint. Для этого надо пройти к пункту меню Правка -> Ваш аккаунт, который вызовет панель с предложением к авторизации:



Нетрудно догадаться, что если аккаунта ещё нет — следует пойти по указанному там адресу. После чего приглашение к авторизации откроется уже в браузере:



Кнопка Register находится на видном месте. Нажав её, можно видеть регистрационную форму:



Заполнение полей — очевидно, кроме последнего, Registration Code. Чтобы получить его, нужно открыть тот самый экран приветствия, который был виден при первом запуске свежеинсталлированного Mint'a. И показ которого, скорее всего, был тогда же и отключён. Однако его можно вызвать в любой момент — это такой же компонент фирменного инструментария, как и все остальные из рассматриваемых в этом очерке. Делается это либо из терминала вводом команды

```
$ mintwelcome
```

либо через главное меню Параметры -> Экран приветствия:



Теперь остаётся только найти пиктограмму с подписью Чат-комната, в эту самую комнату войти и в свободной (но желательно вежливой) форме на английском языке запросить код для регистрации. Практически мгновенно от пользователя mintbotd придёт ответ, гласящий, что код выслан в личном сообщении. Откуда он и вставляется в регистрационную форму, после чего регистрация мгновенно совершается.

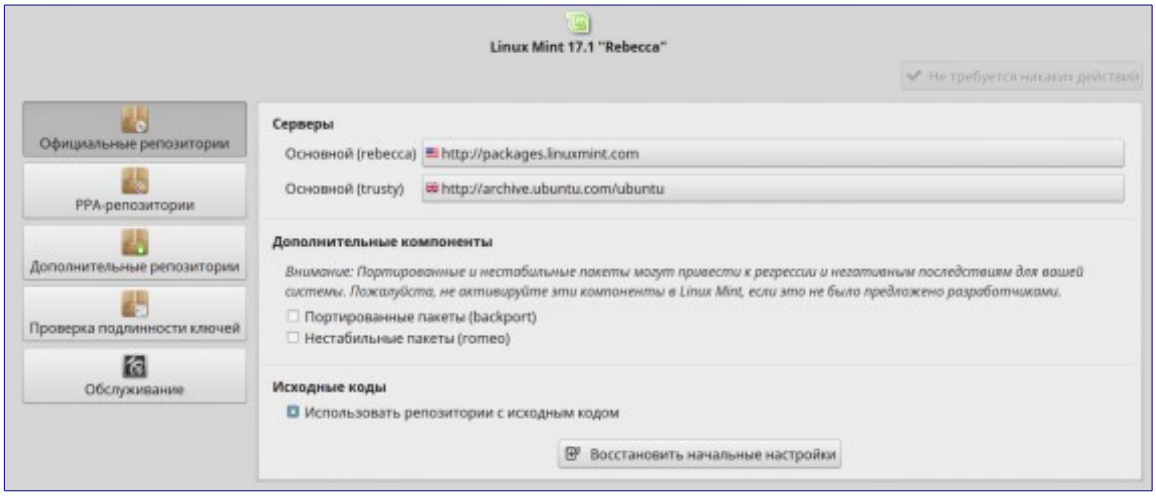
Теперь можно вернуться к форме, вызванной из Менеджера программ, авторизоваться там и оставлять отзывы в своё удовольствие. Причём устанавливать программу, на которую даётся отзыв, совсем не обязательно. А вот оценку ей надо дать непременно.

Разумеется, аккаунт в сообществе нужен не только для того, чтобы оставлять отзывы на пакетах. Будучи членом сообщества, можно получать доступ к тому, что создаёт его мозг (Клемент Лефевр) на ранних стадиях разработки. И даже поучаствовать в тестировании.

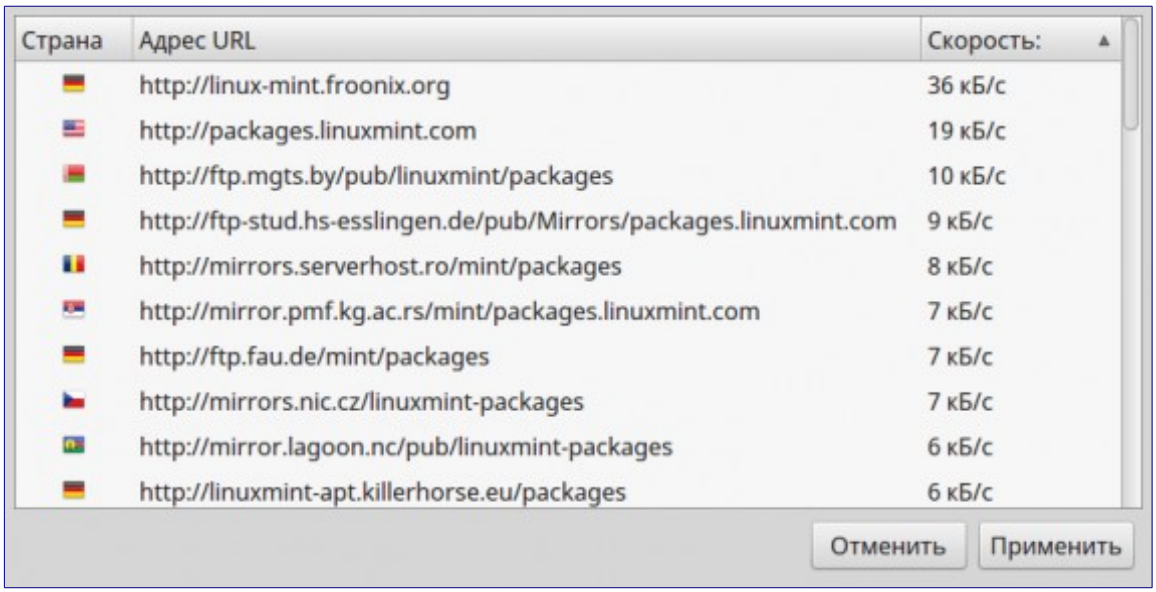
Менеджер репозиториев software-sources

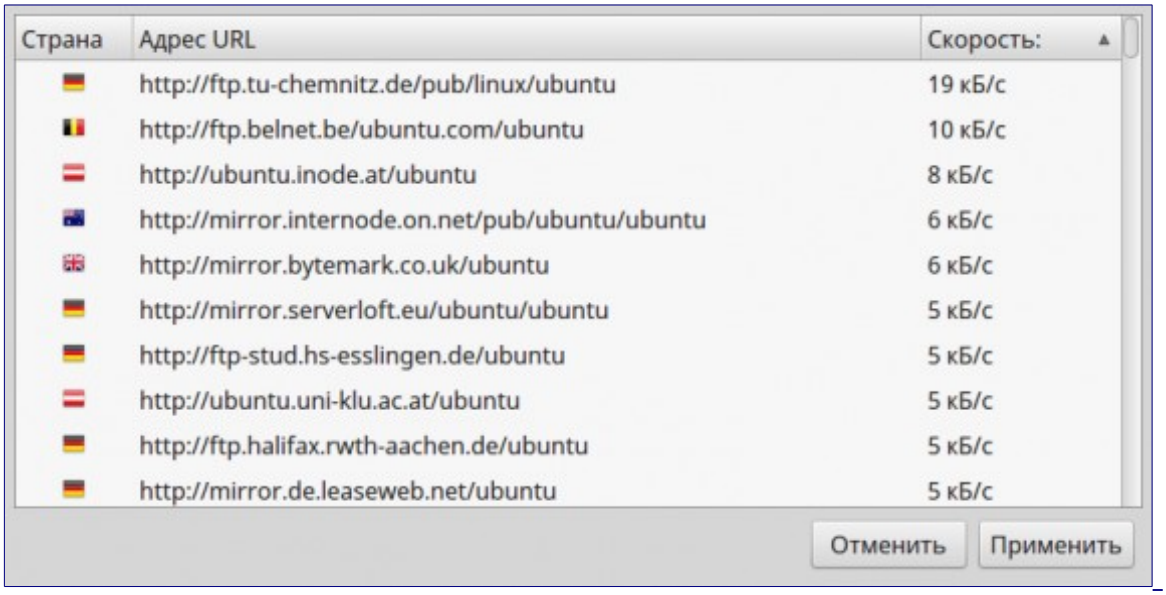
А теперь вернёмся в Менеджер программ с другой целью — проследовать в пункт его меню Правка -> Источники приложений. Через него вызывается самостоятельная утилита фирменного набора, mintsources, она же software-sources (первое имя — символическая ссылка на второе). В разделе Администрирование главного меню ей соответствует пункт Источники приложений (это и есть официальное название программы, менеджер репозиториев — моя отсебятина, придуманная единообразия ради). Наконец, плюс к упомянутой возможности вызова software-sources из Менеджера программ, пиктограмма запуска её есть и в секции Администрирование Системных настроек Cinnamon.

Вне зависимости от способа запуска, после ввода пароля открывается окно software-sources с пятью страницами, переключение между которыми осуществляется экранными кнопками. На первой странице, именуемой Официальные репозитории, выбираются зеркала двух основных репозиториев — собственного и репозитория Ubuntu (вся базовая часть Mint берётся из последнего). Здесь же отмечается, следует ли использовать бэкпорты, нестабильные пакеты, а также исходники:

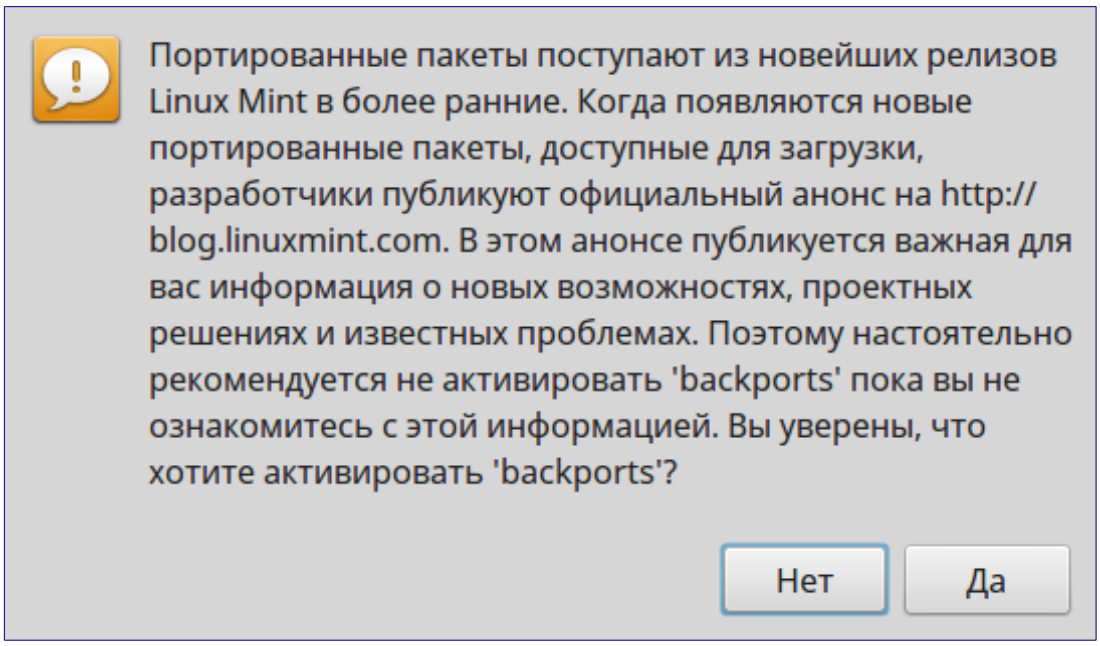


В списке зеркал обоих из основных репозиториев указываются их URL'ы, флажок страны размещёния, а также реальная скорость соединения — последняя колонка появляется по прошествии некоторого времени, необходимого для получения соответствующих данных. Именно по скорости соединения список и сортируется, так что в обоих случаях следует просто выбрать верхнюю строку (в списке зеркал нет ни одного российского, так что выбор по «географическому» принципу смысла не имеет):

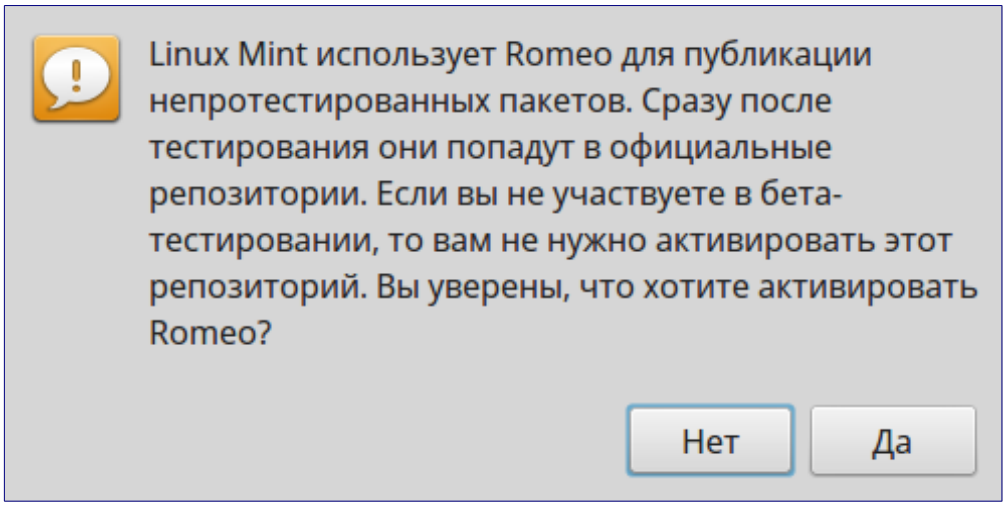




Использование «обратно портированных» (backport) и нестабильных (romeo) пакетов разработчиками настоятельно не рекомендуется, и по умолчанию эти ветви репозиториев отключены. Попытка активировать любую из них вызывает предупреждение, для бэкпортов такое:



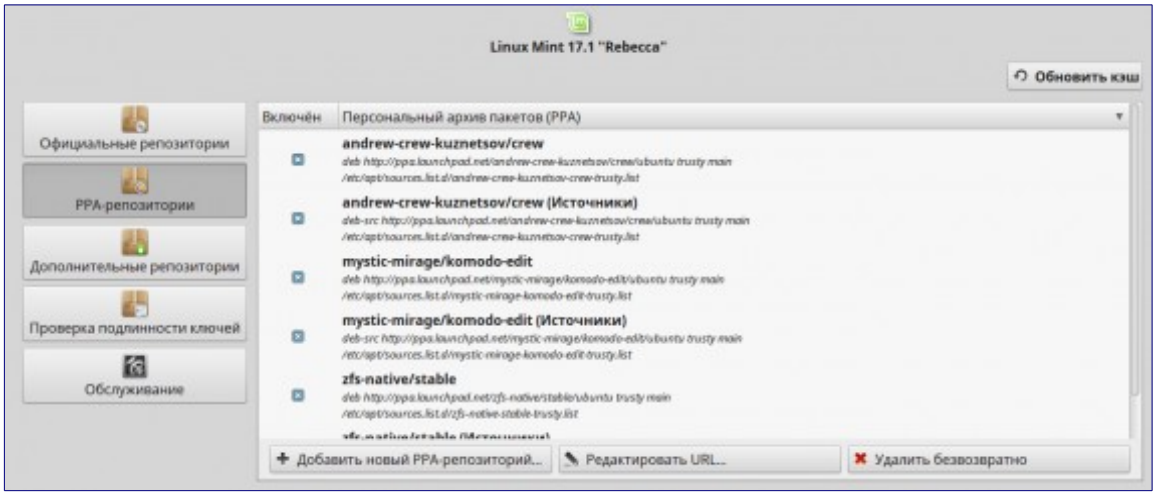
А для нестабильных пакетов — такое:



Не вижу оснований не прислушаться к этим предупреждениям — в любом случае, и к бэкпортам, и к нестабильным пакетам следует подходить индивидуально, а не устанавливать их все гуртом.

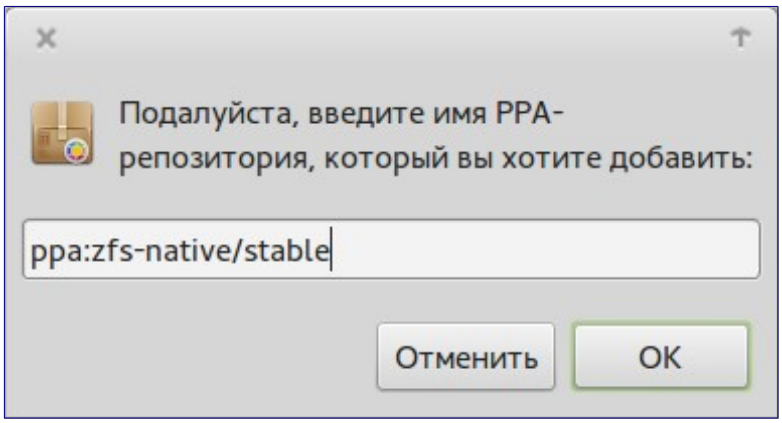
Отключено также использование ветки репозитория, содержащей исходные тексты пакетов. Активация её не несёт никакой опасности, и потому не сопровождается предупреждением. Просто доступ к исходникам нужен далеко не всем применителям, а лишь тем, кто пересобирает пакеты с каким-либо своими специфическими опциями.

Вторая страница — PPA-репозитории, то есть дополнительные PPA-репозитории из централизованного хранилища всех пакетов, собранных независимыми разработчиками и майнтайнерами:

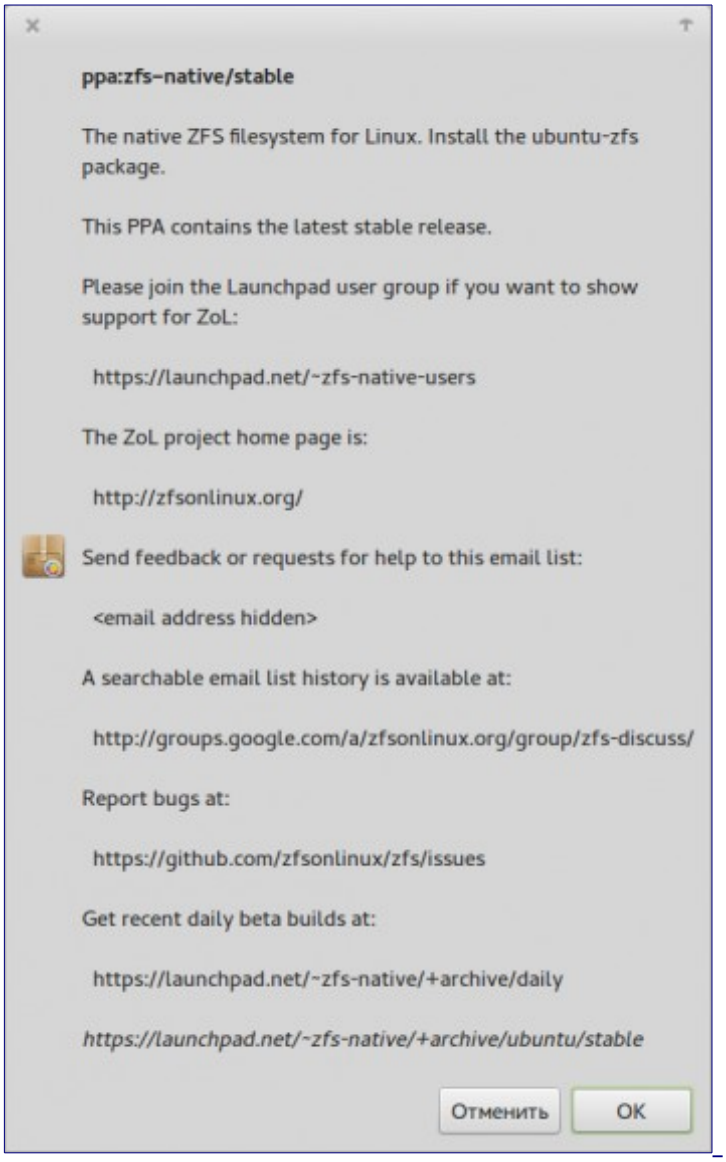


PPA-репозитории предназначены для Ubuntu и её прямых родственников (вроде Kubuntu и Xubuntu). Но, поскольку Mint с Ubuntu полностью обратно совместим, пакеты эти обычно (если не вообще всегда) можно использовать и в нём. По крайней мере, я не только не сталкивался с какими-либо проблемами, но и не слышал о таковых. Для доступа к PPA-репозиториям фирма Canonical разработала специальную систему с web-интерфейсом — Launchpad.

Для подключения дополнительного репозитория его сначала нужно отыскать на Launchpad'e и определить его ppa-имя. Например, для PPA-репозитория с пакетами поддержки файловой системы ZFS оно будет таким: ppa:zfs-native/stable. Затем кнопкой Добавить новый... вызывается панель, в соответствующее поле которой это имя вписывается:



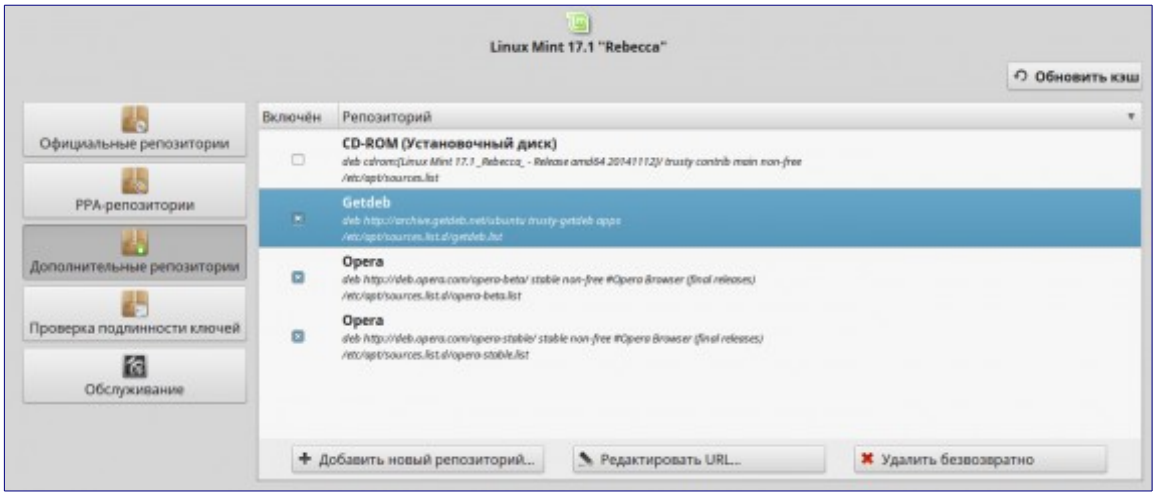
Нажатие кнопки OK вызывает панель с информацией о репозитории:



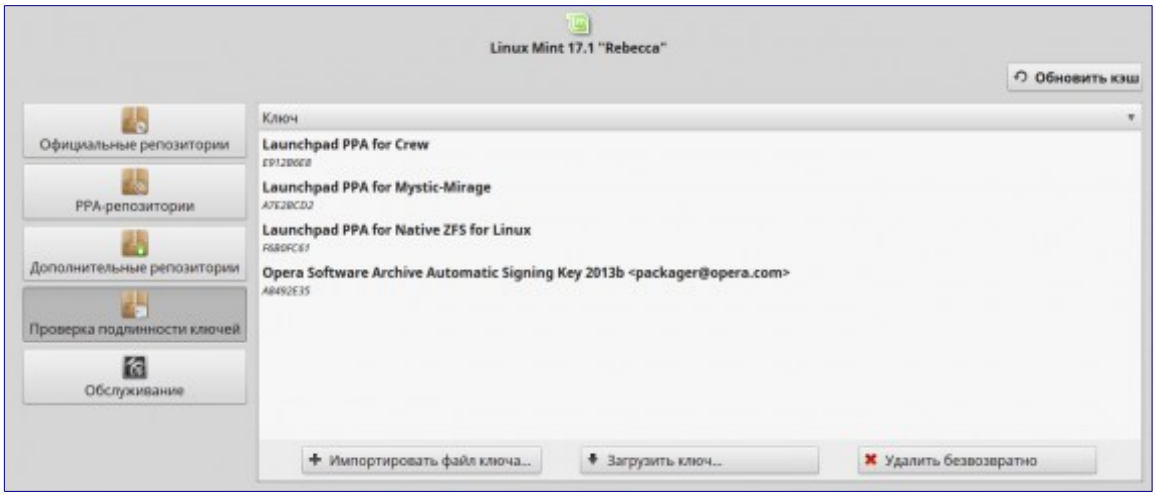
И после подтверждения своих намерений новый репозиторий появляется в общем списке.

В большинстве случаев при подключении PPA-репозиториев автоматически подключаются и их ветки с исходниками (в русском переводе почему-то называемые Источниками) .

На странице Дополнительные репозитории аналогичную процедуру можно выполнить для репозиториев произвольных, в том числе и локальных:



Страница Проверка подлинности ключей предназначена для хранения ключей к подключённым репозиториям — в большинстве случаев они вносятся в список автоматически:



Наконец, на странице Maintenance можно произвести исправление проблем с локальными кешами пакетов, буде таковые возникнут (у меня пока не возникало) и их очистку от продуктов жизнедеятельности при установке пакетов:

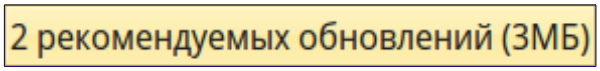


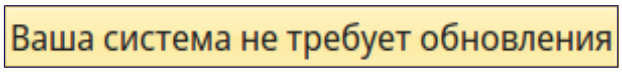
В правом верхнем углу окна программы можно видеть кнопку Обновить кэш. К ней следует обращаться после любых действий с репозиториями — это приведёт локальный кэш пакетов в актуальное состояние.

Впрочем, не будет большого вреда нажать эту кнопку и в том случае, если никаких изменений в составе репозиториев не выполнялась — на всякий пожарный случай.

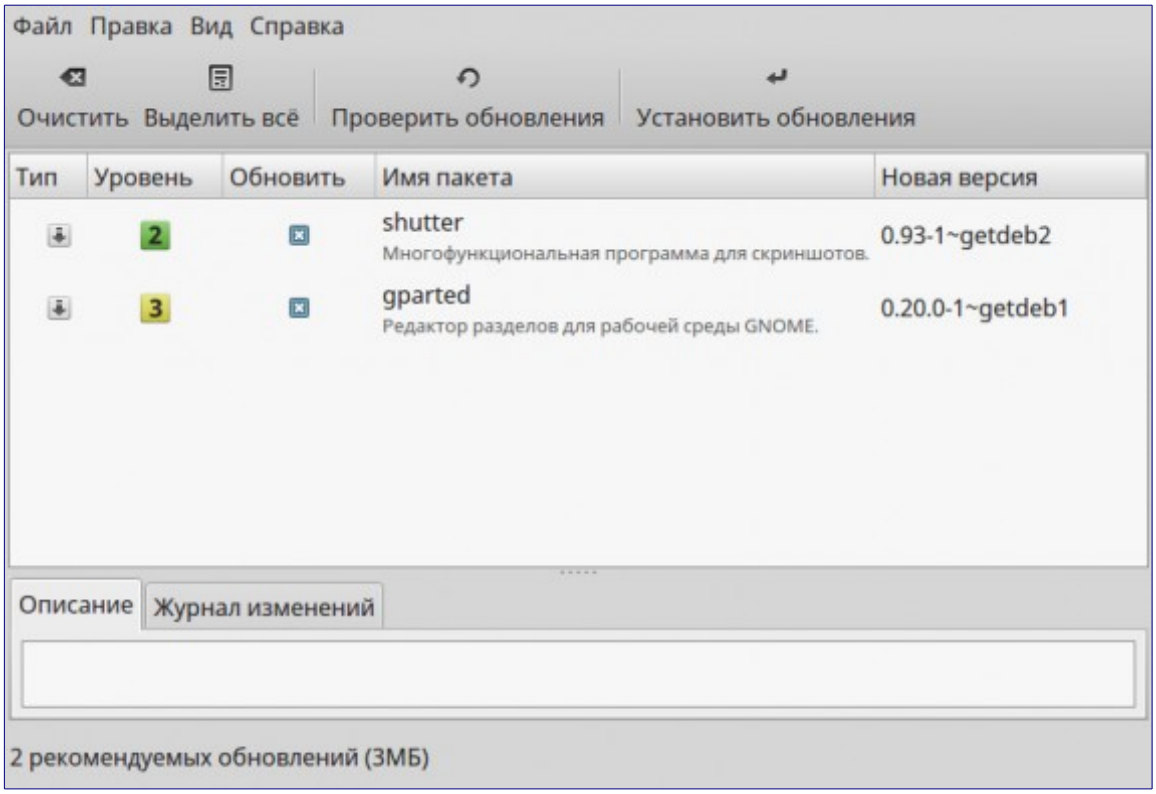
Менеджер обновлений mintupdate

После рассмотрения Менеджера программ и Менеджера репозиториев резонно перейти к средствам, обеспечивающим обновление системы. Таковым в фирменном наборе инструментов Mint является Менеджер обновлений — mintupdate. По умолчанию он включается в автозапуск, и потому применителю не нужно беспокоиться о его запуске: пиктограмма его сидит в трее, изменяя свой вид в зависимости от доступности обновлений: в виде буковки i на голубом фоне в случае их наличия, и в виде зелёной «галочки» — если система обновлений не требует. Соответствующие подсказки всплывают и при наведении курсора мыши на пиктограмму:





При доступности обновлений получить визуальное представление о них можно, щёлкнув мышью на пиктограмме. После этого будет выведен список пакетов, которые могут быть обновлены в данный момент времени:



Строго говоря, начиная с Mint 17.1, вывод не совсем попакетный: в одной строке списка может быть сгруппировано несколько родственных пакетов, которые друг без друга всё равно не устанавливаются, например — cinnamon и cinnamon-common. Эту группировку не следует путать ни с зависимостями, ни с метапакетами — она делается исключительно для компактности представления и лёгкости восприятия.

Далее, некоторых пояснений требуют первые две колонки. Первая — тип обновления. Их два — стандартно обновляемые пакеты по выходе их новой сборки или версии (отмечены серой стрелкой) и обновления безопасности, ликвидирующие выявленные «дыры» в них (отмечены красным восклицательным знаком).

Во второй колонке указывается уровень безопасности обновления пакетов. Здесь под безопасностью понимается не вероятность использования их злодеями, а то, как обновление пакета может повлиять на общую стабильность системы. Уровней безопасности в этом смысле пять:

1. сертифицированные пакеты — обычно те, что непосредственно поддерживаются майнтайнерами Mint;

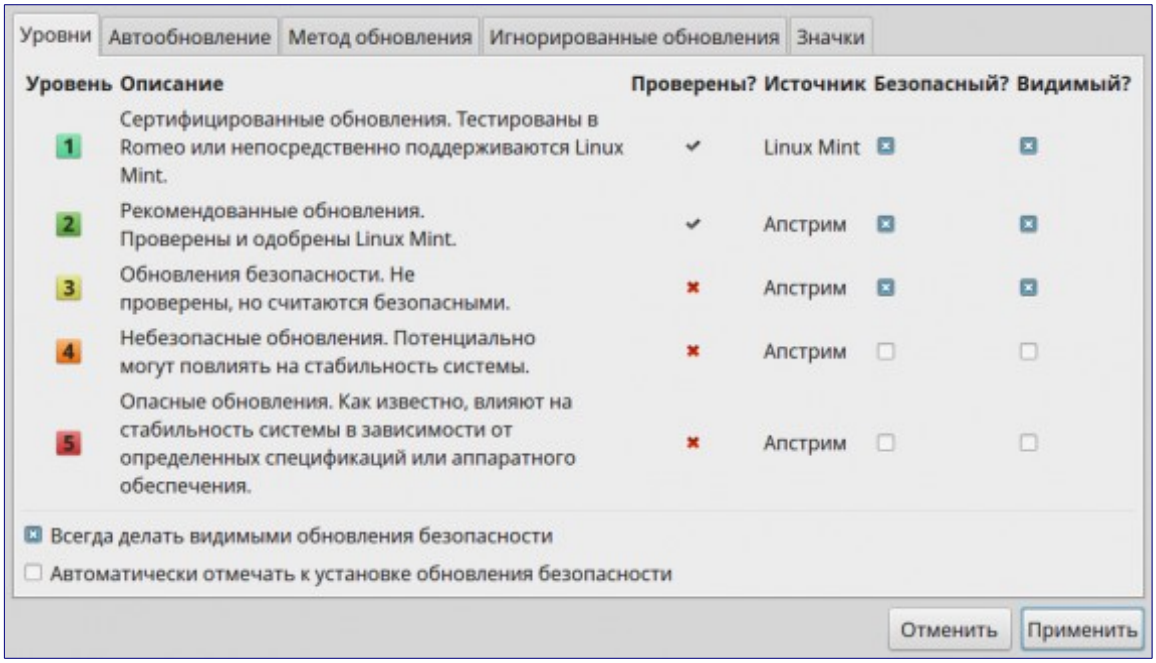
2. рекомендуемые пакеты — проверены и одобрены разработчиками этого дистрибутива;

3. безопасные пакеты — не проверялись разработчиками, но нарушение стабильности системы при их обновлении очень маловероятно;

4. небезопасные пакеты потенциально могут повлиять на стабильность системы;

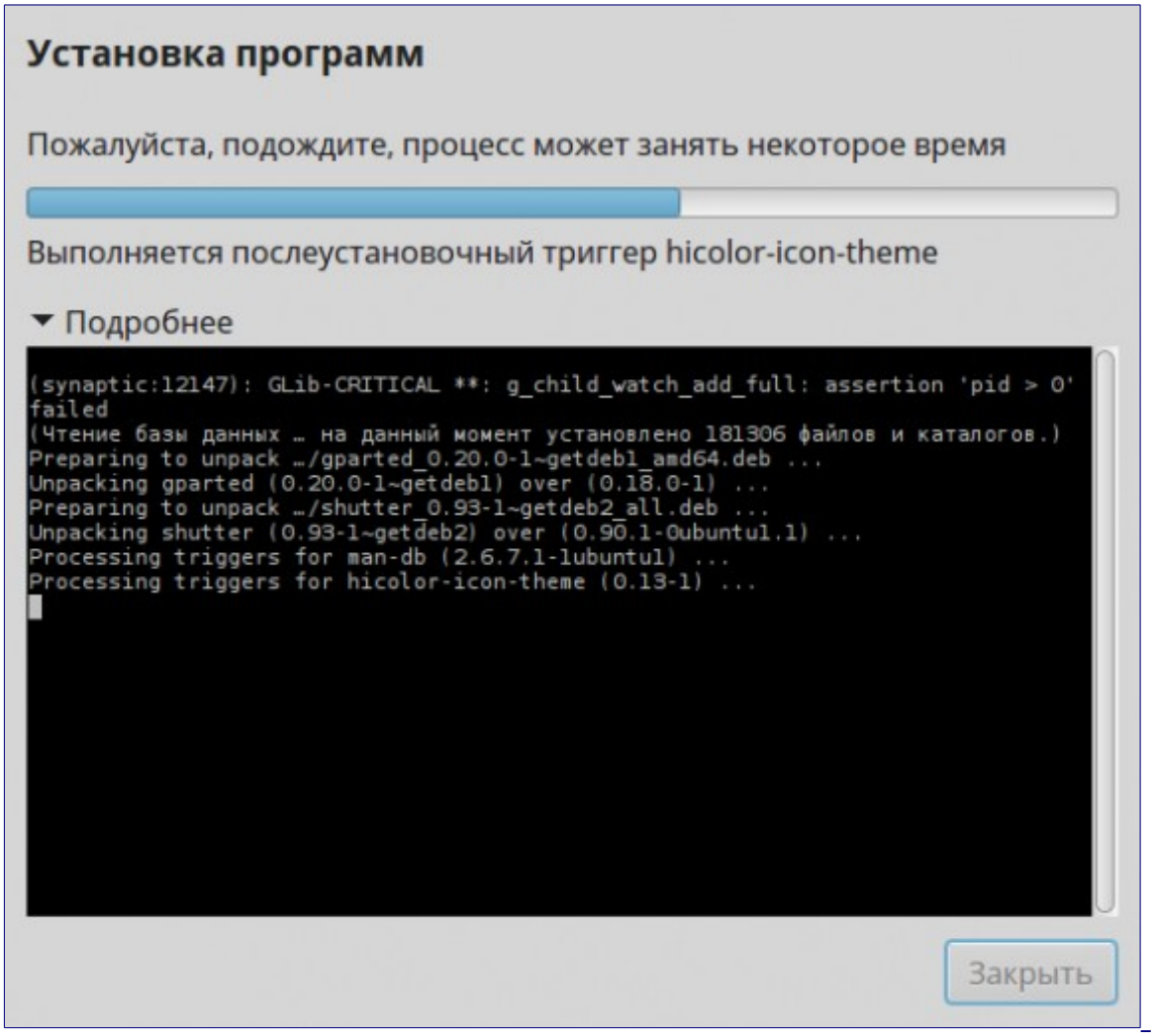
5. опасные пакеты при некоторых условиях могут привести к нестабильности системы.

Забегая вперёд, приведу скриншот окна настройки параметров Менеджера обновлений, наглядно иллюстрирующий сказанное:

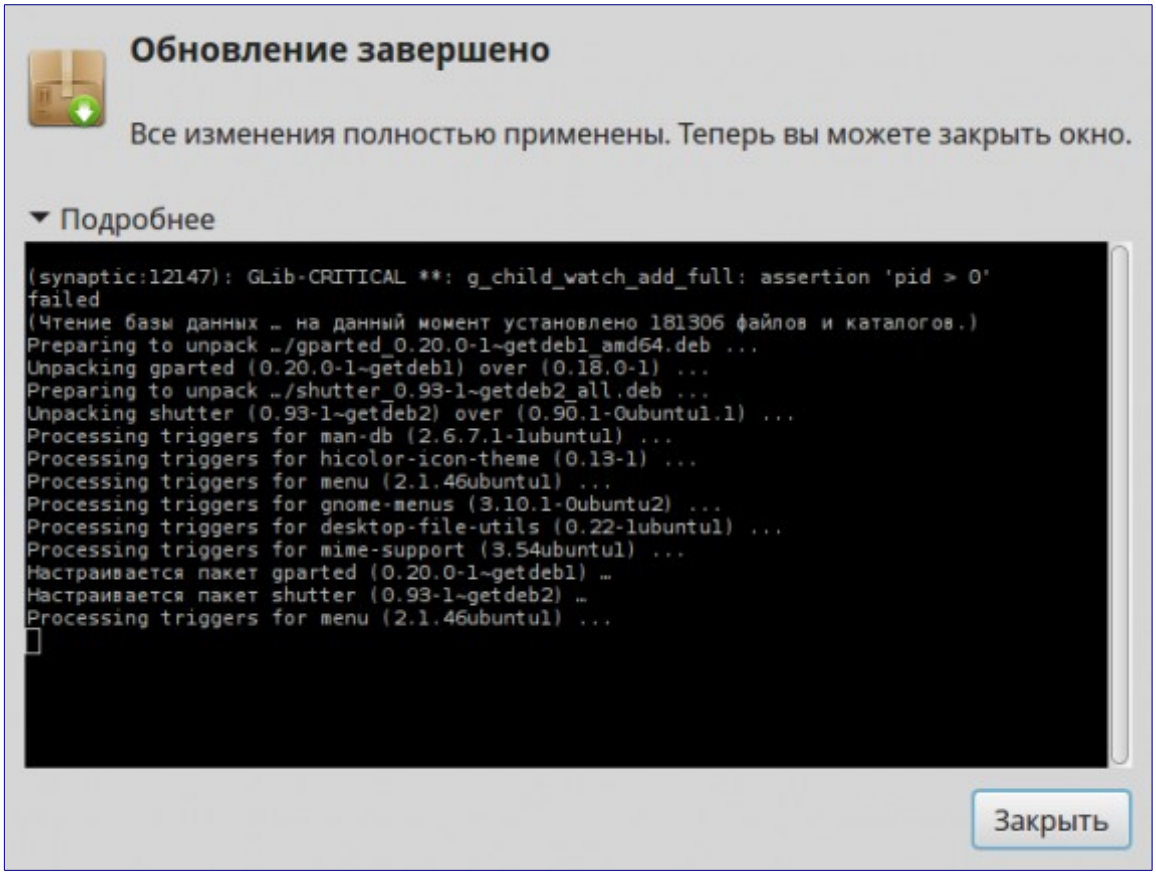


По умолчанию для обновления (третья колонка) отмечаются пакеты уровней 1-3. Для пакетов уровней 4-5 это нужно сделать принудительно. Если оно, конечно, нужно. Разработчики Mint считают, что решение об обновлении таких ключевых пакетов, как ядро, главная системная библиотека glibc и так далее, применитель должен принимать осознанно. Характерно, что режима автоматического обновления в Mint не предусмотрено как класса.

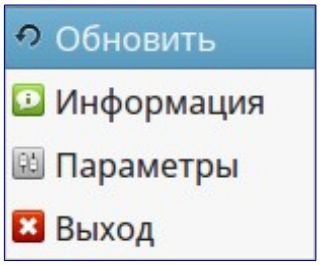
Само по себе обновление выполняется нажатием экранной кнопки Установить обновления и начинается после ввода пользовательского пароля. Развернув пункт Show individual files, можно наблюдать за ходом процесса в деталях (если, конечно, больше заняться нечем):



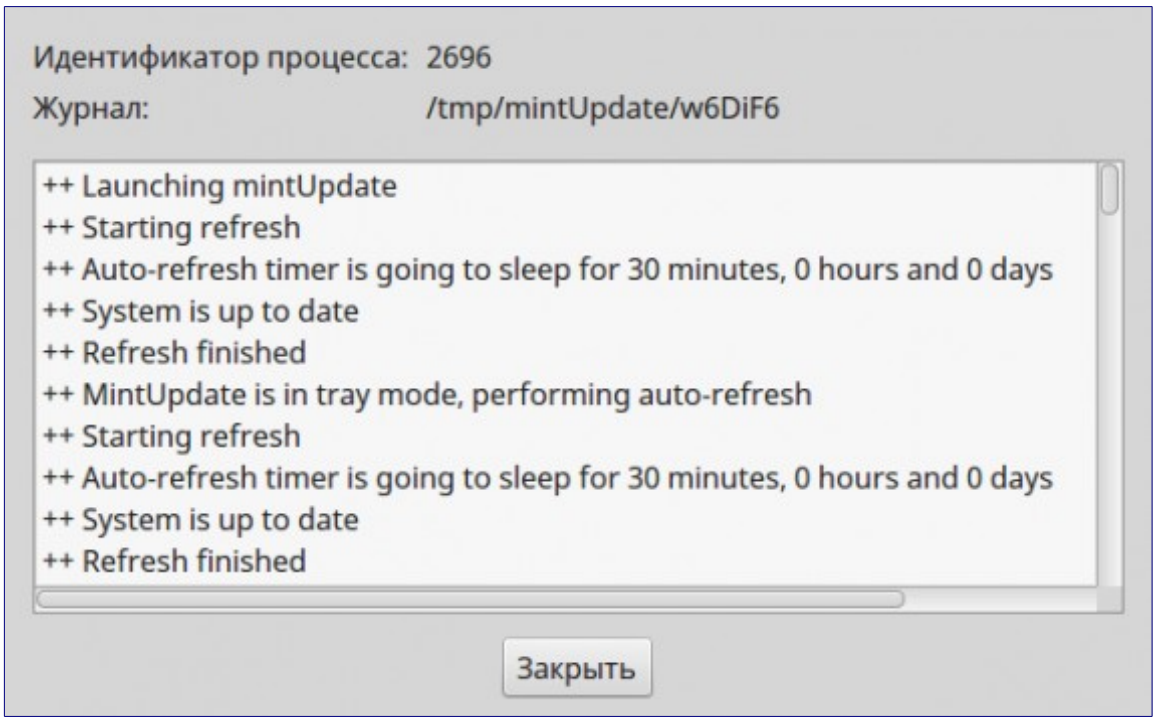
По завершении процесса окно обновлений предлагается закрыть:



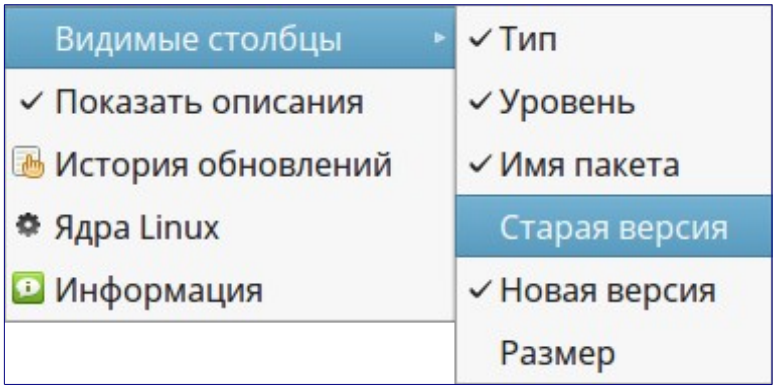
Как я уже говорил, Mint не предлагает автоматического обновления пакетов — Менеджер обновлений нужно запустить руками, или описанным выше способом, или из контекстного меню по щелчку правой кнопкой мыши на его пиктограмме в системной трее:



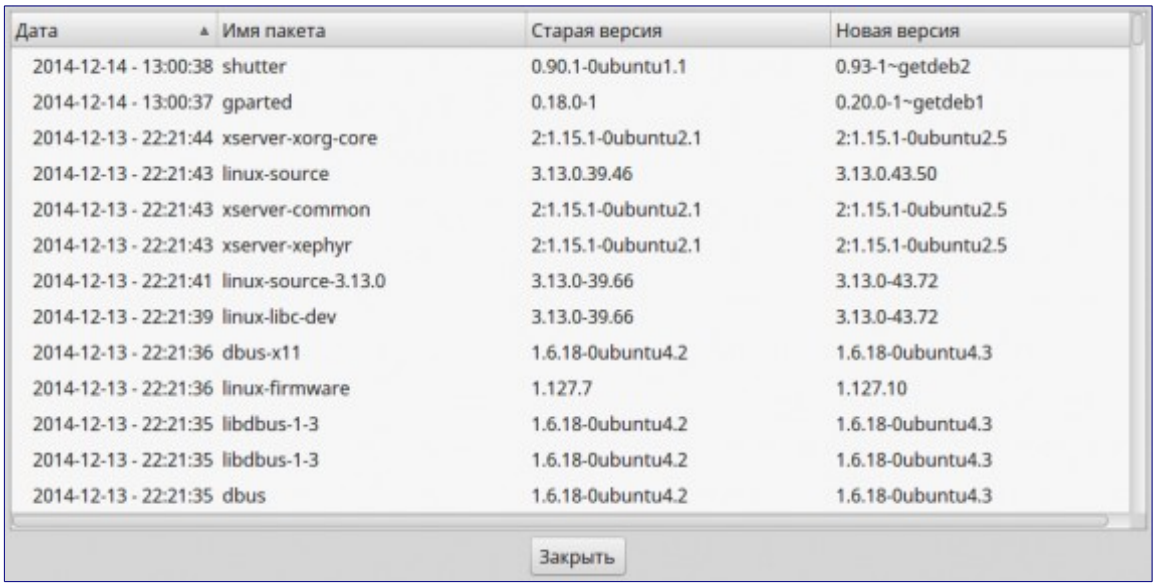
Из него же можно получить доступ к настройкам mintupdate (пункт Параметры), о которых я скажу чуть позже, и к журналу обновлений (пункт Информация):



Меню менеджера обновлений не дублирует кнопки на его панели инструментов. Через пункт меню Файл можно выйти из программы. Пункт Правка содержит подпункты Параметры — это настройка политики обновлений, до чего я скоро доберусь, и Источники приложений — это вызов того самого software-sources, о котором шла речь в предыдущем разделе. А в пункте Вид можно, во-первых, включить или отключить вывод каких-то колонок:



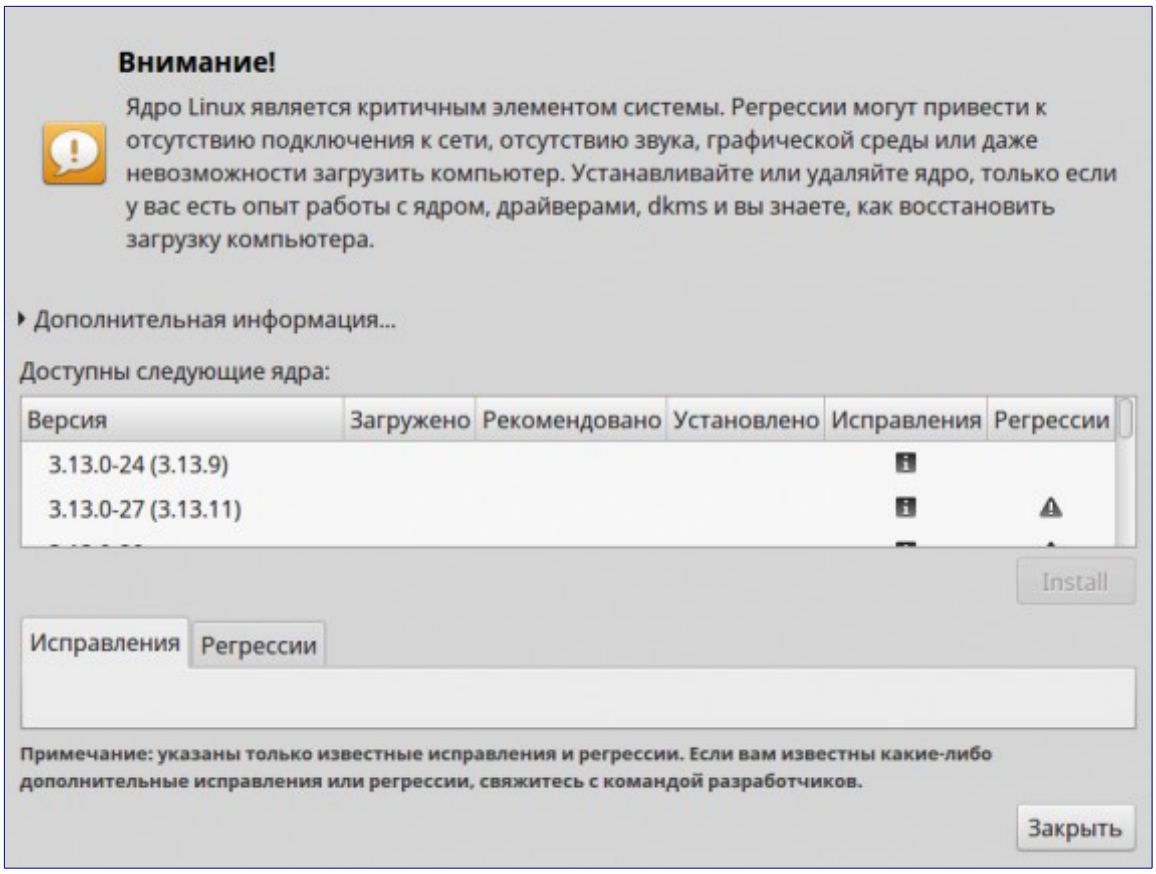
Во-вторых, можно просмотреть историю обновлений:



Дата	Имя пакета	Старая версия	Новая версия
2014-12-14 - 13:00:38	shutter	0.90.1-0ubuntu1.1	0.93-1~getdeb2
2014-12-14 - 13:00:37	gparted	0.18.0-1	0.20.0-1~getdeb1
2014-12-13 - 22:21:44	xserver-xorg-core	2:1.15.1-0ubuntu2.1	2:1.15.1-0ubuntu2.5
2014-12-13 - 22:21:43	linux-source	3.13.0.39.46	3.13.0.43.50
2014-12-13 - 22:21:43	xserver-common	2:1.15.1-0ubuntu2.1	2:1.15.1-0ubuntu2.5
2014-12-13 - 22:21:43	xserver-xephyr	2:1.15.1-0ubuntu2.1	2:1.15.1-0ubuntu2.5
2014-12-13 - 22:21:41	linux-source-3.13.0	3.13.0-39.66	3.13.0-43.72
2014-12-13 - 22:21:39	linux-libc-dev	3.13.0-39.66	3.13.0-43.72
2014-12-13 - 22:21:36	dbus-x11	1.6.18-0ubuntu4.2	1.6.18-0ubuntu4.3
2014-12-13 - 22:21:36	linux-firmware	1.127.7	1.127.10
2014-12-13 - 22:21:35	libdbus-1-3	1.6.18-0ubuntu4.2	1.6.18-0ubuntu4.3
2014-12-13 - 22:21:35	libdbus-1-3	1.6.18-0ubuntu4.2	1.6.18-0ubuntu4.3
2014-12-13 - 22:21:35	dbus	1.6.18-0ubuntu4.2	1.6.18-0ubuntu4.3

Закрыть

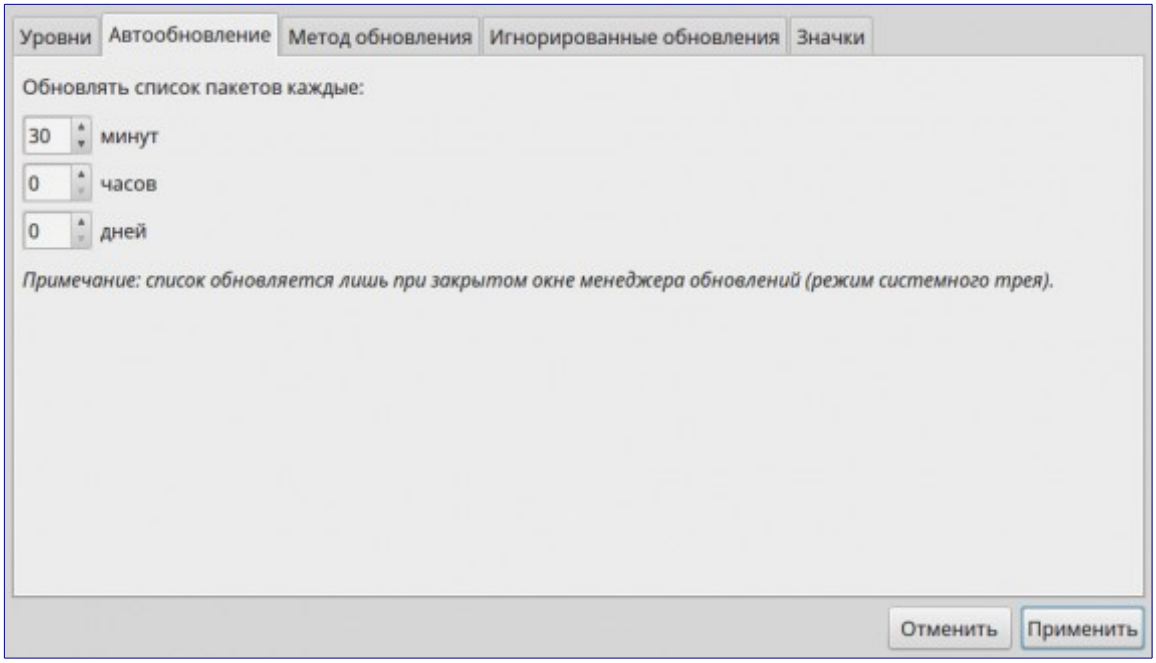
И в-третьих, можно получить информацию об установленном ядре и доступных для обновления версиях:



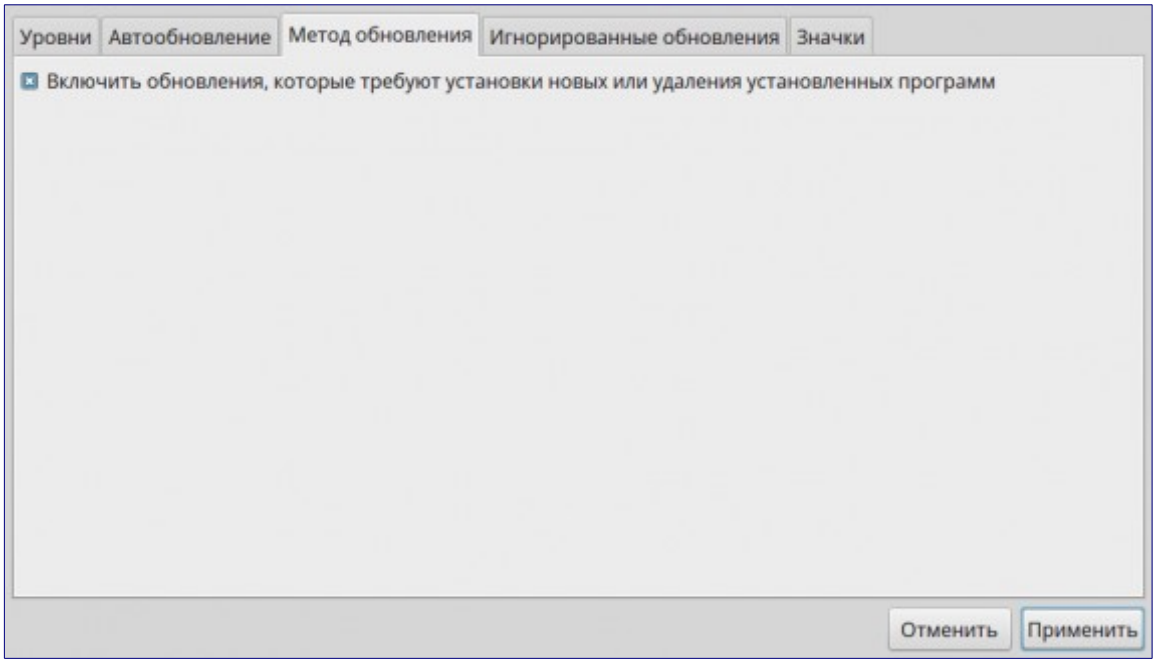
Что такое Информация — я только что говорил. Так что можно вернуться в пункт Правка, где обратиться к подпункту Параметры. Скриншот первой вкладки вызываемого при этом окна я уже приводил, когда говорил об уровнях безопасности. Осталось только добавить, что здесь для пакетов 4-го и 5-го уровней можно включить отметку к обновлению «на постоянной основе». А можно сделать это для всех пакетов, обновляемых по типу обновления безопасности (другой, той, которая от злодеев).

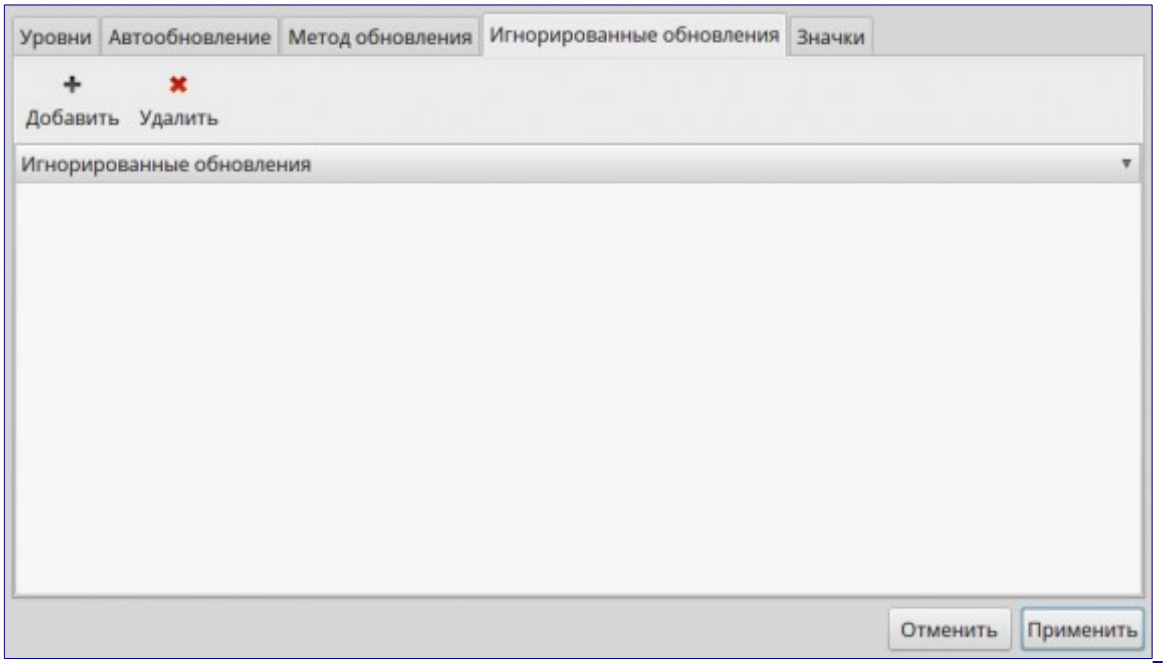
Во вкладке Автообновление задаётся время, через которое обновляется список пакетов. Подчеркну — именно их список сами по себе пакеты

обновляться не будут, если не заказать это явным образом, как было сказано выше:

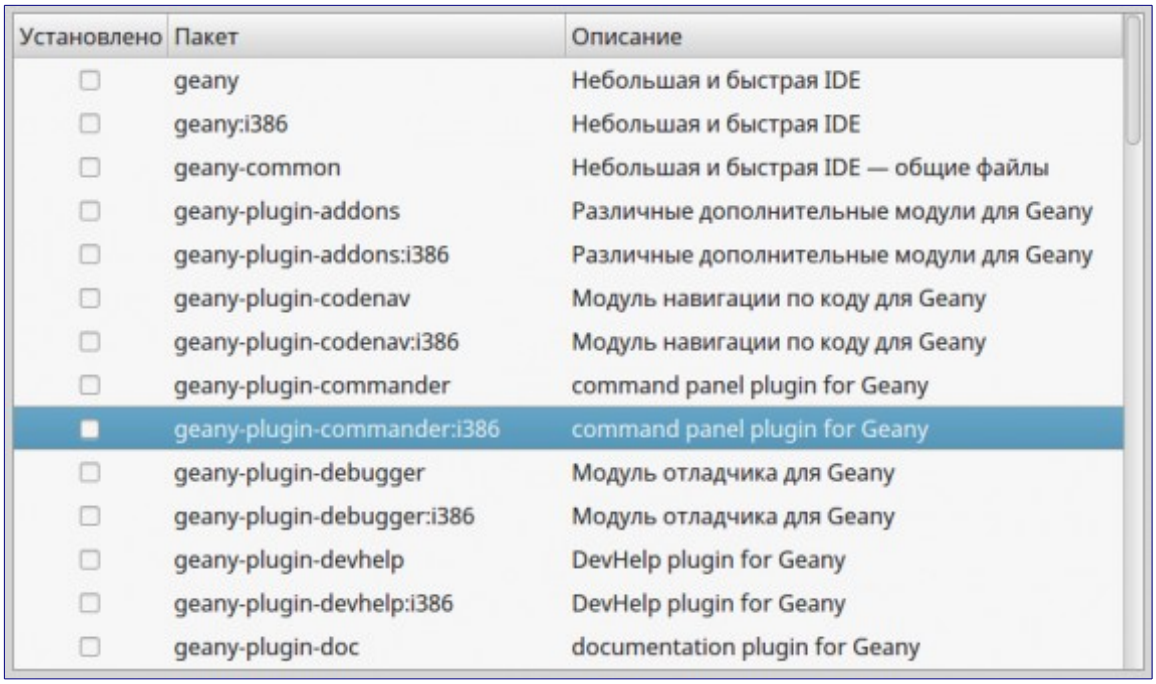


Со вкладками Метод обновления и Игнорируемые пакеты всё понятно без комментариев:





Вкладка Значки — это своего рода легенда (то есть условные обозначения): как выглядит пиктограмма Менеджера обновлений в зависимости от состояния системы и выполняемых им действий:



Как и всякие условные обозначения, каждое из представлений пиктограммы можно изменить (только нужно ли?).

На этом разговор о Менеджере обновлений считаю законченным. Следующим номером нашей программы будет рассказ о визуализаторах вывода.

Средства визуализация пакетов

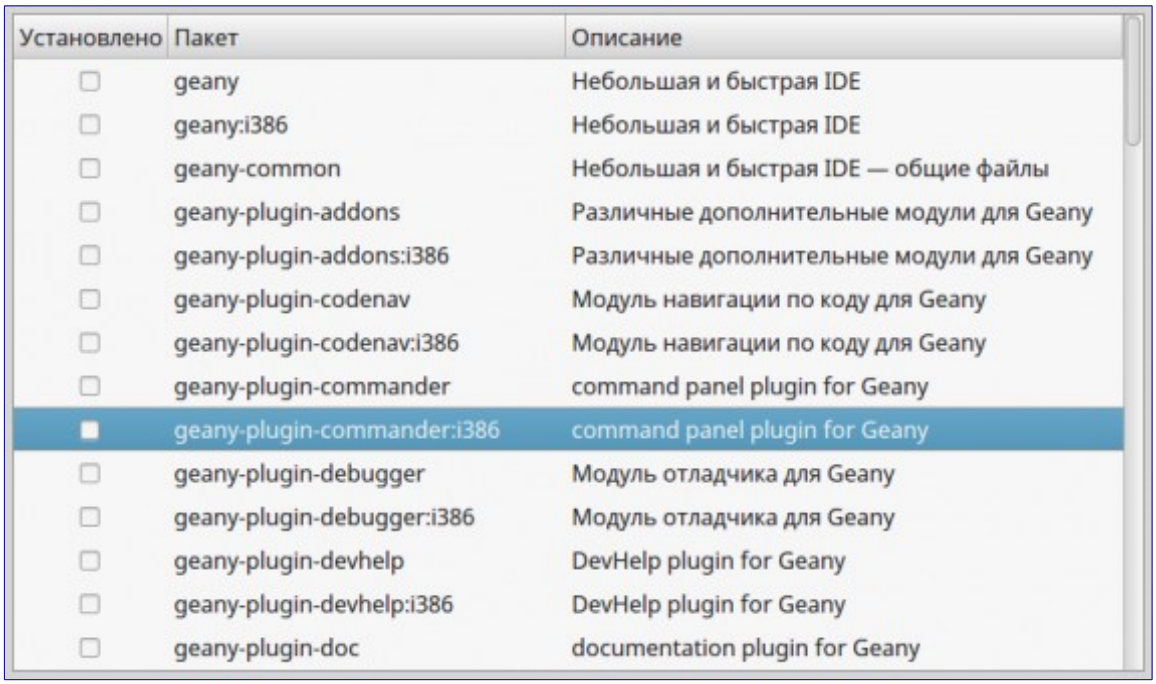
В качестве завершающего штриха в рассказе о mint-инструментах для работы с файлами скажу пару слов об утилитах mint-search-apt и mint-show-apt. Они предназначены для визуального представления результатов поиска пакетов и вывода информации о них. В сущности, это графические фронт-энды для вывода команд CLI apt search и apt show, соответственно (о которых

будет в другом очерке). Подчеркну — именно вывода: сами эти утилиты запускаются из командной строки терминала или из минитерминала.

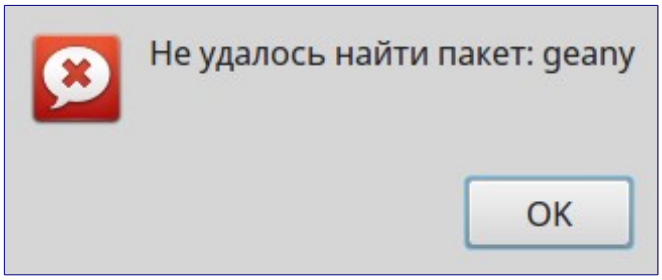
Команда mint-search-apt в качестве аргумента принимает имя пакета или его фрагмент: поиск введённой последовательности символов осуществляется только в именах пакетов. После чего результат поиска выводится во вновь открывающемся окне. Например, ответ на команду

```
$ mint-search-apt geany
```

будет таким:



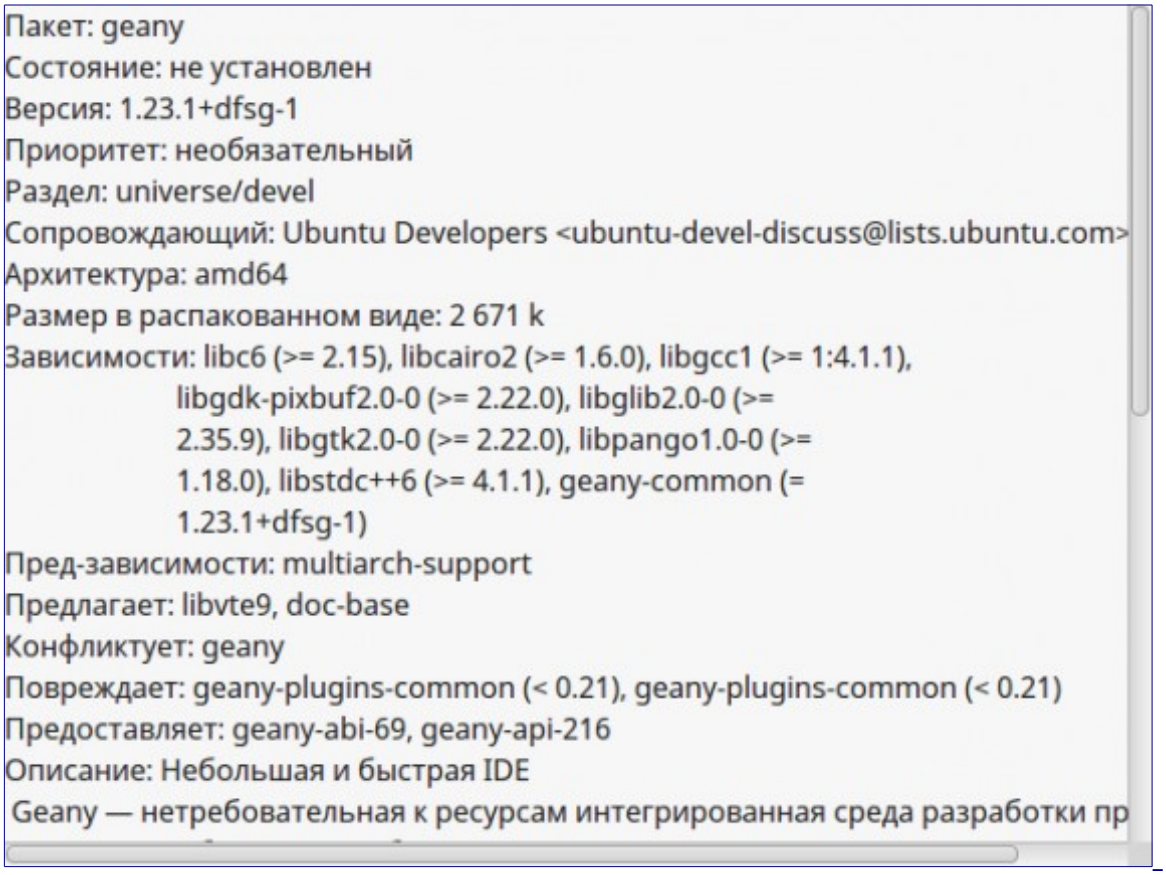
Теоретически двойной клик на имени найденного пакета должен привести к его инсталляции. Однако в реальности последует лишь сообщение об ошибке:



Утилита apt-cache show в качестве аргумента требует имени пакета, после чего в отдельном окне выводит информацию о нём. Например, после команды

```
$ mint-show-apt geany
```

оно будет выглядеть следующим образом:

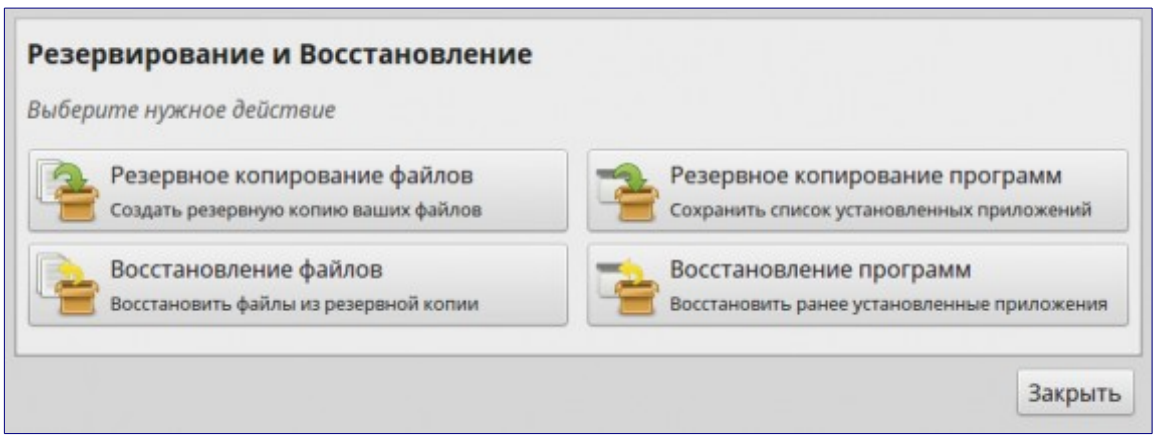


Обе рассматриваемые утилиты входят в пакет mintinstall, который был разработан задолго до появления интегрированной команды apt, о которой будет говориться в очерке про пакетный менеджмент. И в те не столь далёкие времена их применение было оправдано. Ныне же, как мне кажется, использование их большого смысла не имеет. Хотя кому-то, может, и понравится вывод найденных пакетов и информации о них в графическом окне. Почему я и уделил им немного внимания.

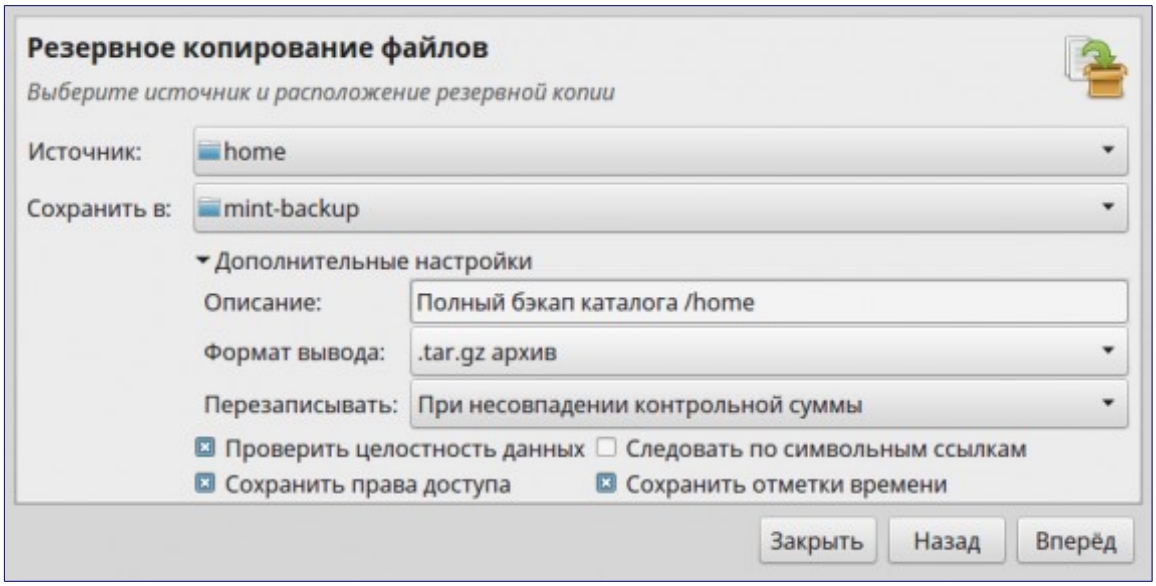
Этой заметкой завершается описание фирменных утилит Mint'a, так или иначе связанных с управлением пакетами. Следующий раздел — о резервном копировании.

Средство резервного копирования mintbackup

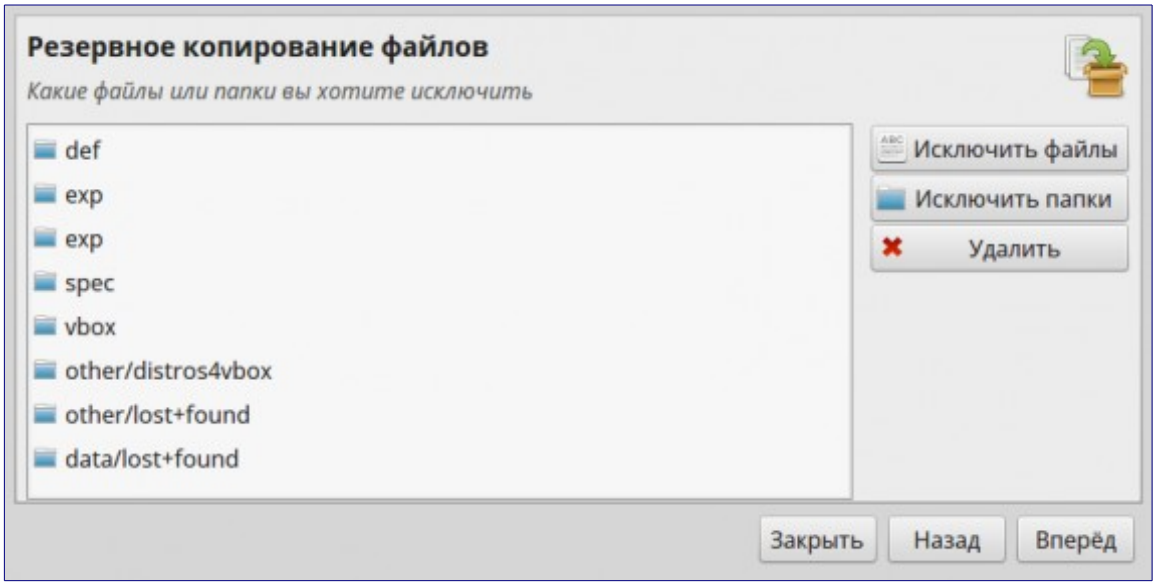
От средств работы с пакетами плавно переходим к средствам работы с файлами. А тут одно из наипервейших дел — резервное копирование. Для чего в составе фирменного инструментария Mint имеется утилита mintbackup. В разделе Администрирование главного меню она так и называется — Резервное копирование. И, после ввода пароля, предстаёт в таком вот виде:



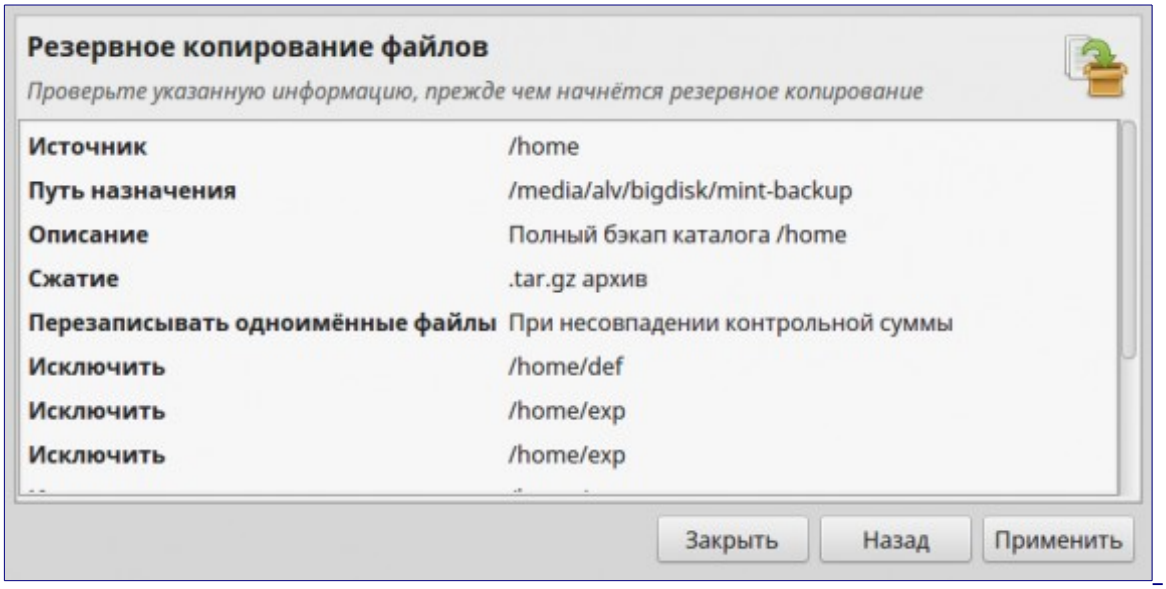
Для начала выполняется резервное копирование, для которого указываются исходный и целевой каталоги, а также дополнительные параметры — простое копирование, архив tar, tar.bz2 или tar.gz, условия перезаписи:



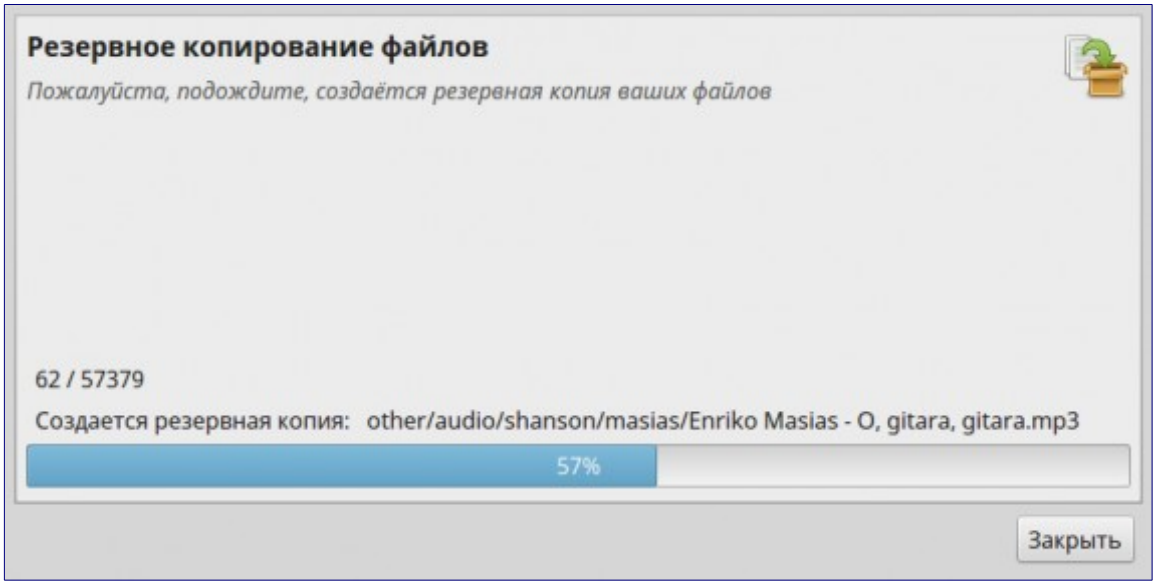
Далее определяются исключения из исходного каталога, не подлежащие архивированию (если, конечно, они нужны):



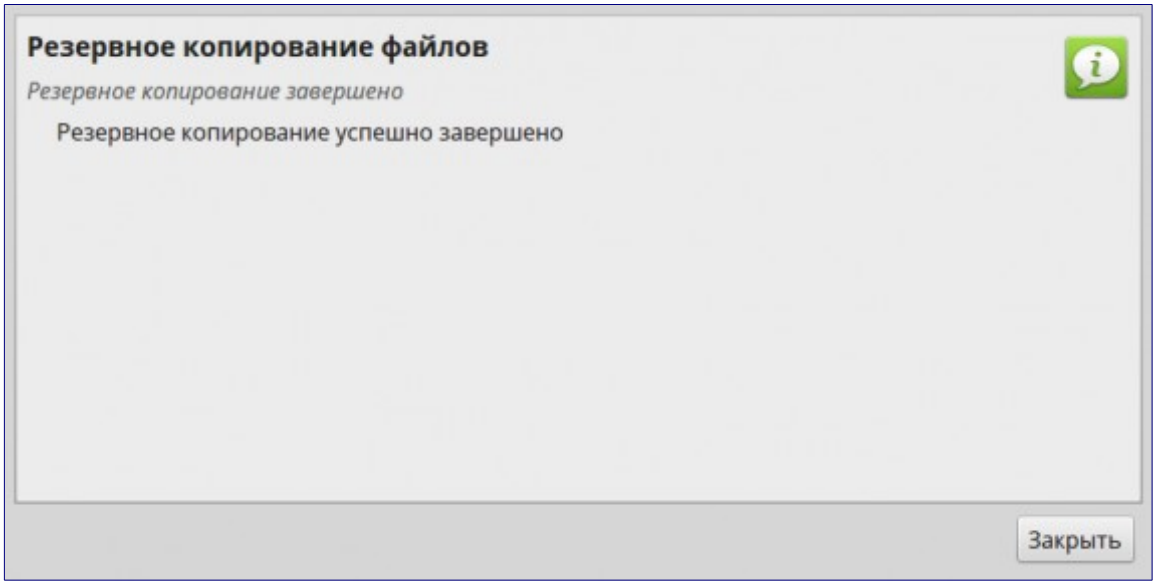
После этого выводится результирующая информация о будущем архиве:



А затем нажатие кнопки Применить вызывает начало процесса:



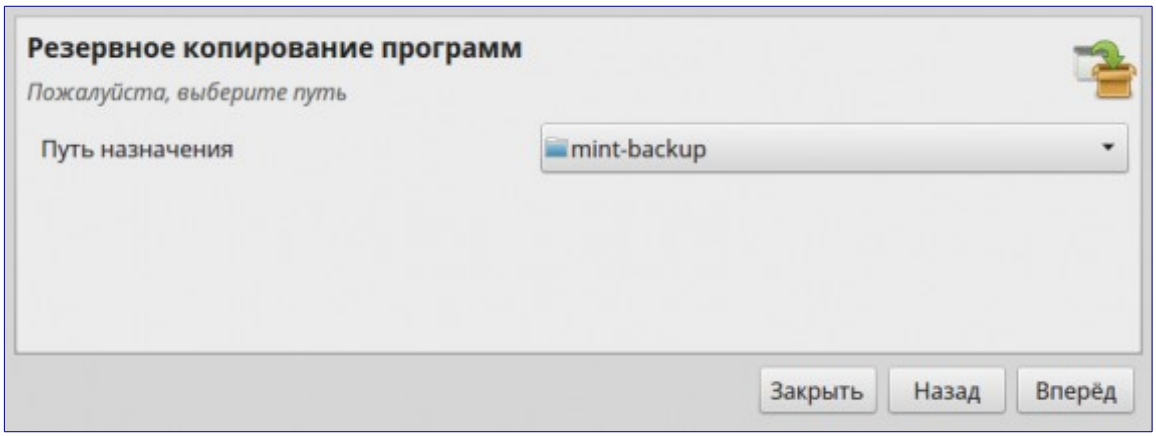
Завершение которого знаменуется таким сообщением:



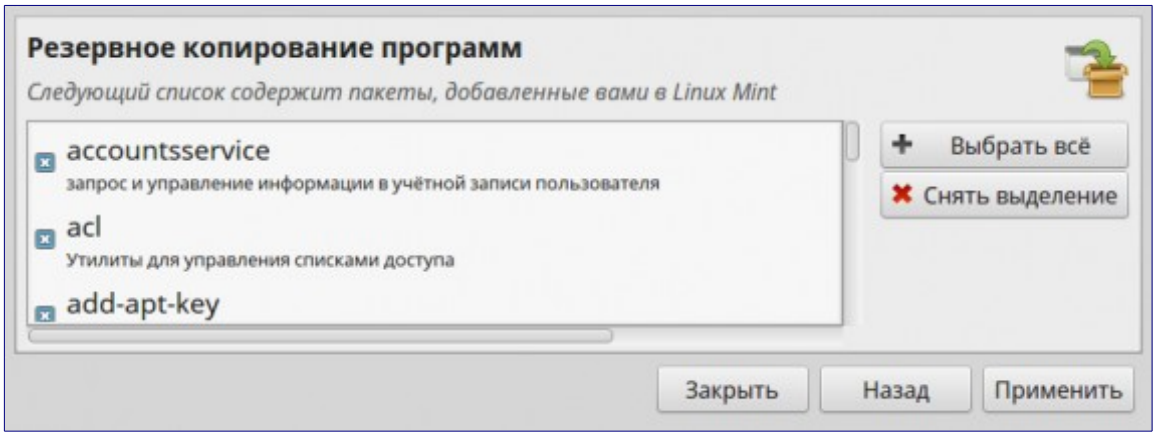
Здесь следует нажать кнопку Закрыть — правда, это приведёт и к закрытию программы, но выбора всё равно нет.

То есть всё просто до банальности — и ничего такого, чего нельзя было бы сделать с помощью утилиты tar и её опций. Но всё это представлено в наглядной форме, избавляющей от необходимости ломать голову над последними. Иными словами, в этой своей части утилита mintbackup заслуживает рекомендации к применению.

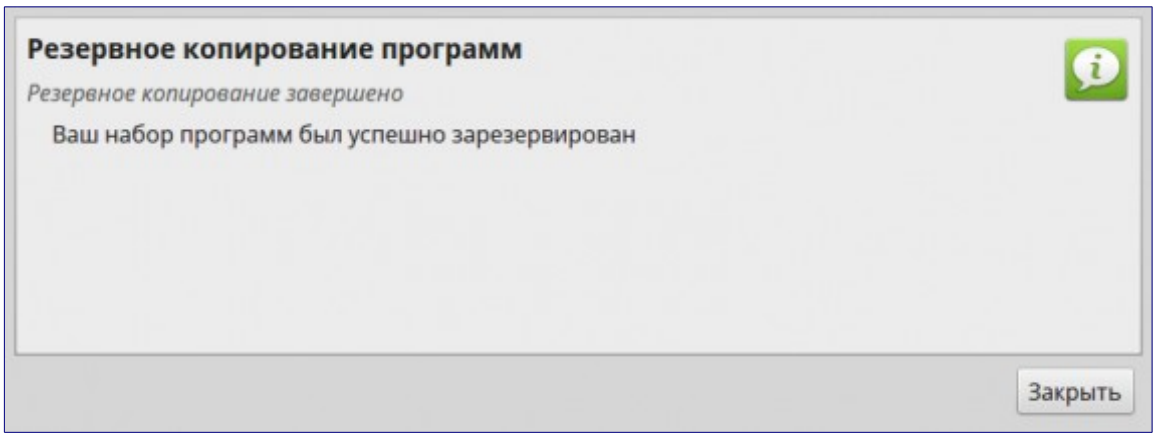
Не менее проста и полезна утилита mintbackup во второй своей части, сохраняющей список установленных пакетов. Здесь всего-то и требуется, что указать целевой каталог:



Затем, при желании, просмотреть список пакетов, установленных в системе:



После чего нажать кнопку Применить — и дождаться появления сообщения об успешном завершении процесса:



После чего в целевом каталоге обнаруживается файл вида software_selection_alv-desk@2014-12-14-1850-package.list. Каковой является

самым обычным текстовым файлом, содержащим список установленных пакетов:

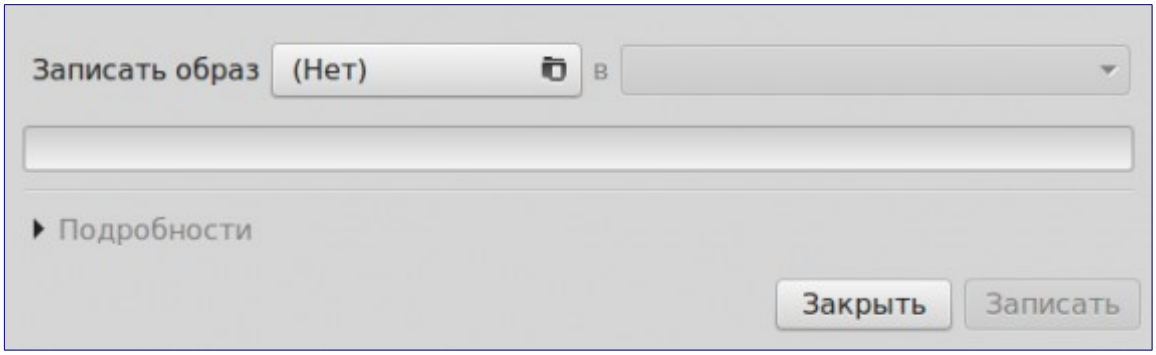
```
accountsservice install
acl install
add-apt-key install
adobe-flashplugin install
aisleriot install
...
```

Пакеты по этому списку могут быть установлены на любой другой машине. Так что и от mintbackup нет никакого вреда, окромя пользы.

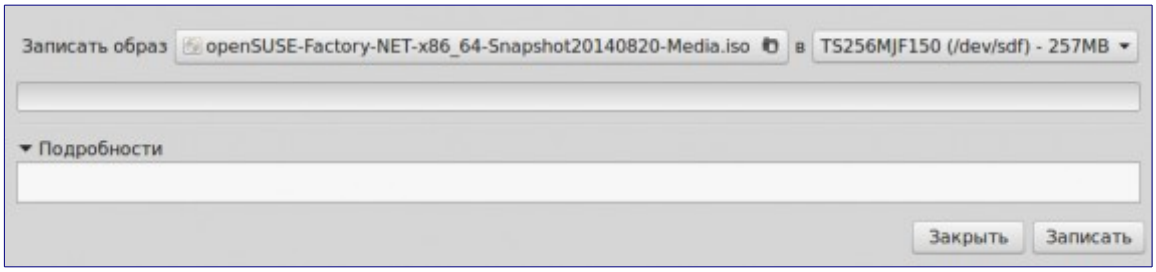
Программа записи USB mintstick

Как известно, оптические приводы постепенно отмирают. И на смену им приходят USB flash и SD-карты. Единственная сфера, где до некоторого времени оптические накопители были не всегда заменимы — это установка системы на чистую машину. Однако ныне все современные дистрибутивы Linux'а или BSD-системы распространяются в виде так называемых гибридных образов, допускающих их запись на твердотельные носители. Что повлекло за собой появление большого числа программ, призванных выполнить эту процедуру.

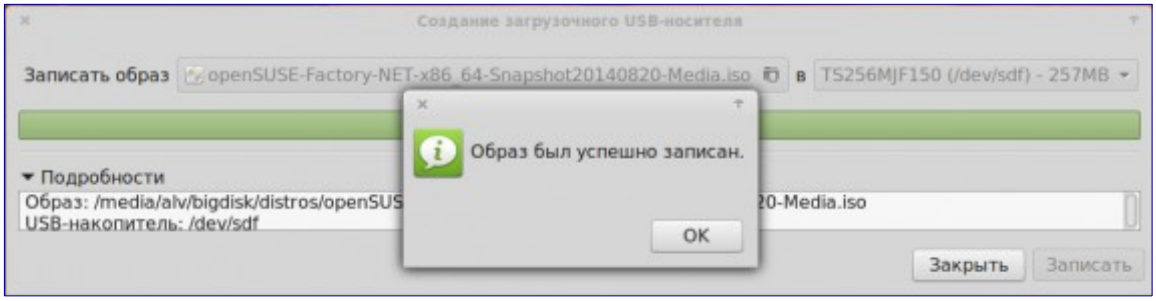
Имеется такая утилита и в составе фирменного инвентаря Mint'а. Это mintstick, которая в главном меню находится в разделе Стандартные под именем Создание загрузочного USB-носителя. И после запуска предстаёт перед глазами применителя в таком виде:



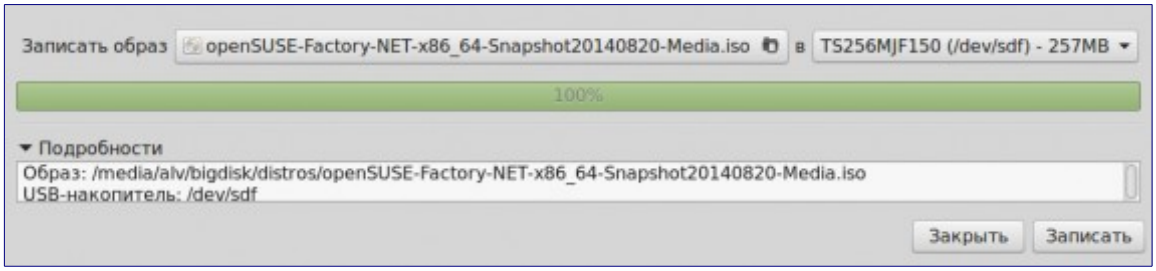
Дальнейшие действия очевидны: надо выбрать записываемый образ и указать, куда он должен быть записан (воткнутая флешка или SD-карта предлагается по умолчанию):



После этого потребуется ввести пароль и подождать завершения процесса, о чем будет сообщено дополнительно:



В поле Подробности будут указаны имена файла образа и целевого устройства:



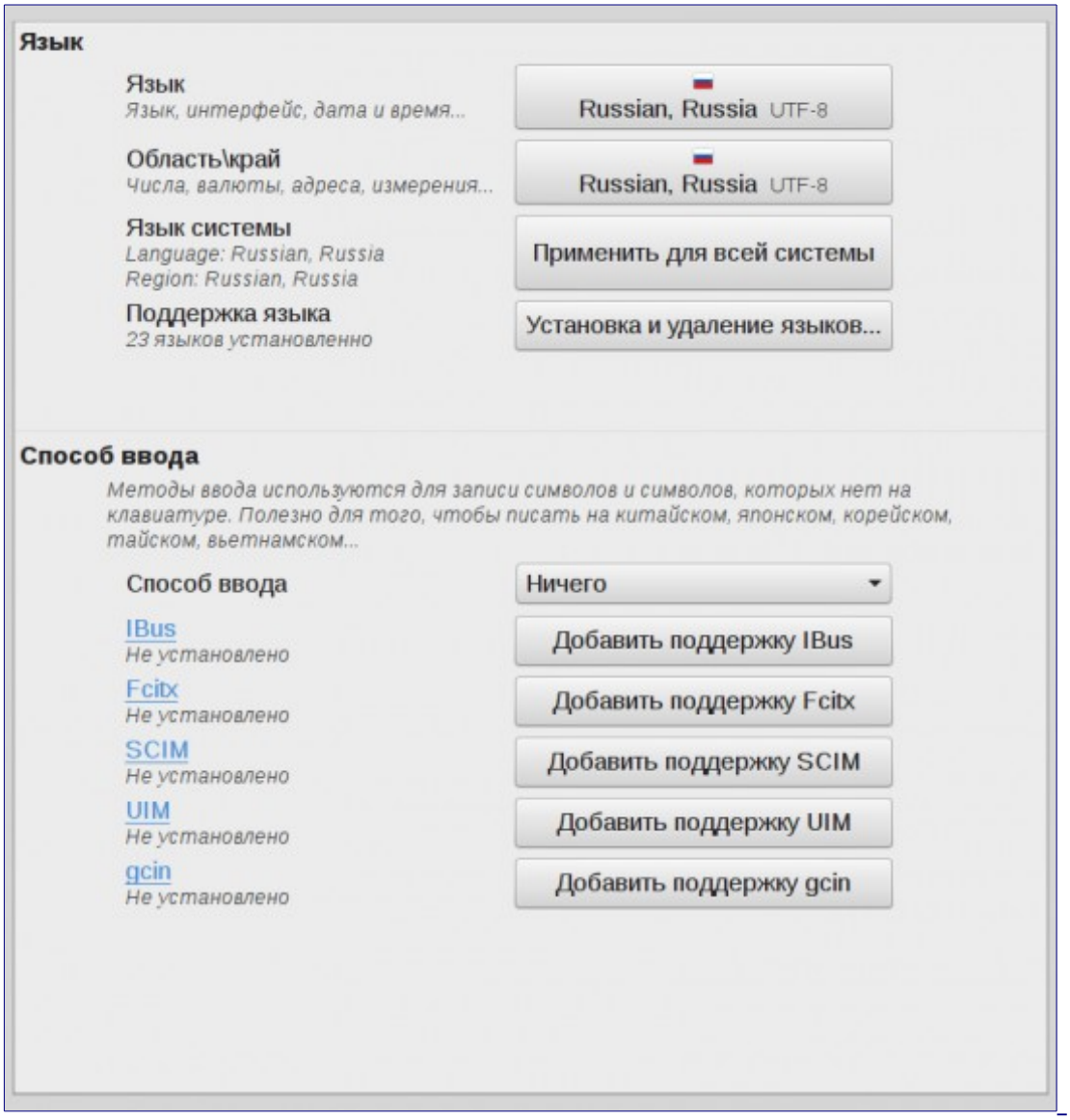
Всё. Можно либо повторить процедуру для другого образа или устройства, либо закрыть программу.

Хотелось бы ещё проще? Не получится — проще уже некуда.

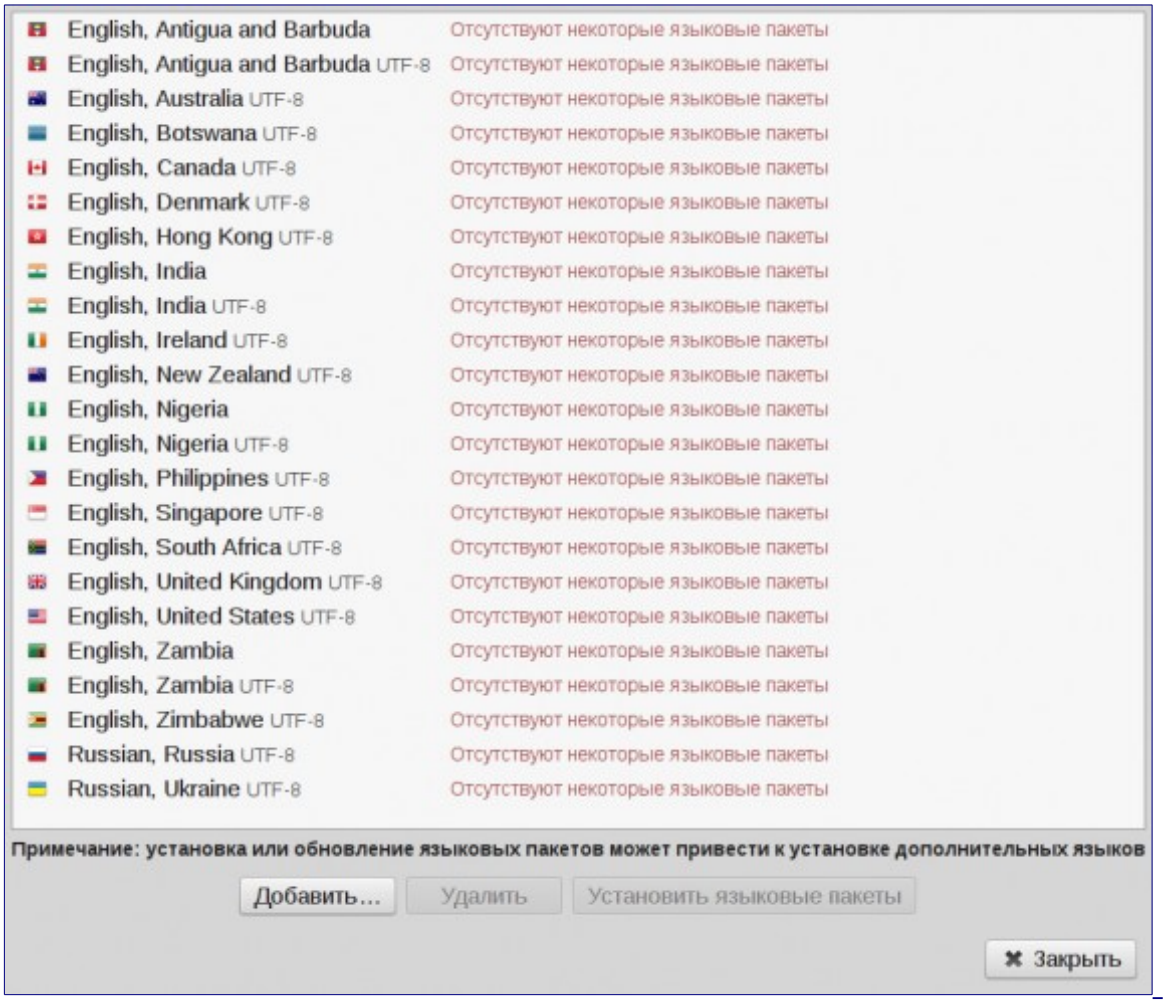
Языковые настройки — mintlocale

В очерке о настройках Cinnamon упоминался модуль Языки в секции Параметры его Системных настроек. И, поскольку он, собственно, принадлежит к семейству фирменных утилит Mint (под именем mintlocale), было обещано рассказать о нём в соответствующее время. Время это наступило.

Модуль Языки можно запустить как из панели Системных настроек, так и из секции Параметры главного меню. Он выполняет двоякую функцию. Во-первых, здесь можно изменить собственно язык интерфейса и прочие параметры, объединяемые понятием locale (формат даты, представление чисел, единицы измерения etc.). При выборе русского в качестве языка инсталляции всё это будет русским Российским (интерфейс, разумеется, русифицируется в меру испорченности перевода):

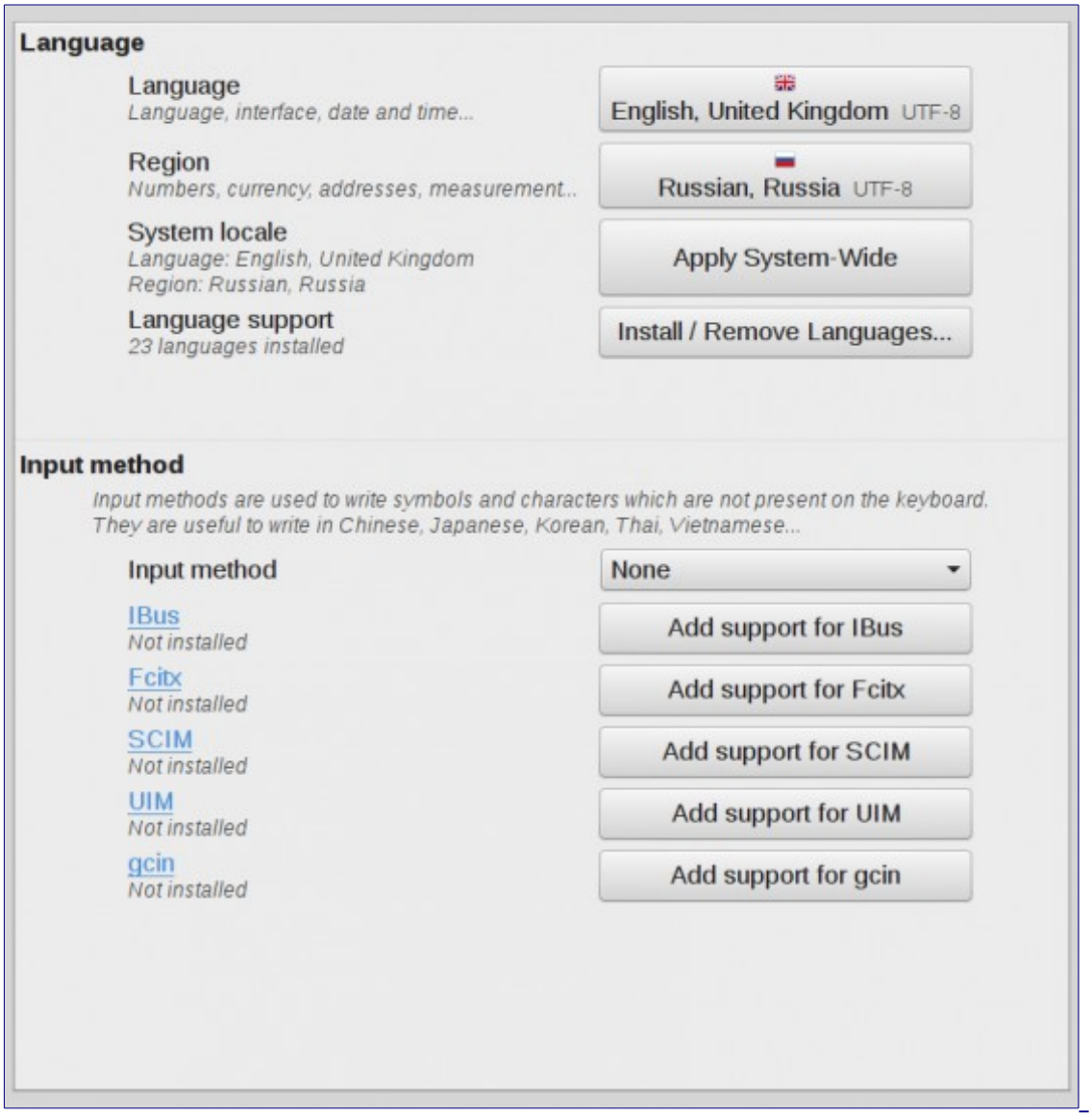


При желании или необходимости можно установить и более иные языки — список доступных превышает два десятка:



Главная особенность нового языкового модуля (он появился в Mint 17.1) — разделение групп параметров Language и Region (в русском переводе — Язык и Область/Край, соответственно). Первая определяет переменные LANG и LC_TIME, то есть собственно язык интерфейса и местное время. Группа же параметров Region задаёт значения всех остальных локально-зависимых переменных — LC_NUMERIC, LC_MONETARY, LC_PAPER, LC_NAME, LC_ADDRESS, LC_TELEPHONE, LC_MEASUREMENT и LC_IDENTIFICATION (представление чисел, единица бабла, формат бумаги, и так далее).

Разнесение переменных по группам может показаться спорным, как и его аргументация Клементом (типа специально для отъезжающих за границу). Однако сама по себе идея отделения языка интерфейса от прочего локально-зависимого хозяйства весьма здрава, особенно если язык этот — смесь нижегородского с оксфордским. Что же до местного времени... С каждым очередным самым последним переводом часов оно теряет всё больше смысла. Так что не пора ли жить по Гринвичу? Вполне реализуемо, как можно видеть на следующем скриншоте:



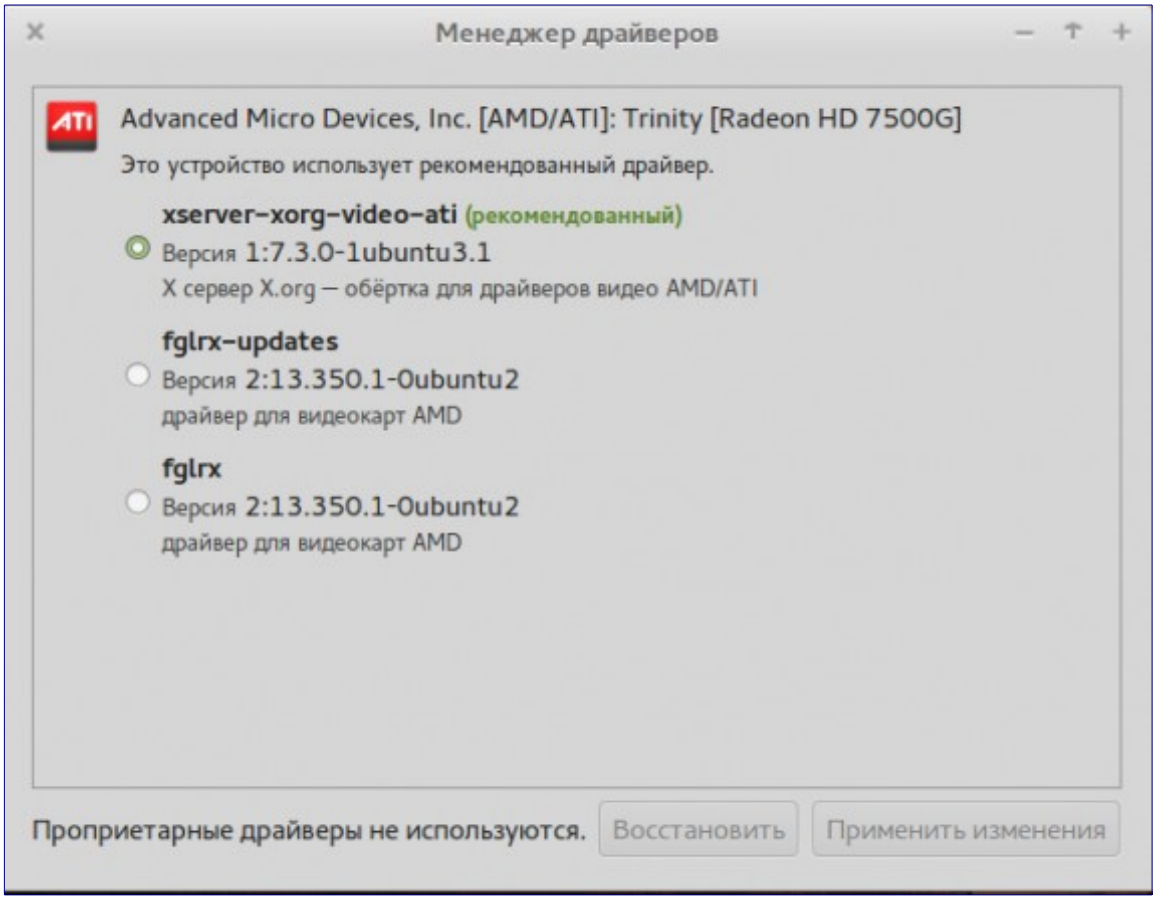
Вторая функция mintlocale — определение так называемого метода ввода. Поддержка их также впервые появилась в Mint 17.1. Методы ввода (Input Method) — это системы обеспечения ввода символов, отсутствующих на клавиатуре от слова «вааще». Например, китайских иероглифов и символов всех генетически связанных с ними систем письма. И потому, конечно, жизненно необходимы жителям соответствующих стран.

Однако мы, индоевропейцы, семиты, тюрки и многие другие, выступая как единый советский народ, не испытываем в них ни малейшей потребности. И потому то, что реально ни один метод ввода не включён (разработчики объясняют это недостатком сил), огорчать нас не должно. Напротив, скорее радовать. Ибо попытки майнтайнеров некоторых дистрибутивов включать поддержку какого-либо из этих методов (мне приходилось сталкивать с IBus), да ещё и по умолчанию, нам, применителям, не давало ничего, кроме лишних проблем.

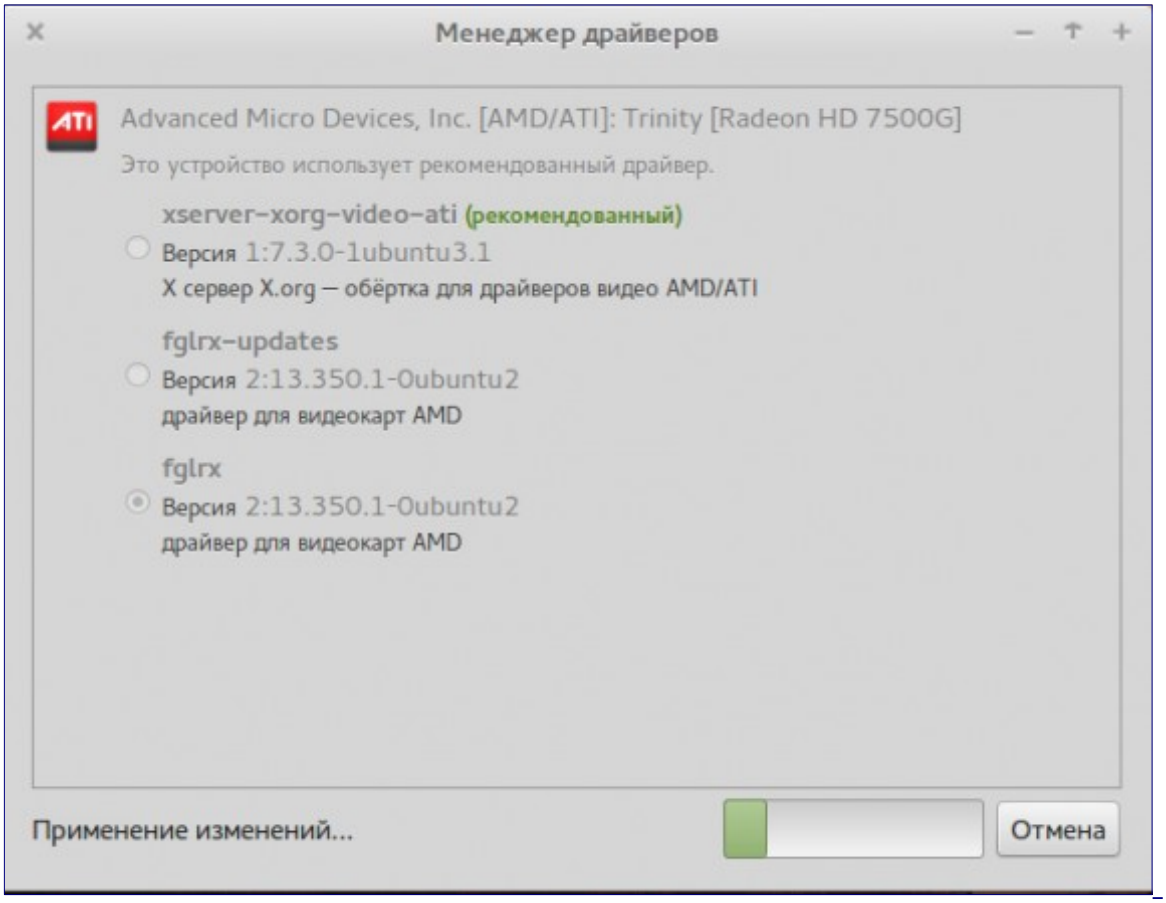
Менеджер драйверов и интегрированное видео AMD

Последний из фирменных инструментов Mint, о котором я хотел сказать пару слов — Менеджер драйверов, он же mintdrivers, предназначенный для управления проприетарными драйверами, например, для видеокарт. Правда, мой личный опыт общения с ним оказался неудачным, но это связано с моим «железом», а не собственно с программой. Так что просто опишу последовательность действий, проиллюстрировав её скриншотами.

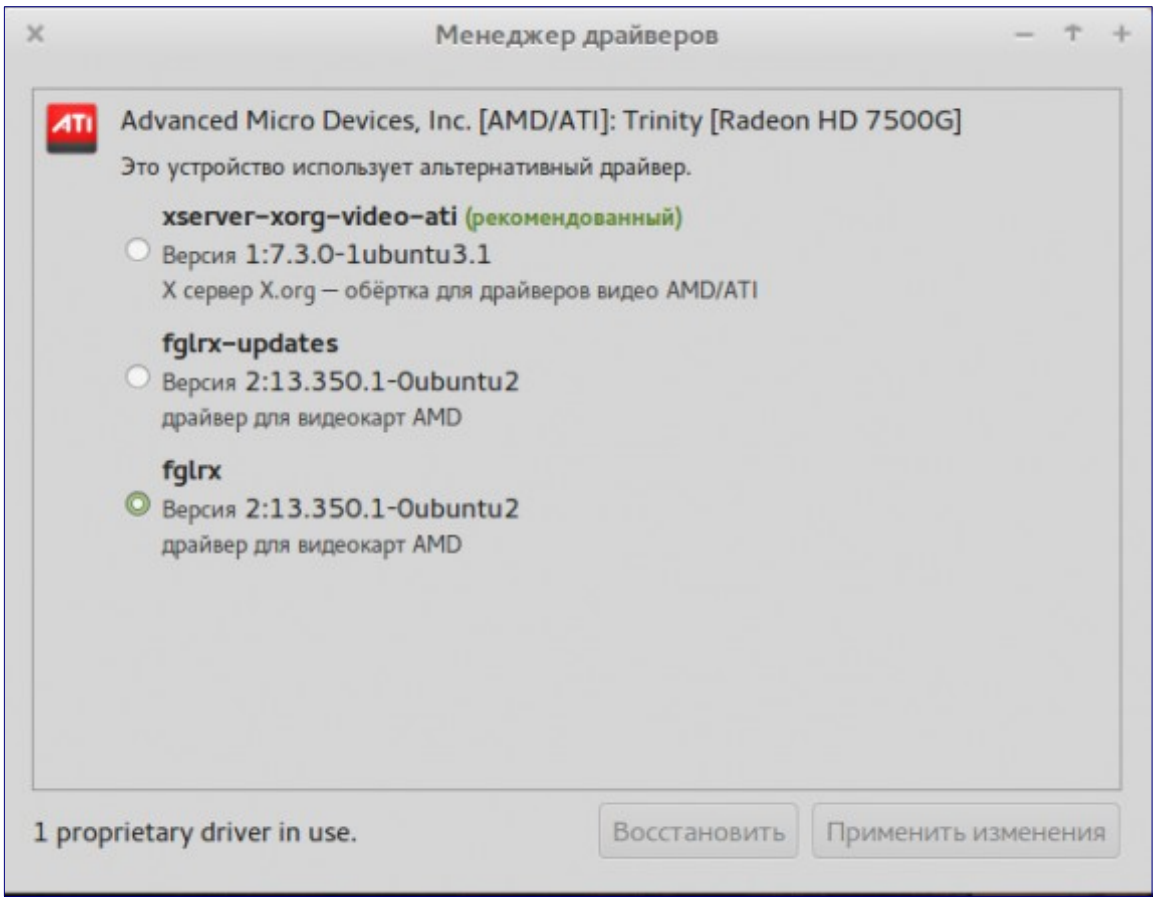
Запуск утилиты происходит из секции Администрирование главного меню, где она носит имя Менеджер драйверов. После чего на экране появляется (в моём случае с интегрированным процессорным видео Radeon HD 7500G) такая картинка:



Представляется очевидным, что для установки проприетарного драйвера достаточно вместо первой строки отметить третью и нажать кнопку Применить изменения. Только я это сделал — как процесс пошёл:



Шёл процесс довольно медленно, так как кроме собственно драйвера fglrx в ходе его устанавливались lib32gcc1, dkms и ещё куча зависимостей, а также регенерация /boot/initrd.img. По завершении всего этого исходная картинка приняла следующий вид:



Заодно был создан и файл /etc/X11/xorg.conf с описанием конфигурации

видеосистемы. Что в моём случае, правда, не помогло: после рестарта машина отказалась загружаться, выдав чёрный экран без возможности переключения в текстовую консоль и реакции на комбинацию из трёх пальцев. Пришлось перезагружаться в recovery mode и заниматься ликвидацией этого безобразия — но это совсем другая история.

Впрочем, проделанный опыт не был совсем уж бесполезным. Оказалось, что теоретически установка проприетарных драйверов через штатный Менеджер драйверов действительно очень проста, хотя в дальнейшем не исключены осложнения — но они, повторяю, скорее всего связаны с особенностями «железа».

На этом я пока поставлю точку в описании фирменного инструментария дистрибутива Mint. За бортом остались ещё несколько утилит, повода обращаться к которым у меня не было, и потому ничего сказать про них я не могу. В их числе:

- mintWifi — средство настройки соответствующего соединения; но у меня на Ноутбучке WiFi волшебным образом заработал сам собой, без всяких настроек;

- mintupload-manager — средство управления закачками на сервера, применения которому я не нашёл;

- mintnanny — блокировщик доменов; без надобности, ибо я не депутат госдумы, чтобы мне везде мерещилась банда педофилов с ихней порнографией.

Что же до утилит mint-make, mint-batch-make, mint-compress, mint-decompress, о назначении которых можно догадаться по их именам, то к ним я обращусь, когда и если (если и когда) в этом возникнет практическая необходимость.

Прочие настройки

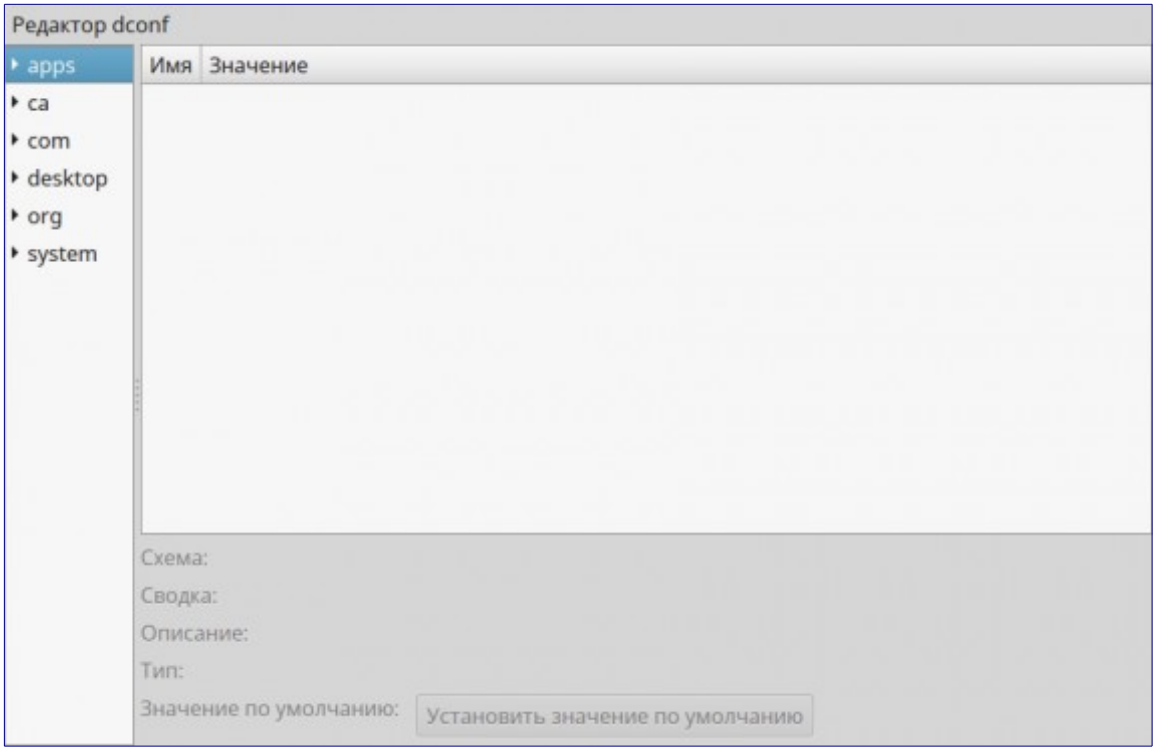
В предыдущих очерках говорилось о штатных инструментах для настройки дистрибутива Mint и его среды Cinnamon. Чтобы закончить с этой тему, скажу несколько слов об инструментах конфигурирования не то чтобы нештатных, но непосредственно ни к дистрибутиву, ни к среде не привязанных. И к которым приходится обращаться не так уж часто.

Редактор dconf

Редактор dconf появился в GNOME 3 и ныне применяется для низкоуровнего конфигурирования, кроме родительсккой среды, также в Unity, MATE и Cinnamon. Он позволяет напрямую редактировать всю базу конфигурации, как общесистемную, так и штатных пользовательских приложений. И, хотя делает это и не самым простым способом, но даёт доступ к тем настройкам, которые разработчики сред посчитали недостойными внимания конечных пользователей. Или, скорее, решили, что народу они не нужны.

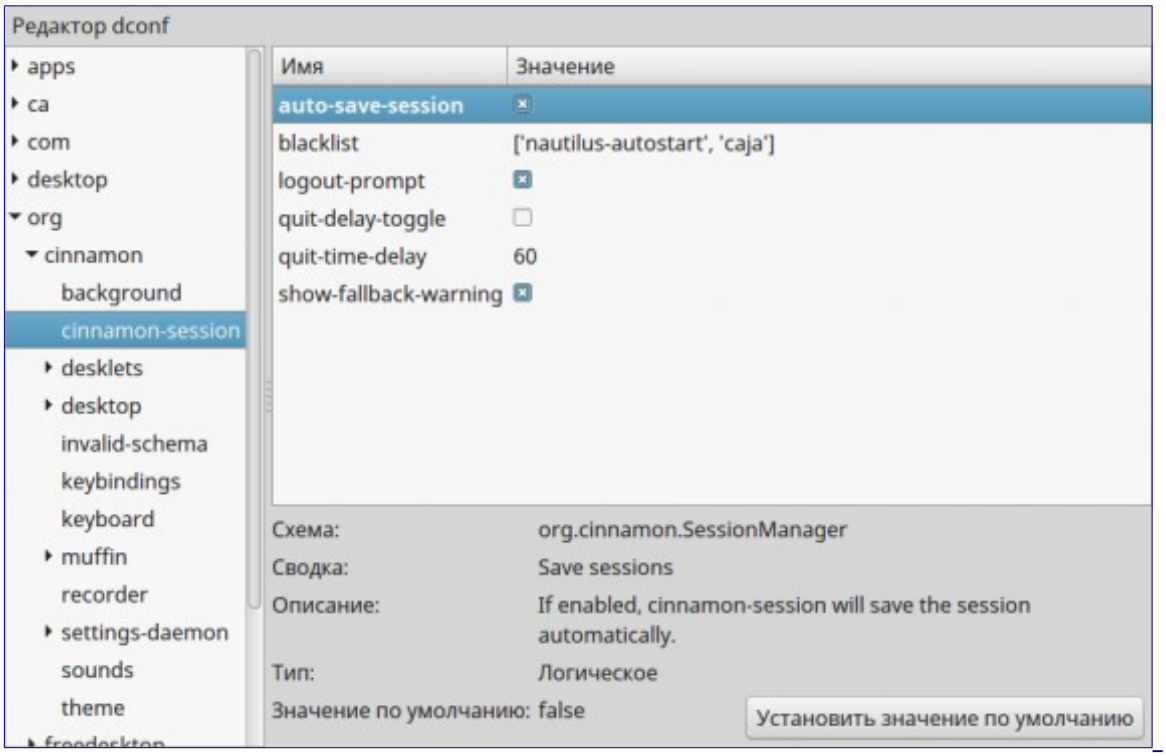
Правда, в Cinnamon таких замаскированных настроек очень мало. И если в Unity обращаться к Редактору dconf приходится постоянно, а в MATE — периодически, то в нашей среде у меня поводы для запуска этой утилиты возникали буквально считанные разы.

Запускается эта утилита, кстати, через пункт Редактор dconf в секции Администрирование главного меню, после чего предстаёт в таком виде:

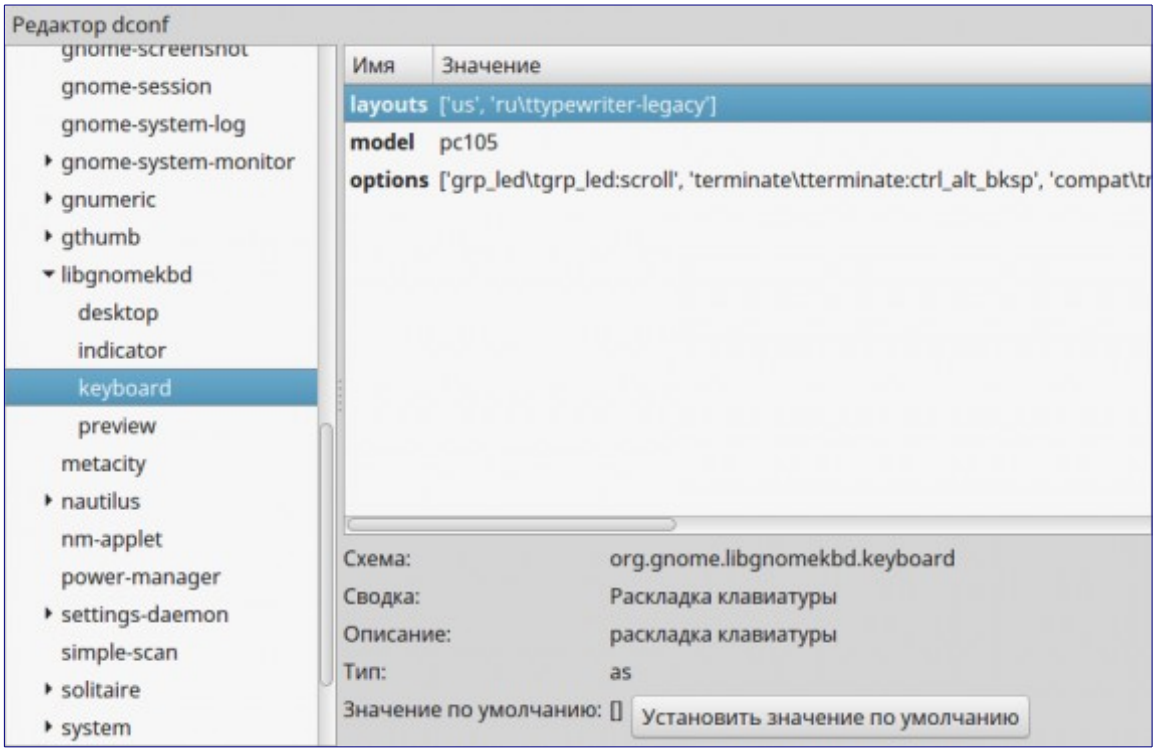


Каждый пункт во фрейме слева разворачивается каскадом вложенных пунктиков и подпунктиков, имя которым далеко превышает легион. И описывать которые я не буду. А дам пару практических рецептов для включения некоторых опций, недоступных через Системные настройки среды Cinnamon.

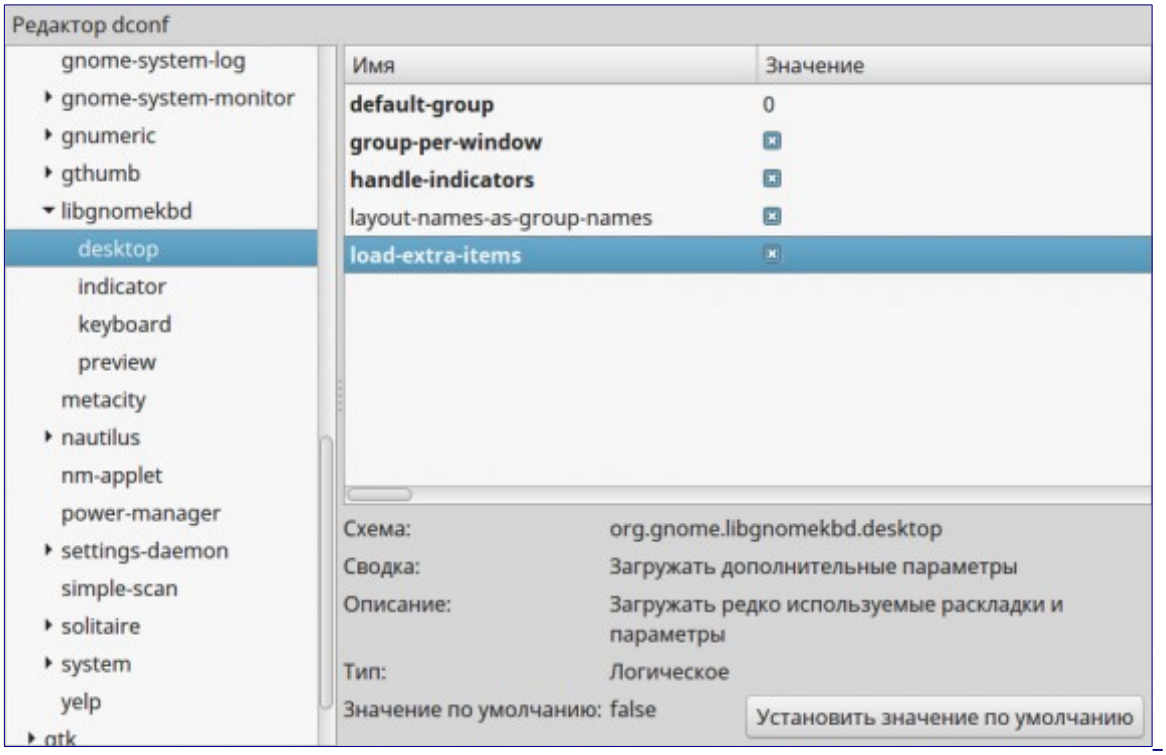
Ранее я упоминал, что в текущей версии Cinnamon возможность сохранения сеанса из пункта Автостарт изъята. Но в Редакторе dconf она осталась. И доступ к ней осуществляется по схеме org.cinnamon.SessionManager, где следует включить параметр (в терминологии dconf — ключ) auto-save-session:



Не так давно мы весьма подробно рассмотрели конфигурирование раскладок клавиатуры и их переключателей через соответствующий модуль Системных настроек Cinnamon. И пришли к выводу, что тут можно настроить всё, что только душе угодно. Тогда же было упомянуто, что всё то же самое можно проделать и в Редакторе dconf. Для чего достаточно проследовать по org.gnome.libgnomekbd.keyboard и заполнить желаемым образом поля layouts, model и options:

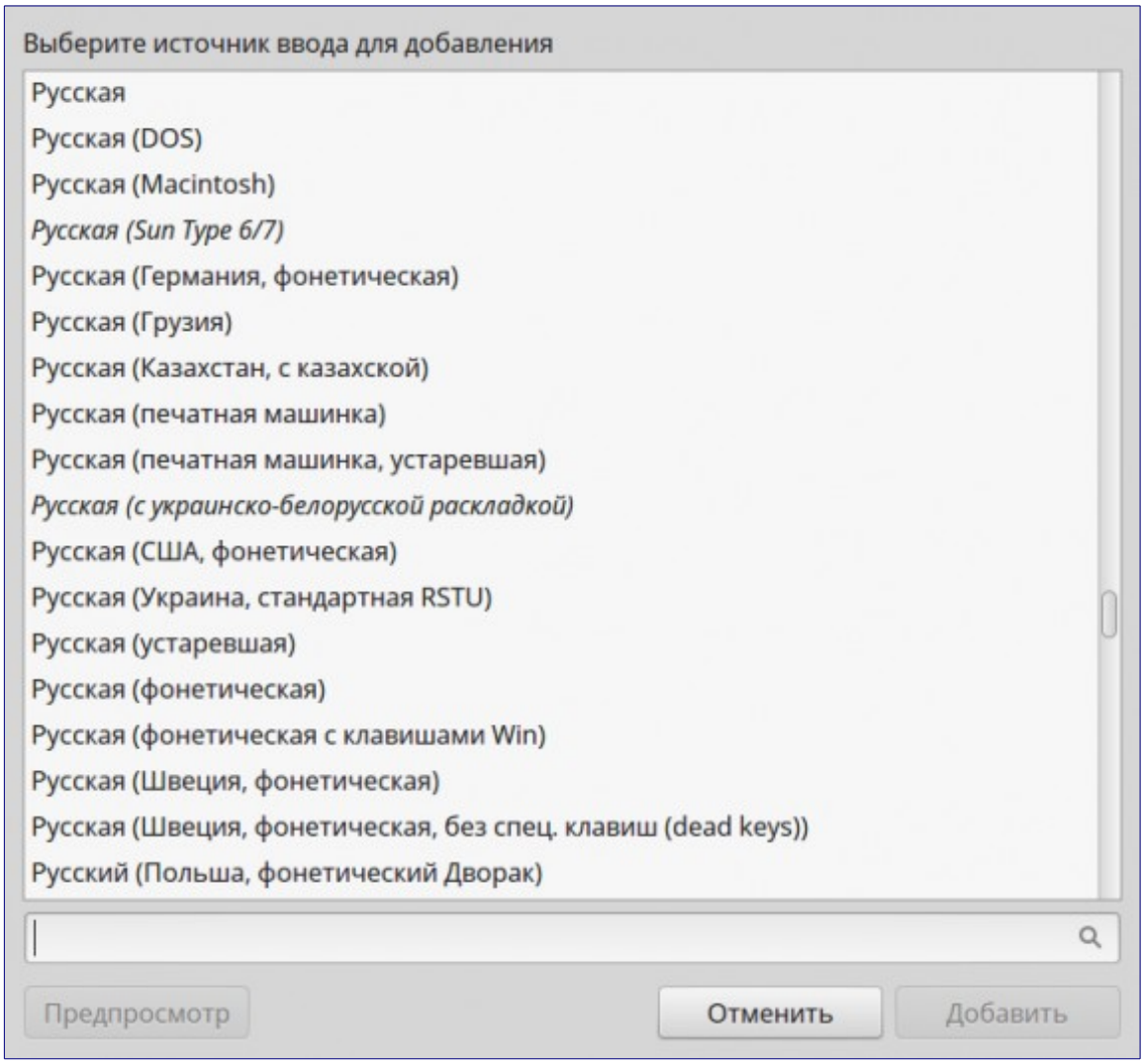


Как заполнить — описывать не буду, потому что через модуль Клавиатура в Системных настройках это сделать проще. А вот если пройти по схеме org.gnome.libgnomekbd.desktop, то можно включить ключ load-exta-items:

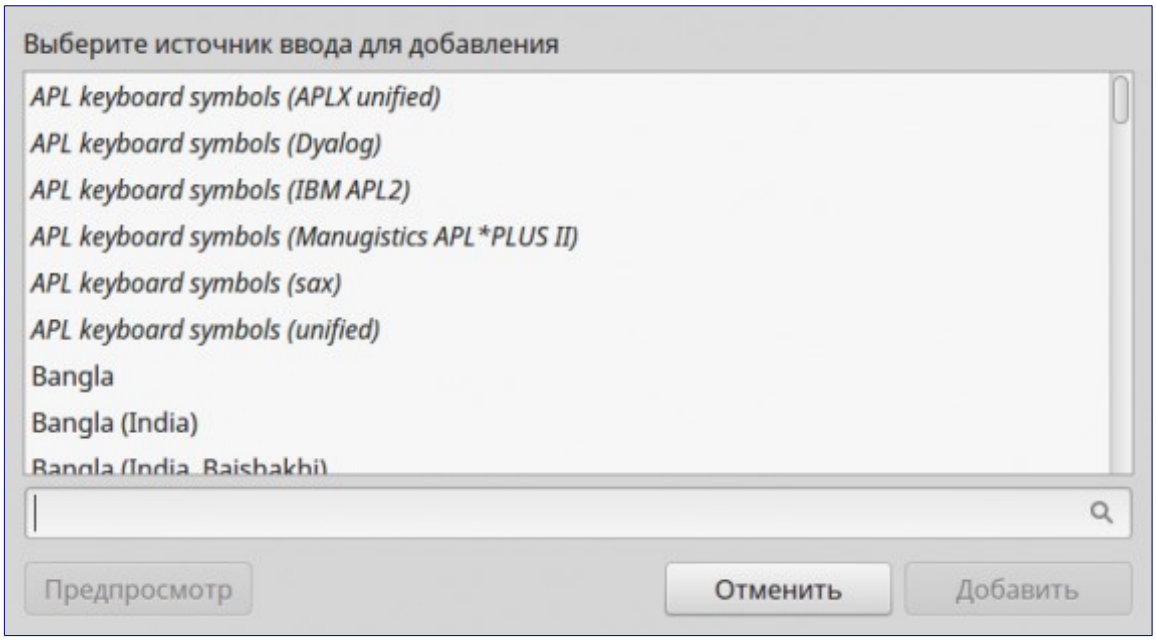


Это обеспечит загрузку редко используемых раскладок клавиатуры. В частности, для русской раскладки окажутся доступными дополнительные

варианты — те, что пыведены курсивом:



В русскоязычном окружении этот ключ практического значения не имеет, но для всяких прочих языцей — может пригодиться. Например, для программирования на APL:



Впрочем, более никаких практических задач для решения в Редакторе dconf я не придумал.

Mint и консоль

Всё, что было сказано о конфигурировании Mint в предыдущих очерках, относилось к настройкам графических сред, даже конкретно одной представительницы их — Cinnamon. Но ведь в Linux'е есть ещё и «текстовый» режим, то есть консоль. «Текстовый» в кавычках — потому что, конечно, чисто текстовой консоли нынче не найти, наверное, ни в одном дистрибутиве, везде frame buffer — но уж такова традиция.

Любителей выполнять в «голой» консоли практическую работу нынче не так уж и много. Наверное, поэтому майнтайнеры практически всех дистрибутивов относятся к настройке консольного режима абы как, если не сказать — никак. А уж когда дело доходит до локализованных систем — это вообще туши свет. Редкий дистрибутив может похвастаться тем, что в его голой консоли в ответ на команду

```
$ date
```

выводятся настоящие русские буквы, а не квадратики или кракозябры.

Повторяю, вряд ли кто будет сочинять в консоли многотомные романы на русском языке — в основном в текстовом режиме приходится загружаться при всякого рода сбоях. Но иногда приходится. Аварийные работы тоже лучше выполнять в комфортной и приятной для глаз обстановке. Да и вообще, как говорил штабс-капитан Мышлаевский, в казарме должен быть порядок, А текстовая консоль большинства современных дистрибутивов от порядка далека.

К чести Mint надо сказать, что этот дистрибутив принадлежит как раз к тем редким, у которых консоль русифицирована «искаропки». Русские буквы здесь и правильно выводятся, и правильно вводятся. Причём вводятся в том самом варианте русской раскладки клавиатуры, который был задан при инсталляции системы, и переключатель раскладок (по комбинации Alt+Shift) оказывается таким же, как в графическом режиме. В общем, оказывается, что приложить руки к улучшательству есть где. Ибо у применителя всё должно быть прекрасно — в том числе и консоль.

Особенно раздражает отсутствие поддержки мыши — без неё работа в консоли, даже аварийно-спасательная, становится мучительной. Ибо мышь в консоли — устройство не указательно-позиционирующее, а копировально-вставляющее. Достаточно выделить мышью фрагмент экрана, как он попадает в экранный буфер, откуда может быть вставлен в любое место, в том числе и в другую виртуальную консоль, щелчком средней кнопки мыши. Функция, незаменимая при правке конфигов — а ведь аварийно-спасательные работы обычно к ней и сводятся.

Поясню (как ни странно, нынче это надо пояснять), что на современных колесовых мышах роль средней кнопки почти всегда выполняет это самое колёсико. А на ноутбучных тачпадах тот же эффект достигается одновременным нажатием обеих кнопок. Правда, что делать на входящих в моду ноутбуках с однокнопочными тачпадами — не знаю; разве что, не покупать их...

Из поставленных задач по улучшательству консоли проще всего решается первоочередная — включение службы консольной мыши, сиречь gpm. Для этого нужно, как это ни парадоксально, установить пакет gpm:

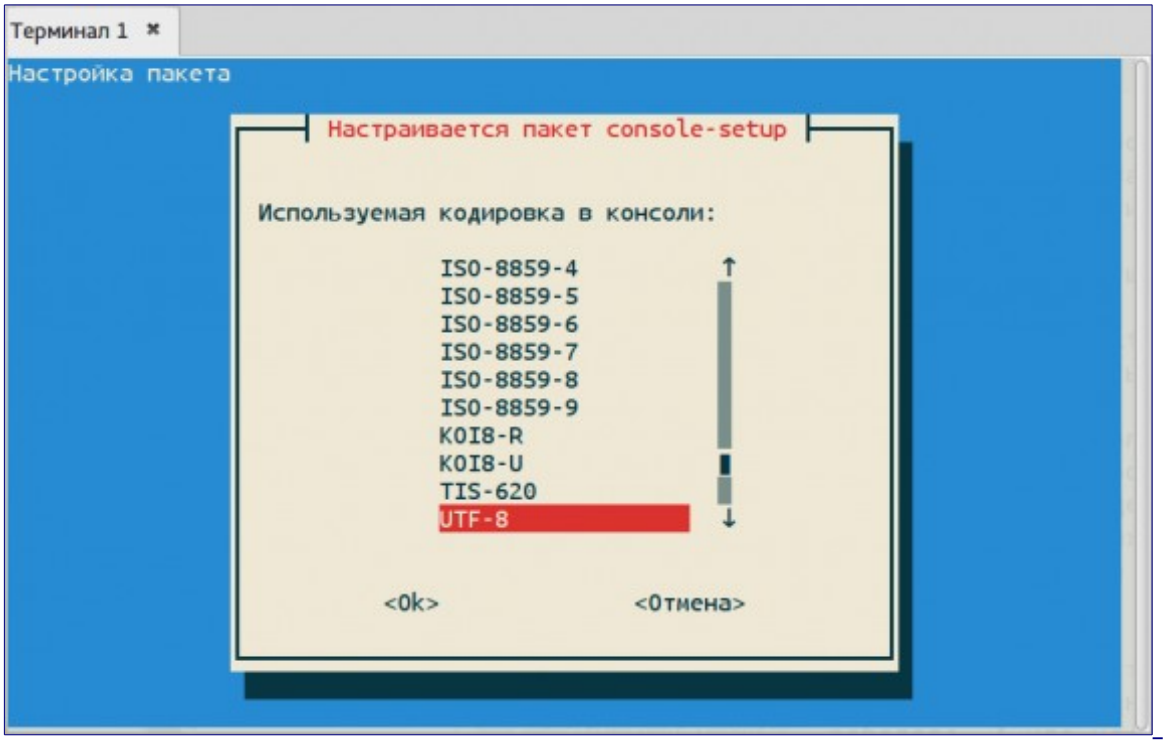
```
$ sudo apt install gpm
```

Если сделать это, находясь в чистой консоли (а перключиться в любую из них можно по комбинации клавиш Control+Alt+F#, # — от 1 до 6; возврат в графический сеанс — Alt+F8), то немедленно после завершения установки можно будет увидеть курсор мыши в виде прямоугольничка. И теперь, по крайней мере, не придётся при всяких ремонтно-восстановительных работах вводить много лишних символов — в распоряжении применителя описанный выше «мышиный» буфер.

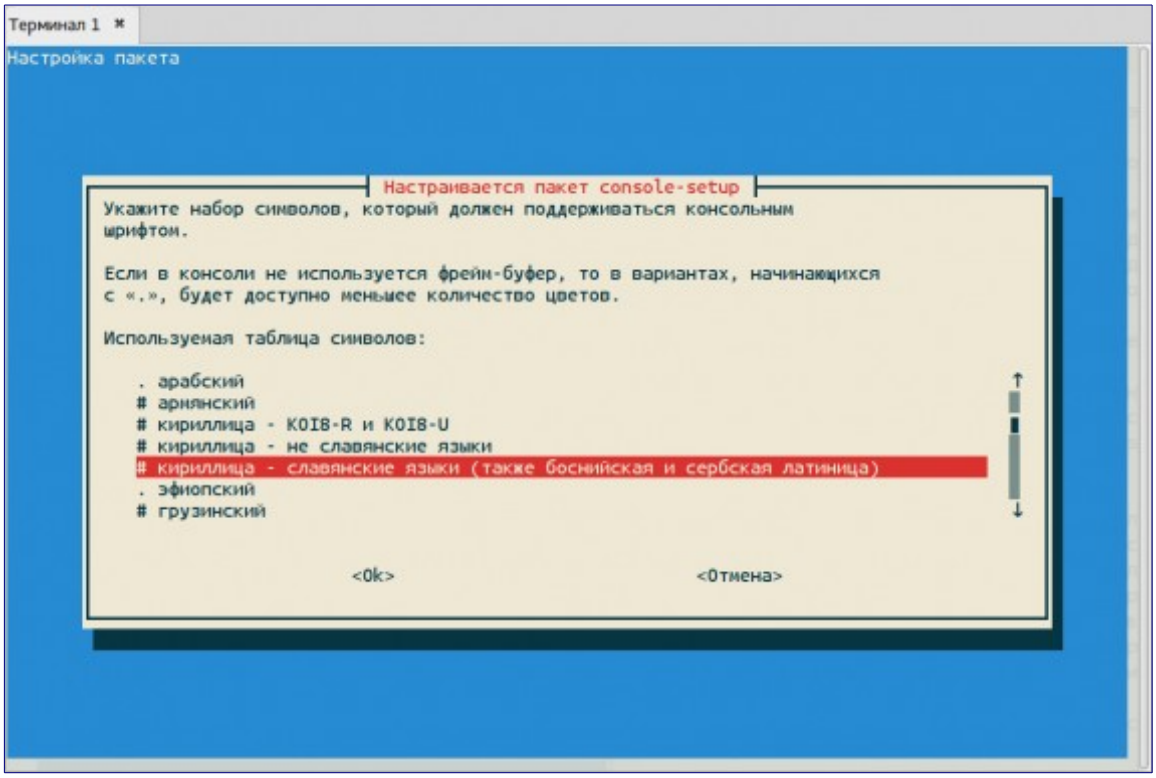
Следующая задача на очереди — установка удобочитаемого экранного шрифта. Проще всего она решается утилитой dpkg-reconfigure. Вызванная в таком виде

```
$ sudo dpkg-reconfigure console-setup
```

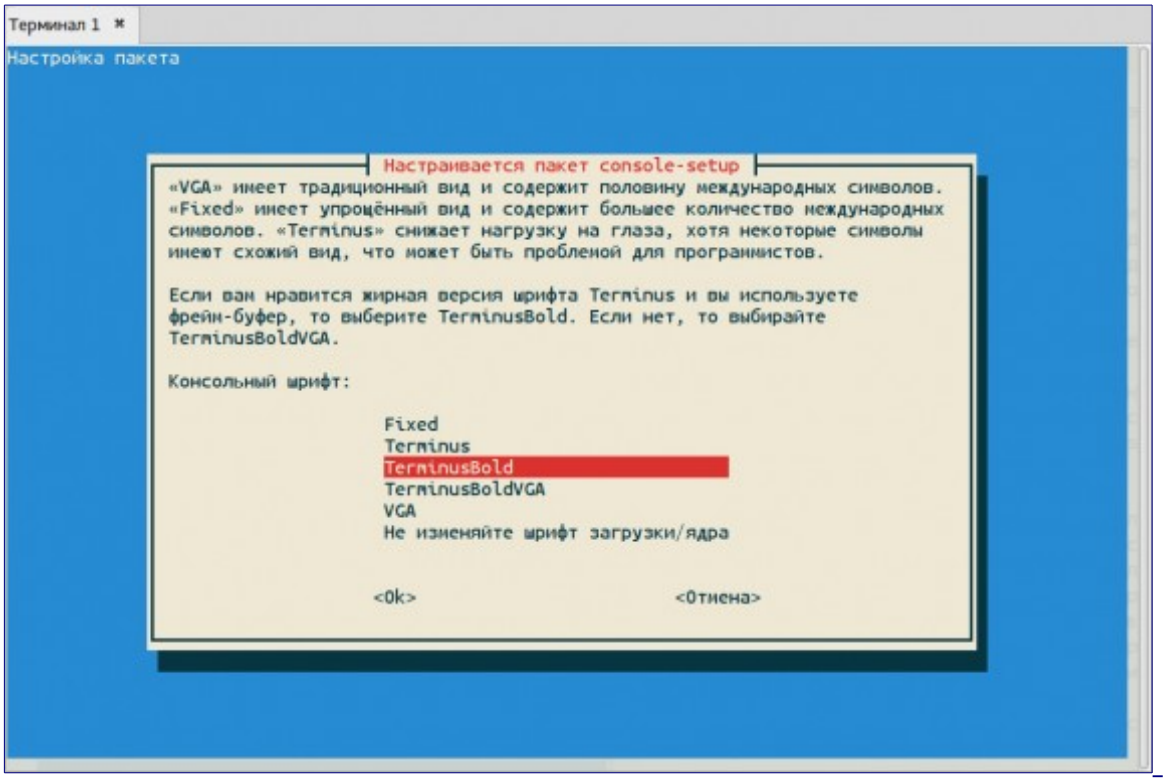
она запустит псевдографическую программу, настройки экранных шрифтов для консоли. Которая сначала попросит выбрать кодировку:



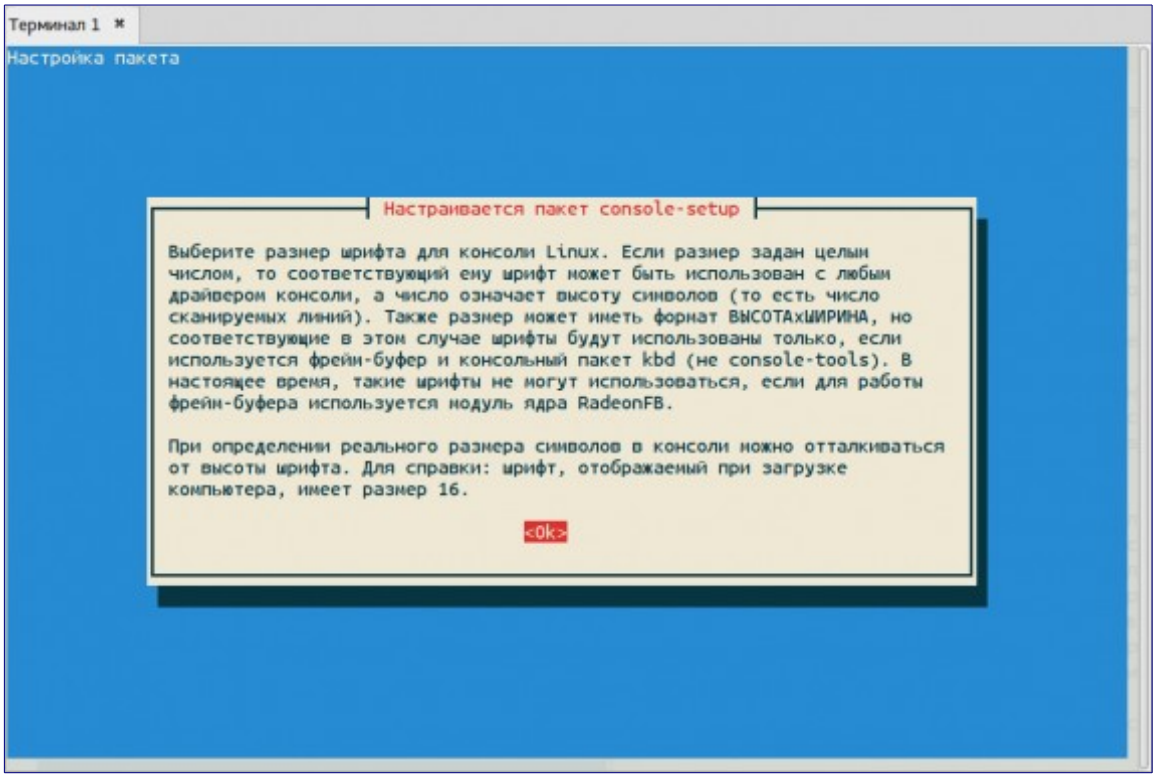
Затем спросит об используемой таблице символов:



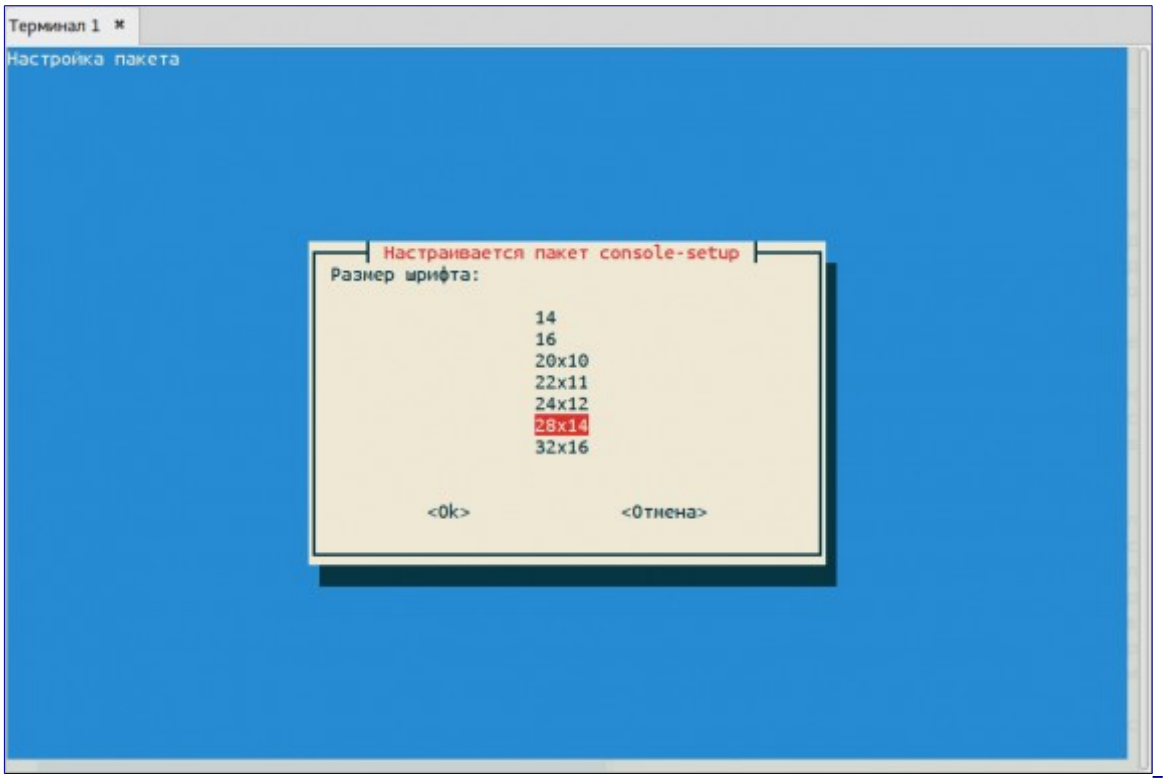
Потом последует предложение выбрать шрифт:



Далее будет проведён маленький ликбез о консольных шрифтах и условиях их использования:



Не советую им пренебрегать — после этого легче сделать осознанный выбор матрицы шрифта (типографские термины к консольным шрифтам не применимы):



После этого происходит выход из интерфейса утилиты, и всё заказанное претворяется в действительность. Процесс этот связан с регенерацией initrd, так что его результат можно будет увидеть только после рестарта — с которым, впрочем, не обязательно торопиться.

Третья задача очень важна для меня — но возможно, что большинству применителей решать её не придётся. Я использую сочетание варианта Typewriter Legacy для кириллической раскладки и CapsLock в качестве

переключателя латиница/кириллица. Когда-то эта была стандартная комбинация (именно такова была первая русская раскладка для UNIX-косоли, созданная Андреем Черновым aka ache), но ныне воспринимается как экзотика. И её «спаривание» для консоли Linux требует некоторых усилий. В частности, в большинстве дистрибутивов мне приходилось прибегать к раскладке, изготовленной собственноручно.

А вот в Mint эти усилия минимальны. Я имел не один раз повод радостно сообщить, что выбранная при установке раскладка клавиатуры и один из её вариантов (среди которых имеется и Typewriter Legacy) наследуется не только Иксами, но и консолью установленной системы. Правда, с переключением раскладок по Alt+Shift, порождённым каким-то умником в недрах Microsoft'a вместе с раскладкой winkeys (также одной из самых неудобных, какую только можно придумать).

Однако задача с изменением переключателя решается очень просто: достаточно отредактировать файл /etc/default/keyboard. Он практически точно совпадает с клавиатурной секцией старого /etc/X11/xorg.conf или современного /etc/X11/xorg.conf.d/10-keymap.conf, и по умолчанию выглядит так:

```
XKBMODEL="pc105"

XKBLAYOUT="us,ru"

XKBVARIANT=",typewriter-legacy"

XKBOPTIONS="grp:alt_shift_toggle,grp_led:scroll"
```

Так что в нём достаточно заменить значение переключателя alt_shift_toggle на желаемое, например, для меня — на caps_toggle. После чего можно с чистым сердцем перегружаться и, авторизовавшись в любой текстовой консоли, любоваться красивыми шрифтами семейства Terminus, созданными Димитром Жековым, набирать русские буквы в привычной раскладке и, при необходимости, копировать набранное из консоли в консоль через «мышиный» буфер.

GRUB2: восстановление загрузчика

В заключение всей «конфигурационной солянки» — пара слов о загрузчике GRUB2. Материалов в Сети на эту тему немерянно (одним из самых полезных мне представляется вот этот), и пересказывать их я не намерен. А остановлюсь только на восстановлении загрузчика в случае его порчи по каким-либо причинам.

Есть несколько методов восстановления GRUB2, я опишу тот, который представляется мне самым простым.

Для начала надо иметь Live-носитель Mint в любом виде — DVD-, флешки, SD-карты, каковой помещается куда следует. При рестарте машины, обычно в момент появления логотипа производителя, нужно успеть нажать клавишу быстрого выбора загрузочного устройства — для современных ASUS'овских «матерей» это F8, для ASRock'овских — F11, для прочих — см. документацию. И в появившемся меню выбрать нужный пункт, он обычно называется явным образом. Происходит загрузка Live-среды, каковая в нашем случае представлена Mint с Cinnamon-окружением.

Далее следует опознать имя файла устройства, несущего корневой раздел системы, загрузчик которой подлежит восстановлению — подчёркиваю, требуется имя устройства, а не раздела. Это можно сделать просмотром вывода либо команды

```
$ sudo fdisk -l
```

либо команды

```
$ ls /dev/sd*
```

Предположим для определённости, что это устройство /dev/sda. Теперь корневой раздел монтируется в файловую систему Live-среды. Прще всего сделать это через Nemo — в его боковой панели, в секции Носители, отражаются все присутствующие в машине дисковые устройства. И достаточно выбрать пункт Присоединить контекстного меню или просто «левокликнуть» на соответствующем имени. Теперь главное — определить точку монтирования. В Mint (как и во всех Ubuntu'идах) временно смотированные устройства попадают в каталог /media/username, устройства с меткой — в подкаталог с её именем, без оной — в подкаталог с универсальным идентификатором устройства (UID), это полуметровая зубодробительная последовательность букв и цифр.

А вот теперь — собственно восстановление загрузчика. Оно выполняется одной командой:

```
$ sudo grub-install --root-directory=/media/username/mount_point /dev/sda
```

Здесь mount_point — метка диска или его очень простой UID вроде d55aebcb-7e7d-4d34-aff4-ed6e494e9b7f. Автодополнение в этом случае не работает, однако, даже если устройство не «помечено», его UID можно взять из файла /etc/mtab, описывающего временно смонтированные устройства, или, открыв его в Nemo, из адресной строки последнего.

На этом дело восстановления загрузчика закончено — можно перезагружаться в нормальном режиме.

Основы командного интерфейса

Поскольку не исключена вероятность, что эту книгу будут читать и совсем начинающие применители Linux'а вообще, тех, для кого Mint оказался первым дистрибутивом этой операционной системы, в этом очерке будут даны некоторые общие сведения об интерфейсе командной строки (CLI — Command Line Interface). Тем более, что это потребуется уже ближайшее время, в очерках, посвящённых управлению пакетами.

Введение в CLI

CLI представляет собой базу, для которой GUI всякого рода являют лишь оболочку. Всякое действие в linux-системе может быть выполнено прямой командной директивой. И его же можно осуществить путем манипулирования объектами. Например, копирование файлов выполняется соответствующей командой — cp, это первый способ. Но его же можно осуществить перетаскиванием мышью объекта, представляющего наш файл зрительно, из того места, где он находился ранее, туда, где мы хотим видеть его копию, а это уже второй способ.

То есть манипуляция объектами в GUI — это обычно более или менее опосредованное выполнение соответствующих данному действию команд. Почему основные навыки работы с CLI не помешают даже тому пользователю, который не вылезает из графической среды. Ибо сфера применения CLI не ограничивается «голой» консолью. Он же используется в эмуляторах терминала в графическом режиме оконной среды X. Более того, в настоящее время это основная среда для применения командного интерфейса — к текстовой консоли обычно обращаются только в аварийных ситуациях.

CLI в большинстве случаев обеспечивается классом программ, именуемых командными интерпретаторами, командными процессорами, командными оболочками или по простому шеллами (shell).

Как легко догадаться по одному из определений, кроме предоставления пользовательского интерфейса, шеллы выполняют и вторую функцию — служат интерпретаторами собственных языков программирования. На этом основывается классификация шеллов — они разделяются на две группы, обычно именуемые Bourne-shell совместимые и C-shell совместимые. В силу ряда причин в качестве стандарта принята одна из оболочек первой группы — так называемый POSIX-шелл. Правда, он представляет собой чистую абстракцию, однако большинство используемых в Unix'ах оболочек с этим стандартом совместимы. А системная оболочка Mint, Dash, довольно точно воспроизводит и его функциональность. И потому все примеры, иллюстрирующие принципиальные вопросы CLI, будут базироваться на наиболее используемых командах, построенных в соответствие с правилами POSIX-шелла.

Командная строка

Основой командного интерфейса является командная строка, начинающаяся с приглашения для ввода. Далее он будет обозначаться милым сердцу россиянина символом длинного зеленого друга — $, если речь идёт о сеансе обычного пользователя, или символом решётки — #, для приглашения строки в сеансе администратора. Это — чистая условность: вид приглашения может быть настроен в широких пределах, причём по разному в разных оболочках. Об этом мы поговорим, когда речь дойдёт до описания конкретного шелла — Zsh.

Командная строка — визуальное представление среды, в которой задаются основные элементы командного интерфейса: командные директивы с их аргументами и опциями.

Командная директива (или просто команда) — основная единица, посредством которой пользователь взаимодействует с шеллом. Она образуется по определенным правилам, именуемым синтаксисом. Синтаксис командной директивы определяется, в первую очередь, языком, принятым в данной командной оболочке. Кроме того, некоторые команды (не очень многочисленные, но весьма употребимые) имеют собственный, нестандартный синтаксис.

Однако в целом базовые правила построения команд имеют много общего. И именно эти базовые правила станут предметом данного раздела. Синтаксические особенности отдельных нестандартных команд будут оговариваться по ходу изложения.

Итак, командная директива образуется:

- именем команды, однозначно определяющим ее назначение,
- опциями, определяющими условия выполнения команды, и
- аргументами — объектами, над которым осуществляются действия.

Очевидно, что имя команды является обязательным компонентом, тогда как опции и аргументы могут и отсутствовать (или подразумеваться в неявном виде по умолчанию).

ещё один непременный компонент командной директивы — это специальный невидимый символ конца строки: именно его ввод отправляет команду на исполнение. В обыденной жизни этот символ вводится нажатием и отпусканием клавиши Enter. Почему обычно и говорят: для исполнения команды нажмите клавишу Enter. Тот же эффект, как правило, достигается комбинацией клавиш Control+M. Символа конца командной строки, знаменующего исполнение команды, мы на экране не видим. Однако важно, что это — такой же символ, как и любой другой (хотя и имеющий специальное значение).

В подавляющем большинстве случаев опции (или их последовательности) задаются непосредственно за именем команды, а аргумент (или группа аргументов) команду завершает, хотя это правило имеет некоторые исключения. Вне зависимости от порядка опций и аргументов, принятых для данной команды, интерпретация их осуществляется слева направо.

Команды, опции и аргументы обязательно разделяются между собой пробелами. Кроме того, опции обычно предваряются (без пробела) символом дефиса или двойного дефиса. Впрочем, немногочисленные (но весьма употребимые) команды могут использоваться с опциями без всяких предваряющих символов.

Как уже говорилось, имя команды определяет выполняемые ею функции. Существуют команды, встроенные в оболочку, то есть не имеющие запускающих их исполняемых файлов, и команды внешние. В последнем случае имя команды однозначно указывает на имя исполняемого файла программы, выполняемой при отдаче соответствующей директивы. Часто встроенные и внешние команды одного назначения имеют одинаковые имена. В этом случае обычно предпочтительно использование встроенных команд — впрочем, они и вызываются в первую очередь. Для вызова одноимённой внешней команды её нужно задать с указанием пути. Так, директива

```
$ time search for Mint in path2/
```

вызовет для определения времени выполнения команды search (о ней будет рассказываться в следующем очерке) встроенную команду time. А в форме

```
$ /usr/bin/time search for Mint in path2/
```

будет задействована внешняя утилита с тем же именем. Кстати, это один из тех случаев, когда второй вариант может иногда оказаться предпочтительней: встроенная и внешняя команды имеют разные форматы вывода, причём в первой он зависит от используемой командной оболочки. И потому она не всегда подходит для прямого сравнения результатов — например, быстродействия в разных системах.

Определить, является ли данная команда встроенной в оболочку или внешней, можно с помощью встроенных команд type или which. Для встроенных команд вывод их будет таким:

```
$ type which

which is a shell builtin

$ which type

type: shell built-in command
```

Или, в некоторых случаях, таким:

```
$ which time

time: shell reserved word

$ type time

time is a reserved word
```

Для внешних команд любой из этих вариантов даст в выводе путь к исполняемому файлу:

```
$ which date

/bin/date

$ type date

/bin/date
```

Некоторые команды могут выступать под несколькими именами. Это связано с тем, что исторически в различных Unix-системах команды, исполнявшие одинаковые функции, могли получать разные названия. В каждой конкретной системе обычно используется только одна из таких команд-дублеров. Но при этом имена дублирующих команд также могут присутствовать в системе — для совместимости. Не следует думать, что это две различные программы одного назначения: как правило, такая синонимичность команд реализуется посредством механизма ссылок (links) или псевдонимов (alias), о которых речь пойдёт позднее.

Иногда команда, вызванная через имя своего синонима, может отличаться по своей функциональности от самой же себя, вызванной под родным именем. В этом случае говорят о эмуляции одной команды другой. Типичный пример — командная оболочка /bin/bash в большинстве дистрибутивов Linux имеет своего дублера — /bin/sh; вызванная таким образом, она воспроизводит базовую функциональность стандарта POSIX-шелла.

Автодополнение

Для правильного применения команд, конечно же, нужно знать их имена и назначение. Однако нас никто не заставляет напрягать пальцы вводом имени команды полностью. Потому что тут на помощь приходит великий метод автодополнения.

Благодаря этому методу для любой команды достаточно ввести первые несколько ее символов — и нажать клавишу табуляции (Tab). И, если введённых буковок достаточно для однозначной идентификации, полное имя команды волшебным образом возникнет в строке. Если же наш ввод допускает альтернативы продолжения имени — все они высветятся на экране (сразу или после повторного нажатия на табулятор), и из них можно будет выбрать подходящую.

Большинство употребимых команд POSIX-систем — коротки и мнемонически прозрачны. И может показаться. что не такое уж это облегчение — заменить ввод двух-трех символов нажатием табулятора (а то ещё и неоднократным). Однако, когда речь дойдет до аргументов команд — тут вся мощь автодополнения станет явной.

И ещё маленькое отступление. Автодополнение — стандартная возможность Bash и всех других командных оболочек, относимых к категории развитых. Но как раз в стандарте POSIX эта возможность не предусмотрена, и потому POSIX shell ее лишён. Нет этой функции и в Dash — системной командной оболочке Mint. Которая, впрочем, в интерактивном режиме не используется.

Ещё один способ облегчения ввода команд — обращение к их истории, о чём разговор будет несколько позже.

Опции

Указания только имени команды достаточно для выполнения некоторых из них. Типичный пример — команда ls (от list), предназначенная для просмотра имен файлов (строго говоря, содержимого каталогов). Данная без аргументов, она выводит список имен файлов, составляющих текущий каталог, представленный в некоторой форме по умолчанию, например, в домашнем каталоге пользователя это будет выглядеть примерно так:

```
$ ls

Desktop/   Downloads/   Music/  Pictures/  Templates/

Documents/  lost+found/  mytmp/  Public/   Videos/
```

Исполнение же многих других команд невозможно без указания опций и (или) аргументов. Для них в ответ на ввод одного её имени часто следует не сообщение об ошибке (или не только оно), но и краткая справка по использованию команды. Например, в ответ на ввод команды для создания каталогов mkdir (от make directory) последует следующий вывод:

```
usage: mkdir [-pv] [-m mode] directory ...
```

Для одних опций достаточно факта присутствия в командой директиве, другие же требуют указания их значений (даваемых после опции обычно через знак равенства). В приведённом примере команды mkdir к первым относятся опции -v (или --verbose), предписывающая выводит информацию о ходе выполнения команды (запомним эту опцию — в том же смысле она используется чуть ли не во всех командах Unix), и -p, которая позволяет создать любую цепочку промежуточных каталогов между текущим и новообразуемым (в случае их отсутствия).

А вот опция -m, определяющая атрибуты доступа к создаваемому каталогу, обязательно требует указания значения — этих самых атрибутов, заданных в символьной форме.

Многие опции имеют две формы — краткую, односимвольную, и полную, или многосимвольную, Некоторые же опции могут быть даны только в многосимвольной форме. Общее правило здесь таково: если одного символа достаточно для однозначного определения опции, могут употребляться обе формы в качестве равноправных. Однако поскольку количество символов латинского алфавита ограниченно (а человеческая фантазия, конструирующая опции — безгранична), при большом количестве опций одной команды некоторые из них приходится делать исключительно многосимвольными.

Продемонстрирую это на примере опций все той же команды mkdir. Полный их список будет следующим:

```
-m, --mode=MODE установить код доступа

      (как в chmod)

-p, --parents не выдавать ошибок,

      если существует, создавать

      родительские каталоги,

      если необходимо

-v, --verbose печатать сообщение

      о каждом созданном каталоге

--help показать помощь и выйти

--version  вывести информацию

      о версии и выйти
```

Очевидно, что для опции --version краткая форма совпала бы с таковой для опции --verbose, и потому первая существует только в полной форме. А вот для опции --help краткая форма в большинстве команд возможна, и она выглядит как -h. Более того, во многих командах вызов помощи может быть вызван посредством опции -?. К слову сказать — приведенный выше список опций команды mkdir получен именно таким способом.

Раз уж зашла речь об опциях --version и -h (--help, -?), давайте и их запомним на будущее. Это — так называемые стандартные опции GNU, в число коих входит и опция -v, --verbose. Назначение «длинной» их формы (--version, --help, --verbose) идентично почти во всех командах, краткой — во многих.

Опять-таки, из того же примера видно, что опции в односимвольной форме предваряются единичным символом дефиса и могут быть даны единым блоком, без пробелов:

```
$ mkdir -vpm 777 dir/subdir
```

При этом, естественно, опция, требующая указания значений, ставится последней, и ее значение отделяется пробелом. Опции же в многосимвольной форме требуют предварения удвоенным дефисом, обязательно должны разделяться пробелами и значения их, если таковые требуются, присваиваются через символ равенства (по научному он называется ещё оператором присваивания):

```
$ mkdir --parents --mode=777 dir/subdir
```

Загадочные семерки после опции -m (--mode) — это и есть те самые атрибуты доступа, данные в символьной нотации, о которых речь пойдёт в соответствующем разделе.

Опции команды именуются также флагами (реже ключами) или параметрами. Однозначной трактовки этих терминов нет. Однако обычно под флагами подразумеваются опции, не требующие указания значений. Термин параметр же применяется к опции, такового требующей, и объединяет опцию и ее значение. Правда, мне встречалось определение параметра как совокупности опций и аргументов, но я буду придерживаться приведенных определений.

Порядок опций, если их приводится более одной, для большинства команд не существенен. Хотя, например, для команды tar, создающей файловые архивы, опция -f, значением которой является имя создаваемого или распаковываемого архива, традиционно указывается последней. И, к слову сказать, именно эта команда — одна из немногих, опции которой не обязаны предваряться символами дефиса. Так, директивы

```
$ tar cf filename.tar dir
```

и

```
$ tar -cf filename.tar dir
```

абсолютно равноценны: и та, и другая создает единый архивный файл filename.tar из отдельных файлов каталога dir.

Особый смысл имеет символ удвоенного дефиса --, если после него не следует никакой опции: таким образом обозначается конец списка опций, и все последующие, отделённые пробелом, символы интерпретируются как аргументы. Одинарный же дефис с последующим пробелом, напротив, подменяет аргументы команды, то есть в качестве таковых рассматривается стандартный ввод: знание этого нам потребуется, когда речь дойдёт до командных конвейеров.

Пример: опции команды ls

Опции определяют условия выполнения команды. На предыдущей странице был приведён пример команды ls без опций. Однако на самом деле отсутствием опций при ней определяется вид выводимого списка по умолчанию — как многоколочночного списка, состоящего из имен файлов без учета т.н. скрытых файлов (а таковыми являются файлы, имена которых начинаются с символа точки, почему они ещё называются dot-файлами), без каких-либо их атрибутов и без визуального различия файлов различных типов.

Различные же опции команды ls определяют состав и формат выводимого списка файлов. Так, в форме

```
$ ls -a
```

она обеспечивает вывод списка имен всех файлов текущего каталога, включая
скрытые файлы вида .* (символ * здесь обозначает шаблон имени, соответствующий любому количеству любых символов — в том числе и нулевому, то есть отсутствию оных), символы текущего (./ каталога и

каталога родительского (../).

В форме

```
$ ls -l
```

дается вывод списка имен файлов в «длинном» формате (отсюда название опции -l — от long), то есть с указанием атрибутов доступа, принадлежности, времени модификации, размера и некоторых других характеристик:

```
drwxrwxr-x. 14 alv alv 4,0K Map 14 08:40 current/

drwxr-xr-x.  2 alv alv 4,0K Фев  8 11:28 Desktop/

drwx------.  5 alv alv 4,0K Map 11 18:34 priv/
```

Форма

```
$ ls -F
```

позволяет получить список файлов с символьным различением файлов различных типов. Например, имя каталога будет выглядеть как dirname/, имя исполнимого файла — как filename* (здесь звездочка — не шаблон имени, а символическое обозначение исполняемого файла), и так далее.

Я столь подробно остановился на команде ls не только из-за многочисленности ее опций: это — одна из самых употребимых команд для просмотра файловой системы. И, должным образом настроенная (в том числе и с помощью приведенных опций), она дает ничуть не менее информативную и зрительно выразительную картину, чем развитые файловые менеджеры типа Midnight Commander или многочисленные файловых менеджеры графического режима.

Аргументы

Таким образом мы подобрались к понятию аргументов командной директивы. Аргументами определяется, как правило, объект (или объекты) действия команды. В большинстве случаев в качестве аргументов команд выступают имена файлов и (или) пути к ним.

Выше говорилось, что при отсутствии аргументов команда ls выводит список имен файлов текущего каталога. Это значит, что текущий каталог выступает как заданный неявным образом (по умолчанию) аргумент команды ls. Если же требуется вывести список имен файлов каталога, отличного от текущего, путь к нему должен быть указан в качестве аргумента команды

явно, например:

```
$ ls /usr/bin
```

Большинство команд допускает указание не одного, а нескольких (и даже очень многих) аргументов. Так, единой директивой вида

```
$ cp file1 file2 ... fileN dir
```

можно скопировать (команда cp — от copy) сколько угодно файлов из текущего каталога в каталог dir (на самом деле на это «сколько угодно» накладываются некоторые теоретические ограничения, определяемые максимально возможной длиной командной строки, но практически предел этот очень далек).

Маленькое отступление. Упоминание команды cp — удобный случай чуть вернуться назад и рассмотреть одну очень важную опцию, почти универсальную для команд POSIX-систем. Для начала попробуем скопировать один каталог в другой:

```
$ cp dir1 dir2
```

Как вы думаете, что получится в результате? Правильно, сообщение о невозможности выполнения этой операции — примерно в таком виде:

```
cp: omitting directory 'dir1'
```

поскольку команда cp в чистом виде для копирования каталогов не предназначена. Что делать? Очень просто — указать опцию -R (от Recursive; в большинстве систем проходит и опция -r с тем же смыслом, но первая форма работает абсолютно везде. В результате в каталог dir2 не только будут скопированы сам каталог dir1 и все входящие в него файлы, но и вложенные подкаталоги из dir1, если таковые имеются.

Маленькое уточнение: вполне возможно, что в дистрибутиве, который имеется в вашем распоряжении, проходит и копирование каталогов просто через cp, без всяких дополнительных опций. Это — потому, что команда cp часто определяется как псевдоним самой себя с опцией рекурсивного копирования, о чем скоро пойдет речь.

Вообще рекурсия (то есть определение некоего выражения через самого себя) — очень важное понятие в Unix, пронизывающее происходящие от нее системы насквозь. И послужившие даже базой для своеобразного хакерского юмора. Однако сейчас для нас важно только то, что рекурсия применима практически ко всем файловым операциям, позволяя распространить действие одной командной директивы не только на файлы данного каталога, но и на все вложенные подкаталоги и их содержимое.

Однако вернемся к аргументам. Действие некоторых команд неоднозначно в зависимости от аргументов, к которым она применяется. Например, команда mv служит как для переименования файлов, так и для их перемещёния в другой каталог. Как же она узнает, что ей делать в данном конкретном случае? Да именно по аргументам. Если дать ее в форме

```
$ mv filename1 filename2
```

то следствием будет переименование filename1 в filename2. А вот если первым аргументом указан файл, а вторым — каталог, например

```
$ mv filename dir
```

то результатом будет перемещёние filename из текущего каталога в каталог dir. К слову сказать, команды типа mv воспринимают разное количество аргументов в зависимости от того, какие они, эти аргументы. В первом примере аргументов может быть только два — имя исходного файла и имя файла целевого. Зато во втором примере в качестве аргументов можно задать сколько угодно файлов и каталогов (с учетом вышеприведенной оговорки относительно «сколько угодно») — все они будут перемещёны в тот каталог, который окажется последним в списке. То есть директивой:

```
$ mv file1 ... fileN dir1 ... dirM dirN
```

в каталог dirN будут перемещёны все файлы file1 ... fileN и все каталоги dir1 ... dirM. Характерно, что для этого команде mv, в отличие от команды cp, ей не требуется каких-либо дополнительных опций — она рекурсивна по самой своей природе.

Пути к файлам

Для правильного построения аргументов команды требуется рассмотрение ещё одного понятия — пути к файлу. Путь — это точное позиционирование файла в файловой системе относительно ее корня (обозначаемого символом прямого слэша — /) или нашего в ней положения — текущего каталога (который, напомню, символически обозначается единичной точкой — .).

Так, если пользователь находится в своем домашнем каталоге (абсолютный путь к нему обычно выглядит как /home/username), то просмотреть содержимое каталога /usr/bin он может двумя способами — тем, который был дан в предыдущем примере, или вот так:

```
$ ls ../../usr/bin
```

Первый путь в аргументе команды ls — абсолютный, отсчитываемый от корневого каталога, второй — задается относительно каталога текущего, ведь ../ — это родительский каталог для него.

Пути в аргументах команд могут быть весьма длинными. Например, чтобы просмотреть список шрифтов, применяемых в интерфейсе Cinnamon по умолчанию, нужно дать команду следующего вида:

```
$ ls /usr/share/fonts/truetype/noto
```

И читатель вправе спросить — неужели мне все это вводить вручную? Отнюдь — отвечу я ему. Потому что автодополнение, о котором упоминалось по ходу разговора об именах команд, действует также для путей в их аргументах — последовательным нажатием клавиши табуляции все недостающие символы будут добавляться

Ещё один способ избежать набора длинных путей к файлам — это определение переменной PATH. Внимательный читатель, вероятно, обратил внимание, что при наборе команды путь к исполняемому её файлу не указывается. Для внутренних команд причина понятна — они прошиты в

самой оболочке. А как мы обходимся без указания путей к командам внешним? Неужели система мистическим чувством определяет, где они находятся?

Отнюдь, ни малейшей мистики, Просто каталоги, в которых находятся команды (а это, как правило, /bin, /sbin, /usr/bin, /usr/sbin) определены в качестве значений переменной PATH, о чём мы подробнее поговорим со временем.

Кое-что об исключениях

Итак, типичная форма POSIX-команды в обобщенном виде выглядит следующим образом:

```
$ command -[options] [arguments]
```

Из этого правила выбиваются немногочисленные, но весьма полезные и часто используемые команды. Однако и для таких команд с нестандартным синтаксисом устанавливаются те же компоненты — имя, опции, аргументы, хотя по ряду причин (в том числе исторических) порядок их может меняться.

Это можно проиллюстрировать на примере полезнейшей команды find, предназначенной для поиска файлов (и не только для этого — она являет собой почти универсальное орудие в деле всякого рода файловых манипуляций). В типичной своей форме она выглядит примерно следующим образом:

```
$ find dir -option1 value -option2 [value]
```

Здесь dir — каталог, в котором выполняется поиск, — может рассматриваться в качестве аргумента команды. Опция -option1 (обратим внимание, что здесь, не смотря на многосимвольность опций, они предваряются единичным символом дефиса) и ее значение value определяют критерий поиска, например, -name filename — поиск файла с указанным именем, а опция -option2 предписывает, что же делать с найденным файлом (файлами), например, -print — вывести его имя на экран. причём опция действия также может иметь значение. Например, значением опции -exec будет имя команды, вызываемой для обработки найденного файла (файлов). Так, директива вида

```
$ find ~/ -name *.tar -exec tar xf {} ;
```

требует отыскать в домашнем каталоге (~/), выступающем в качестве аргумента, файлы, имя которых (первая опция — критерий поиска) соответствует шаблону *.tar (значение первой опции), и выполнить (вторая опция — действия) в их отношении команду tar с собственными опциями, обеспечивающими распаковку архивов (значение второй опции). Интересно, что в этом контексте в качестве значений второй опции команды find выступает не только внешняя команда, но и все относящиеся к ней опции.

В последнем примере имеется несколько символов, смысл которых может показаться непонятным. Надеюсь, он прояснится достаточно скоро — в разговоре о регулярных выражениях.

Псевдонимы

Вернемся на минуту к команде ls. У читателя может возникнуть вполне резонный вопрос: а если я всегда хочу видеть ее вывод с символическим различением типов файлов, да ещё в «длинном» формате? Ну и без вывода скрытых файлов мне никак не прожить. И что же — мне каждый раз вводить кучу опций, чтобы получить столь элементарный эффект?

Отнюдь — ответил бы граф, стуча манжетами о подоконник. Потому что этот вопрос задавали себе многие поколения не только пользователей, но и разработчиков. И ответили на него просто — введением понятия псевдонима команды (alias).

Что это такое? В большинстве случаев — просто некоторое условное имя, подменяющее определённую команду с теми её опциями, которые мы используем чаще всего. Причём, что характерно, псевдоним команды может совпадать с ее именем. То есть, например, — набирая просто ls, мы получаем список файлов не в умолчальном формате, а в том, в каком угодно нам.

Устанавливаются псевдонимы очень просто — одноименной командой alias, в качестве аргументов которой выступают имя псевдонима и его значение, соединенные оператором присваивания (именуемым в просторечии знаком равенства). А именно, если мы хотим ныне, и присно, и во веки веков видеть вывод команды ls с символьным различением типов файлов, нам достаточно дать команду вроде следующей:

```
$ alias ls='ls -F
```

Здесь следует обратить внимание на два момента: а) на то, что имя псевдонима совпадает с именем команды (что отнюдь не препятствует создания псевдонима типа ll='ls -l' специально для вывода файловых списков в длинном формате), и б) на одинарные кавычки, в которые заключено значение псевдонима. Смысл их станет ясен несколькими параграфами позже, а пока просто запомним, что кавычки (и именно одинарные) — обязательный атрибут команды установки псевдонима.

Таким образом мы можем наделать себе псевдонимов на все случаи жизни. В разумных пределах, конечно — иначе вместо упрощения жизни мы создадим себе необходимость запоминания множество невнятных сочетаний символов. Однако на наиболее важных (и обычно определяемых) псевдонимах я остановлюсь.

Вспомним команды типа cp и mv, которыми мы, в частности, можем скопировать или переместить какие-то файлы из каталога в каталог. А что произойдет, если чисто случайно в целевом каталоге уже имеются файлы, одноименные копируемым/перемещаемым? Произойдет штука, могущая иметь весьма неприятные последствия: файлы в целевом каталоге будут заменены новыми, теми, что копируются туда или перемещаются. То есть исходное содержание этих файлов будет утрачено — и безвозвратно, восстановить его будет невозможно никакими силами.

Разумеется, иногда так и нужно, например, при полном резервном копировании старые версии файлов и должны быть заменены их более свежими вариантами. Однако такое приемлемо далеко не всегда. И потому в большинстве команд, связанных с необратимыми изменениями файловой системы, предусматривается специальная опция — -i (или --interactive). Если

задать эту опцию с командой cp или mv, то при совпадении имён исходного и целевого файлов будет запрошено подтверждение на выполнение соответствующего действия:

```
$ cp file1 file2

cp: overwrite  file2'?
```

И пользователь может решить, нужно ли ему затирать существующий файл, ответив yes (обычно достаточно y), или это нежелательно, и должно ответить no (а также просто n — или не отвечать ничего, это равноценно в данном случае отрицательному ответу).

Так вот, дабы не держать в голове необходимость опции -i (ведь, как я уже говорил, пропуск ее в неподходящий момент может привести к весьма печальным результатам), в подавляющем большинстве систем для команд cp и mv (а также для команды rm, служащей для удаления файлов — эта операция также практически необратима) определяются одноименные им псевдонимы такого вида:

```
$ alias cp='cp -i';

$ alias mv='mv -i';

$ alias rm='rm -i'
```

Все это, конечно, очень благородно, заметит внимательный читатель. Но что, если мне заведомо известно, что сотни, а то и тысячи файлов целевого каталога должны быть именно переписаны новыми своими версиями? Что же, сидеть и, как дурак, жать на клавишу Y?

Не обязательно. Потому что все команды рассматриваемого класса имеют ещё опцию -f (в «длинной» своей форме, --force, она также практически универсальна для большинства команд). Которая, отменяя действие опции -i, предписывает принудительно переписать все файлы целевого каталога их обновленными тезками. И никто не мешает нам на этот случай создать ещё один псевдоним для команды cp, например:

```
$ alias cpf='cp -f'
```

Правда, предварительно нужно убедиться, что в системе нет уже команды с именем, совпадающим с именем псевдонима — иначе эффект может быть весьма неожиданным (впрочем, это относится ко всем псевдонимам, не совпадающим с именами подменяемых команд).

Есть и другой способ обойти опции, установленные для команды-псевдонима: просто отменить псевдоним. Что делается командой обратного значения

```
$ unalias alias_name
```

То есть дав директиву

```
$ unalias cp
```

мы вернем команде копирования ее первозданный смысл. Ну а узнать,

какие псевдонимы у нас определены в данный момент, и каковы их значения, ещё проще: команда

```
$ alias
```

без опций и аргументов выведет полный их список:

```
la='ls -A'

less='less -M'

li='ls -ial'

ll='ls -l'

ls='ls -F --color=auto'
```

и так далее.

Когда я сказан о пользовании псевдонимами ныне, и присно, и вовек, — то был не совсем точен. Ныне, то есть в текущем сеансе пользователя — да, они работают. Однако после рестарта системы (или просто после выхода из данного экземпляра командной оболочки) они исчезнут без следа. Чтобы заданные псевдонимы увековечить, их нужно прописать в конфигурационном файле пользовательского шелла. Но этим мы займемся впоследствии. А пока обратимся к переменным.

Переменные

Переменные играют для аргументов команд примерно такую же роль, что и псевдонимы — для команд. То есть избавляют от необходимости мрачного ввода повторяющихся последовательностей символов. Конечно, это — далеко не единственное (а может быть, и не главное) назначение переменных, однако на данном этапе для нас наиболее существенное.

Что такое переменная? Ответ просто — некоторое имя, которому присвоено некоторое значение. Не очень понятно? — Согласен. Но, возможно, станет яснее в дальнейшем.

Имена переменных в принципе могут быть любыми, хотя некоторые ограничения также существуют. Я уже вскользь упоминал о переменных в разговоре про пути к файлам, где фигурировала переменная PATH. Когда дело дойлёт у нас до пользовательских аккаунтов, придётся поговорить о переменных SHELL, USER, HOME.

Все эти (и ещё некоторые) имена зарезервированы за внутренними, или встроенными, переменными оболочки (некий минимальный их набор имеется в любом шелле). То есть значения их определены раз и навсегда. и пользователем не изменяются. То есть он, конечно, может их изменить, если очень хочет — но ничего доброго, кроме путаницы, из этого не выйдет.

Таких встроенных переменных довольно много. Одна из первых по значению — всё та же переменная PATH. Это — список каталогов, в которых оболочка, в ответ на ввод пользователя в командной строке, ищет исполнимые файлы — то есть просто команды. Я уже обращал внимание, что во всех приведённых выше примерах имена команд указывались без всяких

путей к ним (в отличие от файлов-аргументов, путь к которым — обязателен). Так вот, успех её поисков и определяется списком значений переменной PATH. Каковые могут быть просмотрены командой echo:

```
$ echo $PATH
```

Обратим внимание на то, что в качества аргумента команды выступает не просто имя переменной, а оно же, но предваренное символом доллара. Который в данном случае никакого отношения к приглашению командной строки не имеет, а предписывает команде echo подменить имя переменной ее значением (значениями). В большинстве дистрибутивов Linux случае вывод команды для пользователя будет в обязательном порядке включать такие каталоги:

```
/bin:/usr/bin:/usr/local/bin
```

Для администратора системы сюда обязательно добавятся каталоги /sbin, /usr/sbin и /usr/local/sbin. Остальные значения переменной PATH могут варьировать по умолчанию, а также задаваться пользователем (как — поговорим позже).

Обратим вниммание на одно важное обстоятельство: практически во всех дистрибутивах Linux и в более иных ОС в перечне значений переменной PATH отсуствует текущий каталог

Тем временем вернемся к переменной HOME. Значение ее — полный абсолютный путь к домашнему каталогу пользователя. То есть, чтобы перейти в него, пользователю по имени alv вместо

```
$ cd /home/alv
```

достаточно набрать

```
$ cd $HOME
```

и он в своих владениях. Может показаться, что экономия — грошовая (тем паче, что перейти в собственный каталог пользователь может просто командой cd без всяких аргументов), но минуту терпения — и выгоду от использования переменных вы увидите.

Кроме переменных, предопределенных в шелле, пользователю предоставляется почти полная свобода в определении переменных собственных. И вот тут-то и наступает ему обещанное облегчение при наборе аргументов команд.

Предположим, что у нас имеется глубоко вложенный подкаталог с данными, постоянно требующимися в работе. Чисто условно примем, что путь к нему — следующий:

```
/home/alv/data/all.my.works/geology/plate-tectonics
```

Весьма удручающе для набора, даже если исправно работать табулятором для автодополнения, не так ли? Прекрасно, упрощаем себе жизнь определением переменной:

```
$ plate=/home/alv/data/all.my.works/geology/plate-tectonics
```

Дело в шляпе, Теперь, если нам нужно просмотреть состав этого каталога, достаточно будет команды

```
$ ls $plate
```

А вызвать из него любой файл для редактирования можно так:

```
$ joe $plate/filename
```

Подобно псевдонимам, переменные, определенные таким образом (то есть просто в командной строке), имеют силу только в текущем сеансе работы — по выходе из оболочки они утрачиваются. Для того, чтобы они действовали перманентно, переменные должны быть прописаны в конфигурационном файле пользовательского шелла. Однако, в отличие от псевдонимов, и этого оказывается не всегда достаточно. Ибо переменная, определенная посредством

```
$ NAME=Value
```

работает не просто только в текущем сеансе — но ещё и только в конкретном экземпляре шелла. Почему и называется переменной оболочки — shell variable. Звучит это. быть может, пока не очень понятно. Однако практически любое действие в шелле — запуск команды или программы, например, — начинается с того, что оболочка, в которой это действие совершается, запускает новый экземпляр самой себя — дочерний шелл, или, как иногда говорят, субшелл.

Так вот, этот самый субшелл не наследует переменные родительской оболочки. И в итоге запущенная из командной строки программа ничего не будет знать, например, о путях к исполняемым файлам. Что автоматически ведет к невозможности запуска из нее команд просто по имени, без указания точного пути.

Чтобы избежать такой неприятной ситуации, было придумано понятие переменных окружения, или переменных среды — environment variable. Это — те переменные, которые наследуются от родительского шелла всеми дочерними программами. И чтобы сделать их таковыми, переменные следует экспортировать. Как? Командой export, которая может быть применена двояким образом. Можно сначала определить переменную:

```
$ NAME=Value
```

а затем применить к ней команду export:

```
$ export NAME
```

А можно сделать это в один прием:

```
$ export NAME=Value
```

Второй способ применяется, если нужно определить и экспортировать одну переменную. Если же за раз определяется несколько переменных:

```
$ NAME1=Value1;
```

```
$ NAME2=Value2;
```

```
...;

$ NAMEN=ValueN
```

то проще прибегнуть к первому способу, так как команда export может иметь сколько угодно аргументов:

```
$ export NAME1 NAME2 ... NAMEN
```

Традиционно имена переменных окружения задаются в верхнем регистре, переменных оболочки — в нижнем.

Навигация и редактирование

Имя команды, ее опции и аргументы образуют т.н. командные «слова». В качестве словоразделителей выступают пробелы. Кроме того, как разделители «слов» интерпретируется ряд специальных символов — прямой слэш (/) — элемент пути к файлу, обратный слэш (\), служащий для экранирования специальных символов, и операторы командных конструкций, о которых будет сказано ниже.

В некоторых случаях имеет смысл различать «большое слово» и «малое слово». Первые разделяются пробелами, в качестве же вторых интерпретируются символы, лежащие между всеми другими словоразделителями.

Подчеркнем, что командное «слово» прямо не соотносится ни с опциями, ни с аргументами команды. Введение этого понятия призвано просто облегчить навигацию в командной строке и ее редактирование.

Ибо одно из великих достижений командного интерфейса POSIX-систем, заценить которое могут в полной мере только те, кто застал времена «черного DOS'a», — это возможность перемещёния внутри командной строки и внесения необходимых изменений в имя команды, ее опции и аргументы. Делается это различными способами.

Самый привычный и, казалось бы, очевидный способ — использование клавиш перемещёния курсора Left, Right, End и Home, действующих (хотя и не всегда) в командной строке точно так же, как и в каком-нибудь ворд-процессоре для Windows (клавиши Up, Down, PageUp, PageDown зарезервированы для других целей). То есть они позволяют перемещаться на один символ влево и вправо. в начало и конец командной строки. А если добавить сюда ещё клавиши Delete и Backspace, позволяющие удалять символы в позиции курсора или перед ней — то, казалось бы, чего ещё желать?

Оказывается — есть чего, и самый очевидный способ навигации и редактирования оказывается не самым эффективным. Для начала заметим, что в общем случае привычные клавиши перемещёния курсора и редактирования в POSIX-системах не обязаны работать также, как они делают это в DOS/Windows. Это зависит от многих причин, в том числе и исторических. Ведь POSIX-системы по определению предназначены работать на любых практически машинах (в том числе и на тех, клавиатуры которых клавиш управления курсором просто не имели).

Однако это не главное — в большинстве Linux-дистрибутивов командная

оболочка по умолчанию настраивается так, чтобы пользователь при желании мог использовать привычные ему клавиши. Однако тут-то и оказывается, что плюс к этому оболочка предоставляет ему много более эффективную систему навигации по командной строке и ее редактирования. И это — система управляющих последовательностей, так называемых keybindings. То есть сочетания специальных клавиш, именуемых управляющими, с обычными алфавитно-цифровыми.

Управляющие последовательности

Основные управляющиеся клавиши, которые используются в таких последовательностях (и имеются на клавиатурах почти любых машин — как говорят в таких случаях, в любых типах терминалов) — это клавиши Control и Meta.

Пардон — возразит внимательный читатель, — сколько я ни долблю по клавишам моей PC'шки, но клавиши Meta не замечал. Возражение принято, но: на PC-клавиатурах функции Meta выполняют либо а) нажатие и отпускание клавиши Escape, либо б) нажатие и удерживание клавиши Alt.

Впрочем, к этой теме я ещё вернусь. А пока достаточно нескольких простых рецептов, практически универсальных для любых командных оболочек в терминалах любых типов.

Рецепт первый: большая часть управляющих последовательностей состоит из сочетания клавиши Control и алфавитно-цифрового символа. Под сочетанием (или комбинацией, для чего я уже употреблял ранее символ плюс) понимается то, что, удерживая нажатой клавишу Control, мы одновременно нажимаем и какую-нибудь литерную или цифровую.

Так, действие клавишной комбинации Control+F (от Forward — в большинстве случаев регистр алфавитной клавиши управляющей последовательности значения не имеет) эквивалентно нажатию клавиши Right — это перемещёние на один символ вправо, комбинации Control+B (от Back) — нажатию Left (перемещёние на один символ влево). Комбинации Control+A и Control+E действуют аналогично Home и End, перемещая курсор в начало и конец командной строки, соответственно, Ну а с помощью комбинаций Control+D и Control+H можно удалить единичный символ в позиции курсора или перед ней (также, как и клавишами Delete и Backspace, соответственно).

Предвижу резонный вопрос: а какие достоинства в комбинации клавиш Control+Что_то по сравнению с элементарными End или Left? Конечно, одно достоинство — очевидно: при массовом вводе команд (а также, забегая вперед, замечу — и любых иных наборов символов, от исходных текстов до романов), при использовании keybindings руки не отрываются от основной (алфавитно-цифровой) части клавиатуры. И в итоге, по приобретении некоторого минимального навыка, дело движется ну гораздо быстрее. Обосновать тестами не могу (тут какая-нибудь физиометрия понадобится), но не верящим — предлагаю попробовать.

Главное же преимущество клавиатурных последовательностей перед стандартными навигационными клавишами — много более широкая их функциональность. Я не случайно начал этот параграф с упоминания командных «слов» — это, наряду с единичными символами, также навигационные (и, добавлю, редакционные) единицы командной строки. То

есть управляющие последовательности позволяют при навигации и редактировании оперировать не только единичными символами, но и целыми словами.

Например, комбинация Meta+F смещает курсор на одно «слово» вперед, та же Meta в сочетании с B — на одно слово назад, и так далее. Прошу обратить внимание: действие алфавитной клавиши в комбинации с Meta сходно по смыслу ее сочетанию с клавишей Control, но как бы «усилено»: последовательность Meta+D уничтожает не символ в позиции курсора, как это было бы для D в сочетании с Control, а все командное «слово».

Рассматривать ключевые последовательности подробно здесь я не буду: детали их действия зависят от командной оболочки и ее настроек. Отмечу только два существенных обстоятельства. Первое: keybindings предоставляют пользователю полный комплекс приемов для любых действий в командной строке — вплоть до преобразования регистров уже введенных символов и «слов» (из нижнего в верхний и наоборот), «перетасовки» символов в команде или ее аргументах, и так далее.

Значение управляющих последовательностей не ограничивается командной строкой — большинство популярных в POSIX-мире текстовых редакторов, от простых Nano или joe до грандиозного vim и монструозного emacs. построены по тому же принципу. Так что навыки, полученные при работе с keybindings, например, в Bash, весьма поспособствуют виртуозному освоению любого из этих инструментов.

И второе — действие ключевых последовательностей, как правило. не зависит не только от типа терминала и физического устройства клавиатуры, но и от ее раскладки — при переключении на кириллицу они будут работать столь же справно, как и в латинице.

История команд

Возможности навигации и редактирования строки особенно ярко проявляются в сочетании с другой замечательной особенностью, предоставляемой командными оболочками — доступом к истории команд. То есть: раз введенная в строке команда не уходит в небытие после исполнения, а помещается в специальный буфер памяти. Который, как и все в Unix'ах, именуется весьма незатейливо — буфер истории команд. Откуда команда (со всеми её опциями и аргументами) может быть извлечена для повторного использования. Или — для редактирования и исполнения в новой реинкарнации.

Буфер истории команд сохраняется в течении всего сеанса работы. Однако в большинстве случаев командные оболочки настраиваются так, что по выходе из сеанса буфер истории сохраняется в специальном файле в домашнем каталоге пользователя, и таким образом его содержимое оказывается доступным при следующем запуске шелла. Имя этого файла может быть различным в разных оболочках, но обычно включает компонент history (в Bash — ~/.bash_history).Так что, можно сказать, что введенным нами командам суждена вечная жизнь.

Конечно, не совсем вечная. И размер буфера истории команд, и количество строк в файле истории — величины конечные. Так что, если установленный предел превышен, то старые команды вытесняются более новыми. Однако и величину буфера, и количество строк в файле истории можно установить

любыми. Разумеется, в разумных пределах — не знаю, существует ли принципиальное ограничение на их размер, за исключением объёма памяти и дискового пространства. А если учесть, что и из буфера, и из памяти с помощью соответствующих настроек (со временем я расскажу, каких) можно исключить дубликаты и ещё кое-какой мусор — то мое заявление о вечной жизни команд не выглядит столь уж преувеличенным.

Универсальное средство доступа к буферу истории команд — специальная команда, встроенная во все шеллы, таковой поддерживающие — history (в большинстве дистрибутивов Linux она по умолчанию имеет псевдоним — h). Данная без опций, эта команда выводит полный список команд в их исторической (издревле к современности) последовательности, или некоторое количество команд, определенных соответствующими настройками (о которых будет говориться позднее).

В качестве опции можно указать желаемое количество одновременно выведенных команд. Например, директива

```
$ history -2
```

выведет две последние команды из буфера истории вместе с их номерами:

```
1023  joe shell.html
```

```
1024  less ~/.zshrc
```

Любая из команд в буфере истории может быть повторно запущена на исполнение. Для этого достаточно набрать в командной строке символ ! (восклицательный знак) и затем, без пробела — номер команды в списке буфера. Например,

```
$ !1023
```

для приведенного выше примера повторно откроет файл shell.html в текстовом редакторе joe.

Другой способ доступа к командам из буфера истории — комбинации клавиш Control+P и Control+N, служащие для последовательного его просмотра (как бы «пролистывания») назад и, соответственно, вперед (разумеется, если есть куда). Они дублируются клавишами управления курсором Up и Down (назад и вперед, соответственно). Кроме того, последовательности Meta+< и Meta+&rt; обеспечивают переход к первой и последней команде в буфере истории.

Любая извлеченная (с помощью стрелок или управляющими последовательностями) из буфера истории в текущую строку команда может быть повторно запущена на исполнение — нажатием клавиши Enter или дублирующей ее комбинацией Control+M. причём предварительно ее можно отредактировать — изменить опции, или аргументы, — точно так же, как и только что введенную.

Поиск в истории

Во всех современных «развитых» шеллах предусмотрены средства поиска команды в буфере истории — простым перебором (обычно Meta+P — назад и Meta+N — вперед).

Впрочем, не смотря на громкое название, обычный поиск ничем практически не отличается от пролистывания исторического списка курсорными стрелками. Что при обширной истории команд может быть весьма утомительным. И потому для ее облегчения предусмотрена такая интересная возможность, как наращиваемый поиск (incremental search) нужной команды в буфере истории по одному (или нескольким) из составляющих ее символов.

Выполняется инкрементный поиск так: после нажатия (при пустой командной строке) клавишной комбинации Control+R появляется предложение ввести алфавитный символ (или — последовательность символов произвольной длины), заведомо входящий в состав требуемой команды:

```
$ bck-i-search: _
```

Ввод такого символа выведет последнюю из команд, его содержащих. При этом введенный символ будет отмечен знаком курсора. Он не обязан входить в имя команды, но может быть составляющим ее опций или аргументов (имени файла или пути к нему, например). Следующее нажатие Control+R зафиксирует курсор на предыдущем символе, в пределах этой же или более ранней по списку команды, и т.д. Однако вместо этого в строке поиска можно вводить дополнительные символы, детализирующие условия поиска команды (или — ее опций и аргументов).

Процедуру поиска можно продолжать вплоть до достижения требуемого результата — то есть нахождения той команды, которая нужна именно сейчас. Нажатие клавиши Enter в любой из этих моментов запускает найденную (то есть помещённую в командную строку) команду на исполнение, с завершением поиска. Поиск обрывается также и нажатием комбинации Control+C. Перед запуском найденная команда может быть отредактирована стандартными средствами — с использованием управляющих последовательностей.

Регулярные выражения

Как известно, все пользователи-POSIX'ивисты должны быть в обязательном порядке привержены одному из семи смертных грехов. И грех этот — леность, можно сказать, показатель профессиональной пригодности линуксоида. В соответствие со своей леностью разработчики POSIX-систем придумывают способы, как бы им минимизировать свои усилия. А применители из лени изощряются в использовании этих приемов на практике. В частности — в том, как свести к минимуму набор в командной строке.

Собственно говоря, этой цели служили почти все приемы, описанные выше. Осталось осветить немногое. А именно — регулярные выражения, реализуемые с помощью т.н. специальных символов (или метасимволов).

Элементарная, и весьма частая, в духе школьных, задача: из каталога dir1 требуется скопировать все файлы в каталог dir2. Так неужели все они должны быть перечислены в качестве аргументов команды cp? Нет, нет, и ещё раз нет. Ибо для этой цели придуманы шаблоны имен файлов. Самый часто используемый из них — специальный символ * (вроде бы я о нем уже говорил?). Он подменяет собой любое количество любых символов (в том числе — и нулевое, то есть отсутствие символов вообще). То есть для решения предложенной задачи нам достаточно дать команду:

```
$ cp dir1/* dir2
```

Чуть усложним условия: к копированию из dir1 предназначены не все файлы, а только html-документы, традиционно имеющие суффикс html. Решение от этого не становится сложнее:

```
$ cp dir1/*html dir2
```

Однако тут можно вспомнить, что html-документы могут иметь и расширение htm. Не пропустим ли мы их таким образом при копировании? Таким — безусловно, пропустим. Однако нам на помощь придет другой шаблон — символ ?. А соответствует он любому единичному символу (или — его отсутствию, т.е. символу null). И значит, если команда из примера будет модифицирована таким образом:

```
$ cp dir1/*htm? dir2
```

то она гарантированно охватит все возможные маски html-документов.

Вроде все хорошо. Однако нет: из каталога dir1 нам нужно скопировать только три определенных файла — file1, file2, file3. Не придется ли каждый из них указывать в командной строке с полным путем (а ведь они могут быть и в глубоко вложенном подкаталоге типа dir1/dir11/dir111)? Все равно не придется, на столь хитрую... постановку задачи у нас есть прием с левой резьбой — символы группировки аргументов, обозначаемые фигурными скобками. Что на практике выглядит так:

```
$ cp path/{file1,file2,file3} dir2
```

И приведет к единоразовому копированию всех трех файлов в каталог dir2. Заметим, что сгруппированные аргументы разделяются запятыми без пробелов. И ещё: в оболочке Bash группируемые аргументы придется полностью вводить руками. Но вот в Zsh на них распространяется возможность автодополнения, да и запятая после каждого имени появляется автоматически (и столь же автоматически исчезает при закрытии фигурной скобки).

Группировка аргументов может быть сколь угодно глубоко вложенной. Так, команда

```
$ mkdir -p dir1/{dir11/{dir111,dir112},dir12/{dir121,dir122}}
```

в один заход создаст трехуровневую структуру каталогов внутри текущего — если только не забыть про опцию -p, которая предписывает создавать промежуточные подкаталоги в случае их отсутствия.

И ещё несколько примеров. Регулярное выражение для диапазона — то есть вида [...], подменяет любой из символов, заключенных в квадратные скобки. Символы эти могут даваться списком без пробелов (например, выражение [12345] соответствует любому символу от 1 до 5) или определяться в диапазоне, крайние значения которого разделяются дефисом без пробелов (эквивалентное первому выражение — [1-5]). Кроме того, символ ^, предваряющий список или диапазон, означает отрицание: выражение [^abc] подменяет любой символ, исключая символы a, b и c.

Последние примеры регулярных выражений могут показаться

надуманными. Однако представим. что в том же каталоге dir1, кроме html-документов, содержатся также файлы изображений в различных форматах — GIF, JPEG, TIFF и так далее (традиционно имеющие одноименные расширения). И все они должны быть скопированы в каталог dir2, а вот как раз html-файлы нам в данный момент без надобности. No problemas, как говорят у них:

```
$ cp dir1/*[^html] dir2
```

И в каталоге dir2 окажется все содержимое каталога dir1, за исключением html-файлов.

Экранирование

Из приведённых примеров можно видеть, что метасимволы, образующие регулярные выражения, интерпретируются командной оболочкой особым образом, не так, как обычные алфавитно-цифровые символы, составляющие, скажем, имена файлов.

В то же время собственно POSIX-системы накладывают на имена файлов очень мало ограничений. И в принципе система не запретит вам создать файл с именем, содержащим метасимволы. Другое дело, что работать с таким образом именованными файлами может быть сложно — командная оболочка будет пытаться интерпретировать их в соответствии с правилами для регулярных выражений.

Конечно, использовать метасимволы в именах файлов весьма не рекомендуется. Однако а) возможны элементарные ошибки при наборе, и б) файлы, полученные при обмене с другими операционными системами (сами знаете. какими), могут иметь довольно непривычный (и, я даже сказал бы, неприличный) вид.

Вспомним, что MS Word в качестве имени файла спокойно берёт первую фразу документа. А если это — вопрос? И тогда завершающий имя символ ? будет в шелле интерпретироваться как шаблон, а не как элемент имени. Думаю, не нужно обладать очень развитым воображением, чтобы представить последствия. Что делать в таких ситуациях? Для их разрешения резонными людьми придумано было понятие экранирования.

Начнём с первого примера использования экранирования — разрыва длинных строк. Командные директивы, с многочисленными их опциями, особенно в полной форме, и аргументами могут оказаться весьма длинными, не укладывающимися в пределы экранной строки. Правда, обычно командная оболочка по умолчанию настраивается с разрешением так называемого word wrapping'a (то есть переноса «слов» команды без обрыва строки — последнее, как мы помним, достигается нажатием клавиши Enter или комбинации Control+M и приводит к немедленному исполнению введённой команды. Если ввод ее не окончен — последует сообщение об ошибке). Однако перенос «слов» при этом происходит, как бог на душу положит. И в результате командная директива теряет читабельность и становится сложной для понимания.

Тут-то и приходит на помощь понятие экранирования, упомянутое абзацем выше. Знак экранирования — обратный слэш (\), — превращает символ, имеющий специальное значение, например, упоминавшийся ранее шаблон в именах файлов — *, в самую обычную звездочку. А раз конец строки — тоже символ, хотя и специальный, то и он доступен для экранирования. Так что

если завершить введённый фрагмент команды обратным слэшем (некоторые оболочки требуют предварить его пробелом, и лучше так и делать, хотя в Bash или Zsh пробел не обязателен), после чего нажать Enter, то вместо попытки исполнения будет образована новая строка. в которой можно продолжать ввод. Вид приглашения к вводу при этом изменится — это будет так называемое вторичное приглашение командной строки, и его представление настраиваемо, также как и вид приглашения первичного.

Возвращаемся к экранированию обратным слэшем. Действие его распространяется только на непосредственно следующий за ним символ. Если символы, могущие быть воспринятые как специальные, идут подряд, каждый из них должен предваряться обратным слэшем.

У обратного слэша есть ещё одна интересная особенность — я назвал бы ее инвертированием специального значения символов. Для примера: некая последовательность цифр (например, 033), введенная в командной строке, будет воспринята как набор обычных символов. Однако она же может выступать как код какого-либо символа (в частности, 033 — код символа Escape в восьмеричной системе счисления). И подчас возникает необходимость ввода таких кодов (тот же код для Escape, скажем, затруднительно ввести каким-либо иным образом).

И вот тут обратный слэш проявляет свое инвертирующее действие: последовательность \033 будет восприниматься уже не как набор символов, а как код символа Escape (обратим внимание, что тут достаточно единичного слэша). Непосредственно в командной строке такой способ инвертированного экранирования, по понятным причинам, обычно не используется, но находит широкое применение в сценариях.

О кавычках

Есть и экраны, распространяемые на все, что заключено внутри них. Это — кавычки, двойные и одинарные: большая часть символов между ними утрачивает свое специальное значение. Но не все: в двойных кавычках сохраняют специальное значение метасимволы $ и \, а также обратные кавычки (`), о назначении которых я скажу чуть позже. То есть в них сохраняется возможность, с одной стороны, получения значений переменных (как мы помним, с помощью $ИМЯ). А с другой стороны, если нам требуется дать символ бакса в его прямом и привычном значении, у нас есть возможность заэкранировать его обратным слэшем. И если потребуется вывести на экран сообщение «с вас, уважаемый, пятьсот баксов», то это можно сделать таким образом:

```
$ echo "с вас, уважаемый, \$500"
```

ещё одно широко применяемое использование двойных кавычек — экранирование пробелов, предотвращающих разбиение аргументов команды на отдельные «слова». Правда, в случае с командой echo это, как правило, не требуется (хотя настоятельно рекомендуется экранировать ее аргумент таким образом). Однако представьте, что в качестве аргумента команды копирования и перемещёния выступает файл, переписанный с Windows-машины. Ведь там пробелы в именах — вещь обычная. Тут-то экранирование двойными кавычками и придется к месту.

Из сказанного понятно, почему двойные кавычки именуются ещё неполными, или не строгими — они все же допускают внутри себя

использование символов со специальными значениями. В противоположность им, кавычки одинарные носят имя строгих, или полных. Потому что между ними утрачивают специальное значение все метасимволы, кроме их самих — в том числе и символ единичного экранирования. В итоге они используются там, где гарантированно требуется отсутствие специальных символов. Если вы помните, мы применили строгие кавычки при установке псевдонимов. Они же часто оказываются обязательными при определении переменных.

Завершая тему экранирования, осталось сказать только об обратных кавычках. Их функция очень узка: они служат для экранирования команд. То есть, скажем, команда

```
$ echo date
```

в полном соответствие со своим именем, просто выведет нам собственный аргумент:

```
date
```

Однако если аргумент команды закрыть обратными кавычками, то date будет воспринято как имя команды, подлежащей исполнению. И результат этого исполнения (то есть текущая дата и время — а именно для их получения и предназначена команда date) будет замещать имя команды в выводе echo:

```
$ echo `date`

Втр Дек 16 11:45:12 MSK 2003
```

Если вспомнить, что обратные кавычки сохраняют свое специальное значение внутри кавычек двойных, становится ясной польза от их применения: они незаменимы в тех случаях, когда требуется вывод результатов работы одной команды внутри другой. К как в нашем примере с выводом даты, если его (вывод) следует включить в некое выдаваемое командой echo сообщение.

Конечно, в описанном случае добиться той же цели можно было бы гораздо легче — просто командой date. Однако представьте, что у нас возникло желание одновременно и получить сведения о количестве пользователей в системе (для чего предназначена команда who). Тут-то и выясняется. что проще всего это сделать командой типа следующей:

```
$ echo "На момент `date` в системе

      зарегистрированы `who`"
```

Ответом на что будет сообщение, подобное тому, что часто можно наблюдать на главной странице многих сайтов:

```
На момент Сб. дек. 20 06:05:56 MSK 2014 в системе

зарегистрированы alv      tty8         2014-12-15 00:34 (:0)

alv      pts/1        2014-12-15 00:34 (:0)

alv      pts/5        2014-12-19 09:37 (:0)
```

А теперь последнее, чем и закроем тему регулярных выражений вообще. В этом разделе рассматривалось использование метасимволов в командной оболочке (конкретно, в данном случае. в Dash, Bash и Zsh). В других оболочках применение метасимволов и условия их экранирования могут несколько отличаться. И к тому же многие запускаемые из строки шелла команды могут иметь свои правила построения регулярных выражений. Так что в итоге их форма определяется сочетанием особенностей конкретной оболочки и команды, из неё запущенной. Все это при необходимости будет оговариваться в дальнейшем.

А пока переходим к рассмотрению командных конструкций — одной из тех особенностей CLI, которая определяет мощь и универсальность этого интерфейса.

Вводные слова о командных конструкциях

Надеюсь, из того, что было рассказано на предшествующих страницах, посвящённых CLI, читателю стало ясно, что подавляющее большинство команд в POSIX-системах очень просты по сути и предназначены для выполнения какого-либо одного элементарного действия.

То есть команда cp умеет только копировать файлы, команда rm — только удалять их, но зато делают они это хорошо. Подчас — через чур хорошо, что мог ощутить на себе каждый, кому «посчастливилось» по ошибке выдать директиву вроде

```
$ rm -Rf *
```

Для тех, кто не испытал этого волнительного ощущения, поясню: результатом будет полное и безвозвратное уничтожение всех файлов от текущего каталога вниз (включая подкаталоги любой степени вложенности).

А если задать ту же команду от лица администратора, то, в зависимости от текущего положения на файловом древе, можно нечувствительно удалить что угодно, вплоть до системы целиком: одна из причине, почему повседневные действия не следует выполнять под root'ом.

Собственно, разделение любой задачи на серию элементарных операций — это и есть основной принцип работы в POSIX-системах, тот самый пресловутый Unix-way, о котором столько говорят его приверженцы.

Однако вслед за этапом решительного размежевания (эх, неистребимы в памяти нашего поколения слова товарища Ленина) должен наступить этап объединения, как за анализом явления следует синтез эмпирических данных о нём. И целям такого объединения служат командные конструкции.

Командные конструкции — очень важный компонент интерфейса командной строки. Они позволяют объединять несколько команд воедино и выполнять различные команды последовательно или параллельно. Для этого служат специальные символы — операторы: фонового режима, объединения, перенаправления и конвейеризации.

Совместное выполнение команд

Простейшая командная конструкция — это выполнение команды в фоновом режиме, что вызывается вводом символа амперсанда после списка опций и

(или аргументов):

```
$ command [options] [arguments] &
```

В Bash и Zsh пробел перед символом амперсанда не обязателен, но в некоторых шеллах он требуется, и потому лучше возвести его ввод (как и во всех аналогичных случаях) в ранг привычки. После этого возвращается приглашение командной строки и возможен ввод любых других команд (в том числе и фоновых). Команды для параллельного исполнения можно задать и в той же строке:

```
$ command1 & command2 & ... & commandN
```

В результате все команды, перечисленные в строке, кроме той, что указана последней, будут выполняться в фоновом режиме.

Существуют и конструкции для последовательного выполнения команд. Так, если ряд команд разделен в строке символом точки с запятой (;)

```
$ command1 ; command2 ; ... ; commandN
```

то сначала будет выполнена команда command1, затем — command1 и так далее. Молчаливо предполагается, что каждая из этих команд может иметь любое количество опций и аргументов. И, опять-таки, обрамление ; пробелами не обязательно во многих командных оболочках. Сами по себе команды не обязаны быть связанными между собой каким-либо образом — в сущности, это просто эквивалент последовательного их ввода в командной строке:

```
$ command1

$ command2

...
```

и так далее. При этом первая команда может, например, копировать файлы, вторая — осуществлять поиск, третья — выполнять сортировку, или другие действия. Очевидно, что в общем случае выполнение последующей команды не зависит от результатов работы предшествующей.

Однако возможна ситуация, когда результаты предыдущей команды из такой конструкции используются в команде последующей. В этом случае ошибка исполнения любой составляющей команды, кроме последней, делает невозможным продолжение работы всей конструкции. Что само по себе было бы ещё полбеды — однако в некоторых ситуациях исполнение последующей команды возможно только при условии успешного завершения предыдущей.

Характерный пример — сборка программы из ее исходных текстов, включающая три стадии — конфигурирование, собственно компиляцию и установку собранных компонентов. Что обычно выполняется последовательностью из трёх команд:

```
$ ./configure

$ make

$ make install
```

Ясно, что если конфигурирование завершилось ошибкой, то компиляция начаться не сможет и, соответственно, потом нечего будет устанавливать. И потому объединение их в последовательную конструкцию вида

```
$ ./configure ; make ; make install
```

может оказаться нецелесообразным.

Однако для предотвращения таких ситуаций в конструкции из взаимосвязанных команд существует другой оператор, обозначаемый удвоенным символом амперсанда — &&. Он указывает, что последующая команда конструкции должна исполняться только в том случае, если предыдущая завершилась успешно:

```
$ ./configure && make && make install
```

На практике обе приведённые в качестве примера конструкции дадут один и тот же результат — разумеется, если все составляющие их команды будут выполнены без ошибок. Однако в ряде иных случаев различие между этими конструкциями может быть существенным.

Впрочем, предусмотрена и командная конструкция, в которой последующей команде предписано исполняться в том и только в том случае, если предыдущая команда завершилась неудачно. Она имеет вид

```
$ command1 || command2
```

и может служить, в частности, для вывода сообщений об ошибках.

Перенаправление

Следующая командная конструкция — это так называемое перенаправление ввода/вывода. Чтобы понять,что это такое, нужно помнить две вещи:

1. любая команда получает данные для своей работы (например, список опций и аргументов) со стандартного устройства ввода (которым в первом приближении будем считать клавиатуру), а результаты своей работы представляет на стандартном устройстве вывода (коим договоримся считать экран монитора);

2. POSIX-системах любое устройство — не более чем имя специального файла, именуемого файлом устройства.

Таким образом, ничто не запрещает нам подменить специальный файл устройства ввода или устройства вывода любым иным файлом (например, обычным текстовым). Откуда и будут в этом случае браться входные данные или куда будет записываться вывод команды.

Перенаправление вывода команды обозначается следующим образом:

```
$ command > filename
```

или

```
$ command >> filename
```

В первом случае (одиночный символ >) вывод команды command образует содержимое нового файла с именем filename, не появляясь на экране. Или, если файл с этим именем существовал ранее, то его содержимое подменяется выходным потоком команды (точно также, как при копировании одного файла в другой, уже существующий). Почему такое перенаправление называется замещающим (или перенаправлением в режиме замещёния).

Во втором же случае (двойной символ >>) происходит добавление вывода команды command в конец существующего файла filename (при отсутствии же его в большинстве случаев просто образуется новый файл). И потому это называется присоединяющим перенаправлением, или перенаправлением в режиме присоединения.

Перенаправление ввода выглядит так:

```
$ command < filename
```

Простейший случай перенаправления вывода — сохранение результата исполнения команды в обычном текстовом файле. Например, конструкция

```
$ ls dir1 > list
```

создаст файл, содержанием которого будет список файлов каталога dir1. А в результате выполнения конструкции

```
$ ls dir2 >> list
```

к этому списку добавится и содержимое каталога dir2.

При перенаправлении ввода команда получает данные для своей работы из входящего в командную конструкцию файла. Например, конструкция

```
$ sort < list
```

выведет на экран строки файла list, отсортированных в порядке возрастания значения ASCII-кода первого символа, а конструкция

```
$ sort -r < list
```

осуществит сортировку строк того же файла в порядке, обратном алфавитному (вернее, обратном порядку кодов символов, но это нас в данном случае не волнует).

В одной конструкции могут сочетаться перенаправления ввода и вывода, как в режиме замещёния, так и в режиме присоединения. Так, конструкция

```
$ sort -r < list > list_r
```

не только выполнит сортировку строк файла list (это — назначение команды sort) в обратном алфавитному порядке (что предписывается опцией -r, происходящей в данном случае от reverce), но и запишет ее результаты в новый файл list_r, а конструкция

```
$ sort -r < list >> list
```

добавит по-новому отсортированный список в конец существующего файла

list.

Конвейеры

Возможности построения командных конструкций не ограничиваются перенаправлением ввода/вывода: результаты работы одной команды могут быть переданы для обработки другой команде. Это достигается благодаря механизму программных каналов (pipe) или конвейеров — последний термин лучше отражает существо дела.

При конвейеризации команд стандартный вывод первой команды передается не в файл, а на стандартный ввод следующей команды. Простой пример такой операции — просмотр списка файлов:

```
$ ls -l | less
```

Перенаправление вывода команды ls, то есть списка файлов, который при использовании полного формата записи (опция -l) может занимать многие экраны, на ввод команды less позволяет просматривать результат с ее помощью постранично или построчно в обоих направлениях.

Конвейеризация команд может быть сколь угодно длинной. Возможно также объединение конвейеризации команд и перенаправления в одной конструкции. Кроме того, команды в конструкции могут быть сгруппированы с тем, чтобы они выполнялись как единое целое. Для этого группа команд разделяется символами ; и пробелами, как при последовательном выполнении команд, и заключается в фигурные скобки. Так, если нам требуется перенаправить вывод нескольких команд в один и тот же файл, вместо неуклюжей последовательности типа

```
$ command1 > file ; command2 >> file ; ... ; commandN >> file
```

можно прибегнут к более изящной конструкции:

```
$ { command1 ; command2 ; ... ; commandN } > file
```

Как и многие из ранее приведённых примеров, этот может показаться надуманным. Однако представьте, что вам нужно создать полный список файлов вашего домашнего каталога, разбитый по подкаталогам, да ещё и с комментариями, в каком подкаталоге что находится. Конечно, можно вывести состав каждого подкаталога командой ls, слить их воедино командой cat (она предназначена, в частности, и для объединения — конкатенации, — файлов, и речь о ней будет позже), загрузить получившееся хозяйство в текстовый редактор или ворд-процессор, где добавить необходимые словеса. А можно — обойтись единой конструкцией:

```
$ { echo "List of my files" ;  > echo "My text" ;
      ls text/* ;  > echo "My images" ;
      ls images/* ;  > echo "My audio" ;
      ls audio/* ;  > echo "My video" ;
      ls video/* } > my-filelist
```

И в результате получить файл такого (условно) содержания, которое мы для разнообразия просмотрим с помощью только что упомянутой команды cat (благо и для просмотра содержимого файлов она также пригодна):

```
$ cat my-filelist

List of my files

My text

text/text1.txt text/text2.txt

My images

images/img1.tif images/img2.tif

My audio

audio/sing1.mp3 audio/sing2.mp3

My video

video/film1.avi video/film2.avi
```

Понятие о фильтрах

С понятием командных конструкций тесно связано понятие программ-фильтров. Это — команды, способные принимать на свой ввод данные с вывода других команд, производить над ними некоторые действия и перенаправлять свой вывод (то есть результат модификации полученных данных) в файлы или далее по конвейеру — другой команде.

Программы-фильтры — очень эффективное средство обработки текстов, и в своё время мы к ним вернемся для подробного изучения. Пока же важно отметить, что в качестве фильтров могут работать не все команды. Например, команды find или grep фильтруют имена файлов или фрагменты их содержимого, а команда ls фильтром не является.

Сценарии оболочки

Наш затянувшийся разговор о командах и командном интерфейсе подходит к концу. И в заключение этого раздела — ещё немного терпения. Потому что было бы несправедливо не уделить чуть-чуть места тому, что придает командному интерфейсу POSIX-систем его несравненную гибкость и универсальность. Заодно способствуя закоснению пользователя в смертном грехе лености. Итак — слово о сценариях оболочки.

В самом начале я обмолвился, что шелл — это не просто среда для ввода единичных команд и командных конструкций, но и ещё интерпретатор собственного языка программирования. Так вот, сценарии оболочки, именуемые также скриптами, — это и есть программы, написанные на этом языке.

Только не заподозрите меня в гнусном намерении учить вас программерству. Господь борони, и в мыслях не держал (тем паче, что и сам-то этим ремеслом не владею в должной для обучения других степени). Нет,

просто на последующих страницах нам то и дело придётся рассматривать кое-какие примеры готовых сценариев, а подчас и пытаться создавать их собственноручно. Ибо занятие это в элементарном исполнении навыков программирования не требует вообще.

В самом простом случае сценарий — это просто одна или несколько команд или (и) командных конструкций с необходимыми опциями и аргументами, сохраненные в виде обычного именованного текстового файла. И предназначены они в первую очередь для автоматизации часто исполняемых рутинных операций, в частности, ввода длинных последовательностей в командной строке.

Создание пользовательского сценария — просто, как правда. Для этого всего и нужно:

- создать командную конструкцию, достойную увековечивания;
- поместить ее в простой текстовый файл;
- по потребности и желанию снабдить комментариями;
- тем или иным способом запустить файл на исполнение.

С принципами создания команд и командных конструкций мы в первом приближении разобрались раньше. А вот способов помещёния их в файл существует множество. Можно просто ввести (или вызвать из буфера истории) нужную команду и оформить ее как аргумент команды echo, вывод которой перенаправить в файл:

```
$ echo "cp -rf workdir backupdir" > mybackup
```

Таким образом мы получили простейший скрипт для копирования файлов из рабочего каталога в каталог для резервного хранения данных, что впредь и будем проделывать регулярно (не так ли?).

Аналогичную процедуру можно выполнить с помощью команды cat — она, оказывается, способна не только к объединению файлов и выводу их содержимого, но и к вводу в файл каких-либо данных. Делается это так. Вызываем cat с перенаправлением ее вывода в файл:

```
$ cat > myarchive
```

и нажимаем Enter. После этого команда остается в ожидании ввода данных для помещёния их в новообразованный файл. Не обманем ее ожиданий и проделаем это. причём можно не жаться и выполнить ввод в несколько строк, например:

```
cd $HOME/archivedir tar cf archive.tar

      ../workdir gzip archive.tar
```

Завершив ввод тела скрипта, все той же клавишей Enter открываем новую строку и набираем комбинацию Control+D, выдающую символ окончания файла.

В результате получаем сценарий для архивирования в специально предназначенном для этого каталоге archivedir наших рабочих данных (командой tar), а заодно и их компрессии (командой gzip) — в Unix, в отличие от DOS/Windows, архивирование и компрессия обычно рассматриваются как разные процедуры.

Наконец, сценарий можно создать в любом текстовом редакторе. но это не так интересно — по крайней мере, пока. Да и стоит ли вызывать редактор ради двух-трёх строк?

Комментариями в шелл-сценариях считаются любые строки, начинающиеся с символа решетки (#) — они не учитываются интерпретатором и не принимаются к исполнению. Хотя комментарий может быть начат и внутри строки — важно только, что между символом # и концом её больше ничего не было бы. Ясно, что комментарии — элемент для скрипта не обязательный, но очень желательный. Хотя бы для того, чтобы не забыть, ради чего этот сценарий создавался.

Но одна строка, начинающаяся символом решётки, в сценарии практически обязательна. И должна она быть первой и выглядеть следующим образом:

```
#!/path/shell_name
```

В данном случае восклицательный знак подчеркивает, что предваряющий его символ решетки (#) — не комментарий, а указание (т.н. sha-bang) на точный абсолютный путь к исполняемому файлу оболочки, для которой наш сценарий предназначен, например,

```
#!/bin/sh
```

для POSIX-шелла, или

```
#!/bin/bash
```

для оболочки Bash. Здесь следует подчеркнуть, что шелл, для которого предназначается сценарий, отнюдь не обязан совпадать с командной оболочкой пользователя. И полноты картины для замечу, что указание точного имени интерпретатора требуется не только для шелл-скриптов, но и для программ на любых языках сценариев (типа Perl или Python).

Так что по хорошему в обоих приведенных выше примерах ввод команд сценария следовало бы предварить строкой sha-bang. Конечно, отсутствие имени командной оболочки в явном виде обычно не помешает исполнению шелл-сценария: для этого будет вызван системный командный интерпретатор по умолчанию — в Mint /bin/dash. Однако если сценарий предназначен для другой командной оболочки, то без sha-bang он может исполняться неправильно (или не исполняться вообще).

Теперь остается только выполнить наш сценарий. Сделать это можно разными способами. Самый напрашивающийся — непосредственно вызвать требуемый шелл как обычную команду, снабдив его аргументом — именем сценария (предположим, что он находится в текущем каталоге):

```
$ bash scriptname
```

Далее, для вызова скриптов существует специальная встроенная команда

оболочки, обозначаемая символом точки. Используется она аналогично:

```
$ . ./scriptname
```

с тем только исключением, что тут требуется указание текущего каталога в явном виде (что и символизируется ./).

Однако наиболее употребимый способ запуска сценариев — это присвоение его файлу так называемого атрибута исполнения. Эта процедура волшебным образом превращает невзрачный текстовый файлишко во всамделишную (хотя и очень простую) программу.

Так вот, после присвоения нашему сценарию бита исполнения запустить его можно точно также, как любую другую команду — просто из командной строки:

```
$ ./scriptname
```

Опять же — в предположении, что сценарий находится в текущем каталоге (./), иначе потребуется указание полного пути к нему. Что, понятно, лениво, но решается раз и навсегда: все сценарии помещаются в специально отведенный для этого каталог (например, $HOME/bin), который и добавляется в качестве ещё одного значения переменной PATH данного пользователя.

Понятие о функциях

И уж совсем в заключение этого раздела осталось сказать пару слов о функциях командной оболочки. Это — такая же последовательность команд (или даже просто одиночная команда), как и сценарий, но — не вынесенная в отдельный исполняемый файл, а помещённая в тело другого скрипта. В коем она опознаётся по имени, и может быть выполнена неоднократно в ходе работы этого скрипта.

Главное отличие функции от сценария — в том, что она выполняется в том же процессе (и, соответственно, экземпляре шелла), что и заключающий её сценарий. Тогда как для каждого скрипта, вызываемого из другого сценария, создаётся отдельный процесс, порождающий собственный экземпляр шелла. Это может быть важным, если в сценарии определяются некоторые переменные, которые имеют силу только в нём самом.

Функции не обязательно помещаются внутрь сценария — их можно собрать в некоторые отдельные файлы, которые именуются библиотеками функций и могут быть использованы по мере надобности.

Ранее на протяжении всего повествования неоднократно упоминались (и будут упоминаться впредь) системные библиотеки, в частности, главная библиотека glibc. Так вот, это — точно такие же сборники функций, правда, не командной оболочки, а языка Си, и, соответственно, хранящиеся не в виде текстовых файлов, а в бинарном, откомпилированном, виде.

Настройка шелла

Во всех дистрибутивах Linux в качестве пользовательской командной оболочки по умолчанию выступает Bash, и Mint здесь не исключение. Так что, хотя автор этих строк не является ни её любителем, ни, тем более, знатоком, совсем обойти её вниманиемне мог. Так что ниже даётся мини-очерк

настройки этого шелла.

Оболочка Bash поддерживает все интерактивные возможности, столь важные для пользователя, как то: автодополнение для команд и путей к файлам, историю оных (включая средства инкрементного поиска), мощные возможности навигации и редактирования командной строки.

Важно, что существует дополнительный пакет bash-completion: установка его обогащает базовую оболочку множеством опциональных средств настройки автодополнения (в том числе и для командных аргументов). Правда, чтобы эта дополненная оболочка была по настоящему удобной и функциональной, нужно приложить некоторые усилия по её настройке, чем мы сейчас и займёмся.

Схема настройки bash предусматривает наличие пары файлов /etc/profile и /etc/bashrc (для логин-шелла и просто интерактивного его экземпляра), а также соответствующих им пользовательских конфигов — ~/.bash_profile и ~/.bashrc. При авторизации первым в любом случае считывается общесистемный профильный файл /etc/profile, вслед за ним — пользовательский профильный файл ~/.bash_profile, после чего происходит обращение к ~/.bashrc. Файл /etc/profile может занимать особое положение — в него часто помещают переменные окружения (например, локально-зависимые), которые должны быть общими для всех пользователей данной системы. Пользовательские же настройки определяются в файлах ~/.bash_profile и ~/.bashrc. Обычно в ~/.bash_profile определяются переменные окружения, которые должны действовать для всех дочерних процессов, а в ~/.bashrc — параметры, всегда требуемые в интерактивном режиме (например, псевдонимы).

Редактирование командной строки в bash обеспечивается отдельным пакетом — библиотекой функций readline. Она имеет собственные конфигурационные файлы, общесистемный /etc/inputrc и пользовательский ~/.inputrc.

Впрочем, в большинстве современных дистрибутивов Linux, ориентированных на графический режим и, следовательно, использование эмулятора терминала с интерактивным шеллом, не являющимся, тем не менее, шеллом пользовательским (login shell), ~/.bash_profile играет сугубо служебную роль, и содержимое его сводится к отработке файла ~/.bashrc:

```
# include .bashrc if it existsif [ -f ~/.bashrc ]; then


. ~/.bashrc


fi# set PATH so it includes user's private bin if it existsif [ -d ~/bin ] ; then


PATH=~/bin:"${PATH}"
```

```
fi
```

Именно в ~/.bashrc и выполняются при этом все пользовательские настройки.

Большинство настроек Bash по умолчанию разумны, и потому наличные в данном дистрибутиве файлы вполне могут быть взяты за основу. Однако путём некоторых несложных действий их можно дополнить, увеличив удобство интерактивного использования командной оболочки.

По умолчанию в Bash автодополнение клавишей табулятора не работает, например, в аргументах многих команд, таких, как sudo или man.

Решается эта задача очень просто: достаточно файл ~/.bashrc внести следующие строки:

```
# enable bash completion in interactive shells

if [ -f /etc/bash_completion ]; then

 . /etc/bash_completion

fi
```

После этого автодополнение будет работать буквально везде, где только можно себе представить, например, после набора dpkg --sea и нажатия табулятора получится dpkg --search.

Если в файл /etc/inpurc (или в ~/inpurc) добавить такие строки:

```
"e[A": history-search-backward

"e[B": history-search-forward
```

то набор части команды, например, cd /, и последующий перебор стрелками <Up> и <Down> истории команд повлечёт извеление из буфера истории только тех из них, которые начинаются на cd /.

Утилиты CLI

В этом очерке будут рассмотрены утилиты командной строки разного назначения — комплекс так называемых классических UNIX-утилит в их современных свободных реализациях, используемых в дистрибутивах Linux, в том числе и в Mint.

Самая главная команда

Эта рубрика посвящена самой главной команде — man, а также сопутствующим ей материям. Содержание её — не информация о тех или иных командах, или свойствах системы, а метаинформация: информация о том, как получить нужную информацию. То есть выработке некоторых навыков, которые у применителя Linux должны быть доведены до уровня рефлексов.

Команд в свежеустановленной Linux-системе — немерянное количество, только консольных утилит под тысячу. Да ещё почти каждая команда имеет

опции, подчас также в немалом числе. И возникает естественный вопрос: как нормальный человек все это может запомнить? Да никак — последует ответ. Потому что запоминать все это изобилие команд нет не только возможности — но и ни малейшей необходимости: гораздо важнее понимать, каким образом соответствующую информацию можно получить в нужный момент. И тут нам на помощь приходит самая главная команда — команда man.

Команда man предназначена для вызова экранной документации в одноименном формате (Manual Pages, что на Руси ласково переводится как «тетя Маня»). А такая man-документация почти обязательно сопровождает любую уважающую себя программу для POSIX-систем. И устанавливается в принудительном порядке при инсталляции соответствующей программы в любом случае — разворачивается ли она из бинарного тарбалла или собирается из исходников.

Для файлов man-документации предназначен специальный каталог. Обычно это /usr/share/man, разделяемый на подкаталоги, соответствующие восьми нумерованным группам. Назначение этих групп следующее:

1. man1 — команды и прикладные программы пользователя;
2. man2 — системные вызовы;
3. man3 — библиотечные функции;
4. man4 — драйверы устройств;
5. man5 — форматы файлов;
6. man6 — игры;
7. man7 — различные документы, не попадающие в другие группы (в том числе относящиеся к национальной поддержке);
8. man8 — команды администрирования системы.

Кроме того, имеется несколько подкаталогов с локализованными man-страницами, в том числе и русскоязычными, имеющими ту же структуру, хотя и обычно не полную. Так, подкаталог с русскоязычными страницами, /usr/share/man/ru, включает в себя только группы man1, man5, man7 и man8.

Нас, применителей, в первую очередь интересуют команды из 1-й и, поскольку на персоналке каждый юзер — сам себе админ, из 8-й групп, хотя и об остальных категориях забывать не следует, иногда позарез нужные сведения отыскиваются в самой неожиданной из них.

Внутри групповых подкаталогов можно увидеть многочисленные файлы вида filename.#.gz. Последний суффикс свидетельствует о том, что man-страница упакована компрессором gzip. Цифра после имени соответствует номеру группы (то есть в подкаталоге man1 это всегда будет единица). Ну а имя man-страницы совпадает с именем команды, которую она описывает. Если, конечно, речь идет о команде — в разделе 2 оно будет образовано от соответствующего системного вызова, в разделе 2 — от имени функции, и так далее. Но пока нас интересует только информация о командах, так что дальше я этого оговаривать не буду.

Для вызова интересующей документации требуется дать команду man с аргументами — номером группы и именем man-страницы, например:

```
$ man 1 ls
```

Причём номер группы необходим только в том случае, если одноимённые документы имеются в разных группах. Для пользовательских команд он обычно не нужен, так как все равно просмотр групповых каталогов идёт сначала в man1, затем — в man8, и только потом — во всех остальных (в порядке возрастания номеров). Так что для получения информации, например, по команде ls достаточно ввести следующее:

```
$ man ls
```

после чего можно будет лицезреть примерно такую картину:

```
LS(1)   FSF     LS(1)

NAME

       ls — list directory contents

SYNOPSIS

       ls [OPTION]... [FILE]...

DESCRIPTION

       List  information  about

       the FILEs (the current directory by default).

       Sort entries alphabetically if none

       of -cftuSUX nor --sort.
```

То есть в начале man-страницы даются имя команды, которую она описывает (ls), ее групповая принадлежность (1 — пользовательские команды) и авторство (в данном случае — FSF, Free Software Foundations), или название системы. После чего обычно дается обобщенный формат вызова (SYNOPSIS) и краткое описание.

Следующая, основная, часть man-страницы — описание опций команды, и условия их применения. Далее иногда (но, к сожалению, не всегда) следуют примеры использования команды (Examples) в разных типичных ситуациях. В заключении, как правило, даются сведения о найденных ошибках (Bug Report) и приведен список man-страниц, тематически связанных с данной (See also), с указанием номера группы, к которой они принадлежат, иногда — историческая справка, а также указываются данные об авторе.

Большинство man-страниц занимают более одного экрана. В этом случае возникает необходимость перемещёния по экранам и строкам — т.е. некоторая навигация. Сама по себе команда man не отвечает не только за навигацию по странице. но даже за ее просмотр. Для этой цели она неявным образом вызывает специальную программу постраничного просмотре — т.н. pager (это — совсем не то, чем дети лохов в песочнице ковыряются). В Linux

таковым по умолчанию выступает уже известная нам команда less, но на эту роль можно назначить также more или most — это делается указанием значения соответствующей переменной, например:

```
export PAGER=most
```

в конфигурациооном файле пользователя.

Обращение к man-страницам позволяет получить практически исчерпывающую информацию по любым командам, но только в том случае, если пользователь знает название той команды, которая требуется в данном случае. А если он только в общих чертах представляет, что это команда должна делать?

Что ж, тогда можно прибегнуть к поиску man-страниц по ключевым словам, отражающим требуемые функции. Чему призвана служить опция -k команды man. Например, директива

```
$ man -k print
```

выведет на экран список всех man-страниц, описывающих команды, имеющие отношение к печати (причём не только на принтере, но и к выводу на экран — по английски подчас это тоже будет обозначаться как print).

Исчерпывающим руководством по использованию системы Manual Pages является ее собственная man-страница. Доступ к ней осуществляется по команде

```
$ man man
```

которая выводит на экран man-страницу, содержащую описание команды man (эко загнул, а?):

```
MAN(1)  FreeBSD General Commands Manual MAN(1)



NAME

     man — format and display the on-line

     manual pages



SYNOPSIS

     man [-adfhkotw] [-m machine]

     [-p string] [-M path] [-P pager]

     [-S list] [section] name ...



DESCRIPTION
```

Man formats and displays the on-line manual pages.

...

С навигационными возможностями команды less можно ознакомиться, нажав клавишу h — вызов встроенной её помощи. Из которой мы и узнаем, что перемещаться по man-странице можно с помощью управляющих последовательностей.

Управляющие последовательности команды less для большинства навигационных действий весьма разнообразны, но в принципе разделяются на две группы: чисто буквенные и состоящие из комбинаций Control+литера. Так, переместиться на одну строку вперед можно просто нажатием клавиши j, на одну строку назад — клавиши k, сместиться на экранную страницу — с помощью клавиш f (вперед) и b (назад). Однако того же результата можно доиться комбинациями клавиш Control+n и Control+p для построчного перемещёния и Control+v и Control+и — для постраничного (вперед и назад, соответственно).

Аналогично и для большинства других действий (смещёние на половину экранной страницы, например: Control+D и d — вперед, Control+U и u — назад) можно обнаружить минимум одну альтернативную пару управляющих последовательностей. Регистр символов обычно значения не имеет. Одно из исключений — нажатие клавиши g перемещает к первой строке man-страницы, клавиши G — к последней.

Наличие двух типов управляющих последовательностей может показаться излишним усложнением, однако имеет глубокое внутреннее обоснование. За исключением некоторых отщепенцев (в числе коих и автор этих строк), подавляющее большинство записных юниксоидов пользуются одним из двух редакторов — Vim или emacs.

Оба эти редактора относятся к категории командных. То есть все действия по редактированию осуществляются в них обычно не выбором пунктов из меню, а прямыми командными директивами, примерно как в командной строке оболочки. Так вот, одно из кардинальных различий между Vim и emacs — различие управляющих последовательностей для навигации по тексту и его редактированию. vi-образный стиль навигации основан на однобуквенных командных аббревиатурах (команды типа j или k пришли в less именно оттуда). Стиль emacs же подразумевает последовательности, образованные сочетанием клавиши Control и различных алфавитно-цифровых комбинаций.

Поскольку эффективное использование любого редактора командного стиля подразумевает доведенное до автоматизма использование управляющих последовательностей, переключение с vi-стиля на стиль emacs в этом деле может быть просто мучительным. Вот и предусмотрели разработчики pager'ов, в своей заботе о человеке, возможность использования и того, и другого стиля — кто к чему привык.

Раз уж зашла речь о стилях управляющих последовательностей... В большинстве командных оболочек такое переключение между стилями управления также возможно. Только не параллельное, а альтернативное. И устанавливается оно в конфигурационных файлах пользовательского шелла.

Возвратимся, однако, к нашей man-документации. Для навигации по странице можно использовать и обычные клавиши управления курсором,

клавиши PgUp/PgDown, а также некоторые другие. Например, нажатие Enter приводит к смещёнию на одну строку вперед (аналогично клавише Down, а клавиши Spacebar — на один экран вперед (подобно PgDown.

Однако это — не лучший способ навигации. Потому что управляющие последовательности (не зависимо, в стиле ли vi, или в стиле emacs) обладают дополнительной полезной возможностью: они понимают числовые аргументы. То есть если мы нажмем клавишу с цифрой 5, а вслед за ней — клавишу J, то мы сместимся на пять строк вперед, комбинация 3+K — на три страницы назад, и так далее.

Есть и возможность поиска внутри man-страницы. Для этого нажимается клавиша прямого слэша (/), после чего вводится искомое слово (последовательность символов). Для выхода из просмотра man-страницы предусмотрена клавиша q. Кроме того, можно использовать и почти универсальную комбинацию для прекращения выполнения программ — Control+C. Заканчивая разговор об управляющих последовательностях, ещё раз подчеркну: все они относятся не к самой команде man, а к той программе-пейджеру, которая ею вызывается для просмотра.

Команда sudo

Команда sudo — вторая по важности команда в Mint. Это — основной способ получения прав администратора обычным пользователем. А по умолчанию — так просто единственный, ибо при инсталляции этого дистрибутива пароль root'a не задаётся и, соответственно, доступа к аккаунту «чистого» суперпользователя нет (хотя при желании его можно получить). Команда эта дополняется утилитами visudo и sudoedit. Это узкоспециализированные средства для редактирования общесистемных конфигурационных файлов (в том числе и конфига самой sudo) обычным пользователем. Главные особенности sudo таковы:

1. во-первых, sudo по умолчанию требует указания пароля того пользователя, который получает права другого, а не пароля того, чьи права приобретаются; правда, это может быть изменено;

2. Во-вторых, действие sudo распространяется по умолчанию только на одну команду — ту, которая указывается в качестве ее аргумента; хотя и такое поведение можно изменить с помощью соответствующих опций, о чём будет сказано позднее;

3. в-третьих, sudo обеспечивает более гибкое разграничение доступа пользователей к административным правам — не только разрешая или запрещая получение оных, но и позволяя выполнять только определённый круг действий.

Этим достигается две цели: а) возможность выполнения пользователем административных действий без сообщения ему суперпользовательского пароля (и даже, как в Mint, при его отсуствии), и б) снижение риска повредить систему вследствие забывчивости. Да, есть ещё и третья, дополнительная возможность, предоставляемая sudo — протоколирование действий, выполненных в режиме администратора.

В элементарном виде применение команды sudo — элементарно же просто: требуется лишь указать в качестве ее аргумента имя команды, требующей исполнения, со всеми необходимыми последней опциями и аргументами.

После этого запрашивается пароль запустившего её пользователя — и команда исполняется. Например, команда

```
$ sudo fdisk -l /dev/sda
```

данная от лица обычного пользователя, выведет информацию об указанном дисковом устройстве точно так же, как и данная root'ом.

В должным образом настроенной оболочке Bash в отношении команд-аргументов и путей — аргументов последних, будет действовать автодополнение по нажатию клавиши Tab. Как добиться от Bash столь образцового поведения — говорилось в предыдущем очерке.

Если от лица алминистратора нужно выполнить подряд несколько команд, делать это следует быстро — введенный первый раз пароль по умолчанию «действует» в течении 5 минут. То есть в течении этого времени в ответ на команду sudo пароль повторно запрашиваться не будет.

Период действия пароля для команды sudo можно увеличить, уменьшить или вообще ликвидировать, чтобы аутентификация запрашивался всегда. Это достигается редактированием конфигурационного файла утилиты, к чему мы вернёмся чуть позже.

Аналогичным образом пользователь может отредактировать общесистемный конфигурационный файл, например:

```
$ sudo nano -w /etc/fstab
```

Впрочем, для редактирования общесистемных конфигов предназначена специальная команда sudoedit (или просто sudo с опцией -e). Она не требует указания имени вызываемого для этой цели редактора: в качестве такового используется значение переменной EDITOR из окружения того пользователя, который ее вызвал. Если эта переменная не определена — а это обычно делается в профильных файлах регистрационной оболочки пользователя, — для редактирования вызывается редактор Vim (в своей упрощенной ипостаси, эмулирующей классический vi).

Как это ни парадоксально, команда sudo не исключает запуска администраторского сеанса внутри обычного пользовательского. Потому что с ее помощью можно запустить su:

```
$ sudo su
```

И это — даже в тех дистрибутивах, где root-аккаунта как бы и нет; точнее, по умолчанию нет его пароля (к ним, как уже было сказано, относится Mint). Но и задать пароль «настоящего» администратора не запрещается — для этого достаточно дать команду

```
$ sudo passwd
```

чтобы в дальнейшем использовать команду su обычным образом. Правда, не уверен, что это стоит делать. Потому что для перманентного владения правами адмнистратора команда sudo предусматривает «идеологически правильный» метод, и даже не один. Это — опции -s и -i, пролонгирующие, хотя и несколько по разному, действие команды sudo на неопределённый срок, вплоть до завершения «вторичного сеанса» командой exit.

Опция -s, открывая вторичный сеанс root’а, сохраняет все переменные окружения первоначального пользователя. Однако, если к ней добавить опцию -H, то переменные эти будут заново считаны из профильных файлов домашнего каталога администратора, то есть /root, как при запуске интерактивного экземпляра шелла. Однако каталог, бывший текущим в момент ввода команды, при этом не изменится, как не изменится и вид приглашения командной строки.

Опция же -i полностью воспроизводит root-окружение, запуская его командную оболочку как регистрационную (login shell). Разумеется, при этом и текущий каталог меняется на /root, а приглашение командной строки приобретает вид, описанный в соответствующей переменной профильного файла администраторского шелла, то есть /root/.bashrc. Правда, в Mint и его по умолчанию нет.

На практике разница между обеими формами обретения перманентных прав администратора не велика. А вот то, что при использовании опций -H нахождение в перманентно административном режиме никак внешне не проявляется, чревато ошибками. И делает в большинстве случаев применение опции -i предпочтительным.

Возможности sudo не ограничиваются запуском команд от лица администратора: задав опцию -u username, их можно выполнить и от лица того пользователя, чей логин задан в качестве её значения. Это может быть полезным при просмотре или копировании dot-файлов и dot-каталогов другого пользователя, часто открытых для чтения и изменения только их хозяину.

К слову сказать, команду sudo можно запустить так, чтобы она запрашивала пароль пользователя, от имени которого будет выполняться команда (например, администратора), а не того, кто требует его полномочий. Для этого существует опция -targetpw. А чтобы сделать требование root’ового пароля постоянным, достаточно определить, например, псевдоним типа

```
alias sudo='sudo -targetpw'
```

Команда sudo имеет ещё немало опций — выше я привёл только те, которые мне приходилось использовать. Остальные легко посмотреть в man sudo. Из не перечисленных упомяну ещё опцию -b, предписывающую запускать «подсудную» команду в фоновом режиме. Она может быть полезна при выполнении долговременных действий, например, при копировании образов USB на флэшку командой dd.

Таким образом, команда sudo даёт пользователю практически неограниченные полномочия как для любых общесистемных действий, так и для манипуляции чужими пользовательскими данными. В связи с этим зададимся вопросами:

- любой ли пользователь может получить права администратора через команду sudo, и

- все ли действия по администрированию он может ее посредством выполнить?

Если говорить о семействе Ubuntu (в том числе и дистрибутиве Mint), в котором механизм этот был впервые задействован из «коробки» — то «из

коробки» же ответ на первый вопрос будет отрицательным, на второй — положительным. А вообще это зависит от настроек программы sudo, которые описываются в файле /etc/sudoers. И в нем можно задать правила, допускающие к исполнению определенных команд только отдельных пользователей. В обобщенном виде это выглядит так:

```
username        host = command
```

Здесь, как нетрудно догадаться, username — имя пользователя, для которого устанавливается данное правило, host — имя машины, с которой он может к этому правилу прибегнуть, command — конкретная команда, использование которой разрешается данному пользователю с данной машины. Команда должна быть дана с указанием полного абсолютного пути (то есть /sbin/fdisk, а не fdisk). Поле описания команд может включать несколько значений, разделенных запятыми, например:

```
username        ALL = /sbin/fdisk,/bin/mount
```

В Ubuntu’идах по умолчанию правила доступа пользователей к административным привилегиям описываются так:

```
# User privilege specificationroot

ALL=(ALL) ALL

# Members of the admin group may gain root privileges

%admin  ALL=(ALL) ALL
```

То есть пользователь root, как ему и положено, может исполнять любые команды с любых хостов. А вот получить права его могут только пользователи, входящие в группу admin. Пользователь, создаваемый в ходе обычной установки, автоматически становится членом этой группы — и потому все административные права ему доступны без всяких дальнейших настроек. Однако прочие пользователи, чьи аккаунты будут созданы в последствие, этой привилегии лишены. Если, конечно, они не были специально включены в группу admin.

В более иных дистрибутивах, не использующих sudo «из коробки», потребуется редактирование её конфигурационного файла — того самого /etc/sudoers, о котором упоминалось выше.

Файл /etc/sudoers — обычный текстовый, и, соответственно, его можно редактировать в любом текстовом редакторе (или, скажем, средствами ed или sed). Однако при этом существует определённый риск что-нибудь напортачить (за счёт обычных опечаток), вплоть до того, что полностью закрыть самому себе доступ к привилегиям суперпользователя. Конечно, ситуации эти поправимы — например, через перезагрузку в однопользовательском режиме. Однако, лучше в них не попадать. И потому более надёжным средством модификации /etc/sudoers будет использование специально предназначенной для того утилиты — visudo.

Утилита visudo не делает ничего сверхъестественного — она просто открывает /etc/sudoers в текстовом редакторе, описываемом переменной EDITOR суперпользователя (если таковая не определена, им будет опять же классический vi — отсюда и название) и позволяет его отредактировать

обычным образом, после чего выйти из редактора с сохранением результатов штатными его средствами. Однако перед этим результат редактирования проверяется на корректность. И если обнаруживается нарушение синтаксиса, принятого для /etc/sudoers, выдается соответствующее предупреждение. После которого можно вернуться к редактированию, отказаться от сделанных изменений или все-таки принять их (разумеется, под личную ответственность).

Утилита visudo не гарантирует стопроцентного успеха редактирования. Так как проверяет только соответствие синтаксиса, но не «правильность самих правил». То есть если ошибка будет допущена в указании пути к нужной для данного правила команде — эта команда через sudo не сработает.

Впрочем, на деле это выглядит обычно гораздо проще и совсем не страшно. Так, можно было бы предоставить себе по блату возможность использовать sudo без пароля. Для этого потребовалось бы придать строке, описывающей привилегии группы admin такой вид:

```
%admin        ALL=(ALL)       NOPASSWD: ALL
```

А можно ограничиться более «долгоиграющим» действием пароля, вписав изначально отсутствующую строку

```
Defaults    timestamp_timeout=10
```

где значение таймаута указано в минутах. Если же изменить его на ноль --

```
Defaults    timestamp_timeout=0
```

то пароль будет запрашиваться каждый раз при обращении к команде sudo.

Можно, напротив, отключить тайаут на действие sudo, ввдя для него отрицательное значение:

```
Defaults    timestamp_timeout=-1
```

В этом случае пароль будет запрошен только при первом вызове этой команды.

Более пристальное вглядывание в файл /etc/sudoers легко подскажет возможности дать определённым пользователям или группам только ограниченный набор прав. Впрочем, тут уже начинаются тонкости всамделишнего администрирования.

Создание файлов и каталогов

В следующих мини-очерках будут рассмотрены основные команды, предназначенные для файловых операций, вместе с их наиболее используемыми опциями. Чтобы не повторяться, напомню, что почти все описанные ниже команды имеют три стандартные опции (т.н. GNU Standard Options): --help для получения помощи, --version для вывода информации о текущей версии, и --[пробел], символизирующая окончание перечня опций (т.е. любой символ или их последовательность после неё интерпретируются как аргумент). Так что далее эти опции в описаниях команд фигурировать не будут.

Для создания обычных (regular) файлов могут использоваться команды touch, cat и tee. Первая из указанных команд в форме

```
$ touch filename
```

просто создает файл с именем filename и без всякого содержимого. Кроме того, с помощью специальных опций она позволяет устанавливать временные атрибуты файла, о чем я скажу чуть позже.

Для чего может потребоваться пустой файл? Например, для создания скелета web-сайта с целью проверки целостности ссылок. Поскольку число аргументов команды touch не ограничено ничем (вернее, ограничено только максимальным количеством символов в командной строке), это можно сделать одной командой:

```
$ touch index.html about.html content.html [...]
```

Можно, воспользовавшись приемом группировки аргументов, заполнить файлами все подкаталоги текущего каталога:

```
$ touch dirname1/{filename1,filename2}

      dirname2/{filename3,filename4}
```

и так далее. Правда, сама команда touch создавать подкаталоги не способна — это следует сделать предварительно командой mkdir (о которой — чуть ниже).

Для создания пустого регулярного файла может быть использована также команда cat (хотя основное ее назначение — слияние нескольких файлов, о чем будет говориться со временем). Для этого нужно просто перенаправить ее вывод в файл:

```
$ cat > filename
```

затем создать новую строку (нажатием клавиши Enter) и ввести символ конца файла (комбинацией клавиш Control+Z). Разумеется, предварительно в этот файл можно и ввести какой-нибудь текст, однако это уже относится к управлению контентом, о чем речь будет впереди.

Интересно создание файлов с помощью команды tee. Смысл ее — в раздвоении выходного потока, выводимого одновременно и на стандартный вывод, и в файл, указанный в качестве ее аргумента. То есть если использовать ее для создания файла с клавиатуры, это выглядит, будто строки удваиваются на экране. Но это не так: просто весь вводимый текст копируется одновременно и на экран, и в файл. И потому ее удобно применять в командных конструкциях, когда требуется одновременно и просмотреть результаты исполнения какой-либо команды, и запечатлеть их в файле:

```
$ ls dir | tee filename
```

По умолчанию команда tee создает новый файл с указанным именем, или перезаписывает одноименный, если он существовал ранее. Однако данная с опцией -a, она добавляет новые данные в конец существующего файла.

Команда mkdir создает файл особого типа — каталог, содержимым которого

является список входящих в него файлов. Очевидно, что список этот в момент создания каталога должен быть пуст, однако это не совсем так: любой, даже пустой, каталог содержит две ссылки — на каталог текущий, обозначаемый как ./ (т.е. сам на себя) и на каталог родительский, ../ (т.е тот, в список файлов которого он включается в момент создания).

Команда mkdir требует обязательного аргумента — имени создаваемого каталога. Аргументов может быть больше одного — в этом случае будет создано два или больше поименованных каталогов. По умолчанию они создаются как подкаталоги каталога текущего. Можно создать также подкаталог в существующем подкаталоге:

```
$ mkdir parentdir/newdir
```

Если же требуется создать подкаталог в каталоге, отличном от текущего, — путь к нему требуется указать в явном виде, в относительной форме:

```
$ mkdir ../dirname1/dirname2
```

или в форме абсолютной:

```
$ mkdir /home/username/dirname1/dirname2
```

В произвольном, отличном от текущего, каталоге можно одной командой создать несколько подкаталогов, для чего нужно прибегнуть к группировке аргументов:

```
$ mkdir ../parentdir/{dirname1,dirname2,...,dirname#}
```

Такой прием позволяет одной командой создать дерево каталогов проекта. Например, скелет web-сайта, который потом можно наполнить пустыми файлами с помощью команды touch.

А опций у команды mkdir — всего две (за исключением стандартных опций GNU): --mode (или -m) для установки атрибутов доступа и --parents (или -p) для создания как требуемого каталога, так и родительского по отношению к нему (если таковой ранее не существовал). Первая опция используется в форме

```
$ mkdir --mode=### dirname
```

или

```
$ mkdir -m ### dirname
```

Здесь под ### понимаются атрибуты доступа для владельца файла, группы и прочих, заданные в численной нотации (например, 777 — полный доступ на чтение, изменение и исполнение для всех). Не возбраняется и использование символьной нотации: команда

```
$ mkdir -m a+rwx dirname
```

создаст каталог с теми же атрибутами полного доступа для всех.

Опция --parents (она же -p) позволяет создавать иерархическую цепочку подкаталогов любого уровня вложенности. Например,

```
$ mkdir -p dirlevel1/dirlevel2/dirlevel3
```

в один заход создаст в текущем каталоге цепочку вложенных друг друга подкаталогов. Разумеется, и здесь с помощью группировки аргументов можно создать несколько одноранговых подкаталогов:

```
$ mkdir -p dirlevel1/dirlevel2/{dirlevel31,...,dirlevel3#}
```

Аттрибуция файлов

Следующая группа команд предназначена для атрибуции файлов. В ней — chmod, chown, chgrp, umask, а также уже затронутая ранее команда touch.

Команды chown и chgrp служат для изменения атрибутов принадлежности файла — хозяину и группе: очевидно, что все, не являющиеся хозяином файла, и не входящие в группу, к которой файл приписан, автоматически попадают в категорию прочих (other).

Формат команды chown — следующий:

```
$ chown newowner filename
```

По соображениям безопасности, достаточно очевидным, изменить хозяина файла может только суперпользователь. Пользователь обычный в подавляющем большинстве случаев автоматически становится хозяином всех им созданных (и скопированных) файлов, и избавиться от этого бремени, как и от родительского долга, не в состоянии.

А вот изменить групповую принадлежность своих файлов (т.е. тех, в атрибутах принадлежности он прописан как хозяин) пользователь вполне может. Команда:

```
$ chgrp newgroup filename
```

от его лица припишет файл filename к группе newgroup. Однако и здесь есть ограничение — результат будет достигнут, только если хозяин файла является членом группы newgroup, иначе опять придется прибегнуть к полномочиям администратора.

Можно также одной командой сменить (только суперпользователю, конечно) и хозяина файла, и группу, к которой он приписан. Делается это так:

```
$ chown newowner:newgroup filename
```

Или так:

```
$ chown newowner.newgroup filename
```

Где, понятное дело, под именем newowner выступает новый хозяин файла, а под именем newgroup — новая группа, к которой он приписан.

В обеих командах вместо имени хозяина и группы могут фигурировать их численные идентификаторы (UID и GID, соответственно). Это имеет смысл, например, при совместном использовании файлов в разных операционных системах. Так, даже единственный пользователь имя_рек в каком-либо варианте Linux и в BSD по умолчанию имеет разные идентификаторы, и чтобы

сделать его владельцем неких файлов и там, и там, именно численный идентификатор должен фигурировать в качестве параметра команды chown.

Для команд chown и chgrp поддерживается один и тот же набор опций. Наиболее интересны (и важны) две из них. Опция --reference позволяет определить хозяина файла и его принадлежность к группе не явным образом, а по образу и подобию файла, имя которого выступает в качестве значения опции. Так, команда

```
$ chown --reference=ref_filename filename
```

установит для файла filename те же атрибуты принадлежности (хозяина и группу), что были ранее у файла ref_filename. Это весьма полезно при массовой реатрибуции файлов, полученных из разных источников.

Опция -R (или --recursive) распространяет действие обеих команд не только на файлы текущего каталога (излишне напоминать, что в качестве аргументов команд могут использоваться маски типа *, *.ext, name.* и т.д.), но и на все вложенные подкаталоги, вместе с входящими в них файлами. То есть пользователь может поменять групповую принадлежность всех файлов в своем домашнем каталоге одной командой:

```
$ chgrp -R newgroup ~/*
```

А суперпользователь тем же способом может установить единообразные атрибуты принадлежности «по образцу» для всех компонентов любого каталога:

```
$ chown -R --reference=ref_filename

      /somepath/somecat/*
```

Как и следует из ее имени, команда chmod предназначена для смены атрибутов доступа — чтения, изменения и исполнения. В отношении единичного файла делается это просто:

```
$ chmod [атрибуты] filename
```

Атрибуты доступа могу устанавливаться с использование как символьной, так и цифровой нотации. Первый способ — указание, для каких атрибутов принадлежности (хозяина, группы и всех остальных) какие атрибуты доступа задействованы. Атрибуты принадлежности обозначаются символами u (от user) для хозяина файла, g (от group) — для группы, o (от other) для прочих и a (от all) — для всех категорий принадлежности вообще. Атрибуты доступа символизируются литерами r (от read), дающей право чтения, w (от write) — право изменения и x (от execute) — право исполнения.

Атрибуты принадлежности соединяются с атрибутами доступа символами + (присвоение атрибута доступа), - (отнятие атрибута) или = (присвоение только данного атрибута доступа с одновременным отнятием всех остальных). Одновременно в строке можно указать (подряд, без пробелов) более чем один из атрибутов принадлежности и несколько (или все) атрибуты доступа.

Для пояснения сказанного приведу несколько примеров. Так, команда

```
$ chmod u+w filename
```

установит для хозяина (u) право изменения (+w) файла filename, а команда

```
$ chmod a-x filename
```

отнимет у всех пользователей вообще (a) право его исполнения (-x). В случае, если некоторый атрибут доступа присваивается всем категориям принадлежности, символ a можно опустить. Так, команда

```
$ chmod +x filename
```

в противоположность предыдущей, присвоит атрибут исполнения файла filename всем категориям принадлежности (и хозяину, и группе, и прочим).

С помощью команды

```
$ chmod go=rx filename
```

можно присвоить группе принадлежности файла filename и всем прочим (не хозяину и не группе) право на его чтение и исполнение с одновременным отнятием права изменения.

Наконец, команда chmod в состоянии установить и дополнительные атрибуты режима для файлов, такие, как биты SUID и GUID, или, скажем, атрибут sticky.

Приведенные примеры можно многократно умножить, но, думается, их достаточно для понимания принципов работы команды chmod с символьной нотацией атрибутов.

Цифровая нотация — ещё проще. При ней достаточно указать сумму присваиваемых атрибутов в восьмеричном исчислении (4 — атрибут чтения, 2 — атрибут изменения и 1 — атрибут исполнения; 0 символизирует отсутствие любых атрибутов доступа) для хозяина (первая позиция), группы (вторая позиция) и прочих (третья позиция). Все атрибуты доступа, оставшиеся вне этой суммы, автоматически отнимаются у данного файла. То есть команда

```
$ chmod 000 filename
```

означает снятие с файла filename всех атрибутов доступа для всех категорий принадлежности (в том числе и хозяина) и эквивалентна команде

```
$ chmod =rwx filename
```

в символьной нотации. А команда

```
$ chmod 777 filename
```

напротив, устанавливает для всех полный доступ к файлу filename. Для установки дополнительных атрибутов доступа в численной нотации потребуется указать значение четвертого, старшего, регистра. Так, команда для рассмотренного выше примера — присвоения атрибута суидности исполнимому файлу X-сервера, — в численной нотации будет выглядеть как

```
$ chmod 4711 /usr/X11R6/bin/XFree86
```

Как и для команд chown и chgrp, наиболее значимые опции команды chmod — это --reference и -R. И смысл их тот же самый. Первая устанавливает для файла (файлов) атрибуты доступа, идентичные таковым референсного файла, вторая — распространяет действие команды на все вложенные подкаталоги и входящие в них файлы.

Рекурсивное присвоение атрибутов доступа по образцу требует внимания. Так, если рекурсивно отнять для всего содержимого домашнего каталога атрибут исполнения (а он без соблюдения некоторых условий монтирования автоматом присваивается любым файлам, скопированным с носителей файловой структуры FAT или ISO9660 без расширения RockRidge, что подчас мешает), то тем самым станет невозможным вход в любой из вложенных подкаталогов. Впрочем, в параграфе про утилиту find будет показан один из способов борьбы с таким безобразием.

Как было упомянуто в предшествующей главе, для всех вновь создаваемых данным пользователем файлов можно установить некие умолчальные атрибуты доступа. Этой цели служит команда umask — в отличие от прочих, не самостоятельная утилита, а встроенная команда оболочки. Данная без аргумента, она выведет текущее значение субтрактивных (то есть отнимаемых от суммы) прав доступа для новообразуемых файлов:

```
$ umask

22
```

Вывод прав дается в символьной нотации, нули (то есть отсутствие «отъёма» прав у кого-либо) игнорируется. Если же в качестве аргумента указать «отнимаемые» права — все вновь создаваемые файлы будут иметь новые атрибуты доступа. Например, команда

```
$ umask 000
```

приведет к тому, что новые файлы будут иметь всю совокупность атрибутов доступа (чтения, изменения и исполнения) для хозяина, группы и прочих.

Действие команды umask, данной таким образом, распространяется только на текущий сеанс работы. Поэтому обычно она включается в профильный файл пользовательской командной оболочки, определяя умолчальные права доступа на вечные времена.

Кроме атрибутов принадлежности и доступа, файлам свойственны ещё и атрибуты времени — времени доступа (atime), времени изменения метаданных (ctime) и времени изменения данных (mtime) файла. Они устанавливаются автоматически, в силу самого факта открытия файла (atime), смены любых атрибутов, например, доступа (ctime) или редактирования содержимого файла (mtime).

Однако бывают ситуации, когда автоматически установленные временные атрибуты требуется изменить. Причиной может быть сбой системных часов, в результате которого временные атрибуты создаваемых и модифицируемых файлов перестанут соответствовать действительности.

Казалось бы, чего страшного? Ан нет, фактор времени играет в Unix-системах очень существенную роль. Во-первых, команда make (а под ее управлением компилируются программы из исходников) проверяет временные

атрибуты файлов (в первую очередь — атрибут mtime) и при их несоответствии может работать с ошибками. Ну и более обычная ситуация — на основе временных меток файлов можно эффективно осуществлять, скажем, резервное копирование. И потому желательно, чтобы они отражали реальное время создания и модификации файла.

Так вот, для изменения временных атрибутов файлов и предназначена в первую очередь команда touch, которую ранее мы использовали просто для создания пустого файла. Данная же с именем существующего файла в качестве аргумента -

```
$ touch exist_file
```

она присвоит всем его временным атрибутам (atime, ctime, mtime) значения текущего момента времени. Изменение временных атрибутов можно варьировать с помощью опций. Так, если указать только одну из опций -a, -c, или -m, то текущее значение времени будет присвоено только атрибуту atime, ctime или mtime, соответственно. Если при этом использовать ещё и опцию -d [значение], то любому из указанных атрибутов (или им всем) можно присвоить любую временную метку, в том числе и из далёкого будущего. А посредством опции -r filename файл-аргумент получит временные атрибуты, идентичные таковым референсного файла filename.

Навигация по файловой системе

Следующее, что необходимо применителю после создания файлов — ориентация среди существующего их изобилия.

Для начала при неплохо определиться со своим текущим положением в файловой системе. Этому послужит команда pwd. В ответ на нее выводится полный путь к текущему каталогу. Например, если текущим является домашний каталог пользователя, в ответ на:

```
$ pwd
```

последует

```
/home/username
```

Команда pwd имеет всего две опции: -L и -P. Первая выводит т.н. логический путь к текущему каталогу. То есть, таковым является, скажем, каталог /usr/src/vboxhost-4.3.20, являющий собой символическую ссылку на каталог /usr/share/virtualbox/src/vboxhost, то в ответ на

```
$ pwd -L
```

так и будет выведено

```
/usr/src/vboxhost-4.3.20
```

Впрочем, тот же ответ последует и на команду pwd без опций вообще. Если же дать эту команду в форме

```
$ pwd -P
```

то будет выведен путь к физическому каталогу, на который ссылается

текущий, например:

```
/usr/share/virtualbox/src/vboxhost
```

Далее, по каталогам неплохо как-то перемещаться. Что делается командой cd. В отличие от прочих команд, рассматриваемых в этом разделе, это — внутренняя команда, встроенная во все командные оболочки — бесполезно было бы искать соответствующий ей исполняемый файл. Однако это не уменьшает ее важности. Использование ее очень просто —

```
$ cd pathname
```

где pathname — путь к искомому каталогу в абсолютной (относительно корня) или относительной (относительно текущего каталога) форме.

Определить местоположение команды (и вообще исполняемых файлов) в структуре файловой системы можно с помощью команды which (это также встроенная команда оболочки). В качестве аргумента ее можно указать одно или несколько имен файлов, в ответ на что будет выведен полный путь к каждому из них:

```
$ which tcsh zsh bash

/bin/tcsh

/bin/zsh

/bin/bash
```

При наличии одноимённых исполняемых файлов в разных каталогах по умолчанию будет выведен путь только к первому из них: для вывода всех файлов-«тезок» можно прибегнуть к опции -a. При этом не важно, будут это жёсткие или символические ссылки.

Более широкие возможности поиска — у команды whereis. По умолчанию, без опций, она для заданного в качестве аргумента имени выводит список бинарных файлов, man-страниц и каталогов с исходными текстами:

```
$ whereis zsh

zsh: /bin/zsh /etc/zsh /usr/lib/zsh /usr/share/zsh

/usr/man/man1/zsh.1.gz /usr/share/man/man1/zsh.1.gz
```

Соответствующими опциями можно задать поиск файлов одного из этих типов: -b — бинарных, -m — страниц руководств, -s — каталогов с исходниками. Дополнительные опции -B, -M, -S (в сочетании с опцией -f) позволяют определить исходные каталоги для их поиска.

Команды locate и mlocate осуществляют поиск всех файлов и каталогов, содержащих компонент имени, указанный в качестве аргумента и осуществляют вывод содержимого найденных каталогов. Так, в ответ на команду

```
$ locate zsh
```

будет выведен список вроде следующего:

```
/bin/zsh

/bin/zsh-4.0.6

/etc/zsh

/etc/zsh/zlogin

/etc/zsh/zshenv

/etc/zsh/zshrc
```

и так далее. Команда mlocate действует точно так же. Обе они при этом обращаются к базе данных /var/lib/mlocate/mlocate.db. Исходно эта база данных пуста — и перед использованием команды locate или mlocate должна быть наполнена содержанием. Для этого предназначен сценарий /usr/bin/updatedb.mlocate. Он извлекает сведения из базы данных установленных пакетов — /var/lib/dpkg. В Mint этот сценарий автоматически запускается ежедневно посредством cron, обеспечивая регулярное обновление поисковой базы.

Информация о файлах

Наиболее универсальным средством получения практически исчерпывающей информации о файлах является команда ls. Однако для этой цели существуют и другие команды.

Общая форма запуска команды ls —

```
$ ls [options] names
```

где в качестве аргумента names могут выступать имена файлов или каталогов в любом количестве. Команда эта имеет многочисленные опции, основные из которых мы и рассмотрим.

Начать с того, что команда ls, данная без всяких опций, по умолчанию выводит только имена файлов, причём опуская т.н. dot-файлы, имена которых начинаются с точки (это — некие аналоги скрытых файлов в MS DOS и Windows). Кроме того, если в качестве аргумента указано имя каталога (или аргумент не указан вообще, что подразумевает текущий каталог), из списка имен его файлов не выводятся текущий (.) и родительский (..) каталог.

Для вывода всех без исключения имен файлов (в том числе и скрытых) предназначена опция -a. Смысл опции -A близок — она выводит список имен всех файлов, за исключением символов текущего (.) и родительского (..) каталога.

Кроме имени, любой файл идентифицируется своим номером inode. Для его вывода используется опция -i:

```
$ ls -i

12144 content.html
```

и так далее. Как и многие другие, команда ls обладает способностью рекурсивной обработки аргументов, для чего предназначена опция -R, выводящая список имен файлов не только текущего каталога, но и всех вложенных подкаталогов:

```
$ ls -R

unixforall:

about/  apps/    diffimages/  distro/  signature.html  sys/

anons/  content/  difftext/    gentoo/  statistics/     u4articles/



unixforall/about:

about_lol.html  about_lol.txt  index.html



unixforall/anons:

anons_dc.html
```

Опция же -d, напротив, запрещает вывод содержимого вложенных подкаталогов.

В выводе команды ls по умолчанию имена файлов разных типов даются абсолютно одинаково. Для их визуального различия используется опция -F, завершающая имена каталогов символом слэша, исполнимых файлов — символом звездочки, символических ссылок — «собакой»; имена регулярных файлов, не имеющих атрибута исполнения, никакого символа не включают:

```
$ ls -F

dir1/   dir2/   dir3@   file1   file2*  file3@
```

Другое средство для визуального различия типов файлов — колоризация, для чего применяется опция -G. Цвета шрифта, воспроизводящего имена, по умолчанию — синий для каталогов, лиловый (magenta) для символических ссылок, красный — исполнимых файлов, и так далее. Для файлов устройств, исполнимых файлов с атрибутом «суидности», каталогов, имеющих атрибут sticky, дополнительно колоризуется и фон, на котором выводится шрифта, воспроизводящий их имена. Подробности можно посмотреть в секции ENVIRONMENT man-страницы для команды ls. Впрочем, колоризация работает не на всех типах терминалов (и не во всех командных оболочках).

По умолчанию команда ls выводит список файлов в порядке ASCII-кодов первого символа имени. Однако есть возможность его сортировки в порядке времени модификации (-t), изменения статуса (-c) или времени доступа (-tu), а также в порядке, обратном любому из перечисленных (-r). Кроме того, опция -f отменяет какую-либо сортировку списка вообще.

Информацию об объёме файлов можно получить, используя опцию -s,

выводящую для имени каждого файла его размер в блоках, а также суммарные объём всех выведенных файлов:

```
$ ls -s ../book

total 822

656 book.html

4 content1.html

86 var_part2.html

24 command.html

38 part2.html

6 command.txt

8 shell_tmp.html
```

Добавление к опции -s ещё и опции -k (то есть ls -sk) выведет всю ту же информацию в килобайтах.

Как можно видеть из всех приведённых выше примеров, списки файлов по команде ls выводится в многоколоночном виде (чему соответствует опция -C, однако указывать ее нет необходимости — многоколоночный вид принят для краткого формата по умолчанию). Но можно задать и одноколоночное представление списка посредством опции -1:

```
$ ls -1

dir1

dir2

dir3

file1

file2

file3
```

До сих пор речь шла о кратком формате вывода команды ls. Однако более информативным является т.н. длинный ее формат, вывод в котором достигается опцией -l и автоматически влечет за собой одноколоночное представление списка:

```
$ ls -l

total 8

drwxr-xr-x  2 alv  alv  512  8 май 18:04 dir1
```

```
drwxr-xr-x  3 alv  alv  512  8 май 17:43 dir2

lrwxr-xr-x  1 alv  alv    4  9 май 07:59 dir3 -> dir2

-rw-r--r--  1 alv  alv   14  8 май 10:39 file1

-rwxr-xr-x  1 alv  alv   30  9 май 08:02 file2

lrwxr-xr-x  1 alv  alv    2  8 май 10:57 file3 -> f1
```

Можно видеть, что по умолчанию в длинном формате выводятся:

- сведения о типе файла (- — регулярный файл, d — каталог, l — символическая ссылка, c — файл символьного устройства, b — файл блочного устройства) и атрибуты доступа для различных атрибутов принадлежности (о чем было сказано достаточно);
- количество жёстких ссылок на данный идентификатор inode;
- имя пользователя — владельца файла, и группы пользователей, которой файл принадлежит;
- размер файла в блоках;
- время модификации файла с точностью до месяца, дня, часа и минуты (в формате, принятом в данной locale);
- имя файла и (для символических ссылок) имя файла-источника.

Однако это ещё не всё. Добавив к команде ls -l ещё и опцию -i, можно дополнительно получить идентификатор inode каждого файла, опция -n заменит имя владельца и группу на их численные идентификаторы (UID и GUID, соответственно), а опция -T выведет в поле времени модификации ещё и годы, и секунды:

```
$ ls -linT

total 8

694402 drwxr-xr-x  2 1000  1000  512  8 май 18:04:56 2002 dir1

694404 drwxr-xr-x  3 1000  1000  512  8 май 17:43:31 2002 dir2

673058 lrwxr-xr-x  1 1000  1000    4  9 май 07:59:08 2002 dir3 -> dir2

673099 -rw-r--r--  1 1000  1000   14  8 май 10:39:38 2002 file1

673059 -rwxr-xr-x  1 1000  1000   30  9 май 08:02:23 2002 file2

673057 lrwxr-xr-x  1 1000  1000    2  8 май 10:57:07 2002 file3 -> f1
```

Разумеется, никто не запрещает использовать в длинном формате и опции визуализации (-F и -G), и опции сортировки (-r, t, tu), и любые другие, за исключением опции -C — указание ее ведет к принудительному выводу списка в многоколоночной форме, что естественным образом подавляет длинный

формат представления.

Я столь подробно остановился на описании команды ls потому, что это — основное средство визуализации файловых систем любого Unix, при умелом использовании ничуть не уступающее развитым файловым менеджерам (типа Midnight Commander или Konqueror) по своей выразительности и информативности. И отнюдь не требующее для достижения таковых вбивания руками многочисленных опций: со временем будет показано, что соответствующей настройкой последних можно добиться любого «умолчального» вывода команды ls.

Существуют и другие команды для получения информации о файлах. Например, команда под характерным именем file с аргументом в виде имени файла в состоянии определить тип его, а также характер содержания с большой детальностью. Так, для регулярных файлов она распознает:

- исполняемые бинарные файлы с указанием их формата (например, ELF), архитектуры процессора, для которых они скомпилированы, характер связи с разделяемыми библиотеками (статический или динамический);
- исполняемые сценарии с указанием оболочки, для которой они созданы;
- текстовые и html-документы, часто с указанием используемого набора символов.

Последнему, впрочем, для русскоязычных документов доверять особо не следует: кодировка KOI8-R в них вполне может быть обозвана ISO-8859.

Определяет она также каталоги, символические ссылки, специальные файлы устройств, указывая для последних старшие и младшие номера устройств.

Наконец, команда stat (это — встроенная команда оболочки), с именем файла в качестве аргумента, выводит большую часть существенных сведений о файле в удобном для восприятия виде, например, включая идентификатор inode, режим доступа (в символьной форме), идентификаторы владельца и группы, временные атрибуты, количество жёстких и символических ссылок.

Манипулирование файлами

Перейдем к манипуляциям с существующими файлами — копированию, перемещёнию, переименованию, удалению.

Начнем с копирования — это выполняется очень простой командой, cp, имеющей, однако, весьма разнообразные аспекты применения. В самом простом своем виде она требует всего двух аргументов — имени файла-источника на первом месте и имени целевого файла — на втором:

```
$ cp file_source file_target
```

Этим в текущем каталоге создается новый файл (file_target), идентичный по содержанию копируемому (file_source). То есть область данных первого будет дублировать таковую последнего. Однако области метаданных у них будут

различны изначально. Целевой файл — это именно новый файл, со своим иднетификатором inode, заведомо иными временными атрибутами; его атрибуты доступа и принадлежности в общем случае также не обязаны совпадать с таковыми файла-источника.

Новый файл может быть создан и в произвольном каталоге, к которому пользователь имеет соответствующий доступ: для этого следует только указать полный путь к нему:

```
$ cp file_source dir/subdir/file_target
```

Если в качестве второго аргумента команды указано просто имя каталога, то новый файл будет создан в нем с именем, идентичным имени файла-источника. Однако подчеркну, что в любом случае копирования создается именно новый файл, никак после этого не связанный с файлом исходным.

Если в качестве последнего аргумента выступает имя каталога, он может предваряться любым количеством аргументов — имен файлов:

```
$ cp file1 file2 ... file3 dir/
```

В этом случае в целевом каталоге dir/ будут созданы новые файлы, идентичные по содержанию файлам file1, file2 и т.д.

Если в целевом (или текущем) каталоге уже имеется файл с именем, совпадающим с именем вновь создаваемого файла, он в общем случае будет без предупреждения заменен новым файлом. Единственное средство для предотвращения этого — задание опции -i (от interactive) — при ее наличии последует запрос
на перезапись существующего файла:

```
$ cp -i file1 file2

overwrite file2? (y/n [n])
```

Как говорилось в предыдущем очерке, командная оболочка моетт быть настроена так, чтобы по умолчанию не допускать перезаписи существующих файлов. Однако если такая потребность осознанно возникнет, это можно выполнить с помощью опции -f (от force). К слову сказать, она также аннулирует действие опции -i, например, при использовании ее в псевдониме команды cp.

Имя каталога может выступать и в качестве первого аргумента команды cp. Однако это потребует опции -R (иногда допустима и опция -r — в обоих случаях от recursive). В этом случае второй аргумент также будет воспринят как имя каталога, который не только будет создан при этом, но в нем также будет рекурсивно воспроизведено содержимое каталога источника (включая и вложенные подкаталоги).

При копировании файлов, представляющих собой символические ссылки, они будут преобразованы в регулярные файлы, копирующие содержимое файлов — источников ссылки. Однако при рекурсивном копировании каталогов, содержащих символические ссылки, возможно их воспроизведение в первозданном виде. Для этого вместе с опцией -R должна быть указана одна из опций -H или -L. Однако обе они при отсутствии -R игнорируются.

Как уже было сказано, создаваемые при копировании целевые файлы по умолчанию получают атрибуты доступа и времени, не зависящие от таковых файла-источника. Обычно они определяются значением переменной umask, заданной глобально, в профильном файле командной оболочки пользователя. Однако при желании атрибуты исходного файла можно сохранить в файле целевом — для этого предназначена опция -p. Разумеется, атрибуты эти будут сохранены только в том случае, это это допустимо целевой файловой системой: не следует ожидать, что атрибуты доступа и принадлежности будут сохранены при копировании на носитель с файловой системой FAT.

Для выполнения операции копирования файла он должен иметь атрибут чтения для пользователя, выполняющего копирование; кроме того, последний должен обладать правом на изменение каталога, в который производится копирование.

Следующие две часто используемые файловые операции — переименование и перемещёние, — выполняются одной командой, mv.

Как и при копировании, при перемещёнии и переименовании одноименные файлы, ранее существовавшие в целевом каталоге, затираются, замещаясь файлами-источниками без предупреждения. Чтобы этого не случилось, используется опция -i, требующая запрос на подтверждение действия. Напротив, опция -f в принудительном порядке перезаписывает существующий файл.

Операции копирования и перемещёния/переименования выглядят сходными, однако по сути своей глубоко различны. Начать с того, что команда mv не совершает никаких действий с перемещаемыми или переименовываемыми файлами — она модифицирует каталоги, к которым приписаны имена этих файлов. Это имеет два важных следствия. Во-первых, при перемещёнии/переименовании файлы сохраняют первозданными атрибуты доступа, принадлежности и даже времени изменения метаданных (ctime) и модификации данных (mtime) — ведь ни те, ни другие при перемещёнии/переименовании файла не изменяются.

Во-вторых, для выполнения этих действий можно не иметь никаких вообще прав доступа к файлам — достаточно иметь право на изменение каталогов, в которых они переименовываются или перемещаются: ведь имя файла фигурирует только в составе каталога, и нигде более.

Файлы приходится не только создавать, но иногда и удалять. Смысл удаления файлов аналогичен их перемещнию — с самими файлами при этом ничего не происходит, а изменяются содержащие их каталоги. Удаление файлов выполняется командой

```
$ rm filename
```

в которой аргументов, означающих имена подлежащих удалению файлов, может быть произвольное количество. Как и при перемещёнии, при этом не затрагиваются ни метаданные, ни данные файлов, а только удаляются их имена из родительских каталогов. И потому для удаления файлов опять же не обязательно иметь какие-либо права в их отношении — достаточно прав на изменение содержащих их каталогов.

Командой rm файлы-аргументы будут удалены в общем случае без предупреждения. Подобно командам cp и mv, для команды rm предусмотрены

опции -i (запрос на подтверждение) и -f (принудительное удаление вне зависимости от настроек оболочки).

Интересный момент — удаление случайно созданных файлов с именами, «неправильными» с точки зрения системы или командной оболочки. Примером этого могут быть имена, начинающиеся с символа дефиса. Если попробовать сделать это обычным образом

```
$ rm -file
```

в ответ последует сообщение об ошибке типа

```
rm: illegal option — l
```

то есть имя файла будет воспринято как опция. Для предотвращения этого такое «неправильное» имя следует предварить символом двойного дефиса и пробелом, означающими конец списка опций:

```
$ rm — -file
```

В принципе, команда rm ориентирована на удаление обычных и прочих файлов, но не каталогов. Однако с опцией -d она в состоянии справиться и с этой задачей — в случае, если удаляемый каталог пуст. Наконец, опция -R (или -r) производит рекурсивное удаление каталогов со всеми их файлами и вложенными подкаталогами.

Это делает использование опции -R весьма опасным: возможно, набивший оскомину пример

```
$ rm -R /
```

когда при наличии прав суперпользователя уничтожается вся файловая система, и утрирован, но в локальном масштабе такая операция более чем реальна.

Специально для удаления каталогов предназначена команда

```
$ rmdir
```

которая способна удалить только пустой каталог. Кроме того, с опцией -p она может сделать это и в отношении каталогов родительских — но также только в том случае, если они не содержат файлов.

Кроме простого копирования файлов, существует команда для копирования с преобразованием — dd. Обобщенный ее формат весьма прост:

```
$ dd [options]
```

то есть она просто копирует файл стандартного ввода в файл стандартного вывода, а опции описывают условия преобразования входного потока данных в выходной. Реально основными опциями являются if=file1, подменяющая стандартный ввод указанным файлов, и of=file2, проделывающая ту же операцию со стандартным выводом.

А далее — прочие условия преобразования, весьма обильные. Большинство из них принимают численные значения в блоках:

• опции ibs=n и obs=n устанавливают размер блока для входного и выходного потоков, bs=n — для обоих сразу;

• опция skip=n указывает, сколько блоков нужно пропустить перед записью входного потока;

• опция count=n предписывает скопировать из входного потока лишь указанное количество блоков, отсчитываемых с начала файла-источника.

Сфера применения команды dd далеко выходит за рамки простого копирования файлов. Например, именно с ее помощью изготавливаются загрузочные флешки и SD-карты, точные копии CD в файловой системе на винчестере, преобразуются шрифтовые файлы из одного формата в другой, и ещё многое. Эта же команда может применяться для резервного копирования данных.

Архивация…

Для пользователя Windows, привыкшего к программам типа Zip/WinZip и Rar/WinRar, архивация и компрессия неразрывны, как лошади в упряжке. Однако это — вполне разные действия.

Архивация — это сборка группы файлов или каталогов в единый файл, содержащий не только данные файлов-источников, но и информацию о них — имена файлов и каталогов, к которым они приписаны, атрибуты принадлежности, доступа и времени, что позволяет восстановить как данные, так и их структуру из архива в первозданном виде. Компрессия же предназначена исключительно для уменьшения объёма, занимаемого файлами на диске (или ином носителе).

Для архивации и компрессии предназначены самостоятельные команды. Хотя архивацию и компрессию можно объединить в одной конструкции или представить так, будто они выполняются как бы в едином процессе.

Традиционное и самое распространённое средство архивации в Unix-системах — утилита tar. Обобщенный формат ее таков:

```
$ tar [options] archiv_name [arguments]
```

где archiv_name — обязательный аргумент, указывающий на имя архивного файла, с которым производятся действия, определяемые главными опциями. Формы указания опций для команды tar очень разнообразны. Исторически первой была краткая форма без предваряющего дефиса, что поддерживается и поныне. Однако в текущих версиях команды в целях единообразия утверждена краткая форма с предваряющим дефисом или дублирующая ее полная форма, предваряемая двумя дефисами. Некоторые опции (например --help — получение справки об использовании команды) предусмотрены только в полной форме.

Главные опции и указывают на то, какие действия следует выполнить над архивом в целом:

• создание архива (опция c, -c или --create);

• просмотр содержимого существующего архива (опция t, -t или --list);

- распаковка архива (опция x, -x, --extract или --get).

Легко понять, что при работе с архивом как целым одна из этих главных (т.н. функциональных) опций обязательна. При манипулировании же фрагментами архива они могут подменяться другими функциональными опциями, как то:

- r (или --append) - добавление новых файлов в конец архива;
- u (или --update) - обновление архива с добавлением не только новых, но и модифицированных (с меньшим значением атрибута mtime) файлов;
- -A (--catenate или --concatenate) - присоединение одного архива к другому;
- --delete - удаление именованных файлов из архива;
- --compare - сравнение архива с его источниками в файловой системе.

Прочие (очень многочисленные) опции можно отнести в разряд дополнительных — они определяют условия выполнения основных функций команды. Однако одна из таких дополнительных опций — f (-f или --file), значение которой — имя файла (в том числе файла устройства, и не обязательно на локальной машине), также является практически обязательной. Дело в том, что команда tar (от tape archiv) изначально создавалась для прямого резервного копирования на стриммерную ленту, и именно это устройство подразумевается в качестве целевого по умолчанию. Так что если это не так (а в нынешних условиях — не так почти наверняка), имя архивного файла в качестве значения опции f следует указывать явно.

Проиллюстрируем сказанное несколькими примерами. Так, архив из нескольких файлов текущего каталога создается следующим образом:

```
$ tar cf arch_name.tar file1 ... file#
```

Если задать дополнительную опцию v, ход процесса будет отображаться на экране — это целесообразно, и в дальнейших примерах эта опция будет использоваться постоянно.

С помощью команды tar можно заархивировать и целый каталог, включая его подкаталоги любого уровня вложенности, причём - двояким образом. Так, если дать команду

```
$ tar cvf arch_name.tar *
```

файлы каталога текущего каталога (включая подкаталоги) будут собраны в единый архив, но без указания имени каталога родительского. А командой

```
$ tar cvf arch_name.tar dir
```

каталог dir будет упакован с полным сохранением его структуры.

С помощью команды

```
$ tar xvf arch_name.tar
```

будет выполнена обратная процедура — распаковка заархивированных файлов в текущий каталог. Если при архивировании в качестве аргумента было указано имя каталога, а не набора файлов (пусть даже в виде шаблона) - этот каталог будет восстановлен в виде корневого для всех разархивируемых файлов.

При извлечении файлов из архива никто не обязывает нас распаковывать весь архив — при необходимости это можно сделать для одного нужного файла, следует только указать его имя в качестве аргумента:

```
$ tar xvf arch_name.tar filename
```

Правда, если искомый файл находился до архивации во вложенном подкаталоге, потребуется указать и путь к нему — от корневого для архива каталога, который будет различным для двух указанных схем архивации. Ну а для просмотра того, каким образом был собран наш архив, следует воспользоваться командой

```
$ tar tf arch_name.tar
```

Если архив собирался по первой схеме (с именами файлов в качестве аргументов, вывод ее будет примерно следующим:

```
dir2/

dir2/file1

example

new

newfile

tee.png
```

При втором способе архивации мы увидим на выводе нечто вроде

```
dir1/

dir1/example

dir1/new

dir1/newfile

dir1/tee.png

dir1/dir2/

dir1/dir2/file1
```

В данном примере опция v была опущена. Включение ее приведет к тому, что список файлов будет выведен в длинном формате, подобном выводу команды ls -l:

```
drwxr-xr-x alv/alv      0 10 май 11:03 2002 dir2/
```

```
-rw-r--r-- alv/alv      0 10 май 11:03 2002 dir2/file1

...
```

Команда tar имеет ещё множество дополнительных опций, призванных предотвращать перезапись существующих файлов, осуществлять верификацию архивов, учитывать при архивации разного рода временные атрибуты, вызывать для исполнения другие программы. К некоторым опциям я ещё вернусь после рассмотрения команд компрессии, другие же предлагается изучить самостоятельно, воспользовавшись страницей экранной документации man tar.

Здесь уместно добавить пару слов об утилите ar, предназначенной для создания архивов, их модификации, частичной экстракции из них файлов и полного развёртывания. Подобно tar, это — чистый архиватор, не выполняющий никакой компрессии. И, насколько я знаю, практически не используемый для архивирования данных, в частности, для резервного копирования. Но исторически сложилось так, что именно утилитой ar в конечном счёте упаковываются компоненты пакетов deb-формата, используемого в Mint (и многих других дистрибутивах).

… и компрессия

Утилита gzip — это традиционный компрессор Unix-систем, сменивший в сей роли более старую утилиту compress. Простейший способ её использования таков:

```
$ gzip filename
```

где в качестве аргументов будет выступать имя файла. При этом (внимание!) исходный несжатый файл подменяется своей сжатой копией, которой автоматически присваивается расширение *.gz.

В качестве аргументов может выступать и произвольное количество имен файлов — каждый из них будет заменен сжатым файлом *.gz. Более того, посредством опции -r может быть выполнено рекурсивное сжатие файлов во всех вложенных подкаталогах. Подчеркну, однако, что никакой архивации команда gzip не производит, обрабатывая за раз только единичный файл. Фактически форма

```
$ gzip file1 file2 ... file#
```

просто эквивалент последовательности команд

```
$ gzip file1

$ gzip file2

...

$ gzip file#
```

Правда, объединение компрессированных файлов возможно методом конкатенации (с помощью команды cat) или посредством архивирования командой tar.

Команда gzip имеет и другие опции, указываемые в краткой (однобуквенной) или полной нотации. В отличие от tar, знак дефиса (или, соответственно, двойного дефиса) обязателен в обоих случаях. Так, опциями -1 ... -9 можно задать степень сжатия и, соответственно, время исполнения процедуры: -1 соответствует минимальному, но быстрому сжатию, -9 - максимальному, но медленному. По умолчанию в команде gzip используется опция -6, обеспечивающая разумный компромисс между скоростью и компрессией.

Благодаря опции -d (--decompress) команда gzip может выполнить развертывание сжатого файла, заменяя его оригиналом без расширения *.gz. Хотя в принципе для этого предназначена команда gunzip:

```
$ gunzip file.gz
```

Использование этой команды настолько прозрачно, что я задерживаться на ней не буду. Замечу только, что утилита gzip интегрирована в архиватор tar, вызвываясь из него опцией z. То есть для создания компрессированного архива потребуется такая команда:

```
$ tar czf archive.tar.gz path2/
```

А для декомпрессии и развёртывания архива — такая:

```
$ tar xzf archive.tar.gz
```

В обоих случаях не принесёт вреда добавление опции v — она обеспечит вывод на экран сообщеня о ходе процесса.

В относительно недавнее время некоторое распространение получил компрессор bzip2, обеспечивающий большую (на 10-15%) степень сжатия, хотя и менее быстродействующий. Использование его практически идентично gzip, с деталями его можно ознакомиться с помощью страницы экранной документации man bzip2.

Итоговый компрессированный файл получает имя вида *.bz2 и может быть распакован командой bunzip2 (или командой bzip2 -d). Следует только помнить, что форматы *.gz и *.bz2 не совместимы между собой. Соответственно, первый не может быть распакован программой bunzip2, и наоборот.

Как и gzip, утилита bzip2 может быть вызвана из архиватора tar — при создании компрессированного архива так:

```
$ tar cjf archive.tar.bz2 path2/
```

А при развёртывании оного — эдак:

```
$ tar xjf archive.tar.bz2
```

Особо задерживаться на этой утилите не хочется ещё и потому, что, мне кажется, вскоре она выйдет из употребеления. Ибо, обеспечивая меньшую степень сжатия по сравнению с форматом xz (о котором сейчас будет речь), bzip2 отнюдь не превосходит его по скорости компрессии и декомпрессии. И там, где критично именно время упаковки (а также универсальность), будет по прежнему использоваться старый добрый gzip. Там же, где на первый план

выходит степень сжатия, карты в руки новому формату xz. Который, кстати, на мощных машинах по скорости создания и распаковки вплотную приближается к gzip.

Реализацией формата xz является набор утилит XZ Utils, основанный на алгоритме LZMA (Lempel-Ziv-Markov chain-Algorithm). Сам по себе метод сжатия LZMA существует достаточно давно: он был разработан нашим соотечественником Игорем Павловым с использованием достижений предшественников, разработавших алгоритмы LZ77, LZ78 и LZV — что, впрочем, могло бы составить предмет отдельной истории, которую когда-нибудь кто-нибудь напишет.

А метод LZMA был задействован его автором в собственной же разработке — утилите компрессии 7-Zip для Windows, быстро снискавшей славу несравненного «сжимателя» файлов. Инструментарий для разработки программ, использующих данный метод (LZMA SDK) распространялся сначала на условиях лицензии GPL, что позволяло использовать его в соответствующих проектах (например, в GNU tar). Однако в конце 2008 года Игорь Павлов превратил его в общественное достояние (Public Domain). Вслед за чем был создан основанный на этом методе пакет LZMA Utils, немедленно встроенный в tar. Что сделало этот метод компрессии столь же простым и обыденным, как gzip или bzip2. И с тех пор эта возможность, после установки соответствующего пакета, присутствует во всех дистрибутивах Linux.

Правда, вслед за тем появился LZMA2, улучшенная версия того же алгоритма, обеспечивающий более высокую степень сжатия и лучшую поддержку многопоточности. А на его основе был создан пакет XZ Utils — именно он в настоящее время используется в Mint по умолчанию. И включает в себя такие команды:

xz — компрессор и, при указании опции --decompress, декомпрессор;

unxz — собственно декомпрессор;

xzcat осуществляет декомпрессию на стандартный вывод;

xzmore и xzless — pager'ы для lzma-компрессированных текстовых файлов;

xzgrep, xzegrep, xzfgrep — поиск текстовых фрагментов в xz-компрессированных файлах.

Последние три утилиты работают аналогично командам xzgrep, xzegrep, xzfgrep, применённым к некомпрессированным файлам. А команда xzcat является аналогом утилиты cat. Об этих четырёх командах будет подробно говориться в ближайших разделах.

Утилиты пакета XZ Utils могут, с некоторыми ограничениями, работать с файлами, запакованными старым методом LZMA1 (но не наоборот). Хотя сами по себе пакеты XZ Utils и LZMA Utils между собой конфликтуют.

Разумеется, поддержка XZ была немедленно встроена и в tar. Так что теперь для применения компрессии LZMA2 при создании tar-архива достаточно указать соответствующую опцию:

```
$ tar --create --xz --file filename.tar.xz path2/arch_dir
```

Или, в более употребимой простыми людьми краткой форме, так:

```
$ tar cJf filename.tar.lzma path2/arch_dir
```

где опция J и представляет собой алиас для полной формы -xz. Если присваивать архивному файлу суффикс по правилам утилиты tar, опцию J можно заменить на a (что эквивалентно полной форме --auto-compress), обеспечивающей определение типа компрессии по «расширению» *.xz. Более того, скажу по секрету: если архив именован по правилам, то можно опустить даже опцию --auto-compress — она и так будет задействована по умолчанию.

Распаковка xz-компрессированного архива выполняется в обратном порядке:

```
$ tar xJf filename.tar.xz
```

или
$ tar xaf filename.tar.xz

Метод LZMA и особенно LZMA2 вследствие эффективности компрессии быстро нашёл себе применение в сборке дистрибутивных пакетов: именно с его помощью в настоящее время сжимаются deb-пакеты Mint (и всех других дистрибутивов, использующих этот формат пакетов).

Утилита find и xargs при ней

На этих страницах речь пойдет о пакете, известном в проекте GNU как findutils. И в первую голову — о команде find (как, впрочем, и о тесно связанной с ней команде xargs). Столь высокая честь выпадает им потому, что посредством этих двух команд можно выполнить если не все, то изрядную задач, возникающих при работе с файлами.

Итак, апофеоз командного файлового менеджмента — утилита find. Строго говоря, вопреки своему имени, команда эта выполняет не поиск файлов как таковой, но — рекурсивный обход дерева каталогов, начиная с заданного в качестве аргумента, отбирает из них файлы в соответствие с некоторыми критериями и выполняет над отбракованным файловым хозяйством некоторые действия. Именно эту ее особенность подчеркивает резюме команды find, получаемое (в некоторых системах) посредством

```
$ whatis find
```

```
find(1)                  — walk a file hierarchy
```

что применительно случаю можно перевести как «прогулка по файловой системе».

Команда find по своему синтаксису существенно отличается от большинства прочих Unix-команд. В обобщенном виде формат ее можно представить следующим образом:

```
$ find аргумент [опция_поиска] [значение] [опция_действия]
```

В качестве аргумента здесь задается путь поиска, то есть каталог, начиная с которого следует совершать обход файловой системы, например, корень ее:

```
$ find / [опция_поиска] [значение]
      [опция_действия]
```

или домашний каталог пользователя:

```
$ find ~/ [опция_поиска] [значение]
      [опция_действия]
```

Опция поиска — критерий, по которому следует отбирать файл (файлы) из определенных в аргументе частей файловой системы. В качестве таковых могут выступать имя файла (-name), его тип (-type), атрибуты принадлежности, доступа или времени.

Ну а опция действия определяет, что же надлежит сделать с отобранными файлом или файлами. А сделать с ними, надо заметить, можно немало — начиная с вывода на экран (-print, опция действия по умолчанию) и кончая передачей в качестве аргументов любой другой команде (-exec).

Как можно видеть из примера, опция поиска и опция действия предваряются знаком дефиса, значение первой отделяется от ее имени пробелом.

Однако начнём по порядку. Опции поиска команды find позволяют выполнить отбор файлов по следующим критериям (символ дефиса перед опциями ниже опущен, но не следует забывать его ставить):

- name — поиск по имени файла или по маске имени; в последнем случае метасимволы маски должны обязательно экранироваться (например, — name *.tar.gz) или заключаться в кавычки (одинарные или двойные, в зависимости от ситуации); этот критерий чувствителен к регистру, но близкий по смыслу критерий iname позволяет производить поиск по имени без различения строчных и заглавных букв;

- type — поиск по типу файла; этот критерий принимает следующие значения: f (регулярный файл), d (каталог), s (символическая ссылка), b (файл блочного устройства), c (файл символьного устройства);

- user и group — поиск по имени или идентификатору владельца или группы, выступающим в качестве значения критерия; существует также критерии nouser и nogroup — они отыскивают файлы, владельцев и групповой принадлежности не имеющие (то есть тех, учетные записи для которых отсутствую в файлах /etc/passwd и /etc/group); последние два критерия в значениях, разумеется, не нуждаются;

- size — поиск по размеру, задаваемому в виде числа в блоках или в байтах — в виде числа с последующим символом c; возможны значения n (равно n блоков), +n (более n блоков), -n (менее n блоков);

- perm — поиск файлов по значениям их атрибутов доступа, задаваемых в символьной форме;

- atime, ctime, mtime — поиск файлов с указанными временными атрибутами; значения временных атрибутов указываются в сутках

(точнее, в периодах, кратных 24 часам); возможны формы значений этих атрибутов: n (равно указанному значению n*24 часа), +n (ранее n*24 часа), -n (позднее n*24 часа);

• newer — поиск файлов, измененных после файла, указанного в качестве значения критерия (то есть имеющего меньшее значение mtime);

• maxdepth и mindepth позволяют конкретизировать глубину поиска во вложенных подкаталогах — меньшую или равную численному значению для первого критерия и большую или равную — для второго;

• depth — производит отбор в обратном порядке, то есть не от каталога, указанного в качестве аргумента, а с наиболее глубоко вложенных подкаталогов; смысл этого действия — получить доступ к файлам в каталоге, для которого пользователь не имеет права чтения и исполнения;

• prune — позволяет указать подкаталоги внутри пути поиска, в которых отбора файлов производить не следует.

Кроме этого, существует ещё одна опция поиска — fstype, предписывающая выполнять поиск только в файловой системе указанного типа; очевидно, что она может сочетаться с любыми другими опциями поиска. Например, команда

```
$ find / -fstype ext3 -name zsh*
```

будет искать файлы, имеющие отношение к оболочке Z-Shell, начиная с корня, но только — в пределах тех разделов, на которых размещёна файловая система Ext3fs (на моей машине — это именно чистый корень, за вычетом каталогов /usr, /opt, /var, /tmp и, конечно же, /home.

Критерии отбора файлов могут группироваться практически любым образом. Так, в форме

```
$ find ~/ -name *.tar.gz newer filename
```

она выберет в домашнем каталоге пользователя все компрессированные архивы, созданные после файла с именем filename. По умолчанию между критериями отбора предполагается наличие логического оператора AND (логическое «И»). То есть будут отыскиваться файлы, удовлетворяющие и маске имени, и соответствующему атрибуту времени. Если требуется использование оператора OR (логическое «ИЛИ»), он должен быть явно определен в виде дополнительной опции -o между опциями поиска. Так, команда:

```
$ find ~/ -mtime -2 -o newer filename
```

призвана отобрать файлы, созданные менее двух суток назад, или же — позднее, чем файл filename.

Особенность GNU-реализации команды find (как, впрочем, и ее тезки из числа BSD-утилит) — то, что она по умолчанию выводит список отобранных в соответствии с заданными критериями файлов на экран, не требуя дополнительных опций действия. Однако, как говорят, в других Unix-системах

(помнится, даже и в некоторых реализациях Linux мне такое встречалось) указание какой-либо из таких опций — обязательно. Так что рассмотрим их по порядку.

Для выведения списка отобранных файлов на экран в общем случае предназначена опция -print. Вывод этот имеет примерно следующий вид:

```
$ find . -name f* -print

./file1

./file2

./dir1/file3
```

Сходный смысл имеет и опция -ls, однако она выводит более полные сведения о найденных файлах, аналогично команде ls с опциями -dgils:

```
$ find / -fstype ext3 -name zsh -ls

88161  511 -rwxr-xr-x   1 root  root    519320 Ноя 23 15:50 /bin/zsh
```

Важное, как мне кажется, замечание. Если команда указанного вида будет дана от лица обычного пользователя (не root-оператора), кроме приведенной выше строки вывода, последуют многочисленные сообщения вроде

```
find: /root: Permission denied
```

указывающие на каталоги, закрытые для просмотра обычным пользователем, и весьма мешающие восприятию результатов поиска. Чтобы подавить их, следует перенаправить вывод сообщения об ошибках в файл /dev/null, то есть указать им «Дорогу никуда»:

```
$ find / -fstype ext3 -name zsh -ls 2> /dev/null
```

Идем далее. Опция -delete уничтожит все файлы, отобранные по указанным критериям. Так, командой

```
$ find ~ -atime +100 -delete
```

будут автоматически стерты все файлы, к которым не было обращения за последние 100 дней (из молчаливого предположения, что раз к ним три месяца не обращались — значит, они и вообще не нужны). Истреблению подвергнутся файлы в подкаталогах любого уровня вложенности — но не включающие их подкаталоги (если, конечно, последние сами не подпадают под критерии отбора).

И, наконец, опция -exec — именно ею обусловлено величие утилиты find. В качестве значения ее можно указать любую команду с необходимыми опциями — и она будет выполнена над отобранными файлами, которые будут рассматриваться в качестве ее аргументов. Проиллюстрируем это на примере.

Использовать для удаления файлов опцию -delete, как мы это только что сделали — не самое здоровое решение, ибо файлы при этом удаляются без запроса, и можно случайно удалить что-нибудь нужное. И потому достигнем той же цели следующим образом:

```
$ find ~/ -atime +100 -exec rm -i {} ;
```

В этом случае (вне зависимости от настроек псевдонимов командной оболочки) на удаление каждого отобранного файла будет запрашиваться подтверждение.

Обращаю внимание на последовательность символов {} \; (с пробелом между закрывающей фигурной скобкой и обратным слэшем) в конце строки. Пара фигурных скобок {} символизирует, что свои аргументы исполняемая команда (в примере — rm) получает от результатов отбора команды find, точка с запятой означает завершение команды-значения опции -exec, а обратный слэш экранирует ее специальное значение от интерпретации командной оболочкой.

Кроме опции действия -exec, у команды find есть ещё одна, близкая по смыслу, опция — -ok. Она также вызывает некую произвольную команду, которой в качестве аргументов передаются имена файлов, отобранные по критериям, заданным опцией (опциями) поиска. Однако перед выполнением каждой операции над каждым файлом запрашивается подтверждение.

Приведенный на предыдущей странице пример, хотя и вполне жизненный, достаточно элементарен. Рассмотрим более сложный случай — собирание в один каталог всех скриншотов в формате PNG, разбросанных по древу домашнего каталога:

```
$ find ~/ -name *.png -exec cp {} imagesdir ;
```

В результате все png-файлы будут изысканы и скопированы (или — перемещёны, если воспользоваться командой mv вместо cp) в одно место.

А теперь — вариант решения задачи, которая казалась мне некогда трудно разрешимой: рекурсивное присвоение необходимых атрибутов доступа в разветвленном дереве каталогов — различных для регулярных файлов и каталогов.

Зачем и отчего это нужно? Поясню на примере. Как-то раз, обзаведясь огромным по тем временам (40 Гбайт) винчестером, я решил собрать на него все нужные мне данные, рассеянные по дискам CD-R/RW (суммарным объёмом с полкубометра) и нескольким сменным винчестерам, одни из которых были отформатированы в FAT16, другие — в FAT32, третьи — вообще в ext2fs (к слову сказать, рабочей моей системой в тот момент была FreeBSD). Сгрузив все это богачество в один каталог на новом диске, я создал в нем весьма неприглядную картину.

Ну, во-первых, все файлы, скопированные с CD и FAT-дисков, получили (исключительно из-за неаккуратности монтирования, с помощью должных опций этого можно было бы избежать, но — спешка, спешка...) биты исполняемости, хотя были это лишь файлы данных. Казалось бы, мелочь, но иногда очень мешающая; в некоторых случаях это не позволяет, например, просмотреть html-файл в Midnight Commander простым нажатием Enter. Во-вторых, для некоторых каталогов, напротив, исполнение не было предусмотрено ни для кого — то есть я же сам перейти в них не мог. В третьих, каталоги (и файлы) с CD часто не имели атрибута изменения — а они нужны мне были для работы (в т.ч. и редактирования). Конечно, от всех этих артефактов можно было бы избавиться, предусмотрев должные опции монтирования накопителей (каждого накопителя — а их число, повторяю,

измерялось уже объёмом занимаемого пространства), да я об этом и не подумал — что выросло, то выросло. Так что ситуация явно требовала исправления, однако проделать вручную такую работу над данными более чем в 20 Гбайт виделось немыслимым.

Да так оно, собственно, и было бы, если б не опция -exec утилиты find. Каковая позволила изменить права доступа требуемым образом. Итак, сначала отбираем все регулярные файлы и снимаем с них бит исполнения для всех, заодно присваивая атрибут изменения для себя, любимого:

```
$ find ~/dir_data -type f
      -exec chmod a-x,u+w {} ;
```

Далее — поиск каталогов и обратная процедура над итоговой выборкой:

```
$ find ~/dir_data -type d
      -exec chmod a+xr,u+w {} ;
```

И дело — в шляпе, все права доступа стали единообразными (и теми, что мне нужны). Именно после этого случая я, подобно митьковскому Максиму, проникся величием философии марксизма (пардон, утилиты find). А ведь это ещё не предел ее возможностей — последний устанавливается только встающими задачами и собственной фантазией...

Так, с помощью команды find легко наладить периодическое архивирование результатов текущей работы. Для этого перво-наперво создаем командой tar полный архив результатов своей жизнедеятельности:

```
$ tar cvf alldata.tar ~/*
```

А затем в меру своей испорченности (или, напротив, аккуратности), время от времени запускаем команду

```
$ find ~/ -newer alldata.tar
      -exec tar uvf alldata.tar {} ;
```

ещё один практически полезный вариант использования команды find в мирных целях — периодическое добавление отдельно написанных фрагментов к итоговому труду жизни (например, собственным мемуарам). Впрочем, чтобы сделать это, необходимо сначала ознакомиться с командами обработки файлов, к которым мы вскоре обратимся.

А пока — об ограничении возможностей столь замечательной сцепки команды find с опцией действия -exec (распространяющиеся и на опцию -ok). Оно достаточно очевидно: вызываемая любой из этих опций команда выполняется в рамках самостоятельного процесса, что на слабых машинах, как говорят, приводит к падению производительности (должен заметить, что на машинах современных заметить этого практически невозможно).

Тем не менее, ситуация вполне разрешима. И сделать это призвана команда xargs. Она определяется как построитель и исполнитель командной строки со стандартного ввода. А поскольку на стандартный ввод может быть направлен вывод команды find — xargs воспримет результаты ее работы как аргументы

какой-либо команды, которую, в свою очередь, можно рассматривать как аргумент ее самоё (по умолчанию такой командой-аргументом является /bin/echo).

Использование команды xargs не связано с созданием изобилия процессов (дополнительный процесс создается только для нее самой). Однако она имеет другое ограничение — лимит на максимальную длину командной строки. Во всех BSD-системах, которые мне довелось видеть, этот лимит составляет 65536, что определяется командой следующего вида:

```
$ sysctl -a | grep kern.argmax
```

И способы изменить этот лимит мне не известны — был бы благодарен за соответствующую информацию.

Команды обработки текстов: введение

Только что речь шла о командах, которые манипулируют файлами как целыми, не затрагивая их содержания (и, в общем случае, от такового не зависящих). Ныне же речь пойдет о командах, создающих и изменяющих внутреннее содержание файлов, правда, только текстовых.

Конечно, само по себе манипулирование файлами (копирование, перемещёние и т.д.) также подразумевает изменение содержания некоторых файлов, но только одного-единственного типа (а именно - каталогов), однако собственно внутренняя сущность обычных файлов при этом не изменяется. Предметом же настоящей интермедии будут штатные средства POSIX-систем, позволяющие в той или иной мере учитывать контент файлов и манипулировать им. Разумеется, манипулирование контентом возможно только для регулярных файлов. При этом многие их разновидности (бинарные файлы, файлы графических форматов и word-процессоров) требуют для изменения своего содержания специальных средств - а именно, компиляторов и прикладных программ, в которых они создавались. Однако здесь о них разговора не будет - ибо целью моей было продемонстрировать мощь обычных команд для решения многих пользовательских задач. Правда, на самом деле команды модификации контента действенны преимущественно для файлов текстовых.

Однако круг объектов таких команд не столь уж узок, как может показаться. Ведь именно в виде обычных текстовых файлов в ОС POSIX-семейства хранится масса общесистемной информации, исполняемых сценариев, баз данных атрибутов самых разных объектов. Далее - собственно нарративные тексты любого содержания: ведь чисто текстовый формат для них куда роднее, чем всякого рода *.doc и *rtf. Ну и никем не возбраняется использовать такие команды в отношении текстов с разметкой - HTML ли, XML, TeX или ещё чего. Так что поле приложения рассматриваемых команд - достаточно обширно.

Просмотр файлов

Однако прежде чем как-то манипулировать контентом файлов, желательно этот самый контент некоторым образом просмотреть. И тут можно вспомнить о команде cat, посредством которой мы некогда создавали файлы. Данная с именем файла в качестве аргумента, она выведет его содержимое на экран. Можно использовать и конструкцию перенаправления:

```
$ cat < filename
```

Не смотря на то, что в принципе это разные вещи, результат будет тот же - вывод содержимого файла на экран.

Недостаток команды cat как средства просмотра - невозможность перемещёния
по телу файла: выведя содержимое, она завершает свою работу. Конечно, «пролистывание» выведенного возможно, но - средствами системной консоли, а не самой команды.

Поэтому обычно для просмотра содержимого файлов используются специальные программы постраничного просмотра - т.н. pager'ы, очередной пример того, что передача этого термина исконно русским словом «пейджер» (а мне попадалось и такое) может создать совершенно превратное представление о сути дела.

В Unix-системах имеется две основные программы pager'a - more и less. Первая из них допускает только однонаправленный (вперед) просмотр и имеет слабые интерактивные возможности. Почему ныне и представляет лишь исторический интерес, так что о ней я говорить не буду. Тем более, что в современных свободных POSIX-системах она как таковая отсутствует: файл с именем /usr/bin/more, который можно обнаружить во FreeBSD и некоторых дистрибутивах Linux, на самом деле представляет собой жёсткую или символическую ссылку на ту же самую программу, что и команда less. Хотя эта программа проявляет несколько различные свойства, в зависимости от того, какой из указанных команд она вызвана, функции ее от этого не меняются. Так что дальше я буду говорить только о команде less.

Самый простой способ вызова команды

```
$ less filename
```

после чего на экран выводится содержимое файла, указанного в качестве аргумента, по которому можно перемещаться в обоих направлениях, как построчно, так и постранично. В нижней строке экрана можно видеть символ двоеточия - приглашения для ввода команд различного назначения. В частности, нажатие клавиши h выводит справку по использованию less, а клавиши q - выход из программы просмотра (или из просмотра справочной системы, если она была перед этим вызвана). Если команда была вызвана как more (это достигается ещё и специальной опцией - less -m), вместо символа двоеточия в нижней строке будет выведено имя файла с указанием процента просмотра:

```
command.txt 3%
```

что, однако, не воспрещает и здесь давать ее встроенные команды — вводом символа двоеточия (:) и закрепленной за командой литеры (или их сочетания).

Большинство встроенных команд less предназначено для навигации по телу файла. Осуществляется она несколькими способами:

- с помощью стандартных клавиш управления курсором: PageDown или Spacebar (вперед на один экран), PageUp (назад на один экран), Down или Enter (вперед на одну строку), Up (назад на одну строку), Right (на

пол-экрана вправо), Left (на пол-экрана влево);

• с помощью предопределенных клавишных комбинаций, сходных с управляющими клавиатурными последовательностями командных оболочек и таких текстовых редакторов, как emacs и joe (хотя и не всегда с ними совпадающими): Control+V (на один экран вперед), Escape-V (на один экран назад), Control+N (на одну строку вперед), Control+P (на одну строку назад);

• с помощью фиксированных символьных клавиш, иногда подобных таковым командного режима редактора vi: z и w (вперед и назад на один экран), e и y (вперед и назад на одну строку, можно использовать также привычные по vi клавиши j и k, соответственно), d и u (вперед и назад на пол-экрана).

Последний способ интересен тем, что допускает численные аргументы перед символьной командой: так, нажатие 3e приведет к перемещёнию на три строки вперед, а 2w - на два экрана назад.

Помимо «плавной», так сказать, навигации, можно перемещаться по файлу и скачками (jumping): нажатие клавиши с символом g (или последовательности Escape-<) позволяет переместиться к первой строке файла, клавиши G (регистр важен! дублирующий вариант - Escape->) - к последней его строке, клавиши p - к началу файла.

Кроме навигации, имеется и возможность двустороннего поиска - в направлении как конца, так и начала файла. Для поиска вперед требуется ввести символ прямого слэша (/) и за ним - искомое сочетание символов. Поиск в обратном направлении предваряется символом вопроса (?). В обоих случаях в шаблоне поиска можно использовать стандартные регулярные выражения *, ?, [список_символов] или [диапазон_символов]. Нажатие клавиши n (в нижнем регистре) приводит к повторному поиску в заданном ранее направлении, клавиши N (в верхнем регистре) - к поиску в направлении противоположном.

Управляющие комбинации команды less могут быть переопределены с помощью команды lesskey. Формат ее

```
$ lesskey -o output input
```

В качестве входных данных выступает простой текстовый файл (по умолчанию - ~/.lesskey, однако его следует создать самостоятельно), описывающий клавишные последовательности в следующем, например, виде:

```
#command

\r       forw-line

\n       forw-line

...

k        back-line
```

...

Выходные данные - создаваемый из текстового двоичный файл, который собственно и используется командой less. Стандартное для него имя - ~/.less.

Команда less допускает одновременный просмотр нескольких файлов. Для этого ее следует вызвать в форме

```
$ less file1 file2 ... file#
```

после чего между открытыми файлами можно переключаться посредством :n (к следующему файлу), :p (к предыдущему файлу), :x (к первому файлу). Путем нажатия :d текущий файл исключается из списка просмотра. Символ двоеточия во всех этих случаях вводится с клавиатуры в дополнение к приглашению на ввод команд.

Команда less имеет великое множество опций - описание их на странице экранной документации занимает более дюжины страниц, поэтому задерживаться на них я не буду. Следует заметить только, что опции эти могут использоваться не только в командоной строке при запуске less, но и интерактивно - после символа дефиса в приглашении ввода. Так, указав там -m, можно включить т.н. промежуточный формат приглашения (с отображением процентов просмотренного объёма файла), а с помощью -M - длинный (more-подобный) формат, при котором в приглашении дополнительно указываются имя файла, его положение в списке загруженных файлов, просматриваемые ныне строки:

```
command.html (file 2 of 10) lines 1-29/1364 2%
```

Значение команд постраничного просмотра файлов ещё и в том, что именно с их помощью осуществляется доступ к экранной документации (man-страницам). Команда

```
$ man cmd_name
```

как было описано в предыдущей интермедии, на самом деле вызывает определенный по умолчанию pager для просмотра соответствующего файла /usr/share/man/man#/cmd_name.gz. Какой именно - определяется переменной PAGER в пользовательских настройках.

Кроме команд постраничного просмотра, существуют команды для просмотра фрагментарного. Это - head и tail, выводящие на экран некоторую фиксированную порцию файла, указанного в качестве их аргумента, с начала или с конца, соответственно. По умолчанию эта порция для обеих команд составляет десять строк (включая пустые). Однако ее можно переопределитьg произвольным образом, указав опции -n [число_линий] или -c [количество_байт]. Например, команда

```
$ head -n 3 filename
```

выведет три первые строки файла filename, а команда

```
$ tail -c 100 filename
```

его последние 100 байт. При определении выводимого фрагмента в строках

название опции (n) может быть опущено - достаточно числа после знака дефиса.

Существуют и средства просмотра компрессированных файлов. Для файлов, сжатых программой gzip, можно использовать команды zcat и zmore, для спрессованных командой bzip2 - команду bzcat. Использование их ничем не отличается от аналогов для несжатых файлов - в сущности, именно они и вызываются для обеспечения просмотра. В случае команды zmore, как нетрудно догадаться, на самом деле используется команда less (сама по себе она аналога для компрессированных файлов не имеет).

Сравнение, объединение и деление файлов

Следующая важная группа операций над контентом файлов - сравнение файлов по содержанию и различные формы объединения файлов и их фрагментов. Начнем со сравнения. Простейшая команда для этого - cmp в форме

```
$ cmp file1 fil2
```

производит построчное сравнение файлов, указанных как первый и второй аргументы (а более их и не предусмотрено, все указанное после второго аргумента игнорируется). В случае идентичности сравниваемых файлов не происходит ничего, кроме возврата приглашения командой строки. Если же между файлами имеются различия, выводится номер первого различающегося символа и номер строки, в которой он обнаружен:

```
file1 file2 differ: char 27, line 4
```

Это означает, что различия между файлами начинаются с 27-го от начала файла символа (включая пробелы, символы конца строк и т.д.), который имеет место быть в строке 4. С помощью опций -l и -z можно заставить команду cmp вывести номера всех различающихся символов в десятичном или шестнадцатеричном формате, соответственно.

Более информативный вывод обеспечивает команда diff. Она также осуществляет построчное сравнение двух файлов, но выводит список строк, в которых обнаружены отличия. Например, для двух файлов вида

```
$ less file1

line 1

line 2

line 3

line 4

line 5
```

и

```
$less file2

line 1
```

```
line 2

line 3

line 3a

line 4

line 5
```

это будет выглядеть следующим образом:

```
$ diff file1 file2

3a4

> line 3a
```

Если различия будут выявлены более чем в одной строке, для каждого расхождения будет выведен аналогичный блок. Смысл его - в том, какие строки первого файла должны быть преобразованы, и как именно, для того, чтобы файлы стали идентичными. Первая линия блока вывода содержит номер строки первого файла, подлежащей преобразованию, номер соответствующей строки второго файла и обозначенное буквенным символом преобразование, во второй линии приведена собственно строка - предмет преобразования. Символы преобразования - следующие:

- a (от append) указывает на строку, отсутствующую в первом файле, но присутствующую во втором;
- c (от change) фиксирует строки с одинаковым номером, но разным содержанием;
- d (от delete) определяет строки, уникальные для первого файла.

То есть в данном примере для преобразования file1 в file2 в него после строки 3 должна быть вставлена строка 4 из второго файла, что символизирует вторая линия блока - > line 3a, где > означает строку из первого сравниваемого файла. Если же аргументы команды diff дать в обратном порядке, вывод ее будет выглядеть следующим образом:

```
$ diff file2 file1

4d3

< line 3a
```

показывающим, что для достижения идентичности из file2 должна быть удалена
четвертая строка (< line 3a, где < означает строку из второго файла). Если же произвести сравнение file1 с file3, имеющим вид

```
$ less file3
```

```
line 1

line 2

line 3a

line 4

line 5
```

то вывод команды

```
$ diff file1 file3

3c3

< line 3

---

> line 3a
```

будет означать необходимость замены третьей строки из file1 (символ <) на третью строку из file3 (символ >).

Такая форма вывода команды diff называется стандартной. С помощью опции -c можно задать т.н. контекстную форму вывода, при которой на экран направляется не только различающиеся строки, но и строки, их окружающие (то есть контекст, в котором они заключены):

```
diff -c file1 file2

*** file1       Sun May 12 11:44:44 2002

--- file2       Mon May 13 15:17:27 2002

***************

*** 1,5 ****

--- 1,6 ----

  line 1

  line 2

  line 3

+ line 3a

  line 4

  line 5
```

Количество строк контента задается опцией -C:

```
diff -C 1 file1 file2                                   ttyv1

*** file1       Sun May 12 11:44:44 2002

--- file2       Mon May 13 15:17:27 2002

***************

*** 3,4 ****

--- 3,5 ----

  line 3

+ line 3a

  line 4
```

В этом примере значение опции -C (единица) предписывает вывод по одной строке контекстного окружения вокруг различающейся строки. Сами же различающиеся строки помечаются следующим образом: знаком - (минус, или дефис) - строки, подлежащие удалению из первого файла, знаком + (как в примере) - строки, которые должны быть добавлены, знаком ! - просто различающиеся строки.

Кроме контекстного формата, используется ещё и вывод в унифицированном формате, что предписывается опцией -U [значение], в качестве значения указывается число строк. В нем для обозначения изменяемых строк используются только символы + и -, а сам вывод чуть короче, чем при использовании контекстного формата.

С помощью многочисленных опций команды diff сравнение файлов может быть детализовано и конкретизировано. Так, опция -b предписывает игнорировать «пустые» символы пробелов и табуляции в конце строк, а опция -w - вообще «лишние» пробелы (и те, и другие обычно имеют случайное происхождение). При указании опции -B игнорируются пустые строки, то есть не содержащие никаких иных символов, кроме перевода каретки; строки с символами табуляции или пробела как пустые не рассматриваются, для их игнорирования требуется опция -w. Благодаря опции -i при сравнении не принимается во внимание различие регистров символов, а опция -I regexp определяет регулярные вырвжения, строки с которыми также игнорируются при сравнении.

В качестве аргументов команды diff (одного или обоих) могут выступать также каталоги. Если каталогом является только один из аргументов, для сравнения в нем отыскивается файл, одноименный второму аргументу. Если же оба аргумента суть каталоги, в них происходит сравнение всех одноимённых файлов в алфавитном порядке (вернее, в порядке ASCII-кода первого символа имени, разумеется). Благодаря опции -r сравнение файлов может осуществляться и во вложенных подкаталогах.

Вывод команды diff может быть перенаправлен в файл. Такие файлы различия именуются diff-файлами или, применительно к исходным текстам программ, патчами (patches), о которых будет сказано несколько позже. Именно с помощью таких патчей обычно распространяются изменения к

программам (дополнения, исправления ошибок и т.д.).

В принципе, команда diff и придумана была именно для сравнения файлов исходников, над которыми ведут работу несколько (в пределе - неограниченное количество, как в случае с Linux) человек. Однако невозбранно и ее использование в мирных целях - то есть для сравнения просто повествовательных текстов. Единственное, что следует понимать при этом абсолютно ясно - то, что diff выполняет именно построчное сравнение. То есть: сравнение последовательностей символов, ограниченных символами конце строки с обеих сторон. И, соответственно, непрерывная абзацная строка в стиле emacs и vi - совсем не то же самое, что строка, образуемая в редакторе joe на границе экрана. Впрочем, это - вопрос, к которому ещё не раз придется возвращаться.

Как уже было отмечено, команда diff осуществляет сравнение двух файлов (или - попарное сравнение файлов из двух каталогов). Однако, поскольку Бог, как известно, любит троицу, есть и команда diff3, позволяющая сранить именно три файла, указываемые в качестве ее аргументов. По действию она не сильно отличается от двоичного аналога. И потому изучение ее особенностей предлагается в качестве самостоятельного упражнения приверженцам троичной идеологии.

Существуют и средства для сравнения сжатых файлов. Это - zcmp и zdiff. Подобно командам просмотра, ими просто вызываются соотвествтующие команды cmp и diff. И потому использование их не имеет никаких особенностей.

От вопроса сравнения файлов плавно перейдем к рассмотрению способов их объединения. Для этого существует немало команд, из которых по справедливости первой должна идти команда cat, поскольку именно сие есть ее титульная функция (cat — от concatenation, сиречь объединения). Ранее уже упоминалось, что она способна добавлять информацию со стандартного ввода в конец существующего файла. Однако дело этим не ограничивается. В форме

```
$ cat file1 file2 ... file# > file_all
```

она создает новый файл, включающий в себя содержимое всех файлов-аргументов (и именно в том порядке, в каком они приведены в командной строке). Операция, казалось бы, нехитрая - однако представьте, сколько действий потребовалось бы в текстовом процессоре (например, в Word'e) для того, чтобы создать синтетический вариант из полутора десятков фрагментов, раскиданных по разным каталогам?

А вот команда patch выступает в качестве диалектической пары для команды diff, именно она вносит в файл те изменения, которые документируются последней. Выглядит эта команда примерно так:

```
$ patch file1 diff_file
```

в ответ на что последует нечто вроде следующего вывода:

```
Hmm...  Looks like a normal diff to me...

Patching file file1 using Plan A...
```

```
Hunk #1 succeeded at 4.

done
```

В результате исходная версия file1 будет сохранена под именем file1.orig, а сам он преобразован в соответствие с описанием diff-файла. Возможна и форма

```
patch < diff_file
```

В этом случае команда patch попытается сама определить имя файла-оригинала, и, если это ей не удастся, даст запрос на его ввод. Обращаю внимание на символ перенаправления ввода, поскольку если его опустить, имя dif-файла будет воспринято как первый аргумент команды (то есть имя файла-оригинала).

В качестве второго аргумента команды patch могут использоваться dif-файлы не только в стандартном, но и в контекстном или унифицированном формате. Это следует указать посредством опции -c или -u, соответственно.

Сочетание команд diff и patch очень широко используется при внесении изменений в исходные тексты программы. В частности, они применяются для внесения дистрибутив-специфичных изменений в deb-пакеты репозиториев Ununtu и Mint.

Не менее часто, чем в слиянии, возникает и необходимость в разделении файлов на части. Цели этой служит команда split. Формат ее:

```
$ split [options] filename [prefix]
```

В результате исходный файл будет разбит на несколько отдельных файлов вида prefixaa, prefixab и так далее. Значение аргумента prefix по умолчанию - x (то есть без его указания итоговые файлы получат имена xaa, xab и т.д.).

Опции команды split задают размер выходных файлов - в байтах (опция -b) или количестве строк (опция -l). Первой опцией в качестве единицы, кроме байтов, могут быть заданы также килобайты или мегабайты - добавлением после численного значения обозначения k или m, соответственно.

Команда split может использоваться для разбиения файлового архива на фрагменты, соответствующие размеру резервных носителей. Так, в форме

```
$ split -b 1474560 arch_name
```

она обеспечит разбиение архива на части, какждая из которых может быть записана на стандартную трехдюймовую дискету. А посредством

```
$ split -b 650m arch_name
```

архив можно подготовить к записи на носители CD-R/RW. Легко догадаться, что обратное слияние таких фрагментированных файлов можно выполнить командой cat.

В BSD-реализации команды split имеется опция -p (от pattern — шаблон), благодаря которой файл может быть разделена на фрагменты, ограниченные строками, содержащими текст, приведенный в качестве значения шаблона.

Linux-реализация команды split таким свойством не обладает. Однако взамен этому в Linux есть команда csplit, именно для разделения файла по шаблону и предназначенная.

Показать, как она работает, проще всего на конкретном примере. Предположим, у нас имеется книга в формате Plain Text, состоящая из введения и 23 глав, которую надо разбить на соответствующее количество фрагментов. Для этого сочиняется такая командная конструкция:

```
$ csplit -f chapter mybook.txt '/Глава/' {23}
```

Здесь опция -f задаёт маску имён файлов, на которые будет разбит исходный текст (то есть файл mybook.txt). Шаблон, по которому будет выполняться разбиение — слово Глава ограничено прямыми слэшами и заключено в «строгие» кавычки. А число в фигурных скобках указывает, сколько раз надо повторить процедуру разбиения по заданному шаблону. И в результате мы получаем серию файлов вида chapter##, где файл chapter00 будет включать текст от начала до первой строки со словом Глава (которая, как ни странно, будет главой первой), chapter01 — от строки Глава первая до Главы второй, и так далее. Исходный файл при этом останется в неприкосновенности.

Поиск в файлах: grep сотоварищи

В одном из предыдущих разделов говорилось о поиске файлов посредством команды find. Ныне же речь пойдет о поиске внутри файлового контента - то есть поиске текстовых фрагментов. Для этого в POSIX-системах используется семейство утилит grep — собственно grep, egrep и fgrep, несколько различающихся функционально. Впрочем, в большинстве систем все это суть разные имена (жёсткие ссылки) одной и той же программы, именуемой GNU-реализацией grep, включающей ряд функций, свойственных ее расширенному аналогу, egrep. Соответственно, поиск текстовых фрагментов в файлах может быть вызван любой из этих команд, хотя в каждом случае функциональность их будет несколько различаться.

Однако начнем по порядку. Самой простой формой команды grep является следующая:

```
$ grep pattern files
```

где pattern - искомая последовательность символов, а files - файлы, среди которых должен производиться поиск (или - просто одиночный файл). В указании
имен файлов допустимы обычные маски, например, командой

```
$ grep line ./*
```

будут найдены строки вида line во всех файлах текущего каталога. Шаблон для поиска не обязан быть односложным. Правда, если в нем используются последовательности символов, разделенные пробелами, последние должны тем или иным способом экранироваться, иначе в качестве шаблона будет воспринято только первое слово. Например, каждый пробел может предваряться символом обратного слэша (\), или просто все искомое выражение заключается в одинарные или двойные кавычки.

Шаблоны могут включать в себя регулярные выражения. причём список

таковых для команды grep существенно шире, чем для команд манипулирования файлами. Так, кроме маски любой последовательности символов (*), любого одиночного символа (?), списка и диапазона символов ([a...z] и [a-z]), могут встречаться:

- . (точка) - маска любого одиночного (но, в отличие от маски ?, обязательно присутствующего) символа; то есть при задании шаблона lin. будут найдены строки, содержашие line или lins, но не lin;

- ^ и $ - маски начала и конца строки, соответственно: по шаблону ^line, будут найдены строки, начинающиеся с соответствующего слова, по шаблону же line$ - им заканчивающиеся;

- маски повторения предыдущего шаблона, \{n\} - ровно n раз, \{n,\} - не менее n раз, \{,m\} - не более m раз, \{n,m\} - не менее n раз и не более m раз.

Маски повторения относятся к так называемым расширенным регулярным выражениям. Для их использования команда grep должна быть дана с опцией -e или в форме egrep — последняя часто определяется в общесистемном или пользовательском профильном файле как псевдоним команды grep:

```
alias grep='egrep -s'
```

или

```
alias grep egrep
```

в оболочках семейств shell и csh, соответственно.

При этом становятся доступными и другие возможности поиска - например, нескольких текстовых фрагментов (соединиенных логическим оператором «ИЛИ») одновременно. Делается это двояко. Первый способ - просто перечисление искомых фрагментов, каждый из которых предваряется опцией -e (и при необходимости экранируется кавычками):

```
$ grep -e pattern1 -e pattern2 files
```

При втором способе оператор между искомыми шаблонами задается в явном
виде:

```
$ egrep 'pattern1|pattern2' files
```

Таким способом очень легко, например, составить оглавление для длинного текста (при наличии некоторой системы рубрикации в нем, разумеется). Для этого достаточно дать команду вроде следующей:

```
$ egrep 'Часть|Глава|Раздел|Параграф' filename
```

Для текста, включающего html- или TeX-разметку, роль рубрик могут выполнять соответствующие ее элементы, например:

```
$ egrep ' <h1>|<h2>|<h3>|<h4>' filename
```

Вывод команды grep может быть перенаправлен в файл, а при необходимости и предварительно отсортирован с помощью соответствующих командных конструкций перенаправления и конвейеризации.

Разумеется, тем же способом можно создать общее оглавление для серии фрагментов, записанных в виде самостоятельных файлов — для этого достаточно только перечислить их имена в качестве аргументов команды. Так, например, если есть необходимость составления детальной карты сайта, включающей ссылки на подрубрики внутри отдельных html-документов, следует применить конструкцию типа:

```
$ egrep '<h1>|<h2>|<h3>|<h4>' path/*.html > sitemap.html
```

ещё одно замечательное свойство команды grep (и egrep) - возможность получения шаблона не со стандартного ввода (то есть не путем набора его с клавиатуры), а из файла. Так, если для приведенного выше случая создать простой текстовый файл shablon, содержащий строку

```
Часть|Глава|Раздел|Параграф
```

выполнять операцию по сборке оглавления впредь (и в любом тексте, хоть частично совпадающем по структуре с рассмотренным) можно будет выполнять таким образом:

```
$ egrep -f shablon filename
```

Опция -f и указывает команде, что список параметров должен извлекаться из файла, указанного в качестве значения опции.

Список опций команды grep не исчерпывается указанными выше. Так, опция -i предписывает игнорировать различие регистров внутри искомого выражения, опция -h - подавляет вывод имен файлов (выводится только содержание найденных строк), тогда как опция -l, напротив, выводит только имена файлов, содержащих найденный шаблон, но не текстовый фрагмент, опция -n выводит номера найденных строк в соответствующих файлах. Весьма важной представляется опция -v, обеспечивающая инверсию поиска: при указании ее выводятся строки, не содержащие шаблона поиска.

Команда grep имеет и аналоги для поиска в сжатых файлах - команду zgrep и семейство команд xzgrep, о которых говорилось в миниочерке про архивацию и компрессию.

Поиск в файлах: утилита search

В дистрибутиве Mint имеется фирменная утилита для поиска текстовых фрагментов в файлах — search/code>. Она входит в состав пакета mintsystem (о котором подробней говорится в очерке об утилите apt) и располагается в каталоге /usr/local/bin/. Формат её вызова таков:

```
$ search for [искомый фрагмент] in [каталог для поиска]
```

Например, в форме

```
$ search for 'дистрибутив Mint' in /home/current/alv.me
```

она отыщет все абзацы с вхождением дистрибутив Mint во всех файлах

указанного каталога и выведёт их в таком виде:

```
/home/current/alv.me/mint/mint17-cin/mint17-02.txt:9:В числе родственников... нет, не примазавшихся, а настоящих, но пошедших другим путём, был и дистрибутив Mint. Сейчас не время обсуждать его взаимоотношения с прародительской Ububtu, но программу установки он унаследовал от неё практически без изменений. По крайней мере, до недавнего времени макроскопических различий в инсталляторах этих систем не наблюдалось.

...
```

Команда search не является полной заменой утилит семейства grep, в частности, она не поддерживает регулярные выражения. Но вполне может служить более простой в использовании альтернативой во многих тривиальных случаях, столь частых в практике применителей-текстовиков.

Sed: средство потокового редактирования

Весьма часто при обработке текстов встает такая задача: заменить одно слово (или последовательность слов) на другое сразу во многих файлах. Как она решается «подоконными» средствами? Обычно - открытием всех подлежащих изменению документов в word-процессоре и применением функции поиска/замены последовательно в каждом из них.

Таким же способом можно воспользоваться и в POSIX-мире. Это просто, но уж больно скучно. Тем паче, что здесь есть очень эффективная альтернатива — средства потокового (неинтерактивного) редактирования, примером которых является sed, с которым мы уже слегка познакомились в очерке о программах в автозапуске.

Потоковое, или неинтерактивное, редактирование не требует загруки документа в память (то есть открытия), как в обычных текстовых редакторах и word-процессорах. Нет, при нем подлежащий изменению файл (или группа файлов) обрабатываются построчно с помощью соответствующих команд, задаваемых как опции единой командной директивы. В наши дни это выглядит анахронизмом, однако в ряде случаев оказывается чрезвычайно эффективным. Каких? - ответ легко дать на нескольких конкретных примерах.

Так, при настройке системы нередко требуется внести мелкие однотипные изменения в серию конфигурационных файлов. Именно с такой ситуацией мы столкнулись, когда захотели увидеть все автоматически запускаемые при старте Cinnamon приложения. И тогда было самое время вспомнить про sed, с помощью которого эта задача была решена одной командой — не откажу себе в удовольствии напомнить её:

```
$ sudo sed -i 's/NoDisplay=true/NoDisplay=false/' /etc/xdg/autostart/*
```

Другой случай - во многих десятках, а то и сотнях файлов требуется изменить одну-единственную строку, причём — одинаковым образом (например, заменить копирайт Васи Пупкина на Петю Лавочкина). Неужто для этой цели нужно вызывать мощный текстовый редактор, грузить в него немерянное количество документов, перемещаться тем или иным способом перемещаться к нужному фрагменту, вносить требуемое изменение? Отнюдь, ибо sed поможет и здесь, позволив выполнить изменение любого количества файлов в пакетном режиме.

Во всем блесе sed показывает себя при редактировании очень больших файлов (одно пролистывание которых требует немалого времени). А также — при редактировании сложных символьных последовательностей в нескольких файлах. Однажды, после очередной реконструкции моего сайта, передо мной встала задача тотальной модификации всех внутренних ссылок. Долго я с ужасом размышлял, как буду делать это в текстовом редакторе, и сколько ошибок при этом насажаю. Пока, раскинув мозгами, не нашел, как сделать это с помощью sed - быстро и, главное, безошибочно.

В самом общем виде sed требует двух аргументов — указания встроенной его команды и имени файла, к которому она должны быть применена. Впрочем, в качестве аргумента можно задать только простую команду, мало-мальски сложное действие (а команды поиска/замены принадлежат к числу сложных) необходимо определить через значения опции -e, заключенные в кавычки (одинарные или двойные - по ситуации). Что же касается имен файлов — то их в качестве аргументов можно указать сколько угодно, в том числе и с помощью масок типа *, *.txt и так далее. Правда, sed не обрабатывает содержимое вложенных подкаталогов, но это — дело поправимое (как — скоро увидим). Так что поиск и замена слова или их последовательности выполняются такой конструкцией:

```
$ sed -e 's/Вася Пупкин/Петя Лавочкин/' *
```

Здесь s - это команда поиска, Вася Пупкин - искомый текст, а Петя Лавочкин - текст для замены. В приведенной форме команда выполнит поиск и замену только первого вхождения искомого текста. Чтобы заменить текст по всему файлу, после последнего слэша (он обязателен в любом случае, без него sed не распознает конца заменяющего фрагмента) нужно указать флаг g (от global). Важно помнить, что если оставить заменяющее поле пустым, искомый текст будет просто удален.

По умолчанию sed выводит результаты своей работы на стандартный вывод, не внося изменений в файлы аргументы. Так где же здесь редактирование? Оно обеспечивается другой опцией - -i, указание которой внесет изменения непосредственно в обрабатываемый файл. В результате команда для замены, например, всех вхождений html на shtml во всех файлах текущего каталога будет выглядеть так:

```
$ sed -i -e 's/html/shtml' *
```

А что делать, если таким же образом нужно обработать файлы во всех вложенных подкаталогах? Придется вспомнить об универсальной команде find, о которой мы не так давно говорили. В форме

```
$ find . -name * -exec sed -i -e 's/html/shtml' * {} \
```

она с успехом справится с этой задачей.

Я привел лишь элементарные примеры использования sed. На самом деле возможности его много шире, но их описание далеко выходит за рамки этого краткого введения.

Текстовый редактор nano

Только что я попытался показать мощь неинтерактивного редактирования, доступную благодаря потоковому редактору sed. Однако, как бы силён он не

был, иногда при всякого рода конфигурировании возникает необходимость и в настоящем текстовом редакторе, интерактивном. Причём желательно способном работать и в терминальном окне графического сеанса, и в «чёрной» консоли.

Записные линуксоиды обычно в таких случаях советуют начинающим применителям Vim или Emacs, в зависимости от собственной религиозной ориентации. Напрочь забывая о том, что эффективная работа в обоих этих редакторах возможна только при доведённых до автоматизма навыках, и к тому же навыках, постоянно тренируемых — иначе они утрачиваются очень быстро. При необходимости же поправить пару строк в конфиге раз или два в месяц приобретать такие навыки просто не имеет смысла.

А вот редактор Nano вполне может сыграть роль своего рода амортизатора для начинающего применителя. Да, это не Vim, не Emacs, и даже не joe. Но с задачей конфигурирования справляется успешно. А в освоении и`обращении — прост, как грабли. Не случайно во многих дистрибутивах Linux он по умолчанию предлагается в качестве общесистемного. В том числе и в таких юзерофильных, как семейство Ubuntu, представители которого, с одной стороны, имеют штатные, мощные и удобные, инструменты редактирования в своих интегрированных средах, с другой — и Vim'ом эти системы не обделены, да и Emacs им устанавливать не возбраняется. Но даже в этих «борброжелательных» дистрибутивах иногда возникает потребность в простом и лёгком консольном редакторе. А многие ли из начинающих применителей способны сразу же смотреть на Vim и Emacs без содрогания?

Столь длинное вступление направлено к тому, что затратить толику времени на освоение Nano — дело стоящее для многих линуксоидов, в том числе и начинающих применителей Mint. Тем более, что, как уже было сказано, в освоении он прост, а возможностей у него больше, чем может показаться на первый взгляд.

Итак, представляю: редактор Nano, или, точнее, GNU nano. Характеризуется авторами как маленький и дружелюбный. Что в целом соответствует истине. Официальным местопребыванием имеет сайт http://www.nano-editor.org, где его текущая стабильная версия (2.2.6) доступна в виде тарбалла исходников и бинарников в форматах rpm и deb.

Впрочем, применителям Mint заботиться о скачивании бинарников и тем более исходников не придётся: Nano имеется в официальном репозитории этого дистрибутива и, более того, устанавливается по умолчанию с инсталляционного Live-носителя, после чего немедленно готов к работе.

Запускается Nano из командной строки консоли или терминального окна одноименной командой, можно — с указанием имени файла, существующего или нового (в последнем случае, как обычно, файл с таким именем будет создан). Поддерживается несколько опций командной строки, как то: -T #, устанавливающей величину (в символах) табуляции, -i, включающей автоматические отступы, -w, отключающей режим жёсткого переноса строк на границе экрана (что очень важно при редактировании конфигурационных файлов), -$, напротив, включающей режим переноса «мягкого» (так называемый softwrap), при котором визуальный перенос осуществляется без разрыва строки, и так далее. Полный список стартовых опций можно посмотреть посредством

```
$ man 1 nano
```

Кроме того, практически все эти опции могут быть прописаны в конфигурационном файле в качестве умолчальных.

Общесистемный конфигурационный файл редактора — /etc/nanorc. Его можно скопировать в свой домашний каталог

```
$ cp /etc/nanorc ~/.nanorc
```

После чего редактировать в своё удовольствие. Кроме того, в каталоге /usr/share/nano имеется большое количество примеров конфигов, адаптированных для разных языков программирования и разметки, а также специально для некоторых дистрибутивов (Gentoo, Debian).

Впрочем, умолчальный конфиг кажется мне вполне соответствующим своим задачам. Единственное вносимое мной в него изменение — это включение режима «мягких» переносов по умолчанию. Для чего нужно снять комментарий со строки

```
set softwrap
```

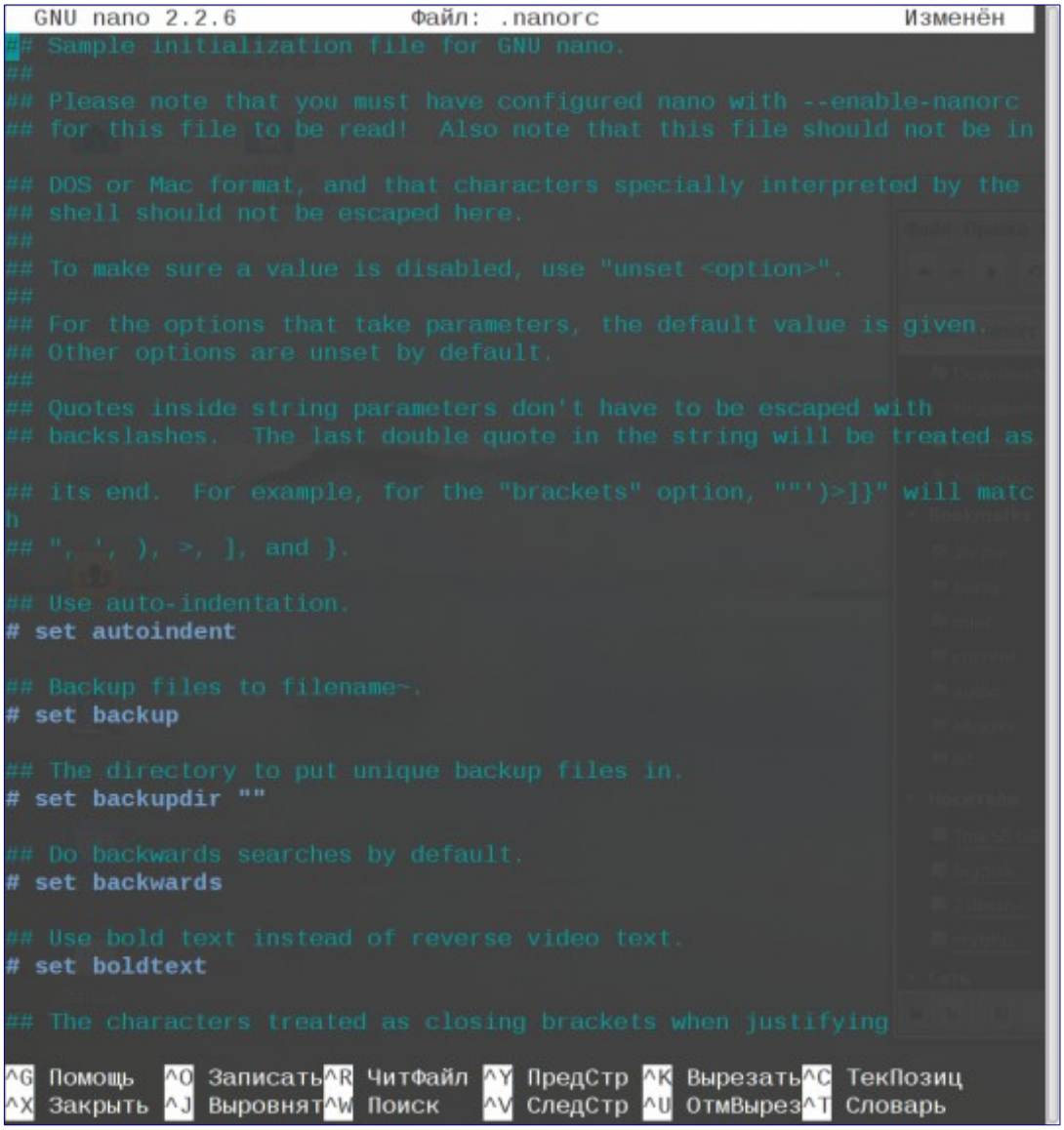
И, напротив, закрыть комментарием строку

```
#set nowrap
```

После этого Nano становится равно пригодным и для редактирования конфигурационных файлов (которые, как известно, не любят произвольных разрывов строк), и для сочинения «мирных» текстов, в которых уходящие за горизонт экрана строки очень мешают, а жёсткие их разрывы также не полезны, ибо затрудняют в дальнейшем поиск.

Поиск, кстати, осуществляется комбинацией клавиш Control+w с последующим нажатием на Enter, повторямыми, сколько требуется. А для замены, в том числе глобальной, используется комбинация Control плюс обратный слэш (\) или Meta+R.

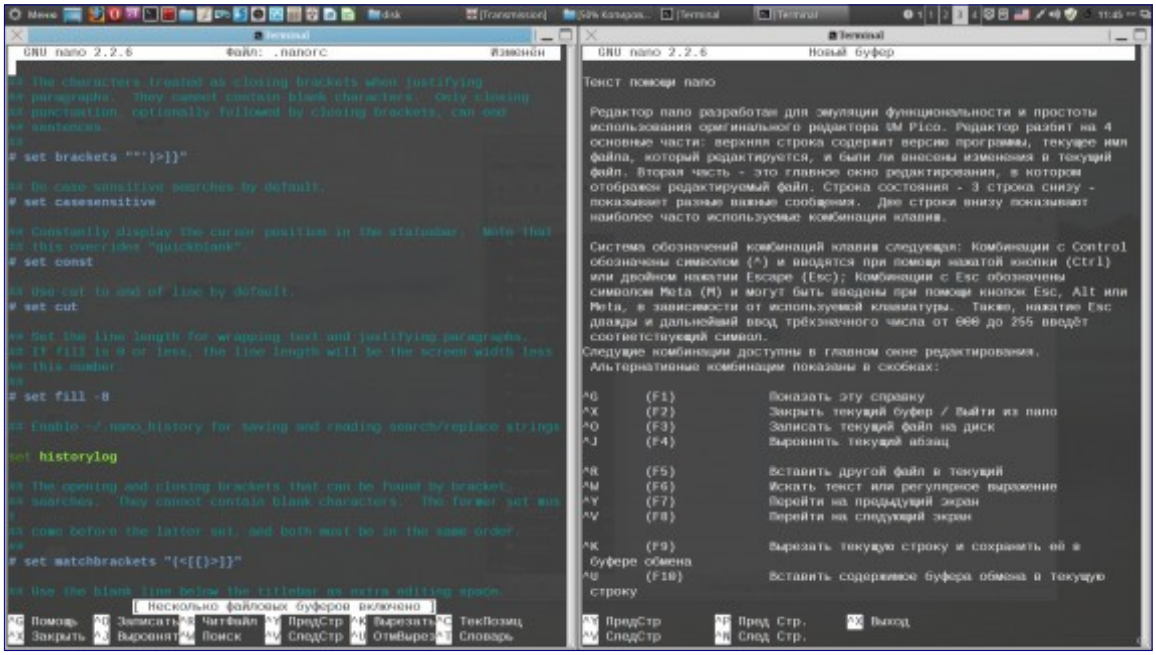
После запуска Nano с указанием существующего файла в качестве аргумента (например, его собственного «домашнего» конфига) перед глазами возникает примерно такая картина:



Вверху — титульная строка, в которой выводятся номер версии программы, имя открытого файла и, в правом углу, сообщение о том, что файл был изменен. В нижней части экрана можно видеть зону подсказки — список основных управляющих клавишных последовательностей (образованных сочетанием Control+литера) с пояснениями на языке установленной локали.

Главнейшей из управляющих последовательностей на первых порах знакомства с редактором будет Control+g (литерные клавиши последовательностей к регистру не чувствительны). Она вызывается весьма подробную справку, в том числе и о тех последовательностях, которые не уместились в двух нижних строках рабочей области. Та же самая справка вызывается и клавишей F1 — если она не перехватывается средой, как это имеет место быть в GNOME Terminal'e, который Cinnamon юзает вместо отсутствующего родного.

Вообще поначалу удобно разместить рядом два терминальных окна, и в одном открывать текст для набора или редактирования, а в другом — вызвать справку и постоянно с ней сверяться:



Потому что, повторяю, управляющих последовательностей в Nano довольно много — больше, чем можно запомнить с одного просмотра, и все они полезны в практической работе.

А ещё они бывают двух видов — в сочетании с клавишей Control и с клавишей Meta. Последней на современных клавиатурах не найти — она эмулируется либо нажатием клавиши Alt, либо нажатием и отпусканием клавиши Escape, хотя поледний способ и не всегда срабатывает..

Собственно управляющие последовательности, Control+литера, предназначены в основном для редактирования текста и операций с файлами. Управляющие последовательности частично дублируются функциональными клавишами F1–F16. Отсутствующие на клавиатуре функциональные клавиши с F13 по F16 вызываются посредством сочетания Shift+F1–F4.

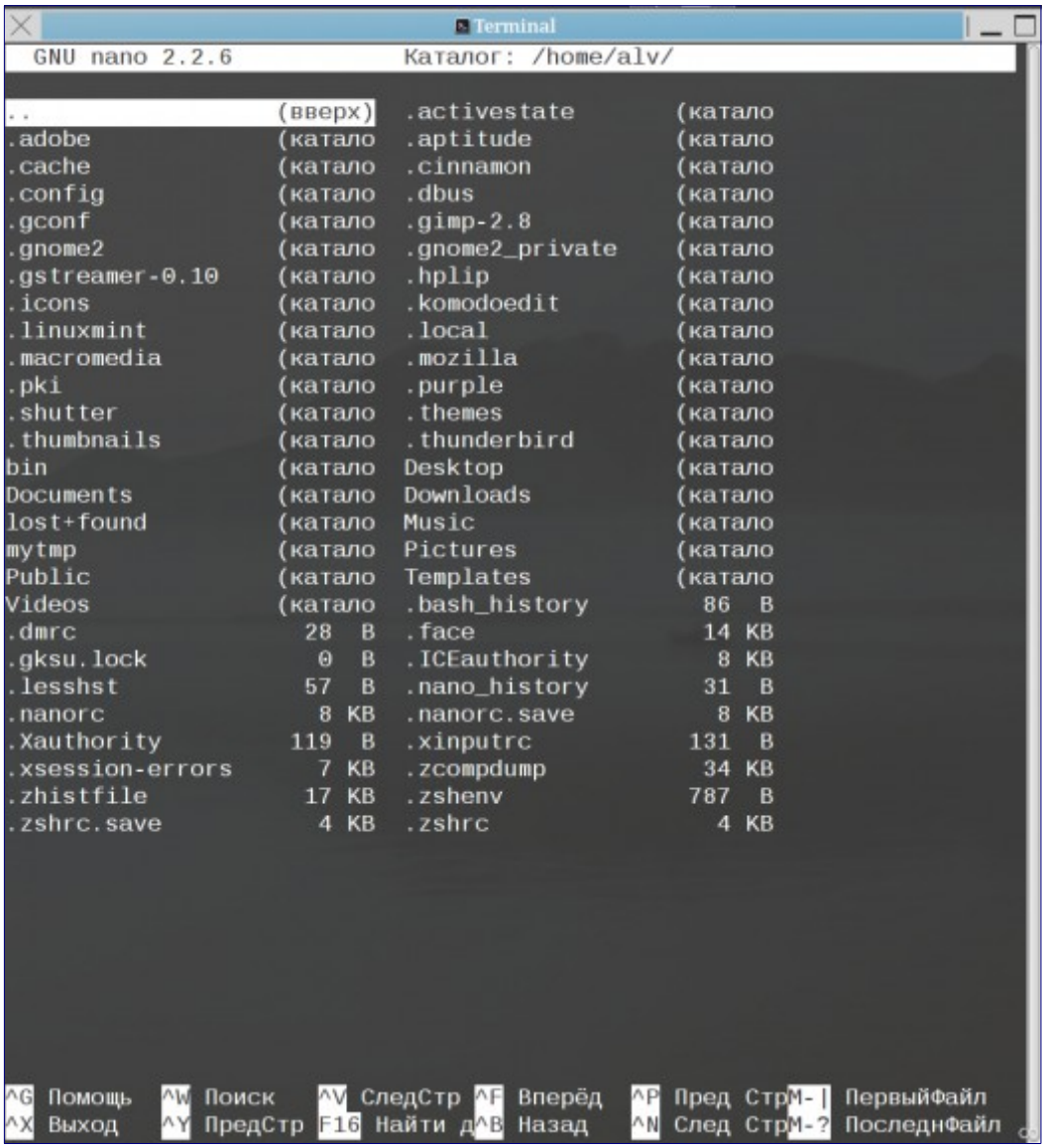
Meta-последовательности (то есть сочетания Meta+литера) теоретически предназначены в основном для временого изменения настроек редактора (тот же результат достигается и опциями командной строки). Однако порою клавиша Meta выступает в роли «усилителя» Control-последовательности.

К слову сказать, в «голой» консоли и Control-, и Meta-последовательности работают при любой раскладке клавиатуры, что латинской, что русской. А вот в терминальных окнах Cinnamon Meta-последовательности при русской раскладке клавиатуры выполнять свои функции отказываются.

Описывать все управляющие и Meta-последовательности не буду — это сделано в той самой экранной подсказке, да в тому же в локализованной системе — на русском языке. Замечу только, что ничего страшного в них нет. Кейбиндинги для навигации по тексту и его редактирования — так называемые Emacs-подобные, примерно такие же, как в шеллах типа Bash или Zsh. И применителю любой из Sh-подобных командных оболочек могут показаться непривычными только те последовательности, которые отражают специфические функции Nano как редактора. Вот эти-то функции рассмотреть стоит.

Как нетрудно догадаться, текстовый редактор может вводить и редактировать тексты, и Nano тут не исключение. Причём он умеет делать это

в нескольких документах параллельно — каждый из них открывается в собственном так называемом буфере. Для чего сначала нужно включить мультибуферный режим — делается это meta-последовательностью Meta+F. Затем каждый новый буфер открывается с помощью комбинации Control+R, либо введя имя файла непосредственно в появившейся строке, либо, нажав Control+T, перейти в режим визуального выбора файла:



Число открытых буферов вроде бы ничем не ограничено — разве что объёмом памяти. Переключение между ними — с помощью Meta+> (вперёд) и Meta+< (назад).

Если при задйствовании нескольких буферов отключить мультибуферный режим (это делается повторением комбинации Meta+F), то открытые буфера никуда не деваются, и переключаться между ними можно по прежнему, но вот открыть новый буфер уже не получится до повторного включения мультибуферного режима. А в днобуферном режиме комбинация Control+R после выбора файла вместо открытия втсавит его содержимое в позиции курсора текущего документа.

Между буферами возможен обмен данными. Так, строка, скопированная (посредством Meta+6) или вырезанная (комбинацией Control+K) в одном буфере, может быть вставлена (с помощью Control+U) в другом. Ну и в «родном», разумеется, тоже.

Надо заметить, что в Nano есть и другой способ одновременного редактирования нескольких файлов. Комбинация клавиш Meta+Z приостанавливает работу редактора (точнее, переводит его в фоновый режим), возвращая приглашение командной строки, в которой Nano можно запустить заново. Но это будет уже другой его экземпляр, и обмен между ними через буфер невозможен — это можно сделать только «мышиным» выделением и вставкой кликом средней её кнопкой. Однако в этой временной командной строке можно выполнить какие-либо команды, а результаты через то же «мышиное» выделение поместить в редактируемый документ. Возврат в который из командной строки — по команде fg.

Очень полезная особенность Nano — возможность включения режима мягкого переноса слов (точнее, символов — softwrap), о которой я уже говорил. Здесь же добавлю, что это можно сделать не только через конфиг или опцию запуска Nano, но и прямо в его сеансе — последовательностью Meta+$. Я уделяю этому вопросу столько внимания, потому что такая, казалось бы, мелочь очень важна для сочинителя нарративных текстов — и не только при доводке их в word-процессоре или программе вёрстки, где лишние разрывы строк — вечный повод для головной боли. Не меньше они мешаются при поиске в архивах собственной нетленки по смутно запомнившимся её фрагментам.

Думаю, важность подстветки синтаксиса понимают не только программисты и профессиональные web-разработчики. Ибо мода сочинять конфиги в XML-формате затронула многие рабочие среды. А разобраться в XML-файле без подсветки — проще удавиться. Да и сочинителям нарративных текстов, не брезгующим одновременно их разметкой (в HTML ли, или в TeX) это тоже лишним не покажется. И Nano поддерживает оную с давних пор, а с некоторого времени эта фича включена в нём по умолчанию. Список поддерживаемых языков программирования и разметки, а также дистрибутив-специфичных файлов можно посмотреть так:

```
$ ls /usr/share/nano
```

И выглядит он следующим образом:

```
asm.nanorc      groff.nanorc     nanorc.nanorc  ruby.nanorc
awk.nanorc      html.nanorc      objc.nanorc    sh.nanorc
cmake.nanorc    java.nanorc      ocaml.nanorc   tcl.nanorc
c.nanorc        makefile.nanorc  patch.nanorc   tex.nanorc
css.nanorc      man.nanorc       perl.nanorc    xml.nanorc
debian.nanorc   mgp.nanorc       php.nanorc
fortran.nanorc  mutt.nanorc      pov.nanorc
gentoo.nanorc   nano-menu.xpm    python.nanorc
```

Правда, возможно, цветовая гамма в них не всем покажется идеальной. Но, как известно, на цвет товарищей нет, и никто не мешает отредактировать её вкусу своих фломастеров.

Далее — проверка орфографии, которая одинаково важна для всех применителей, даже тех, кто, подобно автору этих строк, ею подчас манкирует. Для обеспечения оной в файле ~/nanorc нужно снять комментарий со строки

```
set speller "aspell -x -c"
```

После чего по комбинации Control+T (или по клавише F12, если та не задействована для других целей, например, вызова выпадающего терминала) для спеллинга будет вызываться программа aspell — если она, конечно, установлена и снабжена словарем для требующегося языка. В Mint пакет aspell устанавливается по умолчанию, но сопровождается только английским словарём. Так что для обеспечения проверки орфографии в русскоязычных текстах надо установить пакет aspell-ru.

И, наконец, остается только сделать Nano редактором по умолчанию — чтобы использовать его по умолчанию в командах типа sudoedit и visudo (а также везде, где потребуется впредь). Для чего воспользуемся им самим же, открыв в нем конфигурационный файл своей пользовательской командной оболочки. Например, для Bash — так:

```
$ nano ~/.bashrc
```

И вписав в него такую вот строку:

```
export EDITOR=nano
```

определяющую переменную среды EDITOR. Теперь редактор nano будет вызываться при редактировании системных конфигов, например, командой sudoedit.

Функциональные возможности Nano на фоне Vim или Emacs не производят впечатления исключительно богатых. Однако это — не только минус, но и плюс: ограниченный набор функций легче освоить и особенно — держать в голове при эпизодическом применении. Тем более, что их хватает не только на несложную правку мелких конфигов, но и на сочинение не слишком масштабных нарративных текстов.

Mint и Zsh

В предыдущем очерке работа в CLI была рассмотрена на примере Bash — самой распространённой командной оболочки Linux'а. Однако о ней написаны пуды бумажной литературы и мегабайты сетевых материалов, повторять которые было бы скучно. И к тому же в реальной жизни я её практически не использую. Поэтому далее будет рассмотрена оболочка Zsh.

Кроме борьбы со скукой, есть и немало более технических аргументов в пользу применения Zsh как пользовательской оболочки (login shell):

- функциональность, существенно превосходящая возможности Bash в интерактивной работе;
- расширяемость за счёт дополнительных модулей;
- настраиваемость, ограниченная практически только фантазией

применителя;

• прекрасная документированность — объём официальной документации составляет более 3 МБ в формате HTML (не считая прочих форматов);

• активное сообщество энтузиастов — разработчиков и применителей.

Не последним аргументом в пользу Zsh является его отличная интеграция с утилитой apt, лежащей в основе пакетного менеджмента дистрибутива Mint. До сих пор, описывая действия по управлению пакетами в CLI, я, будучи давним пользователем Zsh, приводил их к общему знаменателю с Bash. В один прекрасный момент мне это надоело. Причём отказываться от мощного функционала Zsh, к которому привык так, что без него как без рук, не не собираюсь. И потому решил впредь помещать в тексты своих сочинений команды в «Zsh'изированной» форме. А для пояснения их сути — написать настоящую серию мини-очерков и включить её в книгу про Mint.

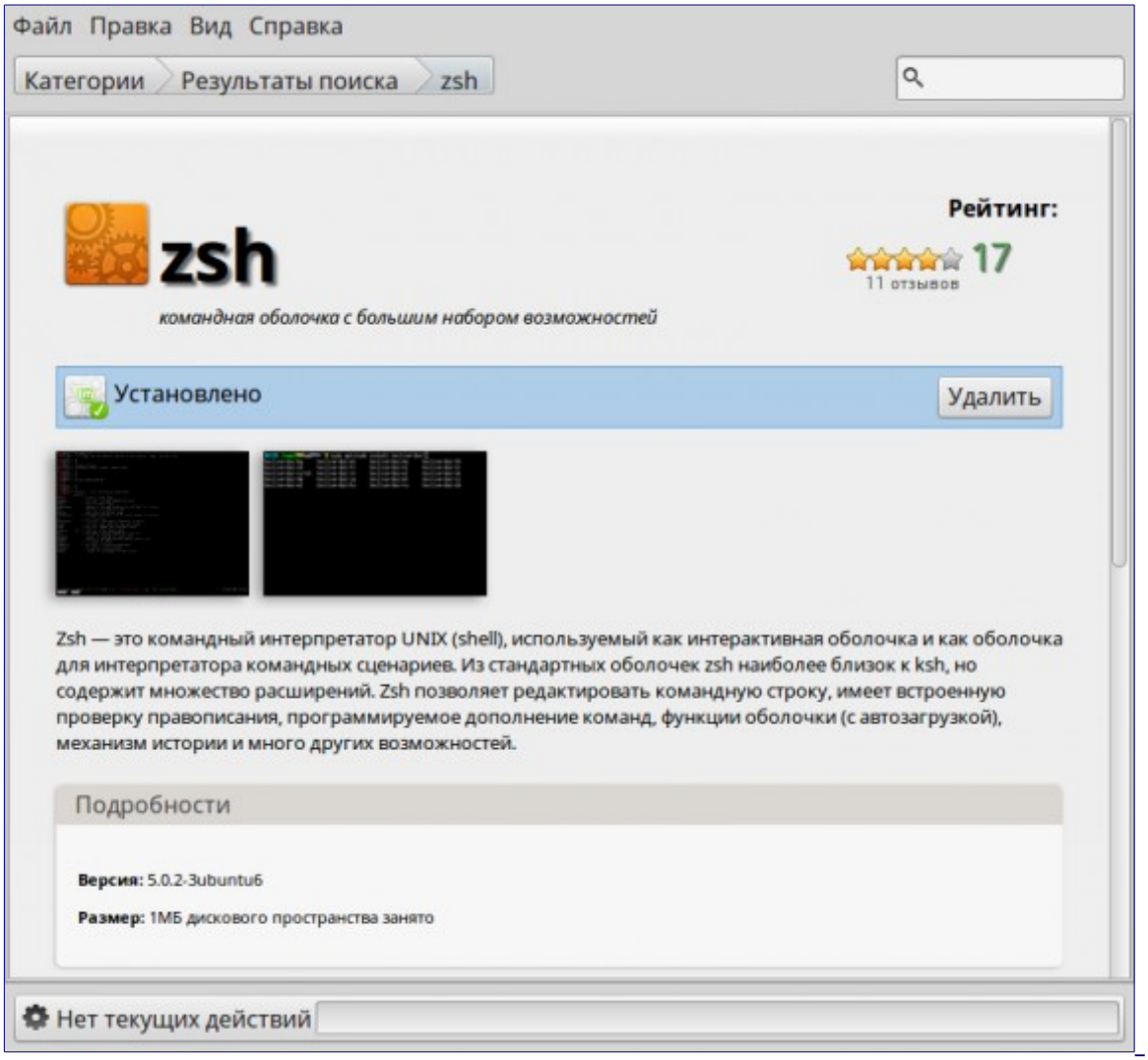
Zsh как login shell

В Mint в качестве системной командной оболочки, то есть интерпретатора общесистемных сценариев, выступает Dash (Debian-клон оболочки Альмквиста, ash), лёгкая и компактная, но имеющая слабые возможности для интерактивной работы. Для последней, как и в подавляющем большинстве дистрибутивов Linux, используется Bash, которая является пользовательской оболочкой (login shell) по умолчанию. Что же до Zsh, она отсутствует в стандартной инсталляции Mint, но доступна в официальном репозитории, из которого легко может быть установлена.

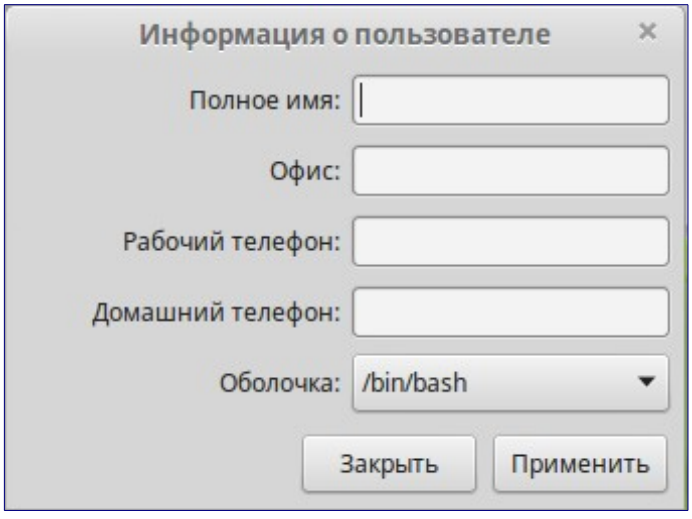
Начинающему применителю Mint проще всего установить Zsh с помощью описанного ранее менеджера пакетов. Для чего сначала надо отыскать соответствующий пакет:



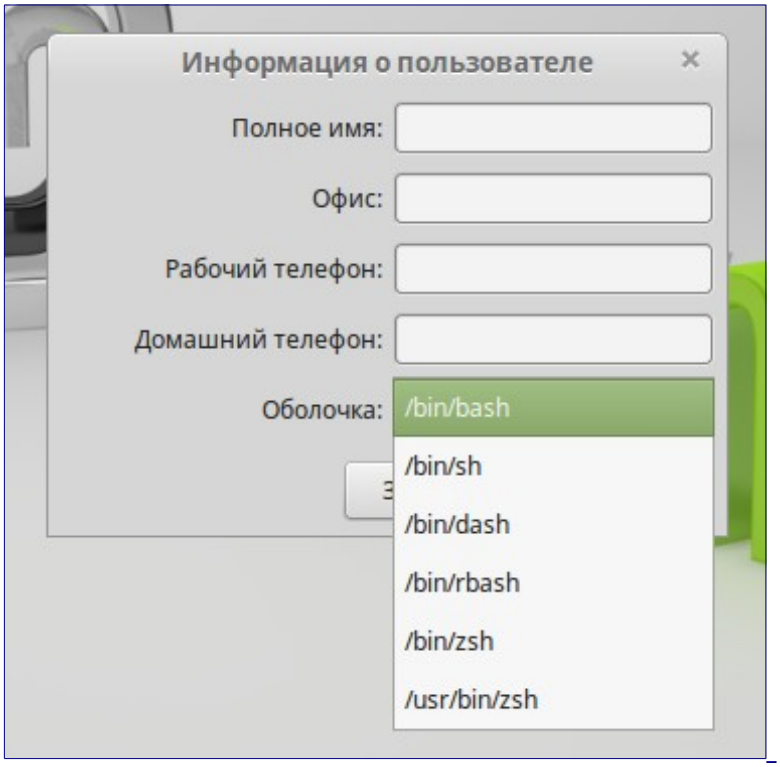
После чего поглядеть на его описание и установить:



Однако просто иметь Zsh мало — его надо сделать регистрационной оболочкой (login shell) в своём аккаунте. Как ни странно, в обоих графических модулях Системных настроек Cinnamon такой возможности нет. Однако можно прибегнуть к графической утилите usermode, предварительно установив её через Менеджер приложений и запустив из главного меню, где она скрывается в секции Параметры под именем О себе и после запуска выглядит таким образом:



После установки Zsh её можно будет выбрать из выпадающего списка в поле Оболочка:



Кажется, это единственное, что может сделать полезного данная утилита. Поэтому возникает вопрос — а стоит ли устанавливать её ради разовой операции? Может быть, лучше прибегнуть к самому простому способу смены login shell — прямой команде? Тот этой:

```
$ chsh -s /bin/zsh
```

Вопрос, как вы понимаете, риторический…

Каким бы образом ни была назначена Zsh ~~любимой женой~~ пользовательской командной оболочкой, следующая авторизация данного пользователя в «голой» консоли однозначно её запустит. В эмуляторах же терминала, возможно, потребуется внести некоторые изменения в их настройках, например, предписать запуск /bin/zsh явным образом, или отметить опцию запуска оболочки как login shell.

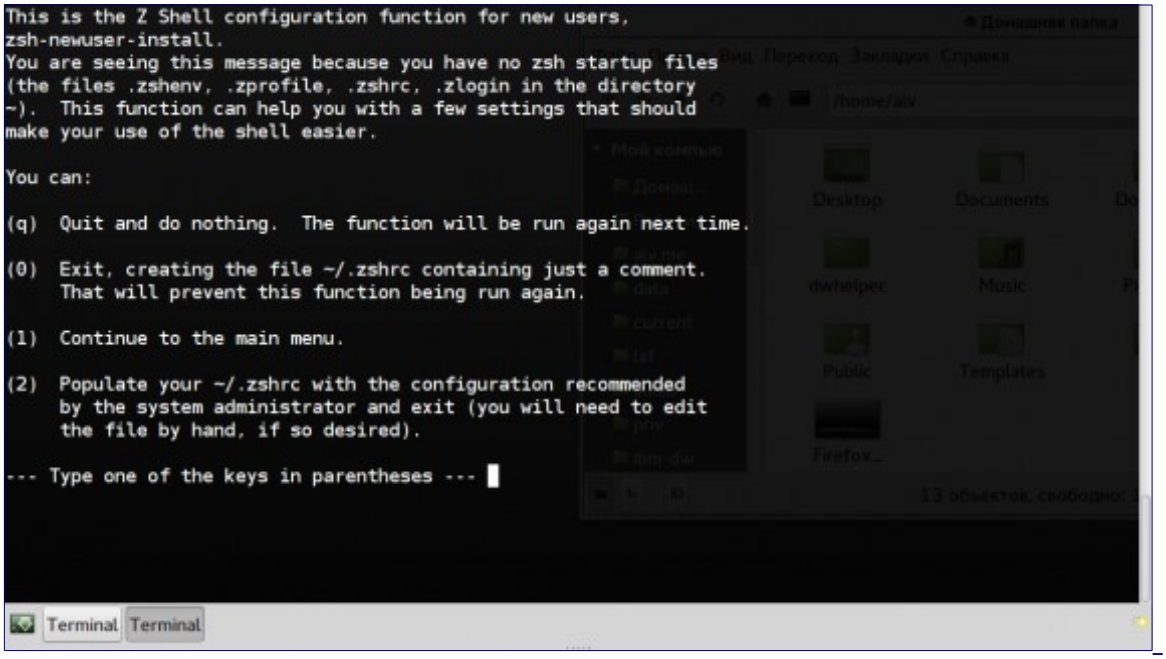
В любом случае первый запуск сеанса пользователя с новой оболочкой предложит такие варианты выбора:

- q — выход из программы автоконфигурирования без последствий; при следующем входе в оболочку вызов её будет повторён;

- 0 — выход из автоконфигурирования с созданием пустого конфига ~/.zshrc, предотвращающем в дальнейшем повторения автоконфигурирования;

- 1 — вызов главного меню;

- 2 — создание конфига ~/.zshrc по образу и подобию эталонного, /etc/zsh/newuser.zshrc.recommended, который в дальнейшем может редактироваться вручную.



С вариантом q всё ясно, это просто откладывание вопроса на потом, вариант 1, с автоконфигурированием, был некогда описан достаточно подробно, и с тех пор процесс этот ничуть не изменился, вариант же 2 зависит от настроек общего конфига оболочки, принятых майнтайнерами данного дистрибутива. Так что я хотел бы сконцентрировать внимание на «нулевом» варианте. И последовательно рассмотреть все настройки, которые потребуется выполнить применителю для создания комфортной среды CLI. Не абстрактно, разумеется, а применительно к целям и задачам себя, любимого. Так что читатель должен воспринимать всё сказанное в этих очерках далее, не как догму, а как руководство к действиям, то есть экспериментам, и к размышлениям о своих потребностях.

Однако прежде отмечу, что применителю не обязательно сразу определять Zsh как login shell. Он может вызвать её из командной строки Bash:

```
$ /bin/zsh
```

Запуск Zsh ознаменуется сменой вида приглашения командной строки с Bash'евской, которая в Mint'е по умолчанию выглядит так:

```
zshuser@alv-cinn ~ $
```

на умолчальную Zsh'еву:

```
alv-cinn%
```

Вот с настройки вида приглашения командной строки я и начну. Добавив только, что после каждого изменения в конфиге ~/.zshrc для вступления его в силу вовсе не обязательно завершать сеанс и авторизоваться заново — достаточно такой команды:

```
alv-cinn% source .zshrc
```

Кстати, конфигурационных файлов для Zsh предусмотрено много, и порядок их считывания тоже определён жёстко. Однако далее речь будет идти, за одним специально оговоренным исключением, только о редактировании ~/.zshrc. Почему? Да потому, что остальные конфиги или

были придуманы для совместимости с оболочкой совсем другого семейства, Tcsh, или не оказывают влияния на пользовательский сеанс.

Документация

Но прежде чем перейти к настройкам Zsh, надо сказать несколько слов о его документации, столь расхваленной мной во вводных словах. И первое, что тут удивляет — отсутствие для его текущих версий (5.0.X) стандартных man-страниц. Раньше они были, причём во множестве: собственные страницы были посвящены отдельным частям этой оболочки (опциям, параметрам, функциям etc.), а сама по себе страница man (1) zsh играла роль оглавления.

Но со временем суммарный объём man-документации достиг такого размера, что ей стало практически невозможно пользоваться в том режиме, в котором мы все привыкли общаться с любимой тётей Маней. И потому разработчики Zsh от man-страниц в составе самого пакета отказались.

Но зато, во-первых, пакет zsh и жёстко с ним связанный zsh-common сопровождается пакетом zsh-doc, который в большинстве дистрибутивов (в том числе и в Mint) следует устанавливать отдельно. Он содержит материалы в форматах info и html общим объёмом 6 МБ, а также включает PDF-руководство на 400 страниц.

Во-вторых, Zsh сопровождается также пакетом zsh-lovers — он также устанавливается отдельно, и его компоненты после этого будут располагаться в каталоге /usr/share/doc/zsh-lovers. Он озаглавлен так: Советы, рекомендации и примеры для Z Shell. И содержит большинство тех самых man-страниц, которые были изъяты из основного пакета — в чисто текстовом формате или в виде gz-компрессированных файлов. А также — заявленные советы, рекомендации и примеры, созданные многочисленными применителями этой оболочки. Все они поимённо перечислены в файле /usr/share/doc/zsh-lovers/README. Своего рода квинтэссенцией пакета является страница man (1) zsh-lovers, в конспективной форме описывающая основные возможности этой оболочки, иллюстрируя их примерами. Собственно, её обзор (OVERVIEW) и начинается словами:

Каждый раз, когда мы заглядываем в руководство по Zsh, мы удивляемся, почему там нет примеров или просто случаев из жизни в командной оболочке. Возможностей у Zsh, много, а руководства, иллюстрирующего их примерами, нет. Поэтому мы написали своё руководство.
...
Это просто развлекухи ради.

И, надо сказать, развлекуха получилась не без пользы для нас, применителей. Кстати, читать эту развлекуху можно также в форматах HTML и PDF.

В-третьих, неисчислимое по объёму количество информации о Zsh'е имеется в Интернете — и всё это богачество доступно по ссылкам с официальный сайт, главнейшей из которых является ссылка на zsh.sourceforge.net. Здесь можно найти руководства по этому шеллу на любой вкус — от «юзерофильного» до исчерпывающего, а также ссылки на книги, wiki, статьи и прочие материалы. Разбираться в этом океане я предоставляю заинтересованным (или заинтересовавшимся) читателям.

В-четвёртых, существует сайт, именуемый Oh My ZSH!. Это коллекция

плагинов, скриптов, конфигов и тем приглашения командной строки. Она инсталлируется на локальную машину и в дальнейшем автоматически синхронизируется с родительским сайтом, который пользуется всенародной популярностью и широкой известностью в узких кругах энтузиастов Zsh.

Наконец, в-пятых, официальными, полуофициальными и общенародными ресурсами информация о Zsh не исчерпывается — существует много «неучтённых» на zsh.sourceforge.net сайтов и блогов, ведомых любителями этого шелла. И на них часто можно найти освещёние неожиданных и интересных нюансов его конфигурирования. В последние годы в их числе появились и русскоязычные ресурсы. Из последних хотелось бы отметить подборку статей на сайте Михаила Мищенкова aka muhas).

Настройка приглашения

Как известно со времён со времён Константин Сергеича Станиславского, театр начинается с вешалки, а дистрибутив — с инсталлятора. Командная же оболочка начинается с приглашения командной строки. Каковая, во-первых, отражает готовность системы к выполнению действий применителя, а во-вторых, несёт (или должна бы нести) некую существенную для него информацию.

Правда, умолчальное приглашение Zsh информативностью не блещёт, сообщая только имя хоста (в примере на предыдущей странице — alv-cinn), и то, что сеанс шелла запущен обычным пользователем — в отличие от Bash'a, здесь это по умолчанию выражается символом %. Однако добавить информации нам никто не мешает. А помогает — файл zshexports.gz из пакета zsh-lovers, упомянутого в позапрошлом очерке. Его можно просмотреть командой

```
$ zcat path3/zshexports.gz
```

отыскать в нём секцию, начинающуюся словами

```
# PS{1,2,3}, RPOMPT, ..

# The "prompt" of the shell

...
```

внимательно изучить её, а также фрагмент конкретных примеров:

```
# Some examples:

#  PS1="PS1='%B%n%b@%m:%4c>'"

...
```

осмыслить прочитанное и опробованное на примерах. После чего решить, какую же информацию вы хотите видеть в приглашении командной строки.

Я, например, не хочу видеть там имени хоста, поскольку не дожил ещё до ситуации из известного аккордеонистого бояна: «Кто я, кто я? Губайдулин я!» Да и вообще, времена, когда каждая машина в сети имела собственное неповторимое имя, канули в лету, и нынче так называемое «хвостнаме» берётся от булды.

А вот имя пользователя в явном виде будет не лишним — у меня на основной машине их обычно не менее трёх: рабочий, экспериментальный и умолчально-восстановительный. Также неплохо иметь представление о своём положении в файловой иерархии, причём в полном виде — одноимённые подкаталоги часто находятся в разных её ветвях. Приглашение получается перегруженным? Отнюдь, потому что в Zsh таковых предусмотрено два — просто PROMPT и RPROMT, и перечисленные элементы можно разнести таким образом:

```
/home/data/current $=>                                  [alv]
```

Или наоборот, таким:

```
[zshuser]$=>                                  [/home/data/current]
```

Добиться этого можно, как вы понимаете, редактированием файла ~/.zshrc. До сих пор он у нас содержал единственную строку комментария:

```
# Created by newuser for 5.0.2
```

Теперь же добавляем в него сторки для получения приглашения первого вида:

```
## Prompt

PROMPT='%~ $=> '

RPROMPT=' [%n] '
```

Или второго:

```
## Prompt

PROMPT='[%n]$=> '

RPROMPT=' [%~] '
```

Раньше мне больше нравился первый вариант, но ныне я перешёл на второй.

Кроме обычного, то есть «левого» приглашения и приглашения «правого», в Zsh поддерживаются также приглашение «вторичное», выводимое в многострочных командах, и «третичное» — предложение вариантов замены при включённой коррекции ошибок, PROMPT2 и PROMPT3, соответственно. Вторичное приглашение у меня имеет вид

```
PROMPT2='%i%U> '
```

В результате в нём выводится номер «вторичной» строки в данном сеансе шелла, указывается стрелкой на то, что ввод следует в ней продолжить, а сам ввод даётся подчёркнутым шрифтоначертанием. Вживе это выглядит так:

```
[zshuser]$=> echo $USER \                               [~]

33> echo $SHELL \
```

```
34> echo $PATH

zshuser echo /bin/zsh echo /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Что же до коррекции ошибок, у меня она отключена (к этому вопросу мы ещё вернёмся).

А вообще, как можно увидеть в файле zshexports.gz, в любом из видов приглашения командной строки могут фигурировать:

- полное или сокращенное имя хост-машины;
- путь к текущему каталогу в различных формах;
- номер текущей команды в буфере истории или строки в данном сеансе работы;
- имя пользователя;
- название командной оболочки;
- номер виртуальной консоли или текущего терминала;
- дата и время в разных форматах;
- индикация работы от лица суперпользователя;
- любые символы типа стрелок, крышечек, скобочек;
- текстовые сообщения (например, поздравление с началом трудового процесса);
- и многое другое.

Кроме того, приглашение могут быть оформлены визуально различно: выделением жирным шрифтом (boldface mode), инвертированием текста/фона (standout mode), полчёркиванием (underline mode), а также цветами. «Раскрашенный» шелл мне нравится не больше, чем «раскрашенный» Штирлиц, инвертирование также вызывает раздражение, а вот выделение полужирным шрифтоначертанием и подчёркиванием я использую. В результате секция настройки вида приглашения в моём ~/.zshrc выглядит так:

```
# Left prompt

PROMPT='%B[%n]$=>%b '

PROMPT2='%i%U> '

#

# Right prompt

RPROMPT=' %B[%~]%b '
```

Как уже говорилось, я не призываю к подражательству, а лишь предлагаю поэкспериментировать, чтобы добиться максимальной информативности приглашения и его внешней выразительности.

Темы приглашений

Только что речь шла о том, как оформить приглашение командной строки Zsh своими руками, в соответствие с собственными вкусами и предпочтениями. Однако можно пойти другим путём, и воспользоваться уже готовыми темами приглашений. Они входят в пакет zsh-common, который всегда, насколько я знаю, устанавливается как зависимость пакета zsh. После установки местоположение готовых тем — каталог /usr/share/zsh/functions/Prompts.

Сами по себе темы приглашения — файлы вида prompt_themename_setup, представляющие собой функции Zsh, описывающие как вид приглашения, так и, часто, некоторый его декор, типа расцветки, которая может быть нескольких видов. Однако разбираться в устройстве этих функций не обязательно — с ними можно ознакомиться визуально.

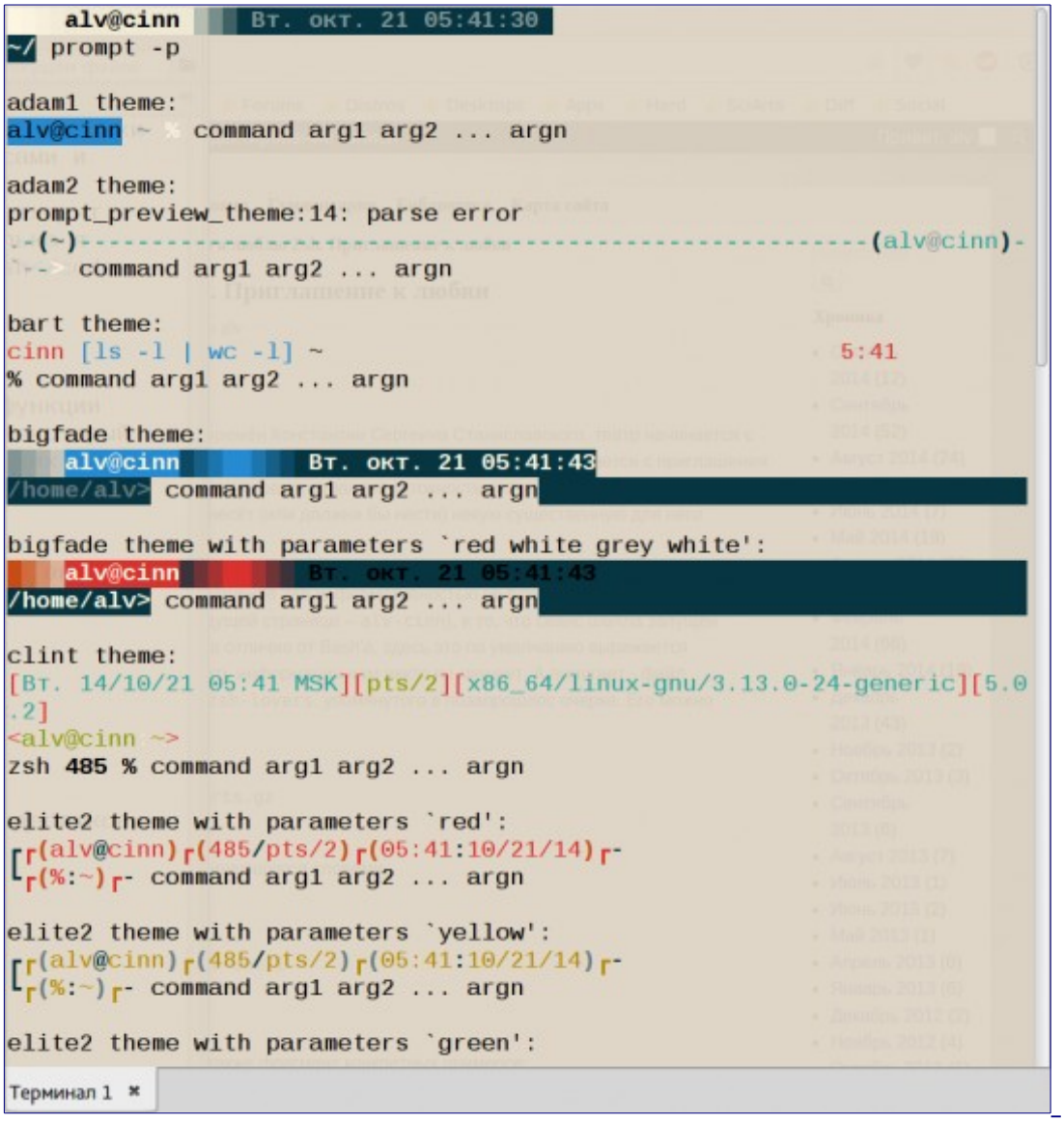
Знакомство это начинается с запуска функций управления видом приглашений:

```
$ autoload -U promptinit && promptinit
```

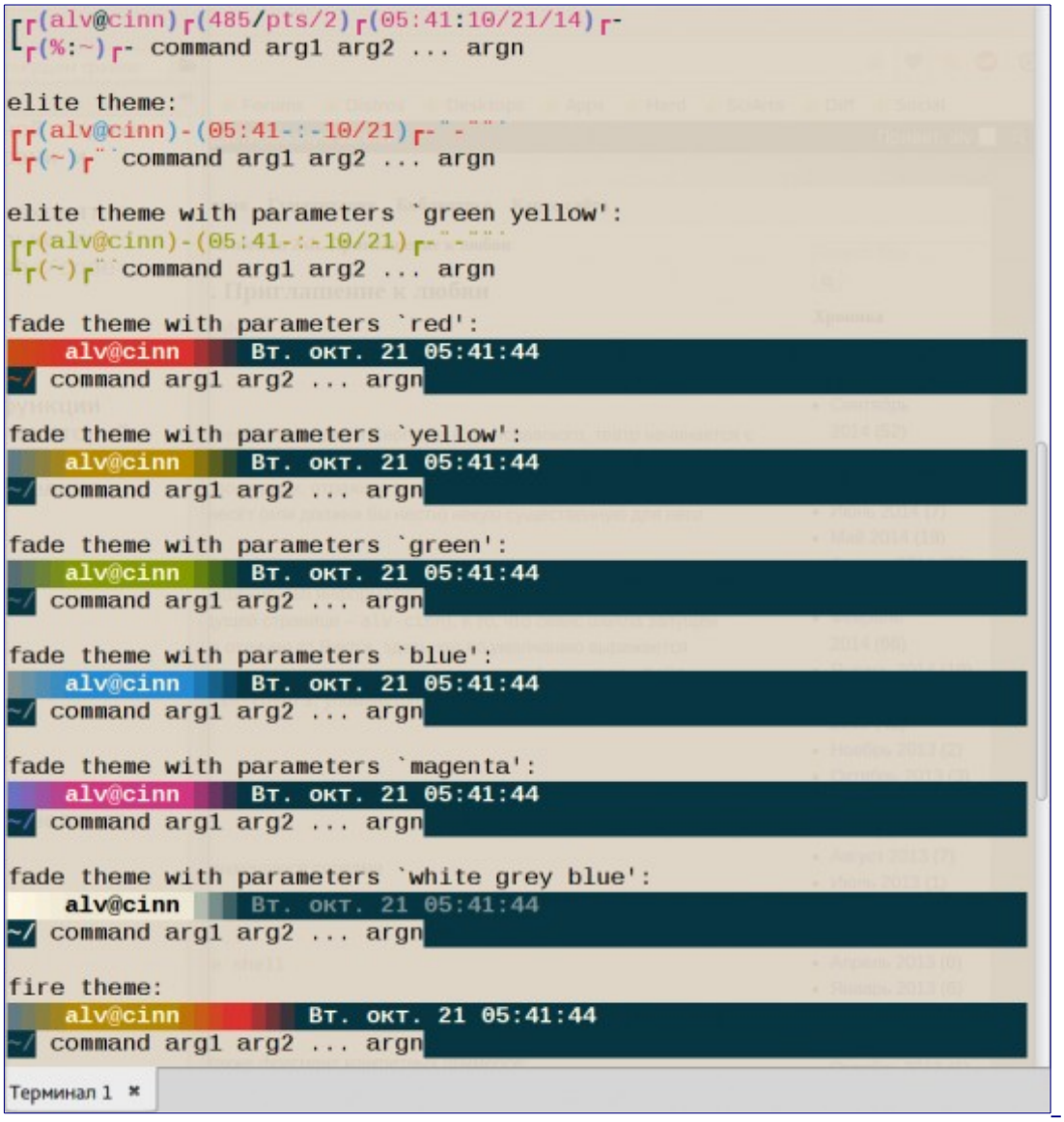
После чего можно давать команду на «смотрины невест»:

```
$ prompt -p
```

которая выведет их все (в моей системе — около двух десятков, плюс цветовые вариации) примерно в таком виде:



Среди «невест» можно видеть весьма пёстро наряженных:



Но и одетых весьма скромно также есть:



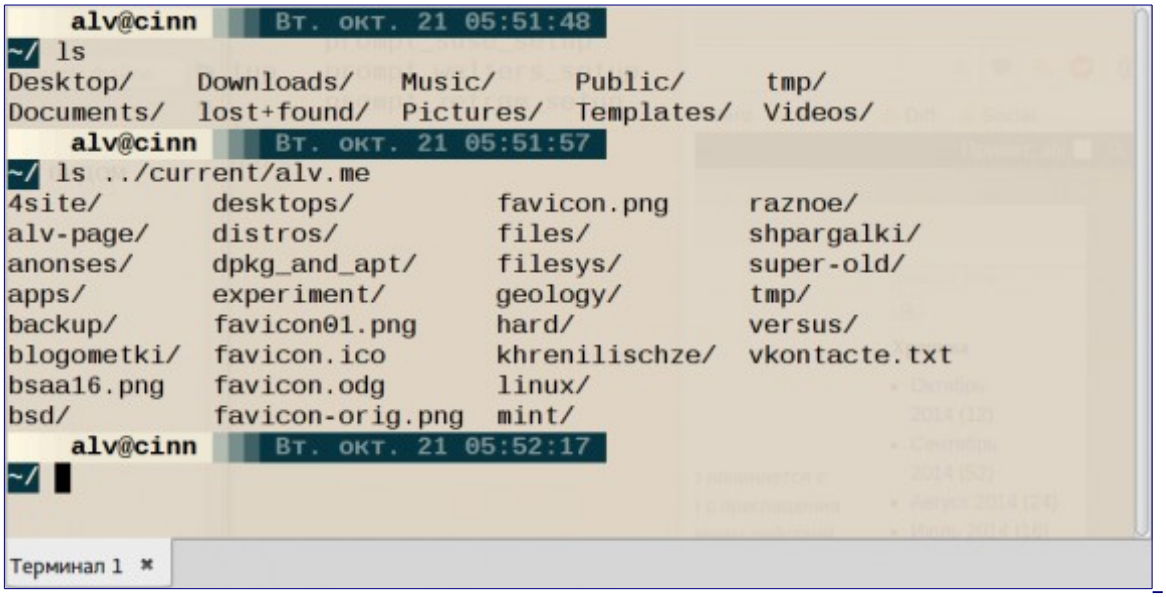
Выбрав подходящую ~~невесту~~ тему, её можно тут же установить командой

```
$ prompt имя_темы
```

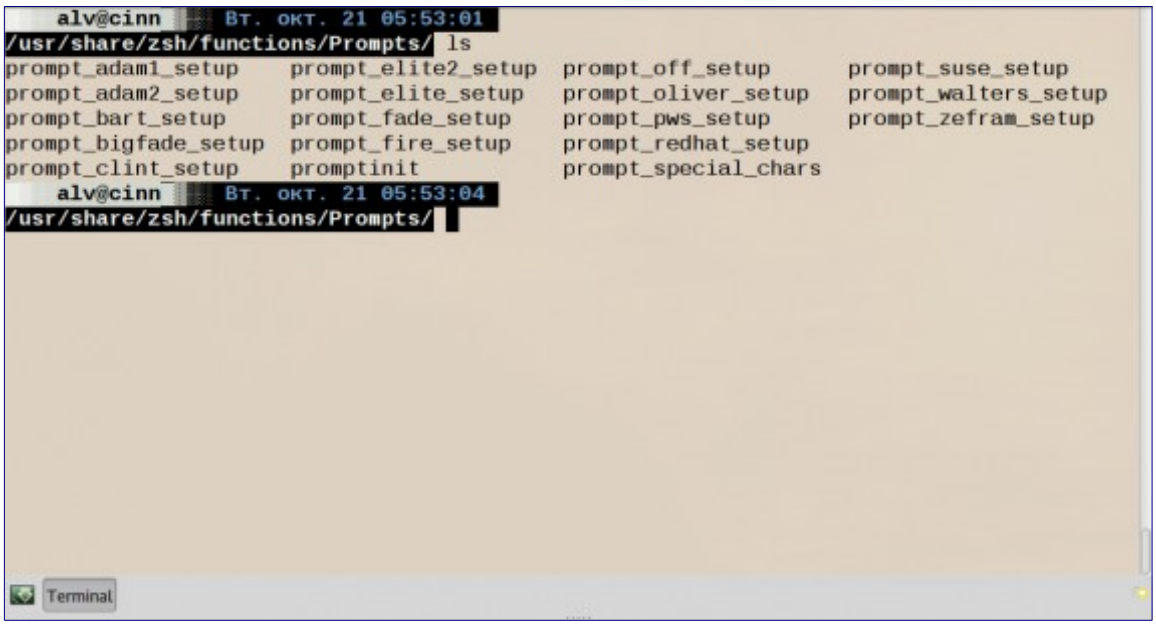
при желании — с указанием цветовых параметров, например:

```
$ prompt fade white grey blue
```

Что в «живом» терминальном окне (терминал Sakura) будет выглядеть так:



А в выпадающем терминале Guake — несколько иначе:



Кстати, а в «голой» консоли вид этой же темы будет существенно скромнее — разбираться с программами для изготовления скриншотов консоли мне было лень, так что прошу поверить на слово.

Установленная таким образом тема будет функционировать только в данном терминальном окне в течении текущего сеанса. Чтобы увековечить её, необходимо вписать в файл ~/.zshrc такие строки:

```
autoload -Uz promptinit

promptinit

prompt clint
```

В примере приведена тема, пожалуй, наиболее информативного приглашения, которое «вживе» выглядит так:



Большое количество тем можно при желании отыскать на сайте Oh My ZSH!, но эти я уже заниматься не стал.

Приёмы навигации

Сознательные граждане, активно применяющие CLI, используют множество команд, как встроенных в их любимый шелл, так и внешних. Но, думаю, что самыми употребимыми в повседневной жизни являются такие:

- pwd для определения своего текущего положения на файловом древе — да-да, иногда, после многократных переходов между подкаталогами, забываешь, не только кто я, но и где я (уж не в Тимирязском ли?);
- ls — для просмотра содержимого текущего каталога;
- cd — для перехода в определённый каталог.

Однако здесь Zsh вносит свои коррективы, здорово облегчающие жизнь его применителя. Только что было показано, как фактическим можно избавиться от команды pwd, выведя путь к текущему каталогу в качестве RPROMPT. Без команды ls, конечно, не обойтись и Zsh. А вот команда cd в Zsh просто... не нужна.

Да, дорогие мои болельщики, в среде Zsh без этой команды не просто можно обойтись, а жить куда комфортней, нежели с ней. Ведь давайте вспомним, что такое переход в каталог имя_рек? Для типа файлов, именуемого каталогом (directory) это то же самое, что исполнение для обычного (ordinary) файла, будь он откомпилированным бинарником или интерпретируемым сценарием.

И потому более чем логично то, что как для запуска скрипта оболочки не требуется никакой внешней команды (хотя и не возбраняется что-нибудь типа . или /bin/sh), так и для перехода в каталог, к которому данный юзер имеет доступ (то есть попадает в число тех, для кого у этого каталога установлен бит исполнения), ему достаточно указать полный путь к нему, без всяких команд. Например, введя к командной строке что-нибудь вроде

```
$ /usr/share/fonts/truetype/
```

можно сразу оказаться в каталоге с TTF-шрифтами.

«Бескомандный» переход в каталоги распространяется и на «символические» обозначения последних. Так, команда

```
$ ~
```

переместит пользователя в его домашний каталог. Как, кстати, и команда

```
$ $HOME
```

Хотя практического смысла последний вариант не имеет. Зато директива

```
$ ..
```

волшебным образом ознаменует переход в каталог, родительский относительно текущего.

Правда, всё это происходит не само собой: для практического воплощения этого волшебства в общесистемном конфиге /etc/zsh/zshrc или пользовательском ~/.zshrc должна присутствовать строка

```
setopt autocd
```

В пару к ней можно добавить ещё и такую строку:

```
cdpath=(~/ /home/current/ /home/data/)
```

где в скобках перечислены каталоги, к подкаталогам которых чаще всего требуется быстрый доступ. И теперь, где бы в пределах файлового древа ни находился пользователь, ввод им директивы

```
$ Documents
```

нечувствительно сделает текущим каталогом /home/username/Documents.

То есть опция autocd и массив переменных cdpath отнюдь не исключают, а прекрасно дополняют друг друга.

Автодополнение

Волшебное свойство клавиши Tab, вызывающей автодополнение — одно из первых, с чем знакомится применитель CLI. Хотя при этом часто забывается, что когда-то, в перворождённом шелле Борна, никакого автодополнения не было. Оно появилось в Csh — и сначала только для путей, но не для команд. Тем не менее, ныне представить себе интерактинуую работу в командной строке без автодополнения невозможно (да и не нужно).

Однако в Zsh клавиша Tab волшебна дважды: она не только дополняет пути и команды после их частичного ввода, но и способна развернуть аббревиатуры для тех и других. Например, нажатие клавиши табулции после набора последовательности

```
$ /u/s/f/tr
```

развернёт её в полный путь к каталогу со шрифтами TrueType

```
$ /usr/share/fonts/truetype
```

а после нажатия клавиши Enter сделает этот каталог текущим, как мы только что видели.

Правда, само по себе развёртывание аббревиатур работать не будет — его надо активизировать такими строками в файле ~/.zshrc:

```
 autoload -Uz compinit

compinit
```

Можно пойти дальше, и не просто разворачивать безальтернативные аббревиатуры, типа приведённый выше, но и выбирать стрелками, как в меню, подкаталги или файлы среди предлагаемый альтернатив. Например, если набрать ту же самую последовательность символов:

```
$ /u/s/f/tr
```

а затем дважды нажать клавишу табуляции, то она не только развернётся в полный путь

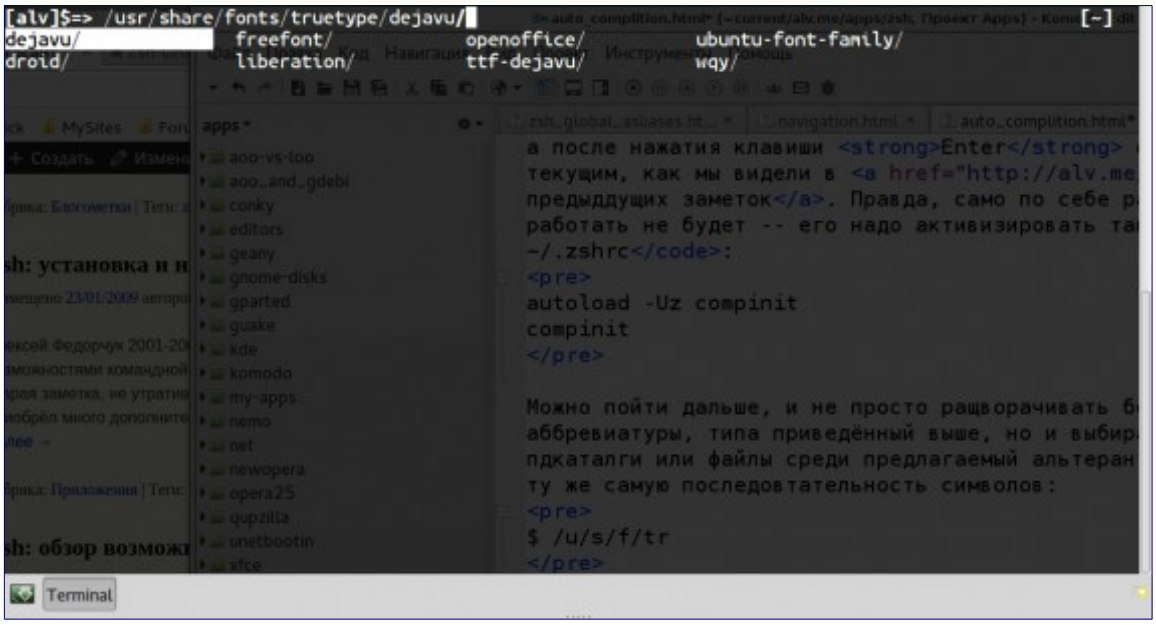
```
$ /usr/share/fonts/truetype/
```

но и выведет список подкаталогов указанного каталога:

```
dejavu/ freefont/ openoffice/ ubuntu-font-family/ droid/ liberation/

ttf-dejavu/ wqy/
```

И выбор нужного среди них можно выполнять либо стрелками управления курсором, либо обычными кейбиндингами типа Control+f, Control+b и им подобными:



Правда, и такая реакция Zsh на Tab возникает не из воздуха, а из присутстствия в файле ~/.zshrc таких строк:

```
setopt menucomplete

zstyle ':completion:*' menu select=1 _complete _ignored _approximate
```

По умолчанию их там нет, а вот стоит ли их вносить — применитель должен решить для себя сам — перебор вариантов традиционным способом, то есть последовательным нажатием клавиши табулции, может показаться более удобным.

История команд

Возможность просмотра истории введённых ранее команд клавишами Up/Down кажется таким же неотъемлемым атрибутом CLI, как и автодополнение командной строки. И, как и последнее, напрочь отсутствовало в перворождённом шелле Борна, однако ныне имеется во всеш развитых шеллах. Причём доступ к истории команд в них не ограничивается командой history и упомянутыми стрелками. В частности, в Bash широко практикуется инкрементный поиск по клавишной последовательности Control+R и вводу последовательности символов одной из предыдущих команд или её аргументов.

В tcsh же испокон веков существовала (и, что характерно, обычно была активирована по умолчанию) другая возможность — так называемый history-substring-search, то есть инкрементный перебор истории команд по вводимым символам. Что это такое — проще пояснить на примере: вы вводите в командной строке один символ (для примера — s) и нажимаете клавишу Up. И тут в перебор включаются только те команды из истории, которые с буковки s начинаются. Вводя дополнительные символы, можно сузить круг поиска: например, последовательность sudo позволяет просмотреть, что было наколбасино от лица суперпользователя вообще.

Поскольку Zsh изначально задумывалась как синтез всех передовых достижений шелло-строительной мысли, аналогичная возможность имеется и здесь. Правда, как и многие другие продвинутые фичи этой оболочки, она требует активации. То есть — внесения в файл ~/.zshrc таких строк:

```
bindkey "^[[A" up-line-or-search

bindkey "^[[B" down-line-or-search
```

Как выяснилось, надо подчеркнуть: перебор history-substring-search и инкрементный поиск по Control+R отнюдь не исключают друг друга, а дополняют: первым способом проще искать ранее введённые директивы по имени команды, вторым — по её аргументам, например, по имени файла.

Справедливости ради надо сказать, что history-substring-search нынче реализован и в Bash, хотя, как и в Zsh, требует активации.

Опытный применитель Zsh, не имевший ранее дела с Ubuntu и её производными (в том числе и с Mint'ом), будет весьма удивлён тем обстоятельством, что эта фича (по моему мнению, одна из самых полезных среди всех достоинств нашей героини), с кондачка работать не будет. Даже при условии правильно настроенного конфига — при внесённых в него строках, указанных выше. Точнее, не будет делать это в окне любого иксового эмулятора терминала, хотя не откажется от выполнения history-substring-search в «голой» консоли. Причём интересно, что это же касается и Bash, хотя

в Tcsh данная фича будет работать «искаропки».

Следствие, проведённое в Джуйке и благодаря участию джуковца @altwazar'a, показало, что это давний известный баг, восходящий к Debian'у, знаменитому своей стабильностью во всех отношениях (в том числе и в отношении багов, вероятно). И бороться с этим можно различными методами. Мне самым простым показался такой: создание в домашнем каталоге файла ~/.zshenv с единственной строкой:

```
DEBIAN_PREVENT_KEYBOARD_CHANGES=yes
```

Разумеется, на поведение Bash это никак не скажется: в нём history-substring-search включается не через его профильный файл, а через inputrc — конфиг для readline. Как именно — оставляю на рассмотрение преданных поклонников этой оболочки.

Разумеется, возможности настройки доступа к истории команд всем сказанным выше не исчерпываются: имеет место быть и исключение из неё дубликатов, и пустых строк, и прочего баласта, а также подключения некоторых полезных фич, вроде ограничения общей истории и истории текущего сеанса. А также — дополнения файла истории. Однако ничего особенного, специфичного именно для Zsh, тут уже нет. Так что к рассмотрению этих вопросов я вернусь под занавес — когда буду говорить о ~/.zshrc для себя, любимого...

Рекурсивный поиск

Все применители CLI знают и любят утилиту find — и любят заслуженно, ибо это апофеоз командного интерфейса: с её помощью можно отыскать в файловой системе всё, что угодно — и почти всё, что нужно, с найденным сделать, конечно, с помощью некоторых дополнительных средств, вроде xargs и конвейеров. Однако для многих рутинных задач мощь этой команды кажется излишней, напоминая знаменитое упражнение по отстрелу мелких пернатых их зенитно-ракетных комплексов. И вот тут Zsh опять позволяет решать такие задачи малой кровью — то есть с минимальным ударением по клавишам. Ибо поддерживает такую штуку, как рекурсивные поиск.

Что это такое — как обычно, проще показать, чем рассказать. Предположим, перед применителем стоит задача отыскать все картинки в каталоге некоего проекта, включая все вложенные в него подкаталоги. Средствами Zsh сделать это очень просто — достаточно дать команду

```
$ ls path3/**/*.png
```

где path3, как нетрудно догадаться, «корневой» каталог поиска, *.png — маска искомых файлов, а «двузвёздие» — так сказать, директива рекурсивного поиска.

Правда, вопреки утверждениям некоторых уж очень правоверных Zsh'истов, эта возможность не делает команду find избыточной, ибо, как все знают, она умеет и многое другое. Но зато такая простая директива позволяет не беспокоить Её Величество по пустякам...

А заодно — конструкции вида **/* можно использовать как аргументы команд управления файлами, таких, как cp, mv, rm. В частности, с помощью команды вида

```
$ rm -f path3/**/*~
```

можно легко гуртом избавиться от всех временных копий, которые по умолчанию так любят сохранять некоторые текстовые редакторы и ворд-процессоры, если им не запретить это самым категорическим образом.

Разумеется, можно фильтровать базар. Давеча в приступе чёрной меланхолии переслушивал я всё, что сочинил и спел Фред Солянов — увы, большинство моих потенциальных читателей о его существовании не подозревают: в отличе от многих всенародно известных так называемых «бардов», он не был популярен при жизни. А когда его верхние люди позвали — люди нижние про него забыли напрочь. И зря — но это из совсем другой оперы. А в нынешней арии мне было интересно, сколько же Фред сочинил песен за ту четверть чека, что ему отпустила на то судьба. И я дал очень простую команду:

```
$ ls path3/fred/**/*.mp3 | wc -l
```

И она мне сказала, что сочинил Фред 168 песен. Откидываем дубликаты, неизбежные в любой коллекции — но здесь их очень мало, на штуки счёт.

Откинем откровенно слабые песенки — ведь даже гений не каждое утро начинает с сочинения чего-то шедеврального. Откинем песенки вторичные — Фред никогда не претендовал на основоположничество, и, в отличе от некоторых более иных авторов, на которых я не хотел бы указывать пальцем, не считал для себя западло называть своего реального учителя в ентом деле, Булат Шавловича...

Для себя откину те песенки, которые лично меня не очень зацепили — их, по сравнению с прочими фильтрами, больше всего, почти полсотни.

Остаётся - около ста песен. За двадцать пять лет. Мало по сравнению с раннеперестроечными сборниками типа «Шестьсот лучших песен имя река»? Да, не много. Но ведь (и это мнение не только моё, а тысяч людей с такими же биографиями) эти песни стали, как нынче принято говорить, культовыми.

Ну, дальше на эту тему распространяться не буду, а вернусь к генеральной линии сюжета. А именно — что маски типа **/*можно использовать в аргументах команды grep и для поиска фрагментов текстов. Так, команда

```
$ grep KDE **/*html
```

выведет все строки с упоминанием KDE в html-файлах каталога текущего и вложенных. А в форме

```
$ grep -i kde **/kde*.html
```

она произведёт аналогичный поиск только в файлах вида kde01.html, kde02.html и так далее. Причём без учёта регистра — но к мадемуазель Zsh, интересы которой я представляю в данный момент, это не имеет никакого отношения.

Перенаправление расширенное и множественное

Что такое перенаправление ввода/вывода — знают все применители CLI. Однако в Zsh возможности его очень широки, почему оно и называется здесь

расширенным перенаправлением. Этот механизм позволяет в ряде случаев обходиться без некоторых команд вообще. Например, обычно для просмотра текстового файла применяют или команду cat, или команды-пейджеры типа more, less, most. Выбор между конкатенатором и одним из пейджеров определяется ситуацией, выбор внутри «тройки по борьбе с ~~басмачами~~ файлами» зависит от привычек или предпочтений. Однако Zsh может избавить применителя от мук буриданова осла, подменяя любую из этих команд оператором перенаправления в виде команды

```
$ < filename
```

Результатом чего будет постраничный вывод содержимого файла, подобный таковому любого пейджера.

С помощью того же оператора можно просмотреть одновременно содержимое двух файлов — то есть, конечно, не одновременно, а последовательно, но в едином потоке. То есть команда

```
$ < {zshenv,zshrc}
```

покажет оба файла как одно целое. Причём в данном случае можно поступить ещё проще, ибо маски имён файлов также не возбраняются:

```
$ < z*
```

Кстати, в терминах Zsh развёртывание масок имён файлов называется globbing — с ним мы уже сталкивались в рассказе о рекурсивном поиске.

Число «оперируемых» файлов ничем не ограничено, кроме здравого смысла и целесообразности. Так, есть резон проглядеть таким образом на скорую руку, как будут выглядеть 5-6 заметок по несколько строк каждая, если их объединить в одну статью. Но просматривать с помощью оператора перенаправления книжку, состоящую из пары десятков глав по много страниц каждая, уже явный перебор.

Однако бывают случаи, когда большое число «оперируемых» файлов очень даже уместно. Например, если требуется объединить ряд текстовых фрагментов в единый файл. И тогда, легким движением рук набрав в командной строке конструкцию

```
$ < chapter[01-10] > mybook
```

мы на выходе из разрозненных глав получаем готовую книгу.

Таким образом мы перешли уже к множественному перенаправлению. Применение которого просмотром файлов не исчерпывается — их содержимое может быть перенаправлено не только на стандартный вывод, но и на ввод какой-либо команды, подменяя командный конвейер. Например, конструкция вида

```
$ sort < file_{1,2}
```

совместно отсортирует строки обоих файлов, file_1 и file_2, точно так же, как это сделал бы конвейер команд

```
$ cat file_1 file_2 | sort
```

Кстати, перенаправление вполне может играть с конвейерами в одной команде. Например, конструкция вида

```
$ time commandname [options] [arguments] > filename | cat
```

занесёт время выполнения некоей команды в файл с одновременным выводом его на экран, заменяя команду tee. Это особенно полезно при всяких «тестированиях на быстродействие», когда надо и сохранить результат для дальнейшей обработки, и не терпится посмотреть на него сразу.

Множественное перенаправление удобно использовать для суммарного подсчёта числа символов в нескольких файлах таким образом:

```
$ wc -m <*txt
```

Что на выводе даст единственное число, например:

```
5382
```

Казалось бы, та же команда в «обычной» форме даже короче на один символ:

```
$ wc -m *txt
```

Однако вывод её будет развёрнут:

```
2820 my_file_1.txt
 606 my_file_2.txt
 401 my_file_3.txt
1555 my_file_4.txt
5382 итого
```

Что при работе во встроенных терминальных окнах текстовых редакторов вроде Geany или Kate , часто небольших по размеру может оказаться лишним. А ведь именно там приёмы, подобные описанным в этом разделе, оказываются весьма эффективными.

В общем, уже за одну только конструкцию < filename разработчики Zsh заслужили памятник, а все остальные возможности расширенного и множественного перенаправления выступают как бесплатное приложение к ней.

Просто псевдонимы и псевдонимы глобальные

Что такое псевдонимы, по простому aliases, — знают все, кто применяет любую командную оболочку: их поддержка существует со времён перворождённого шелла Борна. Это один из простых способов минимизировать ввод командных директив, начиная с простейшего рекурсивного копирования файлов:

```
alias cp='cp -R'
```

И заканчивая бессчётным количеством псевдонимов для команды ls.

Однако в Zsh есть ещё одна фича — глобальные псевдонимы, по сей день не имеющие аналогов, насколько я знаю, во всяких там ваших башах. И даже в почти соплеменных тсишах их нет.

Но начну по порядку. Опять же, кого не раздражала ситуация: в ответ на поиск файла find'ом или поиск фрагмента текста grep'ом выдаётся сто пятьсот экранов сообщений, что доступ к каталогу запрещён?

Разумеется, каждый применитель Bash'а знает, как с этим бороться — достаточно присобачить к конструкции поиска посредством той или другой утилиты маленький аппендикс в виде 2> /dev/null, отправляющий в небытие все сообщения об ошибках.

Сложнее применителям Tcsh — там подавления вывода нежелательных сообщений об ошибках возможно в виде такой конструкции:

```
% (command > out)>& err
```

где command — команда со всеми её опциями и аргументами, out — условное имя файла, в который перенаправляется «полезный» вывод команды, а & в данном контексте представляет весь остаток от оного, то есть сообщения об ошибках, которые помещаются в файл err. Имя последнего также условно, так что никто не запрещает подменить его сакраментальным /dev/null.

Конструкция далеко не столь проста, как в sh-совместимых оболочках типа Bash. Кроме того, для просмотра «полезного» вывода она потребует ещё одной команды — вызова какого-либо пейджера вроде less:

```
% (command > out)>& err ; less out
```

А вот применителям Zsh — проще всех. Им достаточно задать такой глобальный псевдоним:

```
$ alias -g N='2>/dev/null'
```

где -g указывает, что следующий символ (или символы) являют собой на простой псевдоним, а глобальный, N — его имя, а следующая после равенства последовательность в строгих кавычках — подменяемое им выражение. После чего можно практиковать такое:

```
$ find path3 -name [filename] N
```

И больше не заботиться о фильтрации зёрен от плевел.

Глобальные псевдонимы очень полезны в командных конструкциях перенаправления по конвейеру, например, для поэкранного вывода:

```
$ alias -g L='|less'
```

Пример для «пролистывания» вывода команды dmesg:

```
$ dmesg L
```

Для фильтрации по вхождению «слова» можно задать такой глобальный псевдоним:

```
alias -g G='|grep'
```

После чего использовать его в конструкциях, подобных такой:

```
$ dmesg G raid
```

что выведет нечто вроде

```
[    1.434246] md: raid0 personality registered for level 0
[    1.434376] md/raid0:md0: md_size is 390742016 sectors.
...
```

Мне весьма полезен глобальный псевдоним вида

```
alias -g W='|wc -m'
```

Поскольку часто требуется прибегать к такой конструкции

```
$ cat filename W
```

которая в данном случае выведет число символов в текстовом файле — для меня оно важнее числа байт (а при использовании 16-битной кодировки для преимущественно кириллического текста эти значения не совпадают).

К именам глобальных псевдонимов применяются те же требования, что и к именам псевдонимов обычных: они должны быть по возможности короткими, мнемонически прозрачными. И, разумеется, определения всех постоянно используемых глобальных псевдонимов следует занести в свой кондуитик — то есть в ~/.zshrc.

Разумеется, здесь не описаны все возможные случаи употребления глобальных псевдонимов — они лимитируются только потребностями применителя и его фантазией. И, конечно, наказом, который дал атаман Платов небезызвестному Левше:

Не пей мало, не пей много, а пей средственно.

То есть — не придумывайте глобальных псевдонимов больше, чем сможете запомнить.

Псевдонимы-суффиксы

Кроме обычных и глобальных псевдонимов, в Zsh существует ещё одна их разновидность — псевдонимы «суффиксные», более удачного определения на языке родных осин я не придумал, псевдонимы.

Подобно тому, как добаление к команде alias опции -g с помощью магии превращает обычный псевдоним в глобальный, так и опция -s делает обычный псевдоним «суффиксным». То есть привязывает суффикс имени файла (те, кто, подобно автору этих строк, затронуты порчей чёрным DOS'ом, до сих пор часто называют его «расширением») к некоей программе, которая может

сотворить над ним нужное действо. Например, если задать псевдоним такого вида

```
$ alias -s html=links
```

а затем набрать в CLI такое

```
$ path3/некто.html
```

то этот самый некто.html будет открыт в текстовом браузере Links.

Чем, разумеется, возможности «суффиксных» псевдонимов не исчерпываются — как всегда, предел им ставит только фантазия применителя применительно к его задачам. Ограничусь одним примером.

Какой же русский не любит Командера-полуночника? В том числе и потому, что он — один из сыновей прославленного командера Нортона, имя которого, в свою очередь, не более чем alias незабвенного лейтенанта Шмидта (история его чудесного спасения из лап царской охранки и последующей блестящей карьере сначала в ВМС Пендостана, а затем в интернациональном софтверном бузинесе реконструирована нашими замечательными историками из славного Екатеринбурга). Впрочем, со временем наш русский применитель, не смотря на весь свой патриотизм, начинает понимать, что слепая любовь к MC связывает ему руки в операциях с возлюбленной CLI, и хорошо бы с командиром расстаться, как это делают цивилизованные люди — без скандалов и истерик.

Но тут возникает проблема: MC — один из самых удобных способов просмотра того, из чего состоят файлы пакетов (будь то deb, rpm или что ещё из tar.*z-серии). Так вот, механизм «суффиксных» псевдонимов Zsh предлагает нам адекватную замену: если дать команду, например,

```
$ alias -s deb='dpkg -c'
```

а затем набрать в командной строке такое:

```
$ path3/opera-beta_25.0.1614.11_amd64.deb
```

то мы сразу увидим, что же припасли для нас разработчики этого многими любимого браузера в своём полуподпольном пре-релизе за нумером 25 (впрочем, за время сочинения этой книги он стал вполне официальным, приобретя номер версии 27):

```
drwxr-xr-x root/root         0 2014-09-13 03:54 ./
drwxr-xr-x root/root         0 2014-09-13 03:54 ./usr/
drwxr-xr-x root/root         0 2014-09-13 03:54 ./usr/bin/
drwxr-xr-x root/root         0 2014-09-13 03:54 ./usr/lib/
drwxr-xr-x root/root         0 2014-09-13 03:54 ./usr/lib/x86_64-linux-gnu/
...
```

Понятное дело, что аналогичные псевдонимы можно придумать и для

всяких rpm-и tgz-пакетов. И, разумеется, наиболее востребованные из них занести в кондуит... то есть в ~/.zshrc.

Конфигурирование

В качестве обобщения всего сказанного выше в заключение этого очерка я размещаю свой конфигурационный файл ~/.zshrc, прокомментированный, по мере сил, подробно. Этот конфиг существует с 2001 года, кочуя с машина на машину, из системы в систему, постоянно модернизируюсь в соответствие с изменениями моих потребностей и возможностей Zsh. И в текущем состоянии он обеспечивает все функции и особенности, о которых я говорил ранее, и некоторые другие, которые станут понятными после знакомства с Mint-утилитой пакетного менеджмента apt.

Данный конфиг может быть использован полностью или фрагментарно всеми заинтересованными лицами: блоки, заключённые в теги <pre></pre>, пригодны для прямого копирования, за одним исключением, о котором будет сказано в своё время. Однако я отнюдь не призываю к этому, напротив: настоятельно рекомендую, используя данный конфиг и аналогичные, которые можно найти в Сети, по мере сил и возможности создавать конфиг собственный. Ибо хороший (для конкретного применителя) ~/.zshrc — это не результат, а процесс, и причём процесс преувлекательный.

Как и большинство уважающих себя конфигов, мой начинается с секции, закрытой комментариями, в которой сообщается, что:

- это ~/.zshrc — то есть «домашний» конфигурационный файл для командной оболочки Zsh;
- используется только в интерактивных её экземплярах;
- содержит крманды для определения псевдонимов, функций, опций и прочих кейбиндингов;
- укладывается в последовательность считывания конфигов таким образом: zshenv, zprofile, zshrc, zlogin.

Всё это ~~потибрено~~ унаследовано от прототипа, распространяющегося разрабочиками Zsh. От себя я добавил лишь такую строку:

```
#

# Alv's edition for Mint

#
```

Это не значит, что данный конфиг нельзя использовать вне Mint: подавляющая часть его строк будет иметь силу в любых дистрибутивах Linux'а или в BSD-системах. Но отдельные его блоки (специально оговоренные) в них просто не будут иметь смысла.

Далее начинается собственно строки определения конфигурируемых параметров. Для удобства восприятия (по крайней мере, моего собственного) они разделены на блоки «целевого назначения». Последовательность блоков, как и строк внутри них, в большинстве случаев рояля не играет, отдельные

исключения также оговорены специально.

Поскольку всё имеет своё начало, начать свой конфиг мне показалось логичным с блока строк, имеющих отношение к истории команд. Перво-наперво — определение числа команд, сохраняемых в буфере во время данного сеанса, имени файла истории, и числа сохраняемых в нём команд:

```
HISTSIZE=2000

HISTFILE=~/.zhistfile

SAVEHIST=10000
```

Обычно для HISTSIZE и SAVEHIST рекомендуют принимать одинаковые значения (по умолчанию при автоматическом конфигурировании они равны 1000). Однако если действительно трудно представить ввод более чем тысячи команд в течении сеанса, то вот за весь цикл жизнедеятельности оболочки в системе превысить этот лимит достаточно просто.

Кроме того, надо учесть, что в обоих случаях сохраняются не просто команды, а целые директивы с опциями и аргументами, перенаправлениями и конвейерами, подчас достаточно сложными и редко используемыми. В Zsh имеются очень эффективные механизмы извлечения командных строк из сохранённой истории — не только по именам команд, но и по их опциям и аргументам. Обычно этим мало кто заморчивается, однако в некоторых, пусть и не частых, случаях такие командные конструкции могут потребоваться вторично. И тогда приятно сознавать, что они храняться в файле истории, откуда вытащить их всё равно проще, чем пытаться воспроизвести по памяти или отыскивать аналоги в сети.

Так что со временем я, увеличив на всякий пожарный случай HISTSIZE вдвое, отвёл под SAVEHIST 10000 строк. Кстати, когда предупреждают о том, что увеличение обоих значений может привести к торможению, следует учитывать, что в памяти постоянно находится только содержимое HISTSIZE, тогда как из SAVEHIST оно извлекается по мере необходимости. Не говоря уже о том, что при типичных для современных машин объёмах памяти об этом просто смешно говорить.

Имя файла истории я тоже изменяю на ~/.zhistfile. Во-первых потому, что иногда по старой памяти балуюсь Tsch, а в ней файл истории по умолчанию также именуется ~/.histfile (собственно, оттуда он в Zsh и был потибрен, в хорошем смысле этого слова). А во-вторых, просто для удобства восприятия — чтобы все имеющие отношение к Zsh файлы в домашнем каталоге были рядом.

Однако продолжим наши «исторические» опции. Следующие строки задают условия сохранения команд в файле истории:

```
setopt  INC_APPEND_HISTORY

setopt  HIST_IGNORE_ALL_DUPS

setopt  HIST_REDUCE_BLANKS

setopt  HIST_IGNORE_SPACE
```

Они определяют, соответственно:

1. инкрементное наращивание файла истории — без указания этой опции (или одной из однотипных) его прежние команды будут заменены командами текущего сеанса;

2. удаление предыдущих полных дубликатов нововведённых командных конструкций;

3. избавление от пустых строк, возникающих после ошибочного нажатия Enter в «голом» приглашении;

4. удаление лишних пробелов из командной конструкции.

Зачем нужны пункты 2–4 — ясно без комментариев. А вот о пункте 1-м надо сказать несколько слов. Ибо он не просто обеспечивает наращивание файла истории (для этого было бы достаточно опции, APPEND_HISTORY), но делает это в ходе сеанса, не дожидаясь его завершения. В результате команда, введённая в одном терминальном окне или вкладе терминала, будет доступна в истории команд другого терминала или вкладки (хотя и с некоторой задежкой).

Далее следуют две очень важные строки, определяющие одну из полезнейших возможностей Zsh — тот самый механизм history-substring-search, о котором говорилось ранее:

```
bindkey "^[[A" up-line-or-search

bindkey "^[[B" down-line-or-search
```

Следующие две строки касаются уже простого пролистывания истории в командной строке, позволяя делать это клавишами PageUp и PageDown (а не только стрелками Up и Down, которые в этом качестве работают всегда и везде):

```
bindkey "^[[5~" up-line-or-history

bindkey "^[[6~" down-line-or-history
```

Этими строками перебрасывается логический мостик к определению кейбиндингов для клавиш, которые в Zsh по умолчанию работают «неправильно» в большинстве терминалов (если не во всех). У меня это Home, End, Delete — их поведение исправляется такими, соответственно, строками:

```
bindkey "^[OH" beginning-of-line

bindkey "^[OF" end-of-line

bindkey "^[[3~" delete-char
```

Это как раз пример тех строк, которые as is копировать не нужно. Во-первых, в общем случае, могут не работать другие клавиши (скорее, не только эти). Во-вторых же и главных, в более иных терминалах коды тех же клавиш могут быть совсем другими. Какими — легко определить, нажав Control+V, а затем «неправильную» клавишу. Именно таким образом получены коды для Home, End и Delete в системе, в которой сочиняются эти строки.

Теперь — опции, определяющие магию Zsh при навигации по файловой системе:

```
cdpath=(/home/current /home/current/alv.me /etc)

setopt autocd
```

Первая строка позволяет с помощью команды cd переходить в подкаталоги перечисленных каталогов, не набирая никаких путей, ни относительных, ни абсолютных, вторая же — обходиться без команды cd.

На грани между опциями навигации и автодополнения находятся такие строки:

```
setopt menucomplete

zstyle ':completion:*' menu select=1 _complete _ignored _approximate
```

Они в паре обеспечивают «менюобразный» вывод списка доступных дополнений по нажатию клавиши табуляции. И это как раз тот случай, когда последовательность строк имеет значение.

Аналогично и со следующими строками — теми самыми, которые обеспечивают волшебство развёртывания сокращённого ввода пути в полный:

```
autoload -Uz compinit

compinit
```

Расширенные подстановки и дополнения обеспечиваются вот этими строками:

```
setopt extendedglob nomatch notify

zstyle ':completion:*' completer _expand _complete _ignored _correct _approximate
```

Строка

```
zstyle ':completion:*' use-compctl false
```

знаменует собой отречение от старого мира — системы дополнения compctl, в пользу новой системы compsys.

Строка

```
zstyle ':completion:*' matcher-list 'm:{a-z}={A-Z}'
```

устанавливает равноправие при дополнениях символов нижнего регистра с верхним.

А строка

```
zstyle :compinstall filename '/home/zsh/.zshrc'
```

фиксирует файл, в который compinstall (функция автоматического

конфигурирования compsys) будет вносить свои изменения при грядущих её вызовах (если они, конечно, будут).

Пора переходить к псевдонимам. Сначала — серия таковых для команд манипуляции файлами, предписывающие запрос подтверждения на таковые или, напротив, форсированное исполнение, в зависимости от ситуации:

```
alias mv='mv -i'

alias cp='cp -iR'

alias cpr='cp -fR'

alias rm='rm -i'

alias rmf='rm -f'

alias rmrf='rm -fR'
```

Оказывается, что для одной-единственной команды ls можно придумать больше псевдонимов, чем для всех файломанипулирующих команд, вместе взятых:

```
alias ls='ls -F'

alias ll='ls -lh'

alias la='ls -A'

alias li='ls -ial'

alias lsd='ls -ld *(-/DN)'

alias lsa='ls -ld .*'
```

На самом деле их можно придумывать ещё и ещё — этот тот необходимый минимум, который я в состоянии запомнить без вреда для рассудка. Расшифровывать псевдонимы не буду — кому надо, и так могут сорвать с них маски, а кто не знает — так ему это и не нужно.

Далее идёт серия псевдонимов для различных команд и утилит разного назначения. Здесь также расшифровка будет лишней. Ибо они или оболее-менее общеприняты:

```
alias h=history

alias df='df -h'

alias du='du -h'
```

Либо обусловлены давними привычками (как, например, more-образный вывод команды less):

```
alias less='less -M'
```

```
alias wget='wget -c'

alias nano='nano -$'
```

Либо связаны со спецификой деятельности:

```
alias wcl='wc -l'

alias wcw='wc -w'

alias wcm='wc -m'

alias wcc='wc -c'
```

Так что можно переходить к следующей убойной фиче Zsh — определению глобальных псевдонимов:

```
alias -g N='2>/dev/null'

alias -g L='|less'

alias -g G='|grep'

alias -g W='|wc -m'
```

Где, впрочем, комментарии тоже излишни.

А посему перехожу к тем самым дистрибутив-специфическим блокам, которые я предназначил для применения в Mint. Это — псевдонимы для субкоманд её утилиты apt, призванные минимизировать ввод при наиболее частых действиях по пакетному менеджменту:

```
alias aptin='apt install --yes'

alias apter='apt purge'

alias aptup='apt update'

alias aptug='apt upgrade'

alias aptse='apt search'

alias aptsh='apt show'
```

Псевдонимы для внутренних команд apt из APT также имеет смысл определить, по крайней мере один, для получения списка инсталлированных пакетов:

```
alias aptlist='/usr/bin/apt list --installed'
```

Смысл этих псевдонимов будет ясен после знакомства с очерком об утилите apt. И в них нет ничего Zsh-специфичного. В отличие от альтернативного метода, основанного на псевдонимах глобальных, которые определяются для соответствующих аргументов команды sudo. Правда, особенность реализации утилиты apt в Mint такова, что она не требует ввода этой команды в явном виде. И потому здесь у меня осталась единственная строка для псевдонима

команды добавления репозиториев:

```
alias -g Ar='add-apt-repository'
```

Хотя я и утверждал не так давно, что приглашение оболочки — нечто вроде вешалки для театра, сам добрался до этой темы только к концу своего конфига. Однако вот — обычное левосторонне приглашение:

```
#PROMPT='%B[%n]$=>%b '
```

Вторичное приглашение:

```
#PROMPT2='%i%U> '
```

Правостороннее приглашение:

```
#RPROMPT=' %B[%~]%b '
```

А вот это — альтернативы, которыми я баловался во время сочинения раздела про приглашения. Все они начинаются с такой пары строк:

```
autoload -Uz promptinit
```

```
promptinit
```

После которых вызывается уже одна из конкретных тем:

```
#prompt fade
```

```
prompt fade white grey blue
```

```
#prompt clint
```

Естественно, что остальные строки должны быть закомментированы.

Осталось немного — всякая всячина. Например, предотвращение выхода из оболочки после случайного нажатия Control+D в пустой командной строке:

```
setopt IGNORE_EOF
```

Отключение раздражающего звукового сигнала при ошибках набора:

```
setopt NO_BEEP
```

Фиксация emacs-образного поведения клавиш (хотя это и так имеет место быть по умолчанию):

```
bindkey -e
```

И под занавес — определение пары переменных среды, для начала умолчального пейджера. Хотя я не так давно говорил, что расширенное перенаправление делает его практически не нужным, но, кроме всего прочего, это ещё и средство для просмотра man-страниц:

```
export PAGER="less"
```

И умолчальный редактор: не смотря на свою любовь к Joe, навыки работы с ним я утратил напрочь, поэтому так:

```
export EDITOR="nano"
```

Вот вроде и всё. Остаётся последний дистрибутив-специфичный стришок — исправление нехорошего поведения history-substring-search в Mint, унаследованного от Ubuntu. А точнее, поведения никакого — эта фича без дополнительных мер просто не работает. Благо меры эти очень просты — создание файла ~/.zshenv с единственной строкой:

```
DEBIAN_PREVENT_KEYBOARD_CHANGES=yes
```

Вот теперь действительно всё — с конфигурированием Zsh «мануальным» способом покончено.

Пакеты и репозитории

Все дистрибутивы Linux, и Mint тут не исключение, организованы по пакетному принципу. Точно также, в виде пакетов, распространяются и любые дополнительные программы для них, создаваемые независимыми разработчиками. И потому одна из важных задач пользователя — это интеграция пакетов в свою систему. Она решается средствам управления пакетами, предназначенным для установки, обновления и удаления программ, учета и разрешения их зависимостей. Однако, прежде чем говорить о таких средствах, не худо посмотреть, что такое пакеты вообще, deb-пакеты в частности и пакеты дистрибутива Mint в особенности. А также — каким образом они организованы в репозитории.

Пакеты, зависимости, библиотеки

Пакеты — это своего рода программные кванты, на которые делится система или дистрибутив. Это могут быть и простые монофункциональные утилиты (например, строчный текстовый редактор ed или архиватор tar), более или менее обширные наборы функционально связанных программ (скажем, coreutils) или составные части огромных программных комплексов (примером чему — Cinnamon, о котором столько гворилось в прошлых очерках).

Термин пакет (английское package) употребляется в двух смыслах: как авторский набор исходных текстов, созданный разработчиком программы, и как комплект скомпилированных из него исполняемых программ и всех их служебных файлов, собранный майнтайнерами дистрибутива или вообще третьими лицами. Пакеты в первом смысле называются исходниками или вообще «сырцами» (от английского Source), во втором — бинарниками. Далее в этой книге речь пойдёт почти исключительно о пакетах во втором понимании этого термина.

Бинарные пакеты специфичны для семейств некогда родственных дистрибутивов, почему часто говорят о системах rpm based или deb based. Но даже если они собраны в одном формате (например, rpm или deb), бинарные пакеты из разных дистрибутивов далеко не всегда совместимы в рамках одной системы. Впрочем, к формату deb, принятому в дистрибутиве Mint и потому в интересующему нас больше всех остальных, вместе взятых, это относится в наименьшей степени: его пакеты сохраняют почти полную бинарную совместимость с пакетами родительской Ubuntu и частичную — с

пакетами прародительского Debian'а.

Ключевым для бинарных, или дистрибутивных, пакетов является понятие зависимостей. Суть его в том, что пакет packagename1 для сборки, установки и (или) функционирования требует наличия в системе пакета packagename2, тот, в свою очередь, может потребовать пакета packagename3, и так далее.

Следует различать зависимости жёсткие и «мягкие». Удовлетворение первых абсолютно необходимо для сборки и функционирования данного пакета. Так, практически любая программа использует главную системную библиотеку glibc (или libc), любое приложение для системы X — одну из главных Иксовых библиотек: старые — xlib, новые — xcb, любая интегрированная рабочая среда — одно из двух основных семейств высокоуровневых библиотек, Qt/kdelibs или Gtk.

«Мягкие» зависимости данного пакета не критичны для его функционирования — удовлетворение их лишь добавляет ему дополнительные функции (например, печати и сканирования для офисных и графических приложений) или возможности (скажем, доступ к файлам данных определённых форматов для той же графики или мультимедиа).

В deb-формате бинарных пакетов предусмотрено более дробное разделение «мягких» зависимостей, но об этом подробнее будет говориться чуть позже. А пока замечу, что часто приходится учитывать и так называемые конфликтующие зависимости — то есть альтернативные по назначению пакеты, не способные ужиться в одной системе.

Понятие зависимостей пронизывает насквозь UNIX-совместимые системы, и особенно важно для свободных их представителей. В то же время пользователи Windows с ним сталкиваются очень редко, и потому постижение его вызывает у них определённые трудности.

Традиционная модель разработки UNIX-программ (то, что задумчиво именуют UNIX Way) характеризуется ярко выраженным стремлением не множить сущности без крайней необходимости. Или, говоря попросту, не изобретать велосипеды. То есть: если требуемая разработчику данной программы функция уже реализована и включена в какую-либо распространённую библиотеку, то наш разработчик скорее всего этой библиотекой и воспользуется, а не будет переписывать ее с нуля. Благо, поскольку все распространённые и общеупотребимые библиотеки открыты, он имеет полную возможность это сделать.

Ведь все программы, вне зависимости от их назначения, неизбежно должны выполнять некоторые однотипные действия, как то: открыть файл, записать его, вывести на экран его содержимое, и так далее, вплоть до закрытия. Сущность таких действий не меняется, что бы программа ни делала. И потому нет никакого смысла программировать такие манипуляции каждый раз заново.

Вот их, как правило, поддаваясь смертному греху лености, и не программируют «с нуля». А объединяют соответствующие директивы в отдельные программные комплексы, именуемые библиотеками (libraries). Сами по себе они к автономному исполнению не пригодны. Однако любая программа, при необходимости совершить одно из типовых действий, вызывает из такой библиотеки некий фрагмент кода, содержащий требуемую последовательность директив.

Библиотеки обычно привязаны к определённым языкам программирования, синтаксису которого подчиняются описания директив, так называемых функции. Поскольку наиболее употребимым в UNIX-системах и их приложениях является язык C, то его функции и требуются чаще всего. Они собираются в главную системную библиотеку, которая почти во всех дистрибутивах Linux именуется glibc.

Однако главной системной библиотекой список не исчерпывается. В UNIX-подобных системах при создании пользовательских интерфейсов используются библиотеки свойств терминала (например, ncurces) для консольных программ и библиотеки, описывающие процедуры отрисовки окон и управления ими — для графических программ системы X, библиотеки интерфейсных элементов и графических примитивов более высокого уровня (Motif, Qt, Gtk), библиотеки описания графических и мультимедийных форматов файлов и тому подобные «сборники». Иными словами, существует тенденция к вынесению в разделяемые библиотеки всех повторяющихся действий и элементов, какие только возможно.

Если библиотек, используемых в программах для консольного режима, не так много, они достаточно универсальны и легко поддаются учёту, то с библиотеками для обеспечения графического режима существенно сложнее. Даже простое перечисление их заняло бы немало места. Поэтому ограничусь констатацией факта, существенного для нашей темы. Обе основные рабочие среды дистрибутива Mint, MATE и Cinnamon, а также штатные приложения обеих редакций базируются на библиотеках Gtk — 2-й и 3-й версий, соответственно. К ним же примыкает и «левая» редакция с Xfce. И лишь KDE-редакция Mint основывается на библиотеках Qt и kdelibs. Поскольку основной героиней этого романа является Cinnamon, то дальше речь пойдёт в основном о пакетах, связанных зависимостями с библиотекой Gtk 3.

Формат пакетов

Как уже было сказано, в дистрибутиве Mint принят deb-формат пакетов. Будучи разработан ещё в прошлом тысячелетии для дистрибутива Debian, формат этот был унаследован от него Ubuntu, во многом предопределив успех последней. А вслед за ней — и удачливость нашего главного героя. Почему deb-формату и следует уделить некоторое внимание.

Пакет deb-формата — архивный файл (собранный утилитой ar, о которой недавно шла речь), включающий три компонента. Первый — это файлик debian-binary, не содержащий ничего, кроме номера версии deb-формата (в данный момент — 2.0).

Второй файл носит имя data.tar.xz и, как легко догадаться, представляет собой tar-архив, сжатый утилитой xz. Содержимое архива — скомпилированные исполняемые бинарники и необходимые им для работы компоненты (библиотеки, конфиги, документация и так далее). Иными словами, все компоненты, которые при установке пакета будут инкорпорированы в файловую иерархию целевой системы. Например, для пакета cinnamon_2.4.1+rebecca_amd64.deb в этом архиве обнаруживается каталог /usr с подкаталогами /usr/bin, /usr/lib, /usr/share, содержащими исполняемые бинарники, библиотеки и разделяемые компоененты, соответственно.

Третий файл именуется control.tar.gz и представляет собой архив файлов,

содержащих всякого рода метаинформацию — описание пакета, его зависимости, классификационную принадлежность, приоритет и так далее (файл control), контрольные суммы всех исполняемых бинаников (файл md5sums), сценарии, выполняемые при установке и удалении пакета (preinst, postinst, prerm и postrm).

Зависимости в терминах deb-пакетов имеют несколько градаций: обязательные (depends), рекомендуемые (recommends), предлагаемые (suggests), конфликтующие (conflicts). Первая градация — это обычные «жёсткие» зависимости, без удовлетворения которых пакет либо не будет работать, либо вообще не установится. С градацией последней тоже понятно — это, так сказать, анти-зависимости: например, Opera текущей, 26-й, версии конфликтует с пакетом opera-12.16.

Ну а зависимости рекомендуемые и предлагаемые — это две разновидности «мягких» зависимостей. Разница между ними в том, что рекомендуемые пакеты обеспечивают «зависимому» пакету дополнительные функции (например, поддержку мыши в консольных приложениях), а пакеты предлагаемые предоставляют дополнительные возможности, вполне вероятно, полезные, но не жизненно необходимые (например, документацию, в том числе на не-английских языках). То есть первая категория как бы более нужная, нежели вторая. Впрочем, таково субъективное мнение майнтайнера конкретного пакета — вполне возможно, что у применителя будет своё мнение по этому поводу. И потому и пакетный менеджер apt, и его графическая «морда» Synaptic, устанавливающие зависимости автоматически, в Mint по умолчанию не делают этого ни для рекомендуемых, ни, тем более, для предлагаемых пакетов, а лишь выводят их список, дабы применитель сам принял решение по данному вопросу.

Кроме того, спецификой deb-пакетов является ещё и существование так называех пред-зависимостей (pre-depends) — при их нарушении установка пакета даже не может начаться, ибо их наличия требует пре-инсталляционный сценарий «зависящего» пакета. Впрочем, с точки зрения пользователя они немногим отличаются от обычных зависимостей типа depends.

Кроме зависимостей, для пакетов deb-формата важно также понятие их приоритета. Оно отражает степень необходимости пакета для функционирования системы, например: обязательный (required), без которого работа системы невозможна, основной (base) и важный (important), также оказывающиеся практически необходимыми, стандартный, то есть имеющийся практически в любой полнофункциональной Linux-системе, дополнительный (optional) — тут уж степень важности каждый должен решать для себя.

Как это принято в мире Open Source, все бинарные пакеты Mint (а также, конечно, Ubuntu и сородичей) сопровождаются исходными текстами, доступными из соответствующего репозитория дистрибутива. И здесь deb-формат проявляет свою специфику: каждый пакет в исходниках обычно включает три файла — packagename.orig.tar.gz, packagename.dsc и packagename.diff.gz.

Первый — самый обычный тарбалл исходных текстов авторского пакета, что подчеркивается словом orig в его имени: формат архива, имя и система нумерации версий также совпадают с таковыми авторского пакета. Файл packagename.dsc содержит в себе всю метаинформацию, необходимую для

правильного построения из него бинарного deb-пакета. А packagename.diff.gz — это те изменения исходного кода, которые вносятся для адаптации пакета непосредственно к данному дистрибутиву. Если таких изменений не потребовалось (или если пакет писался именно для Ubuntu или Mint), он может и отсутствовать.

Репозитории: введение

Пакеты, входящие в дистрибутив (или, если угодно, образующие дистрибутив), валяются не абы как — они организованы в репозитории. Что это такое?

В переводе на русский язык слово репозиторий означает хранилище — и именно его рекомендуют употреблять языковые пуристы (они же те, кто предпочитает называть себя grammar nazi). Однако, как это обычно бывает по жизни, в народе утвердилось иное их именование — repo или, говоря по нашему, по ~~бразильскому~~ кириллическому, репы. Почему во множественном числе — станет понятно из дальнейшего рассказа.

Сам по себе репозиторий действительно можно в первом приближении определить как место хранения пакетов, специально собранных для данного дистрибутива, к которому возможен свободный (мы ведь ведём речь только о свободных системах) доступ.

Однако доступности сервера, хранящего пакеты, недостаточно, чтобы носить звание репозитория. Пакеты в репозитории должны быть структурированы по определённым, присущим данному дистрибутиву, принципам. Система хранения пакетов должна обеспечивать их пополнение, обновление, а главное — поддержку целостности и непротиворечивости пакетов в отношении зависимостей, причём для всех поддерживаемых на текущий момент версий дистрибутива.

Иными словами, пакеты в репозитории должны сопровождаться базами данных — теми самыми, которые используются системой управления пакетами данного дистрибутива.

Кроме того, весьма желательно, чтобы репозиторий зеркалировался на нескольких независимых серверах — по вполне понятным причинам. Правда, это не является непременным требованием. Тем не менее, наличие зеркал — одно из оснований для употребления слова репозитории во множественном числе.

В последние годы получила распространение точка зрения, что право на гордое имя настоящего дистрибутива даёт только собственный репозиторий пакетов, при его отсутствии ты в лучшем случае ремикс дрожащая, а то и вообще жалкий респин. Причём дистрибутив, претендующий на всенародную любовь, обязан иметь репозиторий тем более всеобъёмлющий, чем шире его претензии.

Автор этих строк и сам активно поддерживает озвученный взгляд. Однако Mint, казалось бы, служит живым его опровержением. Ибо в базовой своей части, начиная с ядра и заканчивая Xorg, он не просто основывается на репозиториях Ubuntu, подобно тому, как последняя основывается на репозиториях Debian ветки testing. Нет, Mint просто напрямую использует соответствующие пакеты из официального репозитория Ubuntu, без всяких

модификаций, патчей, пересборок и тому подобных архитектурных излишеств. Нет, Mint, конечно, имеет и собственный репозиторий, но он выглядит песчинкой по сравнению с глыбой репозиториев прародительской системы.

И, тем не менее в звании дистрибутива Mint никто и никогда даже не пытался отказывать. Почему? Да потому, что репозиторий его, что называется, мад, да удал: в нём поддерживаются пакеты, определяющие своеобразие дистрибутива, такие, как «фирменные» утилиты, рабочие среды Cinnamon и MATE. Причём если для MATE репозиторий Mint является как бы вторичным по отношению к «головному» репозиторию проекта mate-desktop.org, то соответствующий репозиторий Cinnamon выступает самым что ни на есть «головным» для этой среды и всех связанных с ней пакетов (вроде файлового менеджера Nemo). И, само собой, таковым он является и для всех дистрибутив-специфичных пакетов — дисплейного менеджера MDM и комплекса Mint-утилит. Ну а что разработчики Mint не занимаются пересборкой базовых пакетов из репозиториев Ubuntu — вполне понятно: зачем изобретать велосипед, когда имеющийся вполне пригоден для езды. Это позволяет сконцентрировать силы на развитии «генеральной линии» собственной системы.

Пояснение: под «головным» репозиторием я понимаю то, что на вражьей мове называется upstream: они поддерживаются основной командой данного пакета или комплекса пакетов, в них хранятся исходники их текущих и разрабатываемых версий, туда же вливаются (или, по крайней мере, должны вливаться) патчи от независимых разработчиков. И на них основываются сборки бинарных пакетов для всех дистрибутивов, испытывающих необходимость в оных.

В ближайших разделах будет последовательно рассмотрено устройство базового репозитория Ubuntu, а затем собственного репозитория Mint. Кроме этих официальных репозиториев, в Mint могут быть использованы пакеты из PPA-репозиториев Ubuntu, собираемые для этого дистрибутива независимыми майнтайнерами. Так что и них будет сказано под занавес.

Устройство репозиториев Ubuntu

Официальные репозитории Ubuntu располагаются по адресу: archive.ubuntu.com/ubuntu. Это — «головное» хранилище пакетов, имеющее многочисленные региональные зеркала, принадлежность которых к стране указывается стандартным двухсимвольным префиксом, например ru.archive.ubuntu.com/ubuntu/ — российское зеркало. Впрочем, как раз российского зеркала утилита Mintsources (о которой шла речь в соответствующем разделе) автоматически не предлагает.

Проще всего с устройством репозиториев с точки зрения применителя можно ознакомиться просмотром их списка в файле /etc/apt/sources.list.d/official-package-repositories.list. Он создаётся автоматически при инсталляции, но затем может быть изменён с помощью Mintsources или отредактирован в текстовом редакторе. Например, у меня относящесяся к репозиториям Ubuntu строки имеют следующий вид:

```
deb http://gd.tuwien.ac.at/opsys/linux/ubuntu/archive trusty main restricted universe multiverse

deb http://gd.tuwien.ac.at/opsys/linux/ubuntu/archive trusty-updates main
```

restricted universe multiverse

deb http://security.ubuntu.com/ubuntu/ trusty-security main restricted universe multiverse

Здесь первый компонент в каждой строке, deb, означает, что речь идёт о бинарных пакетах (про пакеты с исходниками я скажу чуть позже). Далее следует «базовый» URL репозитория. В первых двух строках он соответствует тому серверу, который был выбран мной с помощью утилиты Mintsources по «скоростным» показателям, в третьей — сохранился в первозданном виде. Затем определяется группа пакетов, соответствующая имени релиза. В данный момент для нас актуален Trusty, потому как именно из него Mint Rebecca (как и предшествовавшая ей Qiana) черпает все свои основные, не специфичные для него, компоненты. Групп этих три:

- просто trusty — в неё входят собственно собственно пакеты дистрибутива;
- trusty-updates — «обычные» обновления пакетов, связанные со сменой версий, сборок и исправлением ошибок;
- trusty-security — как нетрудно догадаться, обновления, латающие «дыры» в безопасности системы.

На самом деле в репозитории Ubuntu имеются ещё группы trusty-backport и trusty-proposed, но в Mint они по умолчанию не задействованы, а trusty-proposed вообще можно подключить только вручную (чего, впрочем, делать не стоит без очень веских причин). В нашем же файле среди «Ubuntu'йских» строк есть такая:

deb http://archive.canonical.com/ubuntu/ trusty partner

Это репозиторий для пакетов, в том числе и коммерческих, разрабатываемых партнёрами фирмы Canonical. Я, кажется, никогда ничего из него не устанавливал, ни в Mint, ни в Ubuntu, и больше говорить о нём не буду.

Далее в каждой группе идёт перечень категорий пакетов. Их четыре:

- main — полностью свободные пакеты, официально поддерживаемые разработчиками Ubuntu;
- restricted — пакеты, также официально поддерживаемые дистрибутивом, но не вполне свободные;
- universe — полностью свободные программы, официально дистрибутивом не поддерживаемые и развивающиеся силами независимых разработчиков;
- multiverse — пакеты, аналогично universe официально не поддерживаемые и, подобно restricted, не вполне свободные.

«Не вполне свобода» пакетов из категорий restricted и multiverse выражается в ограничениях на их распространение (например, мультимедиа-кодеки, использующие алгоритмы, патентованные в отдельных странах) или могут распространяться только в бинарном виде (фирменные драйверы для

видеокарт Nvidia).

До сих пор речь шла о репозиториях бинарных пакетов. Однако существуют и параллельные им репозитории с исходниками. Они подлючаются, если отметить соответствующую опцию в Mintsources — при этом герерируется файл /etc/apt/sources.list.d/official-source-repositories.list. Он устроен точно таким же образом, что и official-package-repositories.list, только в первой позиции каждой его строки будет стоять deb-src:

```
deb-src http://gd.tuwien.ac.at/opsys/linux/ubuntu/archive trusty main restricted universe multiverse
```

и так далее.

Особенности репозитория Mint

Репозитории Mint организованы внешне сходно с таковыми Ubuntu, но на самом деле строятся по несколько иным принципам. В файле official-package-repositories.list они описываются двумя строками:

```
deb http://linux-mint.froonix.org rebecca main upstream import
```

```
deb http://extra.linuxmint.com rebecca main
```

Первая — определяет основной репозиторий для пакетов дистрибутива, распространяемых свободно: это аналог групп main и universe из репозиториев Ubuntu. Как и в последних, в ней указывается URL подключённого по умолчанию или выбранного позднее сервера, а затем имя релиза (на текущий момент — rebecca). Далее следует список категорий, однако тут в это понятие вкладывается несколько иной смысл. Так, категория main включает в себя те самый дистрибутив-специфичные пакеты, о которых я столько говорил раньше: «фирменные» утилиты, MDA, Cinnamon, MATE. В категорию upstream входят пакеты, заимствованные из GNOME 3 и специально пересобранные для совместимости с Mint. Здесь же можно обнаружить пакеты для его Xfce-редакции. Категория же import образована пакетами для KDE-редакции, представленными во всей их полноте.

Кроме трёх перечисленных, в основном репозитории имеются категории backport и romeo. В первой — пакеты, перенесённые из более новых версий в более старые, второй — пакеты в стадии тестирования. Обе эти категории подключаются только в том случае, если в Mintsources были отмечены соответствующие опции (ну или они были прописаны руками в official-package-repositories.list).

Репозиторий extra.linuxmint.com не имеет зеркал (по крайней мере, сейчас) и содержит единственную категорию main, включающую не вполне свободные пакеты — это аналог категорий restricted и multiverse из репозиториев Ubuntu. То есть формально в нём есть и все те же категории, что и в основном репозитории, но они пусты (по крайней мере, в момент сочинения этих строк).

Симметрично строкам для репозиториев бинарных пакетов в файле official-source-repositories.list есть строки для описания репозиториев исходников:

```
deb-src http://linux-mint.froonix.org rebecca main upstream import
```

```
deb-src http://extra.linuxmint.com rebecca main
```

Файлы списков репозиториев можно (почти) безбоязненно править руками в текстовом редакторе. В частности, именно таким образом осуществляется апгрейд с релиза на релиз (по крайней мере, в пределах одной LTS-версии): для этого достаточно заменить, например, все вхождения qiana на rebecca и выполнить тотальное обновление системы, о чём будет рассказано в своё время.

В заключение же разговора о репозиториях Mint напомню, что их содержимое можно посмотреть визуально в браузере — и основного, и extra.

О PPA-репозиториях

Кроме официального репозитория, для Ubuntu существует централизованное хранилище репозиториев дополнительных, объединяемых понятием PPA — Personal Packages Archive, то есть входящих в персональный архив пакетов, пополняемый сторонними разработчиками и майнтайнерами. А их, вследствие популярности дистрибутива, очень немало. И поэтому свежие версии многих программ, как популярных (что важно для начинающих), так и весьма экзотических (что часто критично для многоопытных), в первую очередь появляются как бинарники в так называемых PPA-репозиториях Ubuntu.

Для доступа к PPA-репозиториям фирмой Canonical разработан специальный онлайновый инструмент — Launchpad, размещённый на одноимённом сайте. Это — не открытая и не свободная система. Более того, она имеет и платную версию, предназначенную для коммерческих пакетов. Однако мы ведь не рататую абстрактной свободы, и нас это не волнует, верно? Цели и задачи Launchpad'а не ограничиваются обеспечением доступа к PPA-репозиториям. Однако остальные его функции предназначены для разработчиков, и потому нас, применителей, также не касаются.

Казалось бы, при чём здесь Mint? А при том, что практически все пакеты из PPA-репозиториев прекрасно устанавливаются в нём и работают точно так же, как в родной Ubuntu. И потому обращение к ним неизбежно, как как крах мировой системы социализма: далеко не всегда потребности применителя в пакетах удовлетворяются официальными репозиториями Ubuntu, даже дополненными Mint'овскими.

В разделе про Mintsource мы уже занимались подключением PPA-репозиториев. Теперь посмотрим, что получается в итоге. А вот что: в том же каталоге /etc/apt/sources.list.d/ к спискам официальных репозиториев, official-package-repositories.list и official-source-repositories.list, присоединяются файлы вида *.list — по одному на каждый подключённый репозиторий, где под маской скрывается его так называемое ppa-имя.

Откуда берётся ppa-имя — расскажу в очерке про управление пакетами. А пока — как оно выглядит. Большинство пакетов в PPA-репозиториях собирается и поддерживается майнтайнерами-индивидуалами, и потому здесь нередко можно видеть их имена, фамилии или ники, например, ppa:andrew-crew-kuznetsov/crew — репозиторий, поддерживаемый Андреем Crew Кузнецовым, разработчиком программы XNeur и сборщиком пакета hunspell-ru-ie-yo, словаря для проверки русской орфографии, поддерживающего букву Ё. В других случаях это просто имя пакета, часто с отражением статуса разработки, например, ppa:marlin-devs/marlin-daily — репозиторий «ежедневных» сборок файлового менеджера Marlin. Репозиторий может включать несколько связанных друг с другом пакетов — и тогда

называться по главному из них, например: ppa:zfs-native/stable и ppa:zfs-native/daily — репозитории пакетов поддержки ZFS on Linux стабильной и разрабатываемой ветки, соответственно.

Возможны и более причудливые имена, например, ppa:mystic-mirage/komodo-edit — репозиторий текстового редактора Komodo Edit. Важно, что они в обязательном порядке включают «префикс» ppa:, который в имени соответствующего list-файла отбрасывается. Зато завершается последний обязательным компонентом — именем релиза. Например, для Komodo Edit имя list-файла — mystic-mirage-komodo-edit-trusty.list.

Внутри такого файла — обычно две строки. Например, для пакета komodo-edit они будут такими:

```
deb http://ppa.launchpad.net/mystic-mirage/komodo-edit/ubuntu trusty main
```

```
deb-src http://ppa.launchpad.net/mystic-mirage/komodo-edit/ubuntu trusty main
```

То есть в одном файле описывается и репозиторий бинарников, и репозиторий исходников. Если последний отсутствует — соответствующей строки не будет. Впрочем, в PPA-репозиториях пакетов без исходников не водится. А вот среди «не вполне свободного» софта встречаются, примером чему — браузер Opera: файл opera-stable.list выглядит следующим образом:

```
deb http://deb.opera.com/opera-stable/ stable non-free #Opera Browser (final releases)
```

Однако случаи, когда приходится искать пакеты за пределами PPA-репозиториев, очень редки. Ибо в них, как в Греции есть всё — и ещё немного больше, чем всё.

Подключение PPA-репозиториев

Чтобы воспользоваться всем греческим сокровищем,описанным на предыдущей страницу, необходимо научиться подключать дополнительные репозитории вообще и PPA-репозитории в особенности.

В очерке про фирменный инструментарий Mint был описан один способ их подключения — с помощью утилиты mintsources, она же software-sources. Однако это можно сделать и в CLI — командой add-apt-repository (или apt-add-repository — это опять-таки символическая ссылка на неё). Поскольку очевидно, что для подключения репозиториев требуются права администратора, команда эта должна быть дана в такой форме:

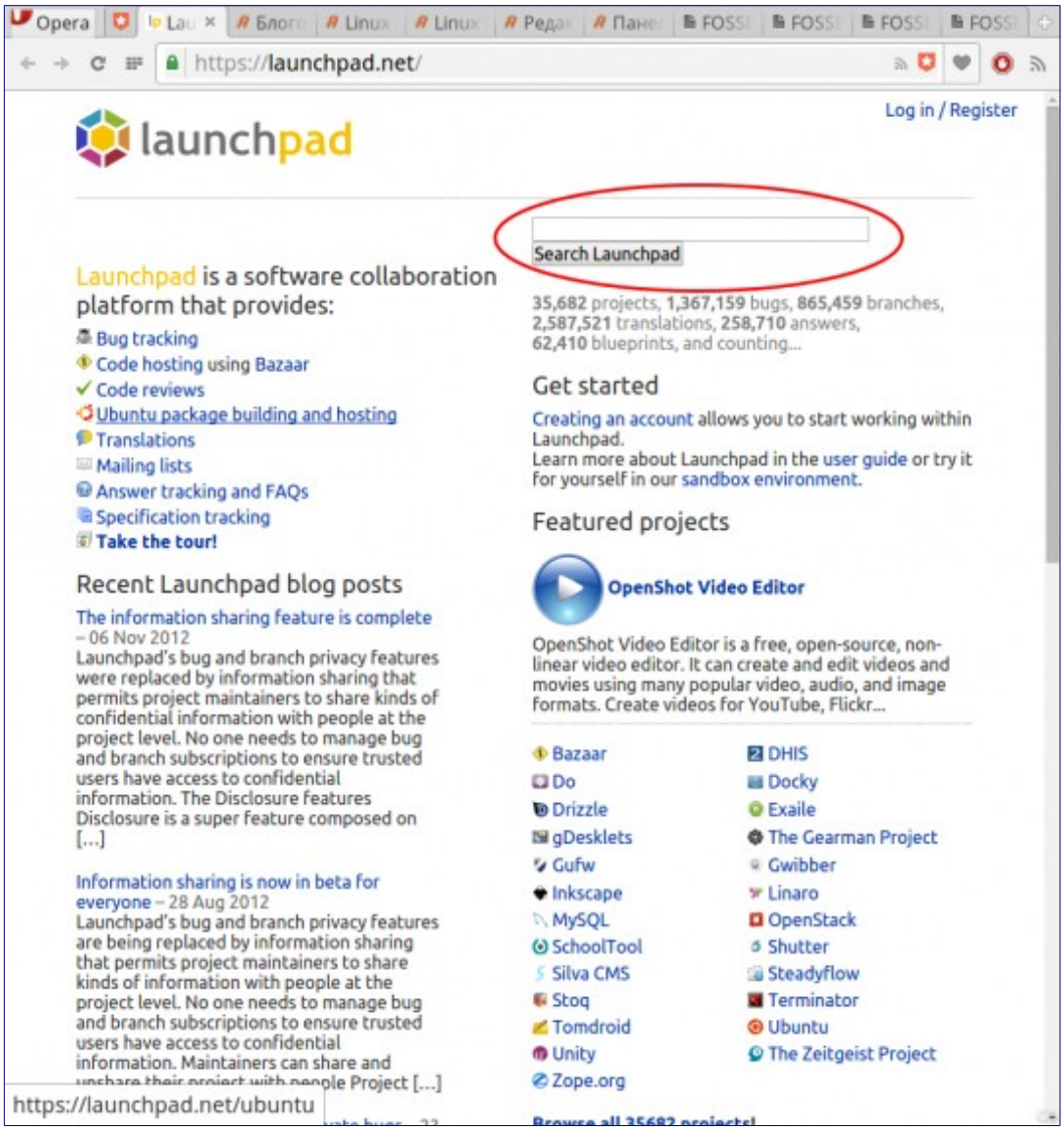
```
$ sudo add-apt-repository ppa-name
```

Если воспользоваться глобальными псевдонимами Zsh, это будет выглядеть примерно так:

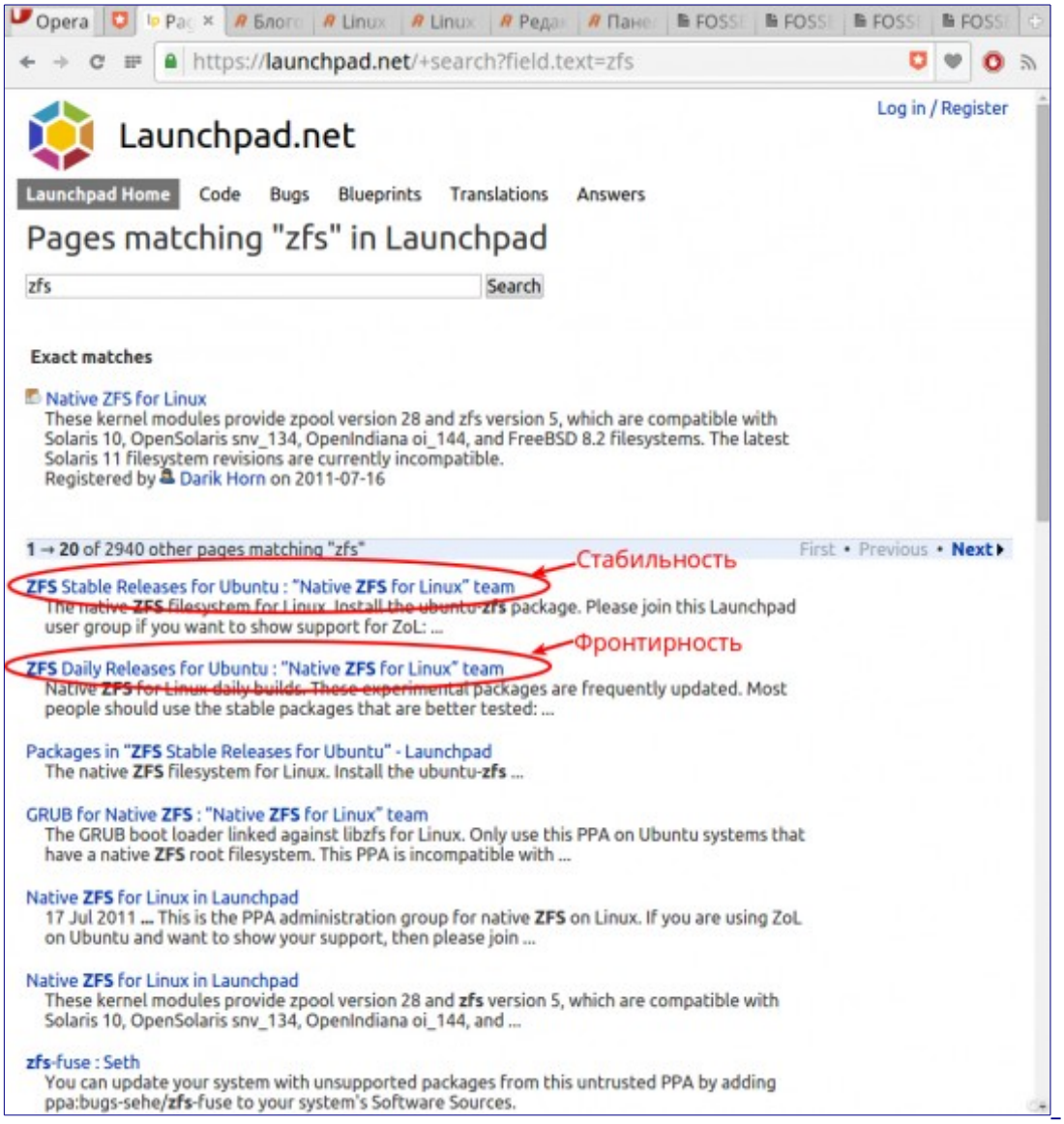
```
$ sudo Ar ppa-name
```

Так что остаётся только сущая мелочь — определить это самое ppa-имя нужного репозитория. Между прочим, та же проблема была и при использовании mintsources — и он никак не может помочь в этом деле. Как её решить?

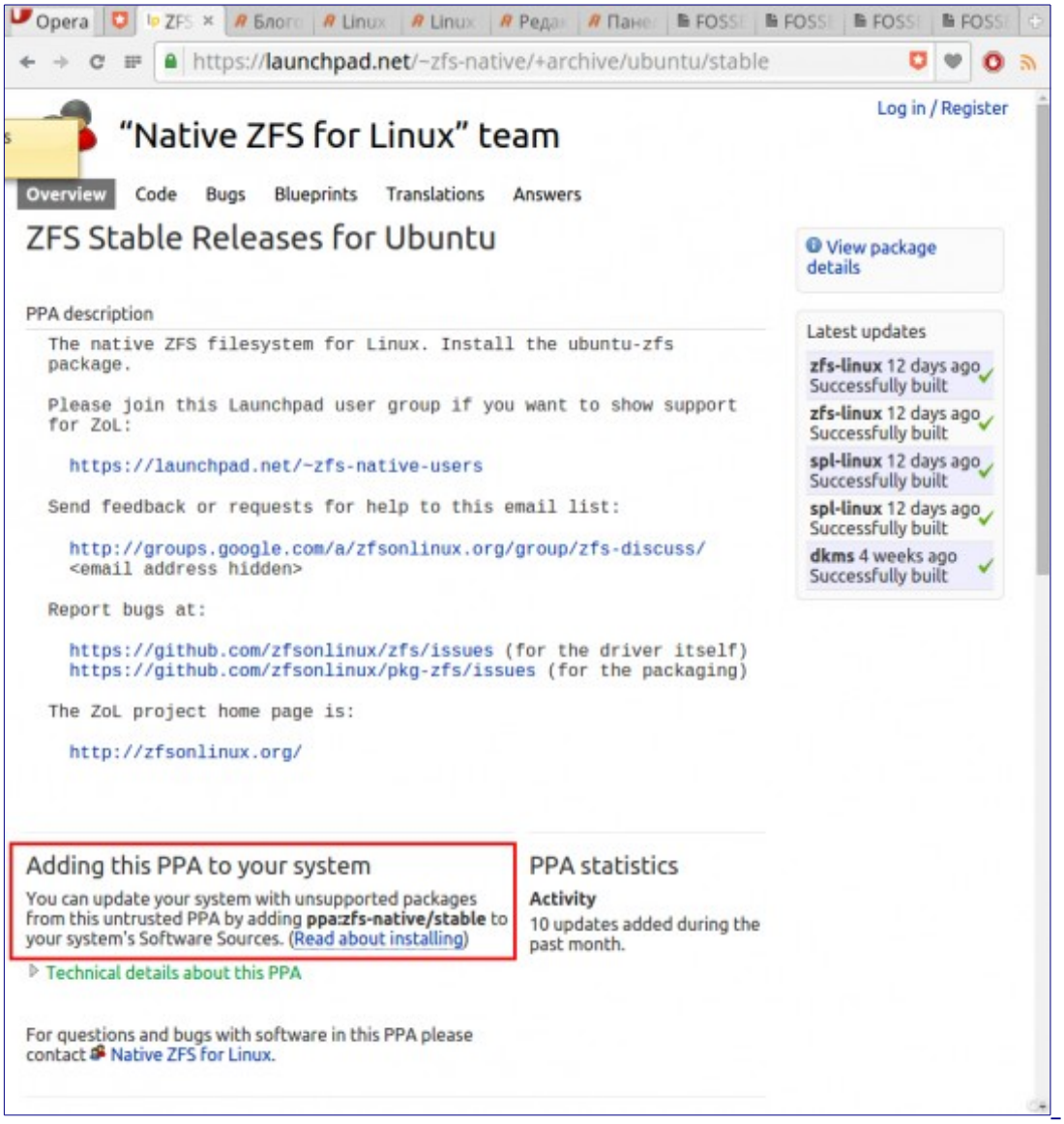
Можно, конечно, походить по форумам Ubuntu'йской тематики, можно сделать запрос к Гоше или Яше, указав имя искомого пакета, можно... да много чего можно сделать, чтобы по прошествии изрядного или ещё большего времени получить нужный результат. А можно, действуя методично и планомерно, прибегнуть к универсальному способу. И для начала зайти на Launchpad:



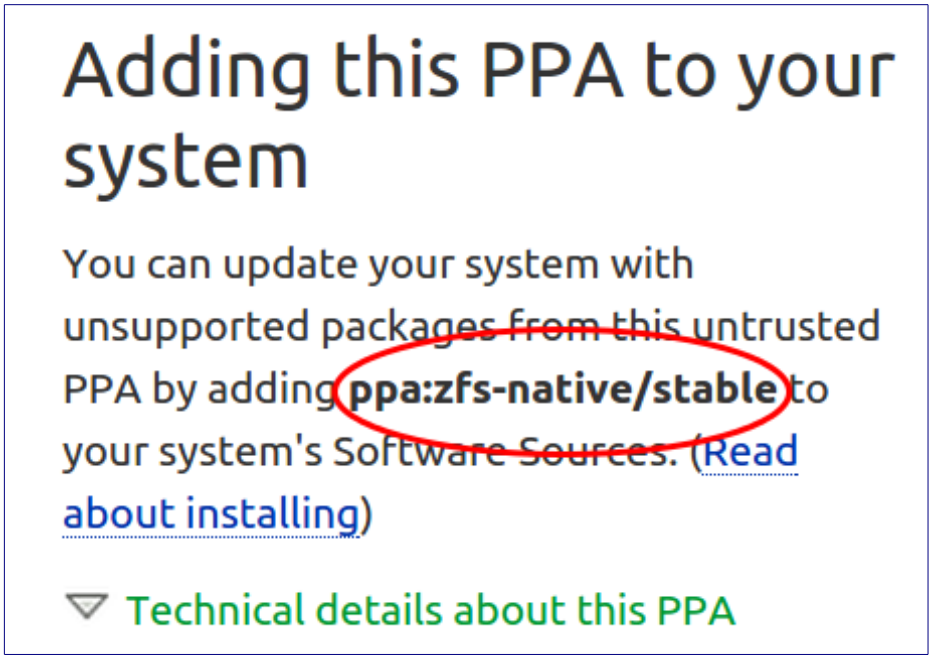
Далее в поле поиска следует набрать имя требующегося пакета или его фрагмент, например, zfs. Далее в списке выдачи результатов нужно отыскать нужную строку — в данном случае это будет ZFS Stable Releases... или ZFS Daily Releases..., в зависимости от требований — стабильности или фронтирности:



Теперь — щёлкнуть на ней (предположим, мы предпочти синицу стабильности журавлю фронтирности), и прочитать раздельчик, озаглавленный Adding this PPA to your system:



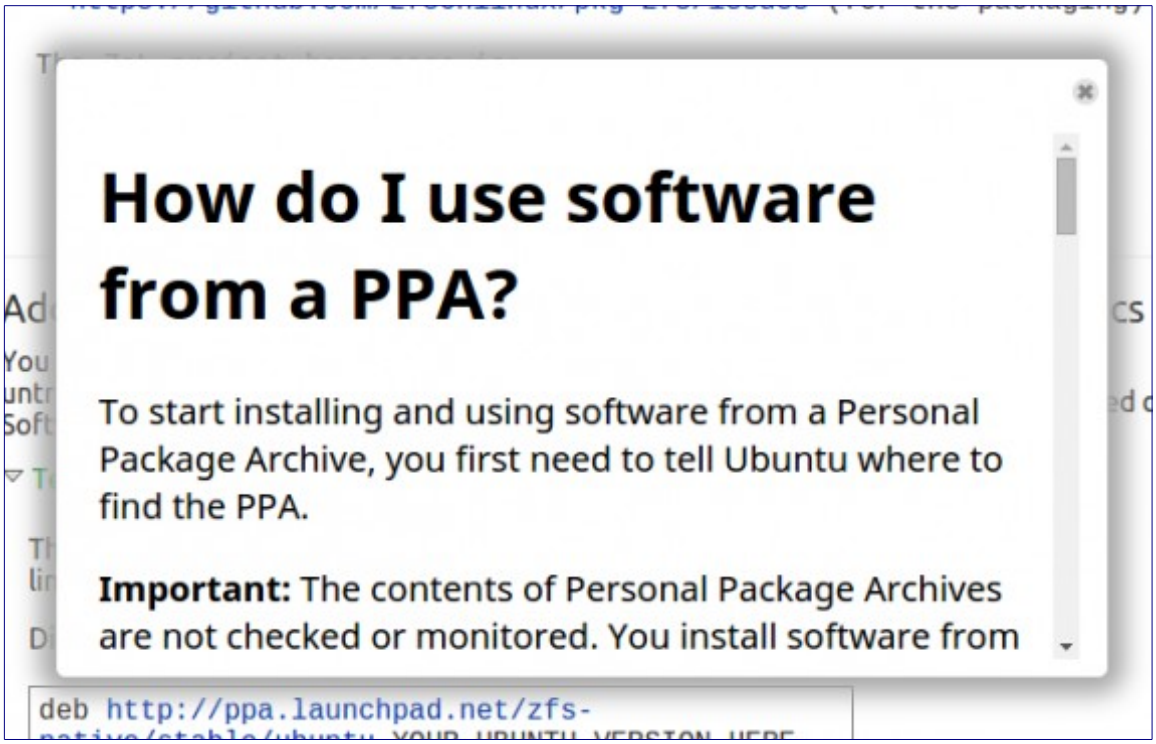
Искомое ppa-имя будет выделено полужирным шрифтом:



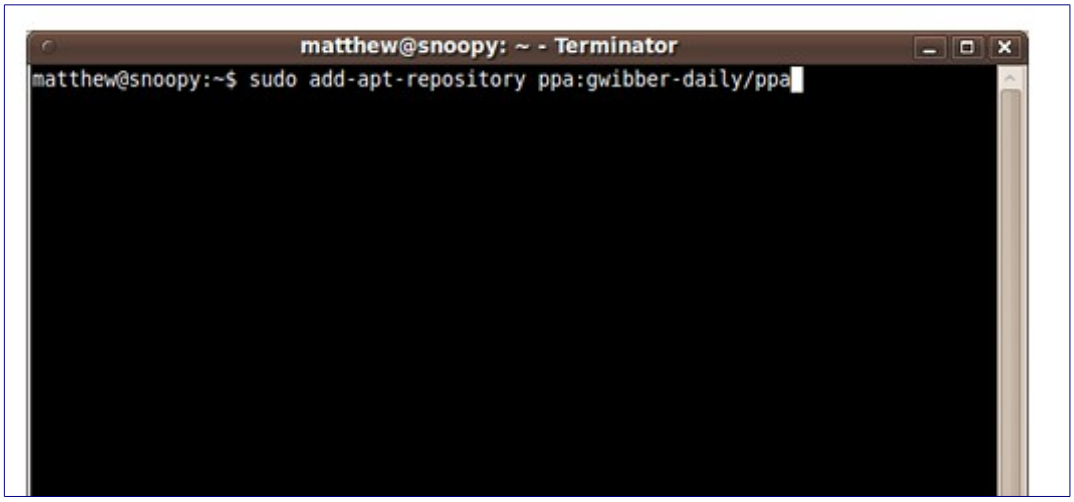
Его и следует подставить в качестве аргумента команды:

```
$ sudo add-apt-repository ppa:zfs-native/stable
```

Дабы развеять все сомнения, можно пройти по ссылке Read about installing. Появится всплывающее окошко, в котором процедура добавления PPA-репозитория будет описана подробно:



И не только описана, но и проиллюстрирована:

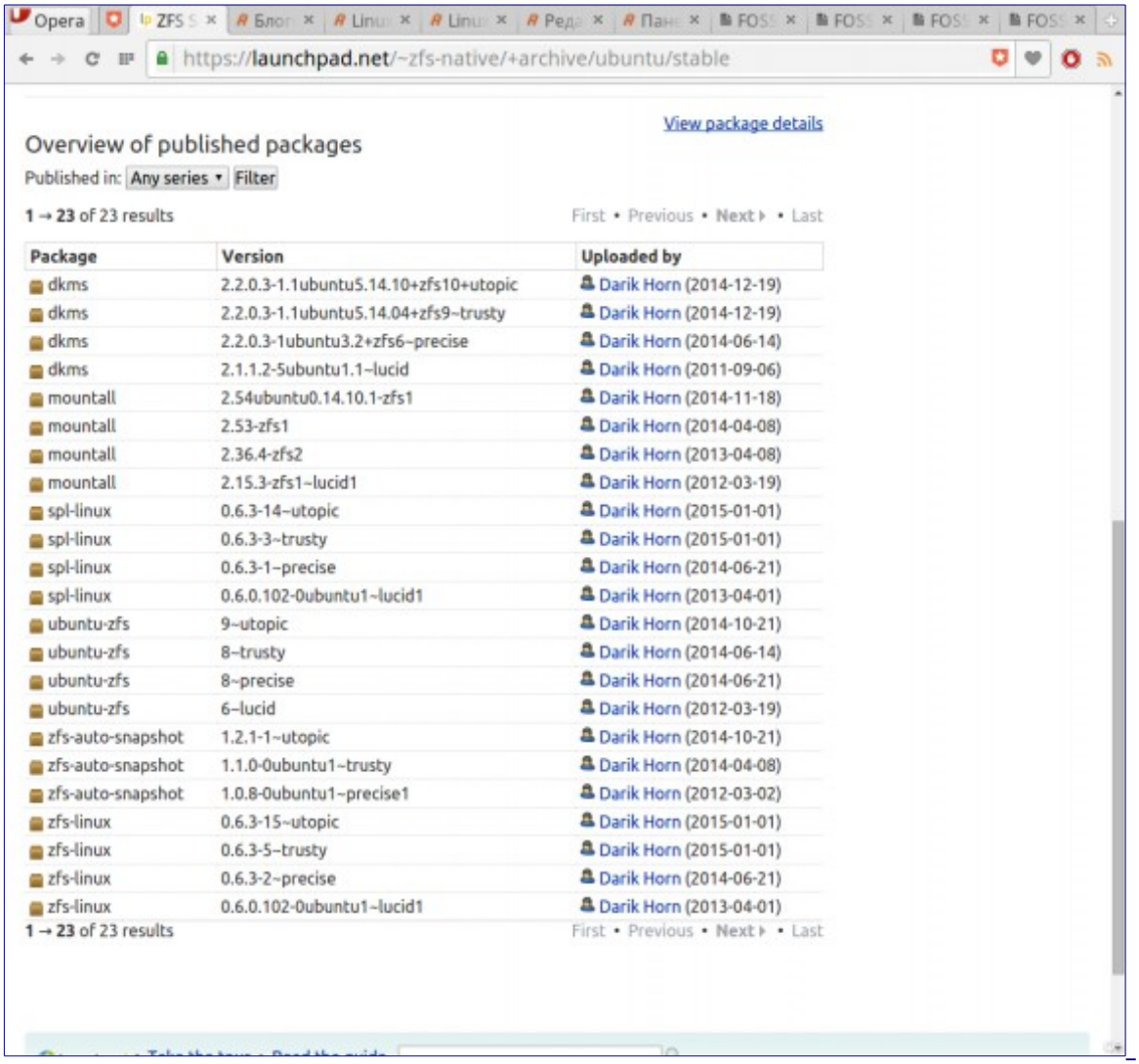


Да, выполнив последнюю команду, нужно ни в коем случае не забыть проделать процедуру апдейта:

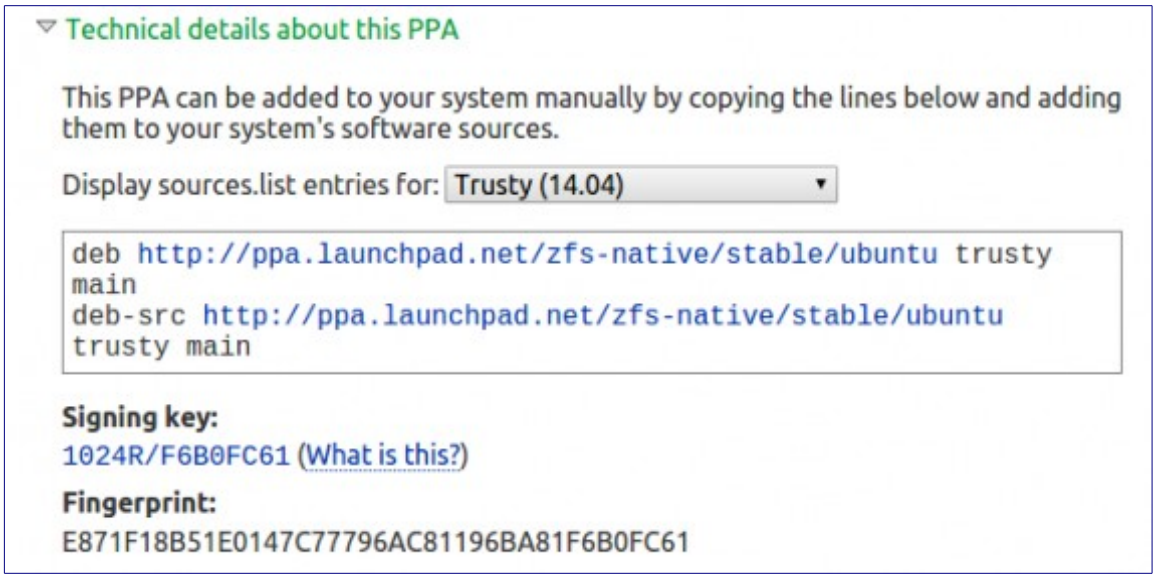
```
$ apt update
```

Обращаю внимание — команда sudo отсутствует: как будет показано в следующем очерке, реализация apt для Mint позволяет применителю не утруждать себя её вводом.

Теперь можно устанавливать пакеты из новообретённого репозитория (о чём также пойдёт речь в следующем очерке). А ознакомиться со списком оных можно ещё на странице Launchpad'a:



Впрочем, можно поступить иначе, обойдясь без команды add-apt-repository: развернуть строку Technical details about this PPA и в выпадающем меню выбрать имя (номер) своего релиза Ubuntu. В нашем случае это будет Trusty (14.04), так как и Mint Qiana, и Mint Rebecca основаны на нём:



Строки из поля ниже просто копируются в новый текстовый файл, создаваемый в каталоге /etc/apt/sources.list.d под именем package_name-status-release_name.list, то есть в нашем примере — zfs-native-stable-trusty.list. После чего опять же не забыть про

```
$ apt update
```

Не правда ли, любой из предложенных способов проще, чем беготня по форумам? Да и Гошу с Яшей не стоит беспокоить по пустякам.

Отдельный случай — подключение репозиториев, содержащих всякие красивости, вроде тем, пиктограмм или обоин. Главным источником таковых является сайт NoobsLab. Здесь также всё просто — в каждой теме или коллекции пиктограмм имеется исчерпывающая инструкция по подключению соответствующего репозитория. В подавляющем большинстве случаев она сводится к выполнению директив

```
sudo add-apt-repository ppa:noobslab/themes
```

```
sudo add-apt-repository ppa:noobslab/icons
```

что, очевидно, нужно проделать единократно, с последующим апдейтом, то есть в нашем случае опять-таки

```
$ apt update
```

Что же до обоин — думаю, каждый уважающий сеья применитель-эстет имеет собственную коллекцию картинок для использования в этом качестве.

Редко, но бывает так, что приходится устанавливать пакеты из какого-либо иного источника, нежели PPA-репозитории. Но в этом случае грамотно сделанный пакет при установке сам добавляет свой репозиторий в общий список — так, например, происходит при установке биаузера Opera версии 26.X для Linux. Либо — сопровождается сведениями о том, как это сделать самостоятельно. Если ни того, ни другого не имеет места быть — возникает вопрос: а стоит ли связываться с таким пакетом?

Управление пакетами

Работа с пакетами предполагает следующие действия — их установку с занесением в локальную базу данных, отслеживание зависимостей (и иногда их разрешение) обновление, удаление, получение информации о пакетах, иногда конфигурирование. Для понимания сути их необходимо дать

Терминологическое введение

В системах пакетного менеджмента deb based дистрибутивов, в том числе и в Mint, пакеты объединяются в категории, секции и группы. Список категорий включает следующие пункты:

- Установленные пакеты — очевидно из названия;
- Обновляемые пакеты — установленные пакеты, для которых в репозитории доступны более новые версии;
- New Packages — пакеты, добавленные в локальный кэш после последней его очистки;
- Неустановленные пакеты — пакеты, отсутствующие в системе, но доступные из репозиториев;

• Виртуальные пакеты — не существующие пакеты, указывающие на другие пакеты, которые нужно использовать или которые предоставляют схожие функции.;

• Задачи (Tasks) — группы пакетов (метапакеты), которые предоставляют лёгкий способ выбора заранее сформированного набора пакетов под определённую цель.

В секции пакеты группируются по назначению: программы для администрирования, базовые пакеты, текстовые редакторы, и так далее.

Группы представляющие собой разделы официального репозитория. В Mint они таковы: main, upstream, import, backport, romeo.

Каждый пакет в терминологии имеет основной статус, обозначаемый строчной литерой; в их число входят:

• i (от install) — установленный пакет;

• p (от purge) — пакет не установленный или деинсталлированный «вчистую» (то есть с удалением его конфигурационных файлов);

• c (от clean) — пакет, деинсталлированный с сохранением конфигурационных файлов;

• v (от virtual) — виртуальный пакет.

Кроме того, пакеты могут иметь один из следующих дополнительных статусов, хотя это и не обязательно:

• A (от Auto) — установленный автоматически, как зависимость другого пакета; пакеты, не имеющие статуса A, считаются установленными вручную;

• h (от hold) — пакет с фиксированной версией (то есть не подверженный апгрейду);

• u (от unpacked) — пакет распакованный, но не установленный;

• H — «недоустановленный» пакет;

• C — пакет установленный, но не настроенный;

• B — «сломанный» пакет, то есть установленный с нарушением зависимостей.

Обращаю особое внимание на пакеты, имеющие статус A: они устанавливаются вместе со своими зависимостями и могут быть удалены только вместе с ними. Правда, как мы увидим дальше, статус установленного пакета может быть изменён, и тогда он станет доступным для индивидуального удаления.

В сущности, все действия по управлению пакетами в Mint сводятся к изменению их статуса. И делается это с помощью инструментов текстового режима (утилиты dpkg и apt) или графических фронт-эндов (Менеджер

программ и Synaptic).

Средства для работы с пакетами. Обзор

Инструментарий для работы с пакетами можно разделить на пять групп:

- установщики пакетов;
- оболочки для них;
- менеджеры пакетов;
- их графические фронт-энды;
- центры приложений.

Первая группа — это низкоуровневые утилиты для работы с единичными пакетами: их установки, удаления, etc. В нашем случае эту роль выполняет семейство утилит dpkg. Отслеживание зависимостей здесь выполняется на уровне удовлетворения или неудовлетворения, попыток автоматического разрешения зависимостей не предпринимается. Семейство это не уникально для Mint, а присутствует во всех дистрибутивах, использующих deb-формат пакетов.

Оболочки для установщиков пакетов выполняют те же самые функции, что и они сами, но посредством не прямых команд, а графического интерфейса. В Mint такой оболочкой для dpkg является Gdebi.

Менеджеры пакетов работают уже не с единичными пакетами, а с их репозиториями. И, кроме перечисленных функций, их непременной обязанностью является не только отслеживание зависимостей, но и их автоматическое, по возможности, разрешение. В большинстве deb based дистрибутивов эту роль выполняет семейство утилит APT. Однако в Mint имеется собственная реализация такого инструмента в лице интегрированной утилиты apt — её следует чётко отличать от одноимённой команды из обще-Debian'овского APT-семейства.

Четвёртая группа — графические фронт-энды для менеджеров пакетов. В Mint она представлена программой Synaptic.

Что же касается пятой группы — это самые высокоуровневые программы, в которых прозрачно для применителя интегрированы функции поиска пакетов в Сети, подключения к содержащим их репозиториями и собственно пакетный менеджмент. Название её заимствовано от Центра приложений Ubuntu — первого представителя этой группы. В Mint её аналогом выступает Менеджер программ, о котором говорилось в очерке про фирменные утилиты этого дистрибутива. Остальные же инструменты работы с пакетами будут рассмотрены в ближайших очерках.

Установщик пакетов dpkg

Утилиты семейства dpkg, предназначенные для работы с единичными deb-пакетами, были исторически первым средством автоматического развертывания пакетов, учитывающим их зависимости. Они лежат в фундаменте всех надстраивающих их систем (apt, synaptic, mintinstall. В ряде

случаев dpkg оказывается наиболее простым средством для установки или удаления пакета, а также получения информации о нем. Кроме того, одна из утилит семейства, dpkg-reconfigure, с которой мы сталкивались во время доводки текстовой консоли, оказывается незаменимой при настройке пакетов установленных.

Вообще, возможности утилит семейства (см. man (1) dpkg) очень широки, и потому заслуживают рассмотрения, хотя бы в минимально необходимом для применителя объёме. Наиболее употребимы из них — следующие:

- собственно dpkg — средство для установки и удаления программ;
- dpkg-query — инструмент создания запросов к базе данных deb-пакетов;
- dpkg-reconfigure — программа для настройки установленных пакетов.

А вообще это семейство включает в себя около 25 утилит — полный их список можно просмотреть таким образом:

```
ls /usr/sbin/dpkg* /usr/bin/dpkg*
```

Утилиты эти входят в состав пакетов dpkg и dpkg-dev; первый, предназначенный для основных действий с бинарными пакетами, устанавливается по умолчанию в ходе первичной инсталляции; второй же, включающий утилиты для манипуляции с пакетами исходников, должен быть установлен дополнительно (или устанавливается как зависимость при инсталляции deb-специфичного сборочного инструментария).

Для начала рассмотрим установку пакетов. Итак, если нам необходимо установить единичный пакет, поступаем так:

```
$ dpkg -i path2/packagename.deb
```

и дело в шляпе — через считанные мгновения пакет packagename.deb будет установлен: это обеспечивает опция -i (от install) вслед за командой dpkg. Дабы в дальнейшем не повторяться, замечу, что все действия по установке и удалению пакетов требуют полномочий администратора, приобретаемых временно командой sudo.

Разумеется, успешной установка пакета будет только при соблюдении следующих условий:

1. физическом наличии его в пределах досягаемости с локальной машины (на подключенной файловой системе, смонтированном компакт-диске или ином носителе);
2. знании точного пути (path2) к нужному файлу пакета (имя его, кстати, должно быть указано полностью);
3. отсутствии неудовлетворенных зависимостей.

Из первого условия следует, что dpkg удобно использовать при доустановке компонентов с инсталляционного CD/DVD (или установке заблаговременно скачанных пакетов). Второе условие самоочевидно. Ну а третье также

выполнимо без особого труда: в случае нарушения зависимостей dpkg выдаст сообщение об ошибке с полным перечнем того, что нужно установить для ее устранения, причём в списке будут перечислены только обязательные зависимости. И достаточно все необходимые пакеты поместить в командную строку:

```
$ sudo dpkg -i path2/packagename1.deb … path2/packagename#.deb
```

для того, чтобы они были установлены единой операцией (если, конечно, все эти пакеты имеются в наличии).

Другое часто требующееся применение команды dpkg — удаление ненужных пакетов. Это делается двояко: команда

```
$ sudo dpkg -r packagename
```

удалит пакет, но сохранит настроечные его файлы, а команда

```
$ sudo dpkg -P packagename
```

произведет полную очистку системы от всех компонентов пакета (кроме конфигурационных файлов в домашнем каталоге пользователя — от них в любом случае придется избавляться вручную). Правда, только если он не связан зависимостями с другими пакетами — в этом случае последует сообщение о невозможности удаления пакета и выведен список его зависимостей, этому препятствующих.

Обратим внимание — в аргументах обеих команд фигурирует уже не полное имя пакета, а только его базовая часть. Это распространяется на все случаи использования dpkg (и других команд ее семейства), когда речь идет об уже установленных пакетах.

Следующая сфера деятельности команд семейства dpkg — получение информации о пакетах. И здесь первое дело — это получение списка пакетов, установленных в системе:

```
$ dpkg -l
```

Что в моей системе даёт примерно такой вывод:

```
ii  accountsservice 0.6.35-0ubuntu7.1 amd64 query and manipulate user account information

ii  acl 2.2.52-1 amd64 Access control list utilities

...

ii  zsh 5.0.2-3ubunt amd64 shell with lots of features

ii  zsh-common 5.0.2-3ubunt all architecture independent files for Z

ii  zsh-doc 5.0.2-3ubunt all zsh documentation - info/HTML format

ii  zsh-lovers 0.8.3-0ubunt all tips, tricks and examples for the zs
```

До появления интегрированной утилиты apt команда dpkg -l была чуть ли не

единственным способом получения списка установленных пакетов. Или, по крайней мере, самым простым.

Для уже установленных пакетов информацию о них проще всего получить с помощью команды dpkg-query, требующей указания какого-либо из операторов действия и имени пакета в качестве аргумента. Операторы действия команды dpkg-query можно вывести так (поскольку получение информации о пакетах никак не влияет на систему в целом, необходимости в правах администратора тут не возникает):

```
$ dpkg-query --help
```

Они следующие:

- -s или --status — вывод детального статуса пакета, включающий:
    - имя пакета, собственно статус (установлен ли он) и приоритет;
    - секция репозитория, к которой пакета относится (например, editors — для текстовых редакторов);
    - размер пакета в установленном виде;
    - имя майнтайнера, архитектура, для которой пакет собран, и номер версии;
    - описание зависимостей и конфликтов;
    - краткое (в один абзац) описание пакета.
- -p или --print-avail — практически то же самое, но в форме, приспособленной для печати;
- -l или --list — тоже своего рода описание статуса, включающее сведения о том, установлен ли пакет, нуждается ли он в обновлении, нет ли ошибок в его настройке, и так далее;
- -W или --show — просто вывод номера версии в форме:

```
 $ dpkg-query -W nano

nano    1.3.8-2
```

- -L или --listfiles — полный список файлов, относящихся к данному пакету, в форме:

```
 /.

/etc

/etc/nanorc

/usr

/usr/share
```

```
/usr/share/doc

/usr/share/doc/nano

...
```

и так далее (пример для текстового редактора nano);

• -S или --search — поиск пакета, к которому относится некий файл, указанный в качестве аргумента; может выполнить и обратную задачу — поиск всех файлов, принадлежащих данному пакету, вывод в этом случае оказывается аналогичным dpkg-query -L.

Повторю, что все сказанное о информации по пакетам, относится к пакетам уже установленным. Для получения же сведений о неустановленных пакетах удобнее использовать графическую оболочку GDebi, о которой будет говориться в следующем разделе.

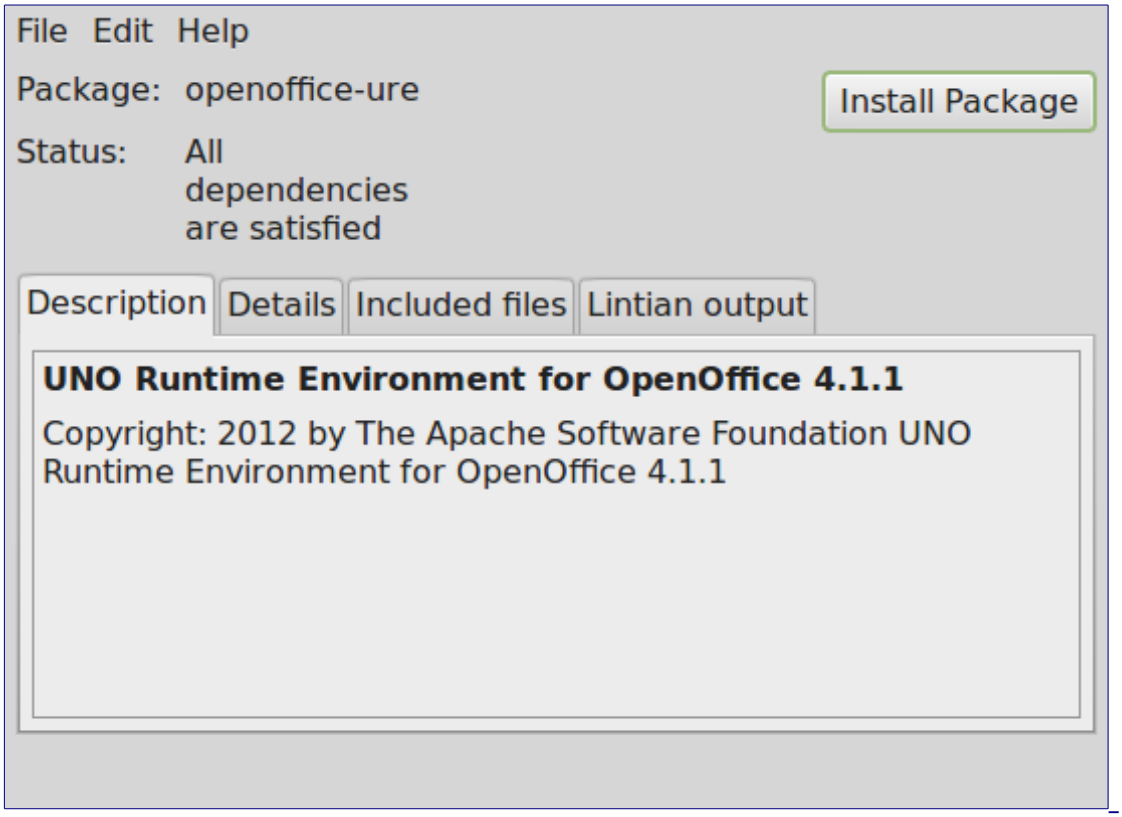
ещё одна важная задача утилит dpkg — выполнение настройки отдельных, уже установленных, пакетов. Предназначенная для этого команда так и называется — dpkg-reconfigure, и запускается таким образом:

```
$ sudo dpkg-reconfigure packagename
```

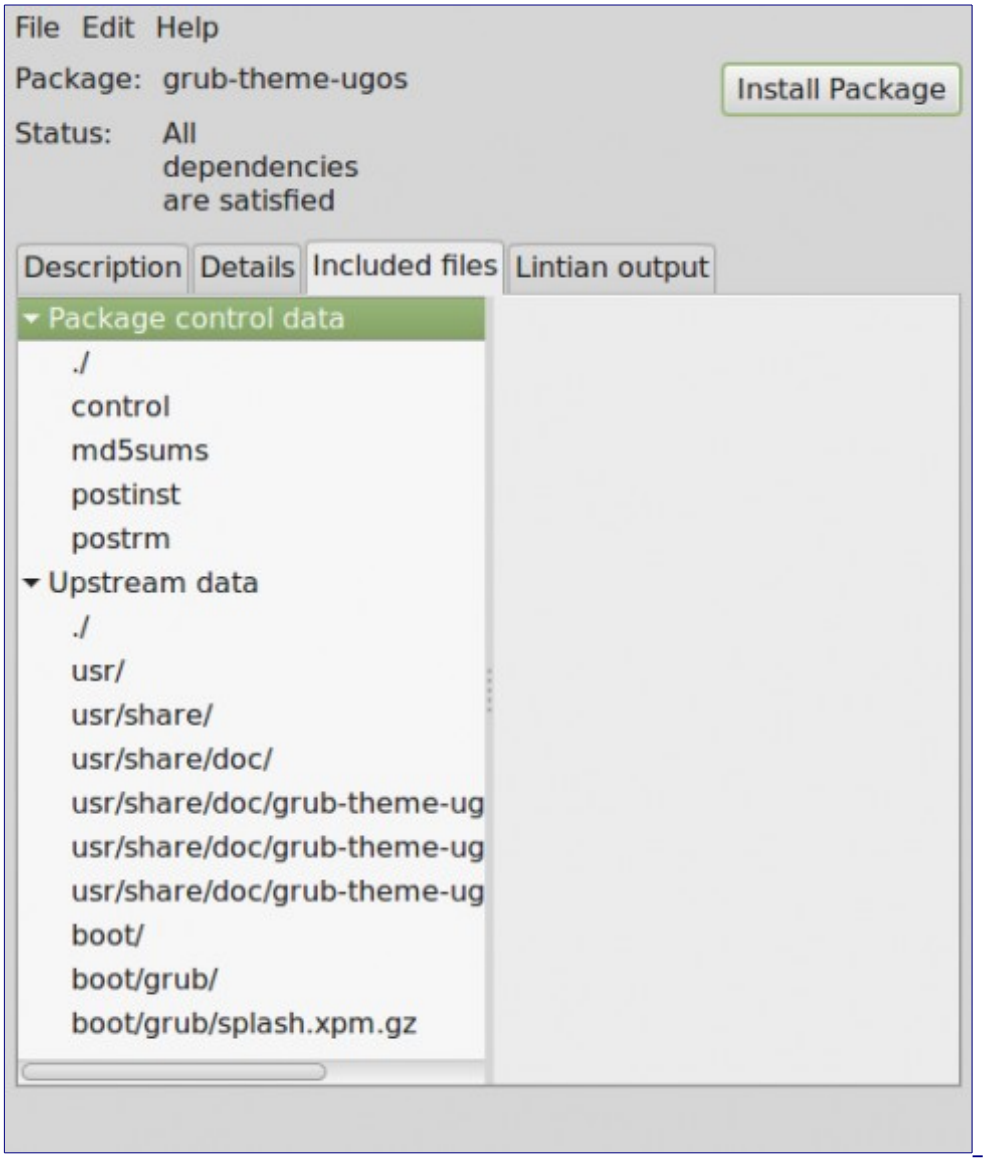
После этого вызывается диалоговая программа конфигурации — debconf, и ответы на серию более или менее тривиальных вопросов позволяют добиться желаемого результата. Каковы эти вопросы — зависит от настраиваемой программы. В частности, ранее dpkg-reconfigure была использована для настройки экранных шрифтов в консоли.

Установщик пакетов GDebi

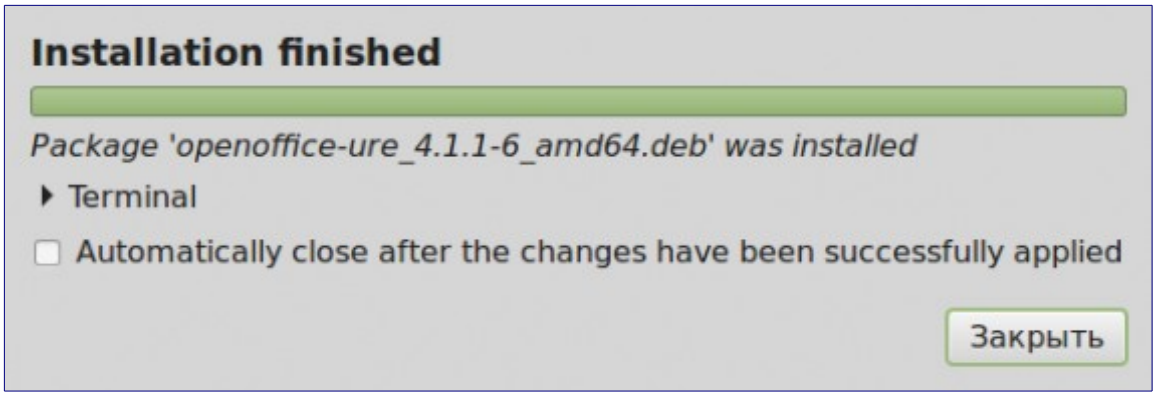
GDebi представляет собой графический фронт-энд для утилиты dpkg. Она разработана фирмой Canonical специально для Ubuntu и потому, естественно, имеется и в Mint (о котором далее и пойдёт речь). Запустить GDebi можно из секции Администрирование главного меню Cinnamon — но тогда придётся обратиться к пункту File -> Open её меню, и потом долго рыскать по файловому древу в поисках нужного имени. Так что более простой способ её запуска — клик на имени deb-файла в файловом менеджере Nemo. Что представит такую картинку:



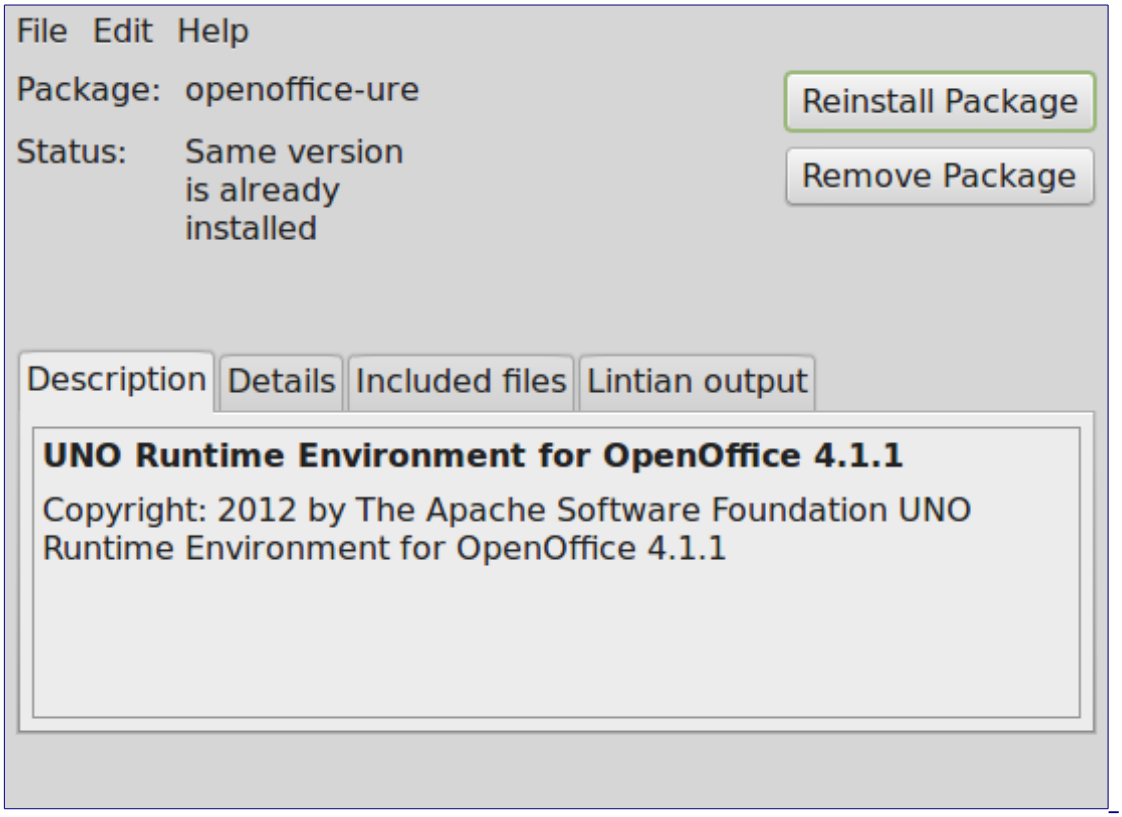
Самая ценная информация здесь — это список файлов, входящих в состав пакета. В отличие от Synaptic'а, о котором речь пойдёт со временем, GDebi выводит его даже для не установленных пакетов:



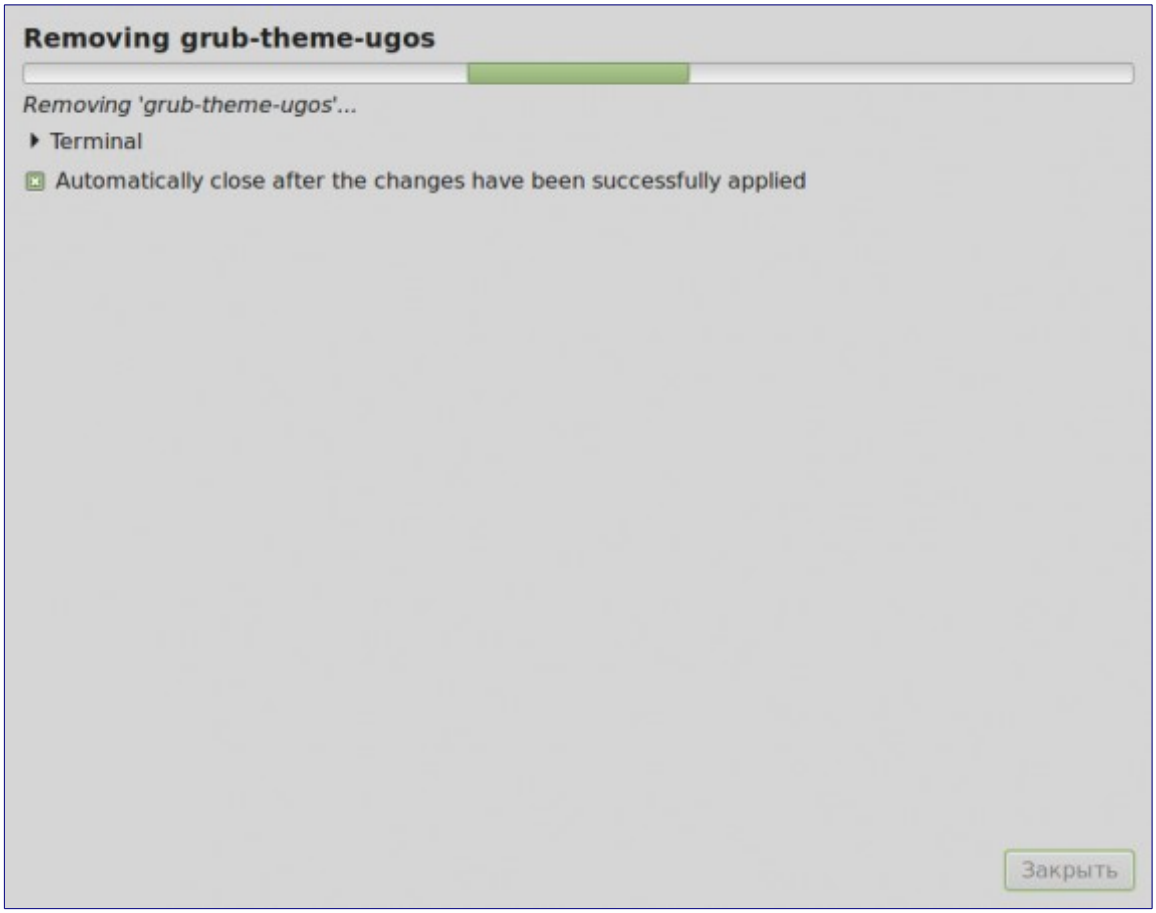
В случае, если в пакете всё устраивает, он устанавливается нажатием одноимённой кнопки, что сначала потребует авторизации, а затем незамедлительно начинается установка:



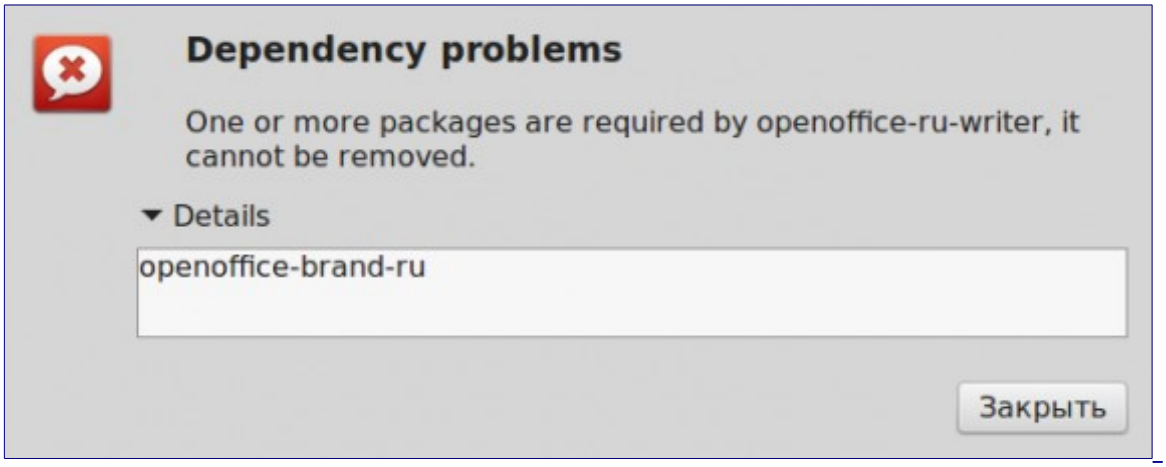
Проверка зависимостей, естественно, осуществляется как и в dpkg — на уровне удовлетворённости или неудовлетворённости. В последнем случае выводится сообщение о том, какие пакеты следует установить для их разрешения. По завершении установки картинка становится такой:



Удаление пакета выполняется тем же образом: авторизация и собственно удаление:



И завершается возвращением исходной картинки. Если от удаляемого пакета зависит какой-либо другой, то опять же последует сообщение об ошибке:



Никаких преимуществ против консольного бэк-энда, то есть собственно dpkg, я в GDebi не усмотрел — кроме разве что наглядности. Для установки большого количества пакетов она оказалась неудобной из-за необходимости авторизоваться на каждый такой чих — при использовании dpkg это можно решить один раз командой

```
$ sudo -i
```

А самая востребованная сфера применения GDebi — установка единичного, не отягощённого завимисостями, пакета на предмет «поиграться и стереть». Но в этом отношении ей нет равных…

Утилита apt. Реализация для Linux Mint

В данном очерке рассмотрены особенности утилиты apt в реализации для дистрибутива Linux Mint и её отличия от семейства утилит, входящих в пакет apt, общий для всех deb based дистрибутивов.

Введение

В процессе сочинения этой книги обнаружилось, что реализация утилиты apt для этого дистрибутива не документирована никак — не только на языке родных осин, но даже на мове Вильяма нашего, Шекспира. В связи с чем я и решил посвятить ей отдельный очерк.

Необходимость в таком материале, как мне кажется, ещё и в том, что многие начинающие пользователи Mint и особенно Linux вообще, судя по сайтам, блогам и форумам соответствующей тематики, даже не подозревают о существовании реализации apt для Mint и её отличиях от тёзки из одноимённого пакета. И потому механически применяют рецепты для чистой Ubuntu и её прямых клонов, на которые так богаты указанные ресурсы. Хотя использование apt для Mint делает эти рецепты излишними — функционал этой утилиты позволяет добиться той же цели быстрей и проще. По крайней мере, путём меньшего количества нажатий на клавиши.

Общее описание

Утилиту apt в реализации для Mint не следует путать ни с одноимённым пакетом, входящим в состав всех deb based дистрибутивов (в том числе и в Mint), ни с одноимённой же утилитой из этого пакета.

Утилита apt для Mint входит в состав пакета mintsystem, что определяется с

помощью её же самой:

```
$ apt contains /usr/local/bin/apt

mintsystem: /usr/local/bin/apt
```

В отличие от стандартной утилиты apt, располагающейся в каталоге /usr/bin, apt для Mint находится в каталоге /usr/local/bin, что определяется такой командой:

```
$ which apt

/usr/local/bin/apt
```

При вводе в командной строке apt без указания пути вызывается именно она, что определяется значениями переменной PATH, определёнными в общесистемном конфиге /etc/login.defs:

```
$ echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

Так что для запуска стандартной утилиты apt из одноимённого пакета для неё следует указывать полный путь, например

```
$ /usr/bin/apt list --installed
```

для вывода списка инсталлированных пакетов — это чуть ли не единственная функция стандартного инструментария пакета apt, отсутствующая в apt для Mint. Ибо последняя перекрывает почти все возможности утилит apt-get и apt-cache, большинство возможностей командного режима aptitude, а также выполняет некоторые функции низкоуровневой утилиты dpkg.

Примечание. Утилита apt — не единственный компонент пакета mintsystem. Кроме неё, он включает ещё четыре утилиты (также располагающиеся в каталоге

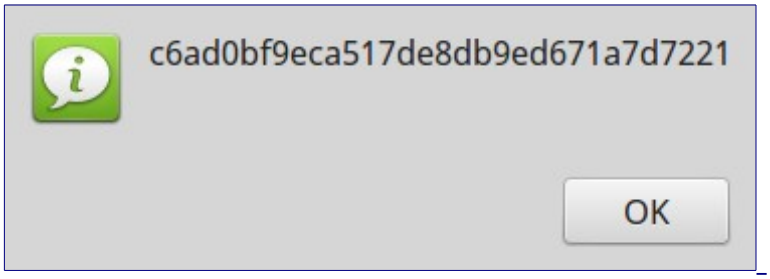
```
/usr/local/bin/search
```

): mint-md5sum, search, highlight и pastebin.

К управлению пакетами некоторое отношение имеет только первая из них, предназначенная для подсчёта контрольных сумм. Так, команда

```
$ mint-md5sum opera-stable_26.0.1656.60_amd64.deb
```

выведет её для пакета opera-stable в таком виде:



О команде search говорилось в очерке про утилиты CLI. А об остальных двух для полноты картины скажу здесь же.

Утилита pastebin предназначена для быстрого размещёния в Сети фрагментов текста, которые почему-либо нежелательно делать доступными каким-либо иным образом. Делается это через сервис, предоставляемый проектом Mint. Так, командная конструкция

```
$ echo 'Утилита pastebin предназначена для быстрого размещёния в Сети' | pastebin
```

даст ткакой вывод:

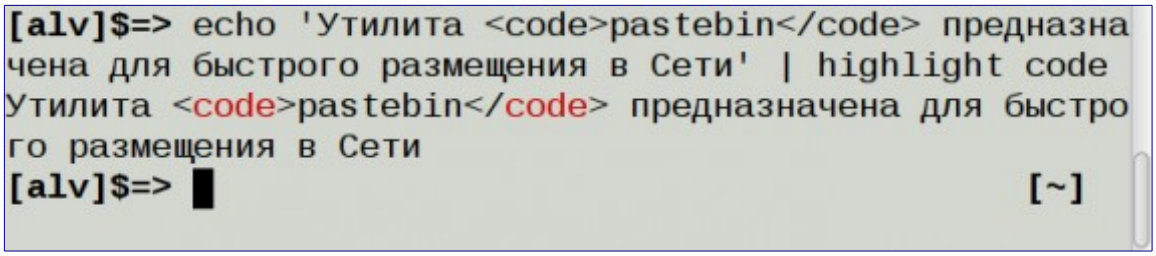
```
http://paste.linuxmint.com/view/u5i0
```

То есть введённый фрагмент будет доступен по указанному в выводже адресу (например, через браузер). Правда, русскоязычный текст по умолчанию окажется там в кодировке ISO 8859-5, так что надо озаботься тем, чтобы браузер поддерживал перекодирование страницы на лету.

Ну а утилита highlight обеспечивает подсветку произвольного текстового фрагмента, заданного как её аргумент. Например, командная конструкция

```
$ echo 'Утилита pastebin предназначена для быстрого размещёния в Сети' | highlight code
```

на выходе даст подсвеченным фрагмент code:



Теоретически рассуждая, если вывод этой конструкции передать по конвейеру команде pastebin, то и в Сети соответствующий фроагмент будет размещён в «подсвеченном» виде. Однако эксперимент показал, что сервис проекта Mint этого не поддерживает.

Применение

Утилита apt для Mint запускается одноимённой командой CLI с указанием внутренней команды, определяющей цель действия и, в большинстве случаев, аргумента (аргументов), в качестве которых выступает имя пакетов (или имена — их может быть сколько угодно):

```
$ apt command pkgname1 ... pkgname#
```

Некоторые часто используемые внутренние команды apt аргументов не требуют.

Полный список внутренних команд apt для Mint можно получить «голой» командой

```
$ apt
```

вывод которой выглядит следующим образом:

```
apt

Usage: apt command [options]

       apt help command [options]



Commands:

autoclean       - Erase old downloaded archive files

autoremove      - Remove automatically all unused packages

build           - Build binary or source packages from sources

build-dep       - Configure build-dependencies for source packages

changelog       - View a package's changelog

check           - Verify that there are no broken dependencies

clean           - Erase downloaded archive files

contains        - List packages containing a file

content         - List files contained in a package

deb             - Install a .deb package

depends         - Show raw dependency information for a package

dist-upgrade    - Perform an upgrade, possibly installing and removing packages

download        - Download the .deb file for a package

dselect-upgrade - Follow dselect selections

held            - List all held packages

help            - Show help for a command

hold            - Hold a package
```

```
install         - Install/upgrade packages

policy          - Show policy settings

purge           - Remove packages and their configuration files

rdepends        - Show reverse dependency information for a package

reinstall       - Download and (possibly) reinstall a currently installed package

remove          - Remove packages

search          - Search for a package by name and/or expression

show            - Display detailed information about a package

source          - Download source archives

sources         - Edit /etc/apt/sources.list with nano

unhold          - Unhold a package

update          - Download lists of new/upgradable packages

upgrade         - Perform a safe upgrade

version         - Show the installed version of a package

                    This apt has Super Cow Powers
```

Здесь для начала следует сказать о внутренних командах version и help. Первая теоретически должны выводить номер текущей версии apt для Mint, но практически не выводит ничего — лишь пустую строку. Команда же help без аргументов выведет список внутренних команд, идентичный приведённому выше. При указании аргумента — любой из внутренних команд она выведет её эквиваленты для apt-cache, apt-get или dpkg. Например:

```
$ apt help search

"apt search" is equivalent to "aptitude search"

$ apt help install

"apt install" is equivalent to "sudo apt-get install"

$ apt help deb

"apt deb" is equivalent to "sudo dpkg -i"
```

Внутренние команды apt для Mint можно разделить на три группы, которые предназначены для:

1. получения информации о пакетах;
2. установки и удаления отдельных бинарных пакетов;

3. общего обновления системы

4. работы с пакетами исходных текстов.

Команды первой группы могут быть выполнены обычным пользователем, второй и третьей — требуют прав администратора. Однако для получения их утилита apt для Mint не нуждается в команде sudo, данной явным образом: она автоматически вызывается при попытке исполнения соответствующих внутренних команд. Например:

```
$ apt install geany

[sudo] password for alv:
```

Тем не менее, внутренние команды apt для Mint целесообразно рассмотреть по трём указанным группам.

Информация о пакетах

Пакетный менеджмент начинается с поиска нужного пакета, для чего предназначена внутренняя команда search, требующая аргумента в виде ключевого слова. Поиск по ключевому слову осуществляется в именах пакетов и их кратких описаниях (т.н. резюме). Например, команда

```
$ apt search geany
```

отыщет одноимённый пакет для установки этого текстового редактора (называемого, однако, «Небольшой и быстрой IDE») и все его плагины:

```
p   geany                    - Небольшая и быстрая IDE

v   geany-abi-69             -

v   geany-api-216            -

p   geany-common             - Небольшая и быстрая IDE — общие файлы

p   geany-plugin-addons      - Различные дополнительные модули для Geany

p   geany-plugin-codenav     - Модуль навигации по коду для Geany

...

p   geany-plugin-xmlsnippets - XMLSnippets plugin for Geany

p   geany-plugins            - Набор плагинов для Geany

p   geany-plugins-common     - Набор плагинов для Geany (переводы)
```

Важное отличие от аналога — команды apt-cache search: apt search показывает основной пакета (i — установленный, p — не установленный или «чисто» удалённый, и так далее) и дополнительный (A — автоматически установленный, h — с фиксированной версией, и так далее) статусы пакетов.

Внутренняя команда held позволяет отсортировать пакеты с фиксированной

версией, то есть те, которые не будут обновляться по команде apt upgrade (о ней буде сказано в следующем разделе).

Подробную информацию об отдельном пакете можно получить с помощью внутренней команды show. Например,

```
$ apt show geany
```

выведет следующее:

```
Пакет: geany

Состояние: не установлен

Версия: 1.23.1+dfsg-1

Приоритет: необязательный

Раздел: universe/devel

Сопровождающий: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

Архитектура: amd64

Размер в распакованном виде: 2671 k

Зависимости: libc6 (>= 2.15), libcairo2 (>= 1.6.0), libgcc1 (>= 1:4.1.1),
                libgdk-pixbuf2.0-0 (>= 2.22.0), libglib2.0-0 (>=
                2.35.9), libgtk2.0-0 (>= 2.22.0), libpango1.0-0 (>=
                1.18.0), libstdc++6 (>= 4.1.1), geany-common (=
                1.23.1+dfsg-1)

Пред-зависимости: multiarch-support

Предлагает: libvte9, doc-base

Конфликтует: geany

Повреждает: geany-plugins-common (< 0.21), geany-plugins-common (< 0.21)

Предоставляет: geany-abi-69, geany-api-216

Описание: Небольшая и быстрая IDE

 Geany — нетребовательная к ресурсам интегрированная среда разработки программ,

 маленькая и быстрая, с небольшим количеством зависимостей от других пакетов.
```

использует только GTK2, поэтому для запуска Geany необходимы только

runtime-библиотеки GTK2.

The basic features of Geany are:

* syntax highlighting

* code completion

* auto completion of constructs like if, for and while, XML and HTML

* call tips

* folding

* many supported filetypes like C, Java, PHP, HTML, Python, Perl, Pascal

* symbol lists

* embedded terminal emulation

Сайт: http://www.geany.org

А сведения о смене версий пакета получаются с помощью внутренней команды changelog. Для Geany это выглядит так:

geany (1.23.1+dfsg-1) unstable; urgency=low

* [3b1ced4] Imported Upstream version 1.23.1+dfsg

* [b418909] Update debian-branch in gbp.conf

— Chow Loong Jin <hyperair@debian.org> Mon, 20 May 2013 00:18:56 +0800

geany (1.23+dfsg-2) unstable; urgency=low

* Upload to unstable, fixes FTBFS (Closes: #707368)

* [a472a80] Enable parallel builds

* [17a6378] No-change bump of Standards-Version to 3.9.4

* [ea78f31] Add README.source describing git branch structure

```
 — Chow Loong Jin <hyperair@debian.org>  Fri, 10 May 2013 15:27:35 +0800

...
```

И так далее.

Более подробные, нежели вывод команды show, сведения о зависимостях пакета даёт пара внутренних команд depends и rdepends. Первая выводит полный список пакетов, от которых зависит заданный в качестве её аргумента — жёстких, рекомендуемых, предлагаемых и конфликтующих:

```
$ apt depends geany

geany

  Зависит: libc6

  Зависит: libcairo2

  Зависит: libgcc1

  Зависит: libgdk-pixbuf2.0-0

  Зависит: libglib2.0-0

  Зависит: libgtk2.0-0

  Зависит: libpango1.0-0

  Зависит: libstdc++6

  Зависит: geany-common

  ПредЗависит: multiarch-support

    multiarch-support:i386

  Предлагает: libvte9

  Предлагает: doc-base

  Ломает: geany-plugins-common

  Ломает: <geany-plugins-common:i386>

  Конфликтует: geany:i386
```

Команда же rdepends решает обратную задачу — выводит список пакетов, зависящих от данного:

```
$ apt depends geany

geany
```

Reverse Depends:

geany:i386

geany-plugins-common

geany-plugins

geany-plugin-xmlsnippets

geany-plugin-webhelper

geany-plugin-vc

geany-plugin-updatechecker

geany-plugin-treebrowser

geany-plugin-tableconvert

geany-plugin-spellcheck

geany-plugin-shiftcolumn

geany-plugin-sendmail

geany-plugin-scope

geany-plugin-prj

geany-plugin-prettyprinter

geany-plugin-pg

geany-plugin-numberedbookmarks

geany-plugin-multiterm

geany-plugin-miniscript

geany-plugin-markdown

geany-plugin-macro

geany-plugin-lua

geany-plugin-lipsum

geany-plugin-latex

geany-plugin-insertnum

geany-plugin-gproject

geany-plugin-geniuspaste

```
  geany-plugin-gendoc

  geany-plugin-extrasel

  geany-plugin-doc

  geany-plugin-devhelp

  geany-plugin-debugger

  geany-plugin-commander

  geany-plugin-codenav

  geany-plugin-addons

  geany-common

  geany-common

 |deb-gview
```

Все приведённые выше внутренние команды дают информацию как об установленных пакетах, так и о пакетах, доступных в подключённых репозиториях. А вот команды contains и content работают только для установленных пакетов. Первая позволяет определить, к какому пакету принадлежит данный файл — именно таким способом была определена выше принадлежность утитлиты apt:

```
$ apt contains /usr/local/bin/apt

mintsystem: /usr/local/bin/apt
```

А команда content выводит список всех файлов пакета с указанием их положения в файловой иерархии:

```
$ apt content mintsystem

/.

/etc

/etc/apt

/etc/apt/preferences.d

/etc/apt/preferences.d/official-extra-repositories.pref

/etc/bash_completion.d

/etc/bash_completion.d/apt-linux-mint

/etc/init.d

/etc/init.d/mintsystem
```

```
...

/usr/share/nemo

/usr/share/nemo/actions

/usr/share/nemo/actions/mint-md5sum.nemo_action
```

Наконец, последняя из «информационных» команд — policy. При указании в качестве аргумента имени установленного пакета она выводит такую о нём информацию:

```
$ apt policy mintsystem

mintsystem:

 Установлен: 7.9.7

 Кандидат:   7.9.7

 Таблица версий:

*** 7.9.7 0

       700 http://linux-mint.froonix.org/ rebecca/main amd64 Packages

       100 /var/lib/dpkg/status
```

А для пакета не установленного она будет такой:

```
$ apt policy geany

geany:

 Установлен: (отсутствует)

 Кандидат:   1.23.1+dfsg-1

 Таблица версий:

    1.23.1+dfsg-1 0

       500 http://gd.tuwien.ac.at/opsys/linux/ubuntu/archive/ trusty/universe amd64 Packages
```

Где числе перед URL — приоритет репозитория, в который входит пакет, оно берётся из файлов каталога /etc/apt/preferences.d. Большее число соовтетствует более высокому приоритету.

Внутренняя команда policy была придумана для утилиты apt-cache дистрибутива Debian, где использовалась для управления приоритетами при совмещёнии в одной системе пакетов из его многочисленных веток — stable, testing, unstable, experimental. Не уверен, что она востребована в дистрибутиве Mint.

Работа с бинарными пакетами

Главное действие в отношении пакетов, которые были сочтены полезными — их установка. А основным инструментом установки является внутренняя команда install. В качестве аргументов она принимает имена пакетов — те самые, которые были найдены командой apt search и в полезности которых можно было убедиться командой apt show. Например, для установки чрезвычайно полезного текстового редактора Geany следует дать команду

```
$ apt install geany
```

которая сначала запросит пароль пользователя с административным типом аккаунта:

```
[sudo] password for alv:
```

А затем, после считывания локального списка пакетов и построения дерева зависимостей, сообщит о необходимости таковых, объёме скачиваемых пакетов и увеличении занятого дискового пространства после установки, запросив подтверждение серьёзности намерений:

```
Чтение списков пакетов... Готово

Построение дерева зависимостей

Чтение информации о состоянии... Готово

Будут установлены следующие дополнительные пакеты:

  geany-common

НОВЫЕ пакеты, которые будут установлены:

  geany geany-common

обновлено 0, установлено 2 новых пакетов, для удаления отмечено 0 пакетов, и 37 пакетов не обновлено.

Необходимо скачать 3808 kБ архивов.

После данной операции, объём занятого дискового пространства возрастёт на 9872 kB.

Хотите продолжить? [Д/н]
```

Согласие предполагается по умолчанию, так что тут достаточно нажать Enter. После чего начинается скачивание пакетов из содержащего их репозитория, распаковка и инкорпорация компонентов в файловую иерархию, а также регистрация в базе данных и включение, если требуется, исполняемого файла в главное меню (для Geany — в секцимю Прграммирование, так как эта программа позиционируется её авторами как IDE — Integrated Development Environment, то есть интегрированная среда разработки). Основной статус пакета geany изменится на «установленный»:

```
$ apt search geany | head -n 1
```

```
i   geany          - Небольшая и быстрая IDE
```

А пакет geany-common приобретёт ещё и статус автоматически установленного:

```
$ apt search geany-common

i A geany-common    - Небольшая и быстрая IDE — общие файлы
```

Если в системе уже был установлен данный пакет более старой версии — он будет обновлён. А вот переустановить пакет той же версии (например, если он был безнадёжно испорчен в ходе экспериментов) команда install откажется, сообщив, что

```
Уже установлена самая новая версия geany.

обновлено 0, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 37 пакетов не обновлено.
```

Однако на этот предмет существует специальная команда reinstall, аргументом которой указывается установленный пакет, нуждающийся в исправлении.

Локально отдельные пакеты могут быть установлены с помощью внутренней команды deb, аргументом которой должно быть полное имя файла пакета, если нужно, с указанием пути. Например, команда

```
$ apt deb sublime-text_3065_amd64.deb
```

установит текстовый редактор Sublime — разумеется, предварительно файл этого пакета должен быть скачан.

Поскольку внутренняя команда deb — полный эквивалент конструкции sudo dpkg -i, она не занимается разрешением зависимостей, а только сообщает об их нарушении. Например, попытка установить в окружении Cinnamon файловый менджер Caja из среды MATE вызовет следующие сообщения:

```
$ apt deb caja_1.8.2-0+rebecca_amd64.deb

Выбор ранее не выбранного пакета caja.

(Чтение базы данных ... на данный момент установлен 188621 файл и каталог.)

Preparing to unpack caja_1.8.2-0+rebecca_amd64.deb ...

Unpacking caja (1.8.2-0+rebecca) ...

dpkg: зависимости пакетов не позволяют настроить пакет caja:

 caja зависит от libcaja-extension1 (= 1.8.2-0+rebecca), однако:

  Пакет libcaja-extension1 не установлен.

 caja зависит от libmate-desktop-2-17, однако:
```

```
 Пакет libmate-desktop-2-17 не установлен.

 caja зависит от mate-desktop, однако:

 Пакет mate-desktop не установлен.

 caja зависит от caja-common (= 1.8.2-0+rebecca), однако:

 Пакет caja-common не установлен.


dpkg: error processing package caja (--install):

 проблемы зависимостей — оставляем не настроенным

Processing triggers for man-db (2.6.7.1-1ubuntu1) ...

Processing triggers for mime-support (3.54ubuntu1.1) ...

Processing triggers for gnome-menus (3.10.1-0ubuntu2) ...

Processing triggers for desktop-file-utils (0.22-1ubuntu1) ...

Processing triggers for ubuntu-system-adjustments (2014.11.19) ...

При обработке следующих пакетов произошли ошибки:

 caja
```

В отличие от внутренней команды install, команда deb не только обновит пакет до более новой версии, но и переустановит его версию текущую.

Установленные пакеты иногда требуется и удалять. Этой цели в apt для Mint служат две внутренние команды — remove и purge, аргументами которых служат, очевидно, имена удаляемых пакетов. Первая удаляет файлы пакета, но сохраняет его общесистемные конфиги, вторая — удаляет также и их. Различие между ними отражается в основном статусе удалённого пакета — в первом случае его значение будет c, во втором — p, как и у пакетов, которые никогда не устанавливались.

И remove, и purge автоматически удаляют все зависимые пакеты, список их выводится после ввода пользовательского пароля:

```
$ apt purge libreoffice-impress

[sudo] password for alv:

Чтение списков пакетов… Готово

Построение дерева зависимостей

Чтение информации о состоянии… Готово

Пакеты, которые будут УДАЛЕНЫ:
```

```
  libreoffice-impress* libreoffice-ogltrans*

  libreoffice-presentation-minimizer*

обновлено 0, установлено 0 новых пакетов, для удаления отмечено 3 пакетов, и 5 пакетов не обновлено.

После данной операции, объём занятого дискового пространства уменьшится на 6031 kB.

Хотите продолжить? [Д/н]
```

Список удаляемых пакетов нужно читать очень внимательно, чтобы случайно не удалить что-нибудь жизненно необходимое.

Пакеты, от которых зависит удаляемый, автоматически не удаляются ни remove, ни purge. В этом случае apt предлагает воспользоваться внутренней командой autoremove для очистки системы от «осиротелых» зависимостей:

```
$ apt purge geany

Чтение списков пакетов… Готово

Построение дерева зависимостей

Чтение информации о состоянии… Готово

Следующий пакет устанавливался автоматически и больше не требуется:

  geany-common

Для его удаления используйте «apt-get autoremove».

Пакеты, которые будут УДАЛЕНЫ:

  geany*

обновлено 0, установлено 0 новых пакетов, для удаления отмечено 1 пакетов, и 5 пакетов не обновлено.

После данной операции, объём занятого дискового пространства уменьшится на 2671 kB.

Хотите продолжить? [Д/н]
```

Разумеется, в нашем случае мы обращаемся не к команде apt-get, всё в той же утилите apt для Mint:

```
$ apt autoremove
```

Она не нуждается в аргументах и выполняет свою работу молча, не задавая вопросов. Перед её выполнением не вредно выполнить другую внутреннюю команду — check, проверяющую систему на предмет «сломанных» зависимостей.

```
apt check
```

```
[~]
```

Что при хорошем раскладе после ввода пароля должно дать такой результата:

```
Чтение списков пакетов... Готово
```

```
Построение дерева зависимостей
```

```
Чтение информации о состоянии... Готово
```

А при плохом... плохого у меня до сих пор не было.

Перед установкой пакетов из репозитория они предварительно скачиваются и помещаются в каталог /var/cache/apt/archives/. Со временем файлов пакетов накапливается много, а нужны они бывают только в исключительных случаях. Для избавления от них существуют в apt для Mint предусмотрены команды autoclean и clean. Первая удаляет из кеша только пакеты устаревших версий, сохраняя те, версии которых установлены в системе. Вторая же полностью очищает каталог /var/cache/apt/archives/.

Обновление системы

Сказанное выше касалось единичных пакетов или их серий — любая из перечисленных субкоманд принимает любое количество аргументов. Однако в утилите apt предусмотрены и внутренние команды для общего обновления пакетов, а также для тотального системы. Однако, прежде чем выполнить любую из них, необходимо провести обновление локального кеша пакетов, то есть получить списки новых и обновлённых пакетов. Делается это внутренней командой update:

```
$ apt update
```

Её же в обязательном порядке следует выполнять после каждого изменения в репозиториях — подключения новых или отключения имевшихся. Теоретически для редактирования списков репозиториев в apt для Mint предназначена команда sources. Однако практически она бесполезна, так как вызывает текстовый редактор по умолчанию (nano) для редактирования /etc/apt/sources.list. В нашем же дистрибутиве этот файл содержит только репозиторий локального оптического диска, а все реально подключённые репозитории описываются в файлах каталога /etc/apt/sources.list.d.

Для обновления всех, по возможности, пакетов установленной системы в apt для Mint существует внутренняя команда upgrade. Она выявит все пакеты, для которых в репозиториях доступны более свежие версии, выведет их список, объём для скачивания и прирост объёма занятого дискового пространства после выполнения процедуры, а также запросит подтвержения:

```
$ apt upgrade
```

```
Чтение списков пакетов... Готово
```

```
Построение дерева зависимостей
```

```
Чтение информации о состоянии... Готово
```

```
Расчёт обновлений...Готово

Пакеты, которые будут обновлены:

  gir1.2-gudev-1.0 libegl1-mesa libegl1-mesa-drivers libgbm1

...

обновлено 35, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 0 пакетов не обновлено.

Необходимо скачать 36,3 МБ архивов.

После данной операции, объём занятого дискового пространства возрастёт на 124 kB.

Хотите продолжить? [Д/н]
```

В ходе выполнения upgrade обновляются по возможности все пакеты, за исключением тех, для разрешения зависимостей которых обновление потребует доустановки новых пакетов или удаления существующих. Для таких пакетов текущие версии будут сохранены.

При использовании команды upgrade следует учитывать, что она обновляет в том числе и те компоненты, которые по умолчанию заблокированы для обновления через фирменный инструмент mintupdate — ядро и всё, что с ним связано, glibc, и так далее (пакеты уровней 4 и 5). Так что, прежде чем применять эту команду, следует либо всё взвесить и решиться на обновление указанных компонентов, либо явным образом зафиксировать их версии.

Фиксация версий пакетов может потребоваться и в ряде других случаев — например, при использовании более неподдерживаемых, но по прежнему необходимых пакетов, пакетов, пересобранных с собственными опциями, и ещё некоторых. Она выполняется внутренней командой hold с указанием имени фиксируемого пакета (пакетов). После чего пакет приобретает основной статус h и не затрагивается обновлениями. Обратная процедура, то есть снятие фиксации, если в ней пропала необходимость, выполняется внутренней командой unhold.

Для тотального обновления системы предназначена внутренняя команда dist-upgrade: она не только обновляет все пакеты, для которых обновления доступны, но может доустанавливать новые пакеты и удалять существующие, если это необходимо для разрешения зависимостей. Эта субкоманда применяется, например, при смене релиза дистрибутива — например, для перехода с Mint 17.0 Qiana на 17.1 Rebecca достаточно прописать одноимённые репозитории в файлах их описаний. И после этого дать команду

```
$ apt dist-upgrade
```

Есть и ещё несколько случаев, требующих применения dist-upgrade, а не просто upgrade — например, обновление версии рабочей среды (в данном случае Cinnamon) и некоторых других базовых компонентов системы.

При тотальном обновлении через dist-upgrade следует помнить о том, что выше было сказано про upgrade. И если в данном случае обновление базовых компонентов системы (ядра etc.) необходимо, то о фиксации самосборных и

неподдерживаемых пакетов посредством команды hold нужно позаботиться заблаговременно.

В apt для Mint предусмотрена ещё одна внутренняя команда общего обновления — dselect-upgrade, эквивалентная конструкции sudo apt-get dselect-upgrade. Она выполняет обновление в соответствие со статусом пакетов, установленным по умолчанию для древней утилиты dselect — предшественницы aptitude. Поскольку самой этой утилиты в стандартной инсталляции Mint нет, изменить (и даже посмотреть эти умолчания затруднительно). Так что я бы воздержался от использования dselect-upgrade, тем более что ни малейшей необходимости обращаться к ней нет.

Работа с пакетами исходных текстов

Всё сказанное выше относилось к бинарным пакетам. Однако в утилите apt предусмотрены и средства для работы с пакетами исходных текстов. Так, с помощью внтуренней команды source можно просто скачать пакет, указанный в качестве её аргумента — разумеется, для этого должен быть подключён репозиторий исходников. Внутренняя команда build (эквивалент sudo dpkg-buildpackage) выполнит построение бинарного пакета (что требует соответствующего инструментария в установленном виде). А внутренняя команда build-dep ограничится конфигурированием необходимых для этого зависимостей, например:

```
apt build-dep ubuntu-zfs
```

Это потребовалось мне в Rebecca 17.1 при сборке пакетов поддержки ZFS on Linux.

Итог

Можно видеть, что и по части манипулирования пакетами возможности утилиты apt широки и многогранны. То есть это действительно универсальное средство управления пакетами, в обыденной жизни способное почти всегда заменить все прочие — от низкоуровневой dpkg (обращение к которой потребуется только в исключительных случаях) до графического front-end'а — Synaptic'а. Ибо не уступает последнему в наглядности вывода информации о пакетах, позволяя манипулировать ими проще и быстрее. Рядом с apt для Mint его тёзка из одномиённого пакета (общего для всех deb based дистрибутивов) Debian/Ubuntu выглядит ограниченным функционально, а традиционные apt-cache и apt-get — несколько усложнёнными синтаксически. Что же до aptitude, то она в этом контексте кажется вообще излишней: apt для Mint обеспечивает почти все функции её командного режима, а в интерактивном режиме эта программа в дистрибутивах семейства Ubuntu и её клонах уже давно работает не вполне корректно.

Примечание: кратко об apt из APT

Если apt-cache и apt-get полностью заменяются утилитой apt для Mint, то, как ни странно, apt из одноимённого пакета имеет некоторые дополнительные функции, и потому заслуживает хоть и краткого, но рассмотрения. Как уже говорилось, в нашем дистрибутиве его следует запускать с указанием полного пути:

```
$ /usr/bin/apt
```

В приведённом виде эта команда выведет справку о внутренних командах этой утилиты — краткую, но вполне достаточную:

```
CLI for apt.

Basic commands:

 list - list packages based on package names

 search - search in package descriptions

 show - show package details


 update - update list of available packages


 install - install packages

 remove  - remove packages


 upgrade - upgrade the system by installing/upgrading packages

 full-upgrade - upgrade the system by removing/installing/upgrading packages


 edit-sources - edit the source information file
```

Назначение большинства внутренних команд понятно без комментариев, из их имён и сопуствующих пояснений: search, show, install, remove, update и upgrade суть аналоги одноимённых внутренних команд apt для Mint (а также apt-cache и apt-search), full-upgrade идентична команде dist-upgrade, а edit-sources — команде sources.

Так что единственной внутренней командой, не имеющей аналогов ни в связке apt-cache и apt-search, ни в apt для Mint, оказывается list. Но зато командой очень полезной:

- с опцией --installed она выводит список установленных пакетов (который иначе можно получить только командой dpkg -l или всякими конструкциями с grep);
- опция --upgradable выводит список пакетов, для которых имеются обновления;
- опция же --all-versions выдаёт на гора полный список доступных пакетов, специально отмечая установленные.

Тот же результат достигается командой

```
$ /usr/bin/apt list
```

без всяких опций. Так что внутренняя команда list в ряде случаев оказывается востребованной, почему я и придумал для неё специальный глобальный псевдоним в конфиге ~/.zshrc.

Управление пакетами: Synaptic

Система управления пакетами Synaptic — графический фронт-энд для утилит семейства APT, обычно используемыми для работы с пакетами deb-формата, а в некоторых дистрибутивах — и с пакетами rpm.

Введение

Как ни странно, Synaptic появился не в Debian, и вообще не в deb based системах: первые его версии были созданы в бразильском дистрибутиве Connectiva — том самом, разработчики которого впервые прикрутили apt-get для управления rpm-пакетами (под именем apt-rpm).Создателем Synaptic’а был Альфредо Кодзима (Alfredo Kojima), а позднее им занимался Густаво Нимейер (Gustavo Niemeyer), оба являвшиеся тогда, на рубеже тысячелетий, сотрудниками фирмы Connectiva. И именно и исключительно фронт-эндом к apt-rpm и выступал Synaptic в начальную пору своей жизни.

После покупки Connectiva фирмой Mandrakesof (в январе 2005 года) связка apt-rpm и Synaptic была благополучно похерена в недрах объединённой Mandriva — в пользу собственных инструментов, urpmi и rpmdrake. Однако сама идея оказалась очень продуктивной — и ещё в 2001 году Михаэль Фогт (Michael Vogt) «дебианизировал» Synaptic, приспособив его для работы с собственно deb-пакетами. Хотя Фогт и по сей день является основным майнтайнером upstream-версии пакета, среди пользователей Debian’а, насколько мне известно, он широкого распространения не получил — предпочтение здесь отдавалось сначала собственно apt-утилитам, а затем и поныне — aptitude.

Звёздный час Synaptic’а наступил с появлением в октябре 2004 года первой версии Ubuntu. Будучи основанным на библиотеке Gtk, он сразу и гармонично вписался в тогдашнее GNOME-окружение этого дистрибутива. А с возникновением в ноябре 2006 года Mint был включён в состав этого дистрибутива, чтобы с тех пор верой и правдой служить во всех его вариантах, включая даже KDE-редакцию. И, хотя здесь он «снизу» подпирается собственной, чрезвычайно функциональной реализацией apt, а «сверху» перекрывается Менеджером программ, в ряде случаев Synaptic оказывается самым эффективным средством для работы с пакетами.

Обзор

Как только что говорилось, Synaptic — это интегрирующая надстройка над утилитами семейства apt, но не над Mint'овской реализацией apt. Тем не менее, он предоставляет все функции, обеспечиваемые последней, а также командами apt-get и apt-cache. В их числе:

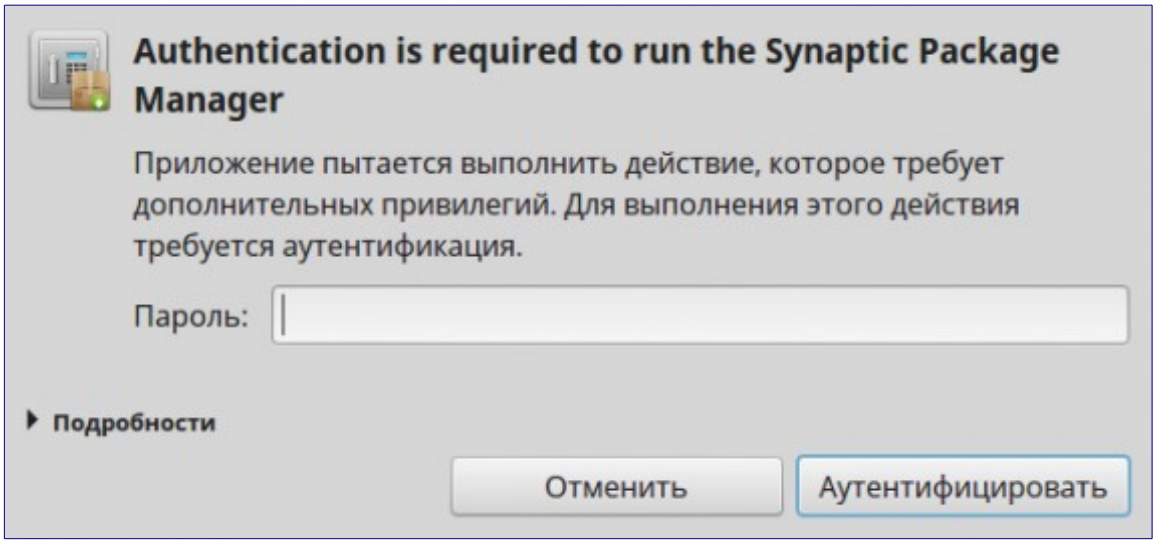
- поиск пакетов в репозиториях с определением их состояния и статуса;
- их установку и обновление с автоматическим разрешением зависимостей;

- удаление пакетов, в том числе и включая их зависимости;
- обновление базы данных пакетов из репозитория;
- тотальное обновление системы.

Кроме того, Synaptic включает средства настройки — в частности, доступа к репозиториям. В Mint для этой цели вызывается собственная утилита smintsource.

Запуск Synaptic’а выполняется через главное меню панели приложений (Администрирование -> Менеджер пакетов Synaptic) или любым другим традиционным для Mint способом.

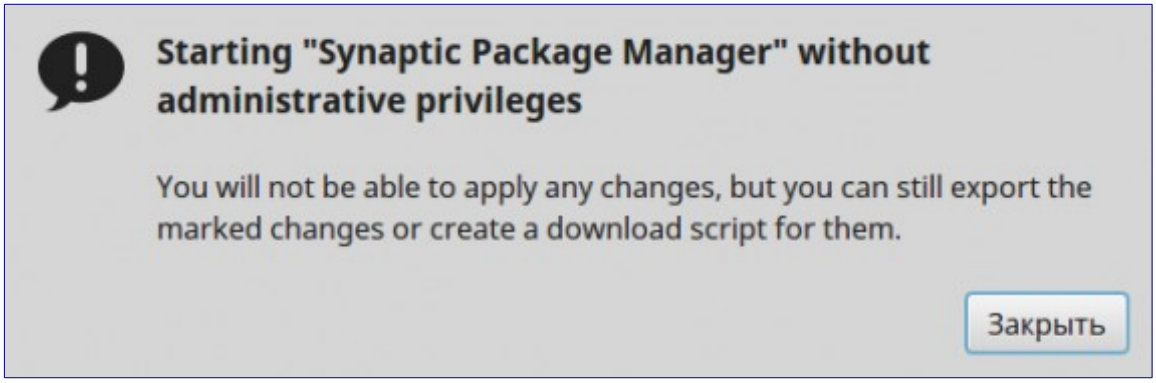
Очевидно, что установка и удаление пакетов потребует прав администратора, запрос на получение каковых (посредством механизма sudo, то есть с вводом пользовательского пароля) и последует после вызова Synaptic’а через меню:



Если отказаться от ввода пароля, то Synaptic таким способом запущен не будет. Однако его таки можно запустить и от лица обычного пользователя — из командной строки терминала или минитерминала прямой командой:

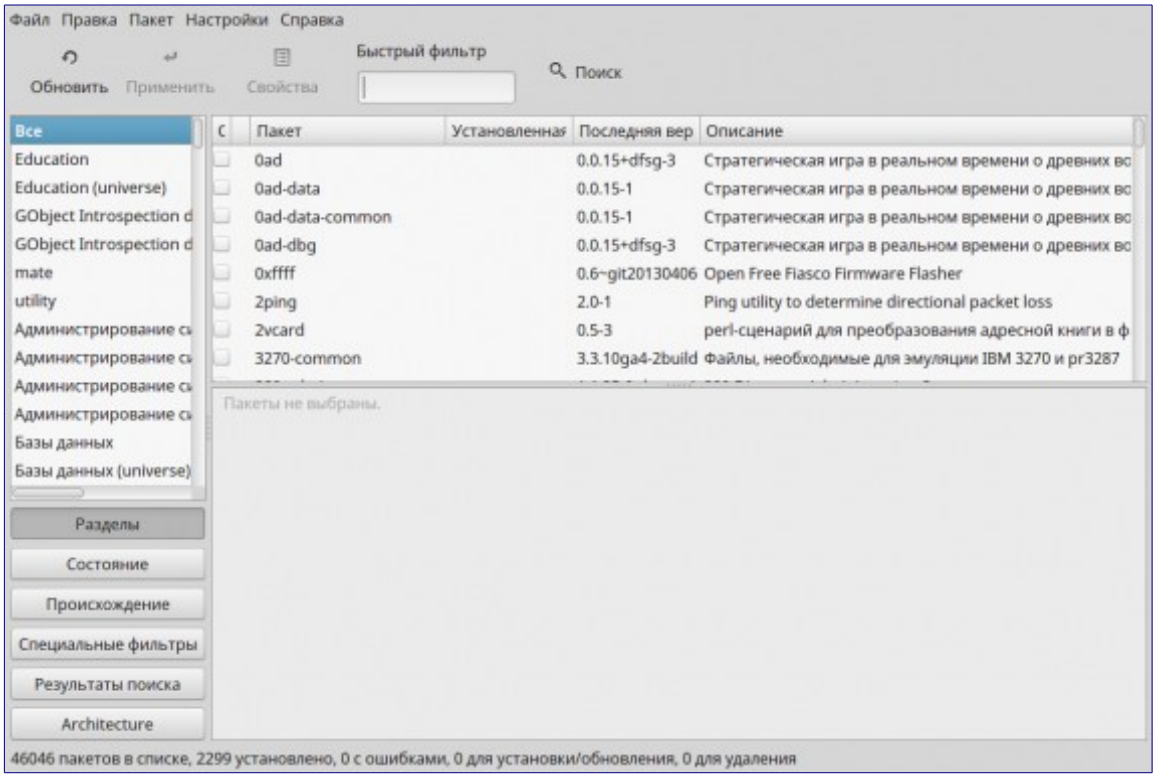
```
$ synaptic
```

В этом случае появится такое предупреждение:



Из которого явствует, что запущенный в пользовательском режиме Synaptic можно использовать для поиска пакетов и получения информации о них.Тем

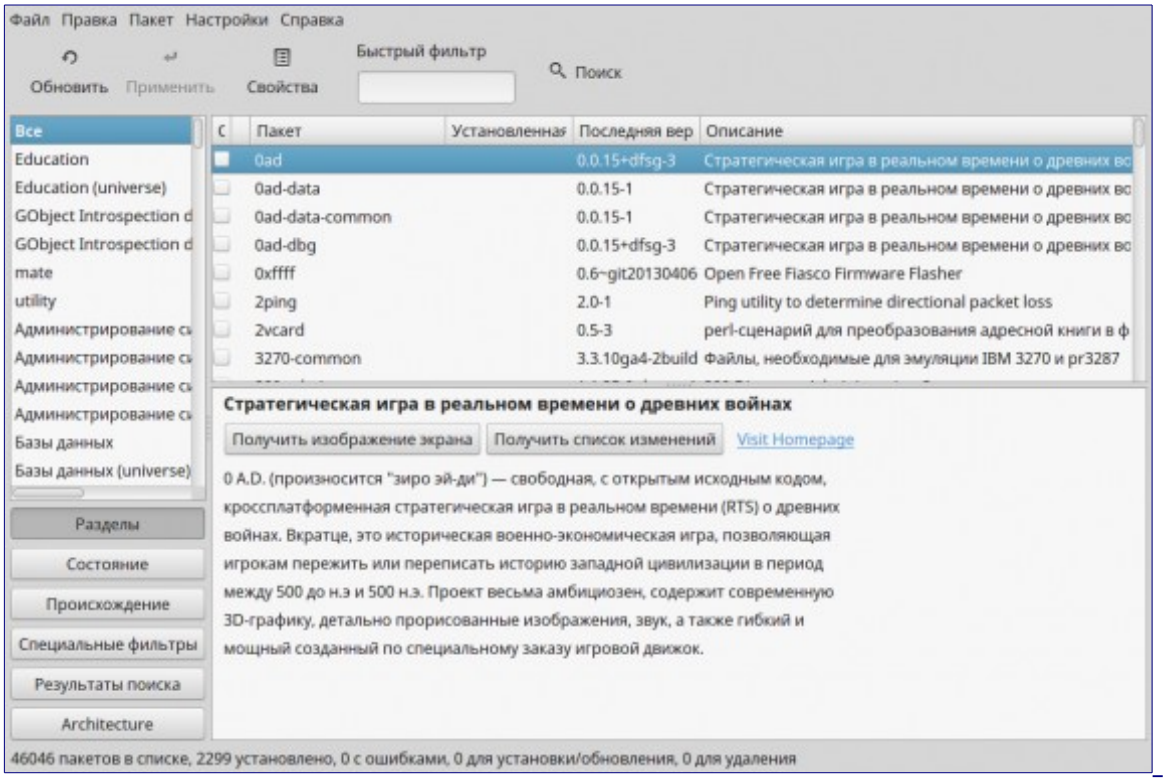
не менее, нормальный режим работы Synaptic’а — административный. И после ввода пароля пользователя (надо отметить, что по умолчанию во время появления панели для его ввода экран пригасает, а все управляющие элементы интерфейса блокируются) появляется окно примерно такого вида:



Как явствует из скриншота, в окне Synaptic’а мы имеем следующие основные элементы интерфейса:

- строку меню;
- панель инструментальных кнопок;
- два главных фрейма — список разделов репозитория и список пакетов выбранного раздела (по умолчанию показываются все пакеты);
- фрейм с кнопками выбора критериев для вывода пакетов;
- фрейм свойств конкретного пакета.

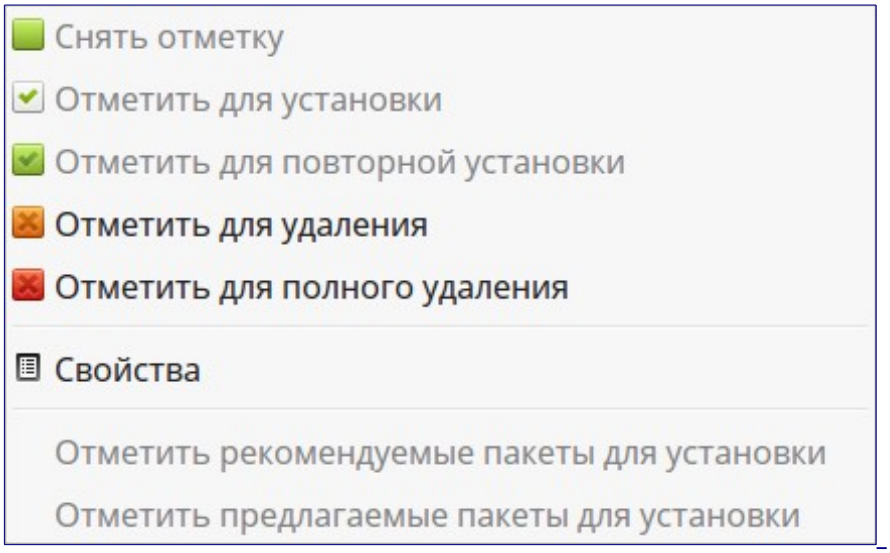
Последний фрейм пуст, если в правом главном фрейме не отмечен ни один пакет, как на предыдущем скриншоте. Но заполняется контентом при свершении выбора: в правом нижнем фрейме мы увидим описание пакета (если доступно, на русском)



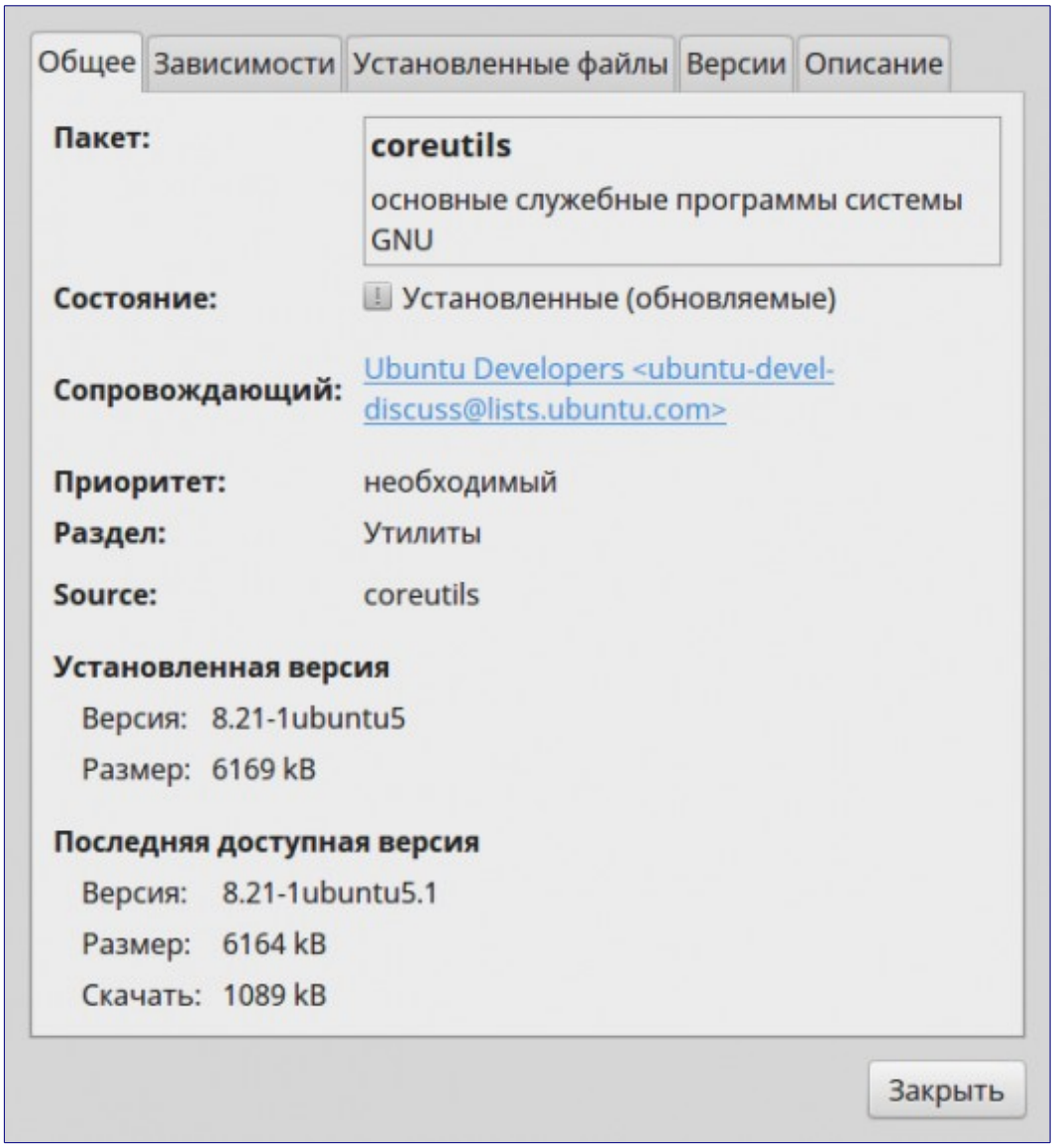
Если при этом нажать на кнопку Получить снимок экрана — то появится скриншот соответствующего пакета (буде таковой существует и имеет смысл):



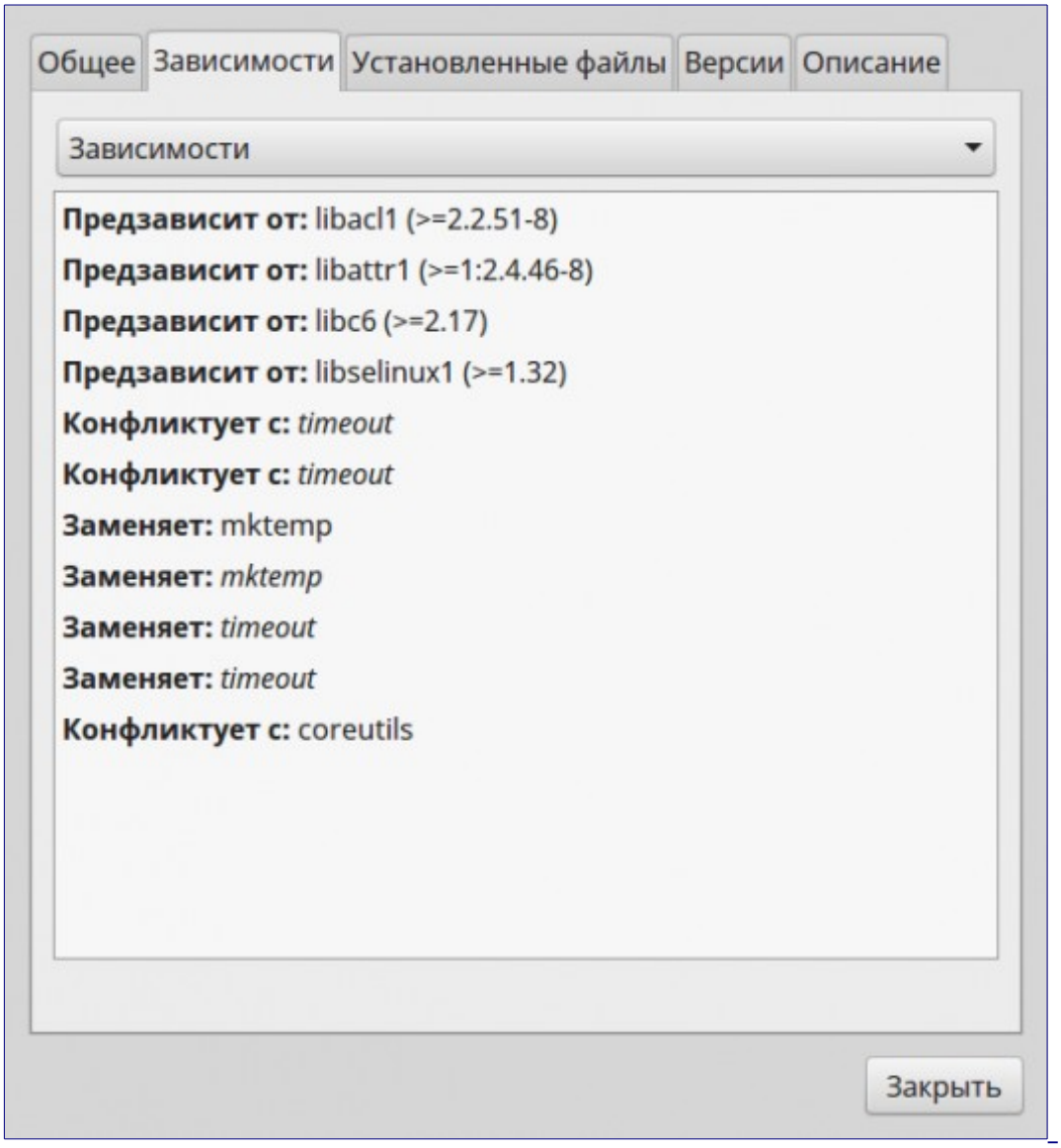
А при правом клике на имени пакета появляется контекстное меню:



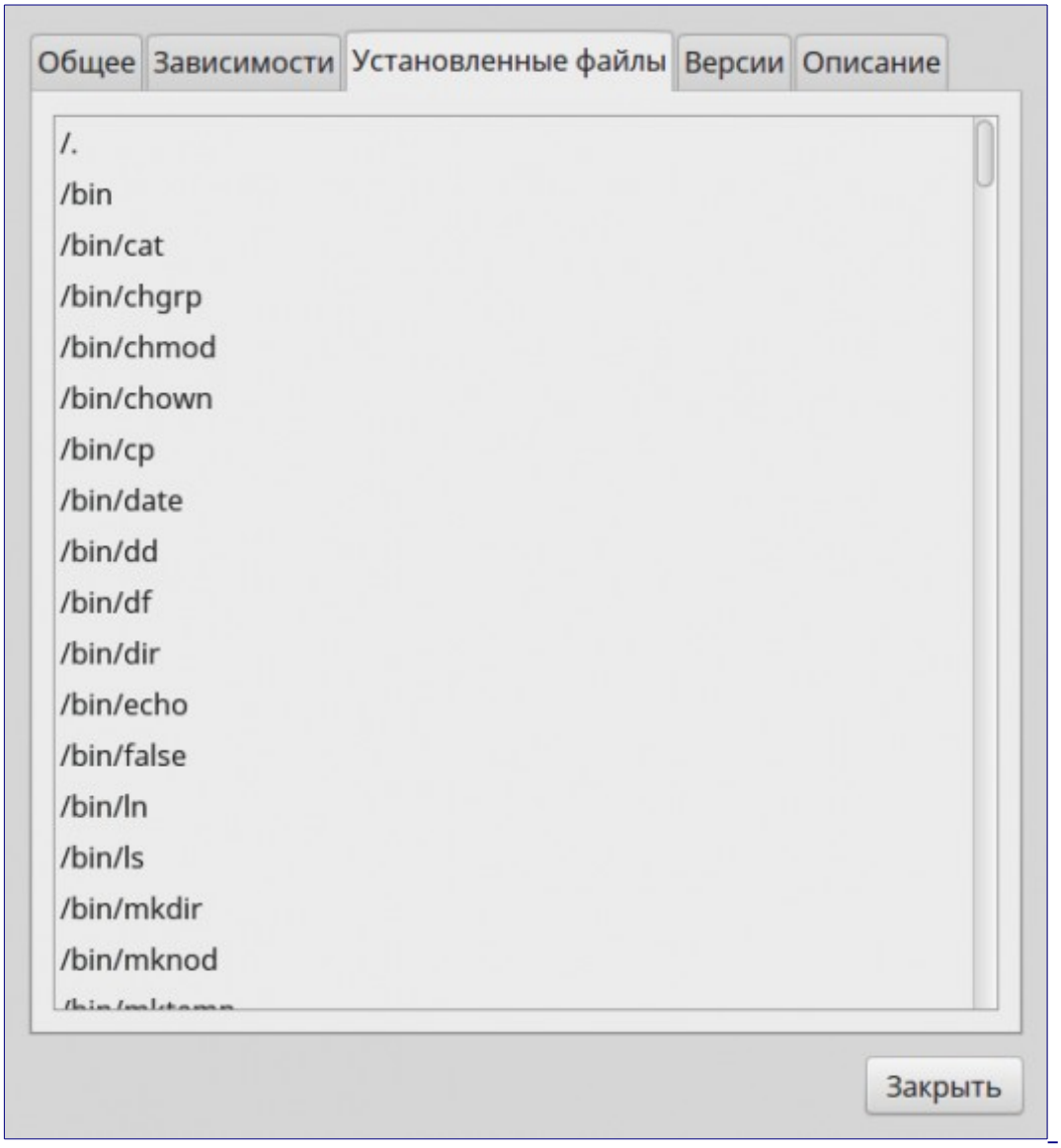
Здесь-то, в пункте Свойства, и содержится главная информация о пакете — общие сведения



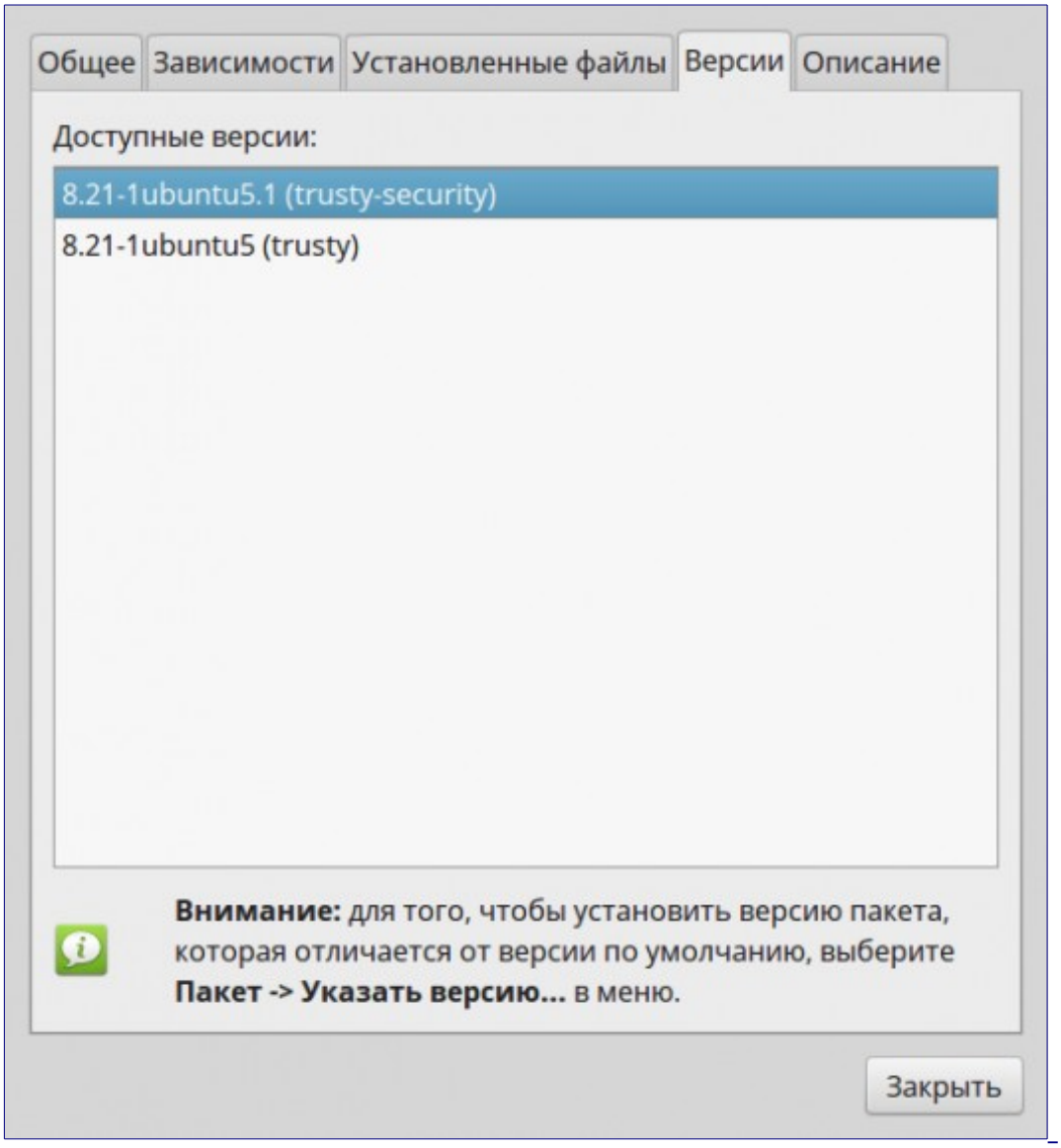
перечень зависимостей



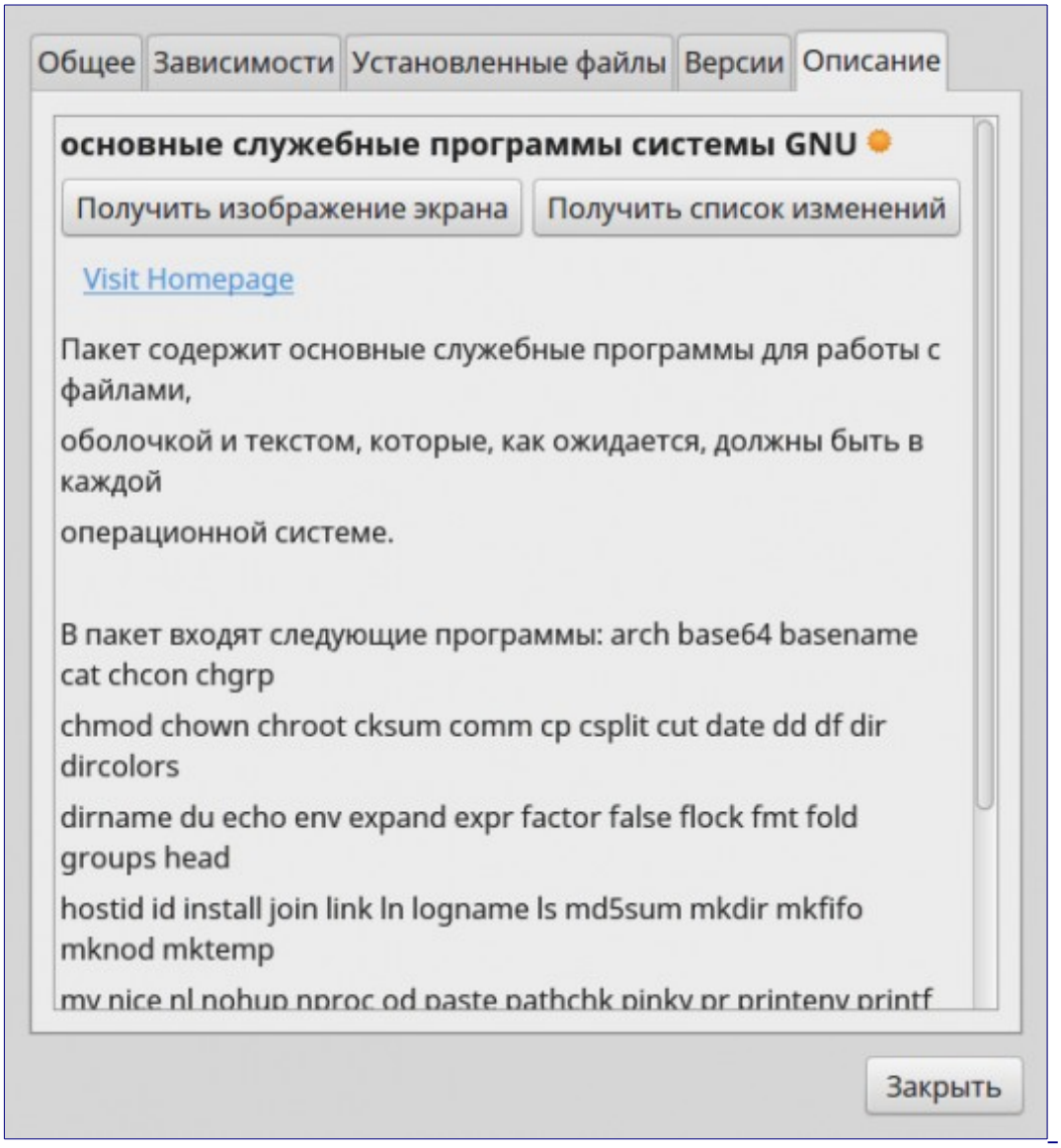
список установленных файлов и путей к ним (доступно только для установленных пакетов)



доступные версии

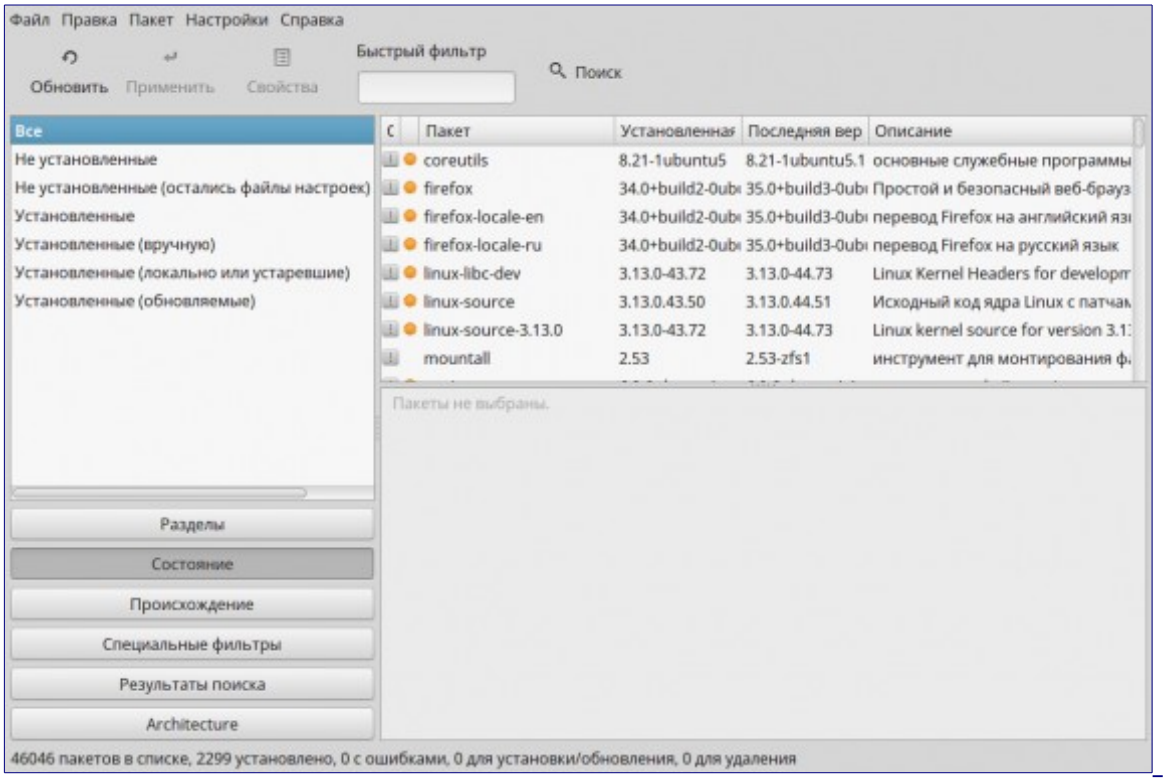


и, наконец, описание пакета

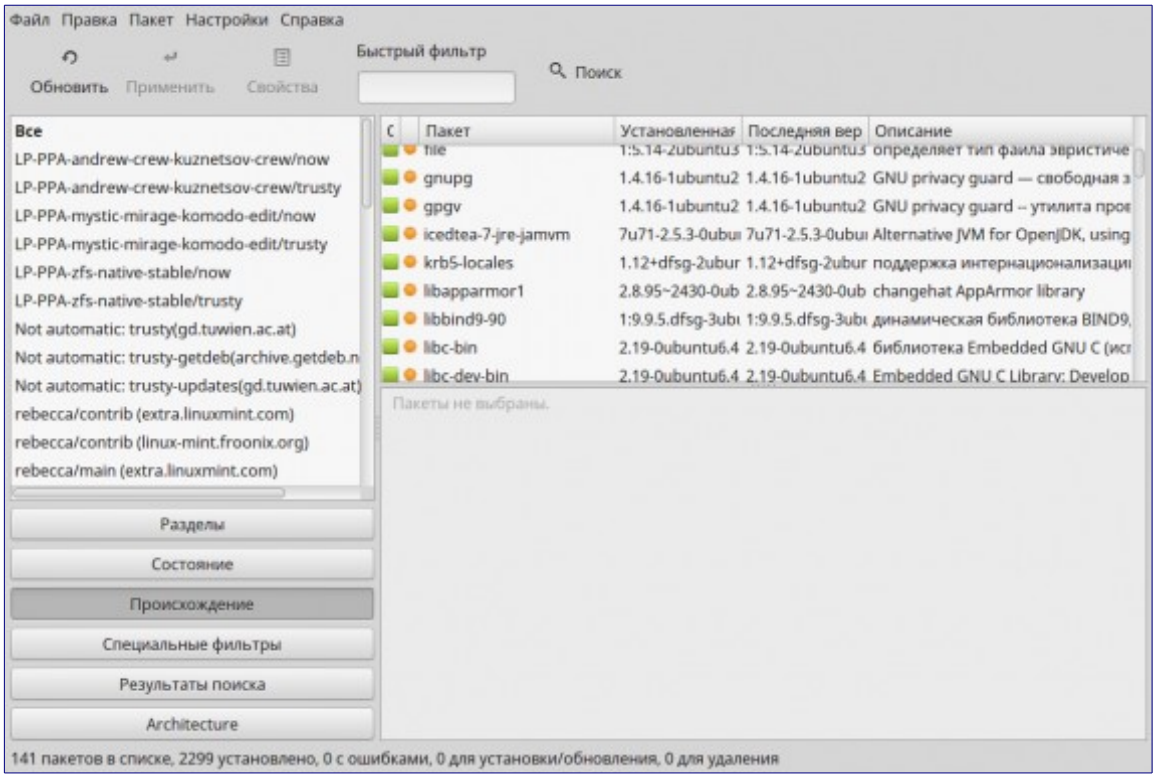


Теперь пробежимся по критериям вывода пакетов. С группировкой пакетов по разделам всё более-менее ясно, тем более, что названия разделов почти все даны в русском переводе, а те немногие, что оставлены в оригинале (например, World Wide Web), и без перевода понятны.Следующий критерий отбора — по состоянию пакетов. После нажатия соответствующей кнопки в левом главном фрейме выводятся следующие категории:

- Все;
- Не установленные;
- Не установленные (остались файлы настроек);
- Установленные;
- Установленные (вручную);
- Установленные (локально или устаревшие);
- Установленные (обновляемые).



По происхождению пакеты разделяются на установленные локально (то есть с инсталляционного носителя или предварительно скачанные на диск) и из различных репозиториев — rebecca, trusty, PPA и так далее:



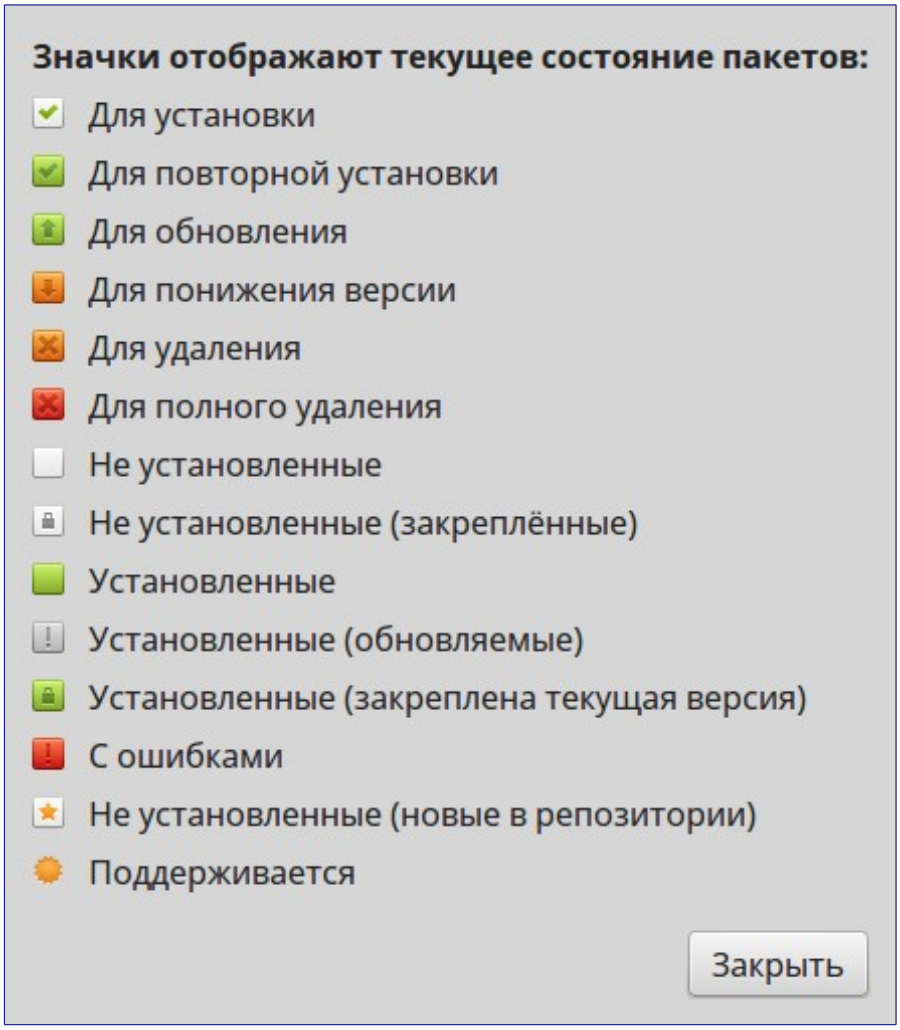
Про архитектуру всё понятно без комментариев — их нынче осталось всего ничего.

Что касается кнопок Специальные фильтры и Результаты поиска, то о них мы поговорим позднее.

А пока обратимся к спискам файлов, выводимых в правом главном фрейме. Если поглядеть на него внимательно, то слева можно увидеть две колонки

иконок, причём вторая может либо изображать нечто жёлтенькое, либо быть пустой. Факт наличия жёлтой иконки указывает, что данный пакет поддерживается официально разработчиками дистрибутива. А отсутствие пиктограммы во второй колонке говорит о том, что пакет либо поддерживается сообществом (точнее, некими конкретными его представителями), либо, в рамках дистрибутива, не поддерживается вообще.

Пиктограммы же первой колонки отражают статус пакет: установленный (зелёный квадратик), не установленный (квадратик не залитый), и так далее. Полную расшифровку значений пиктограмм можно получить через систему встроенной помощи: меню Справка -> Описание значков:



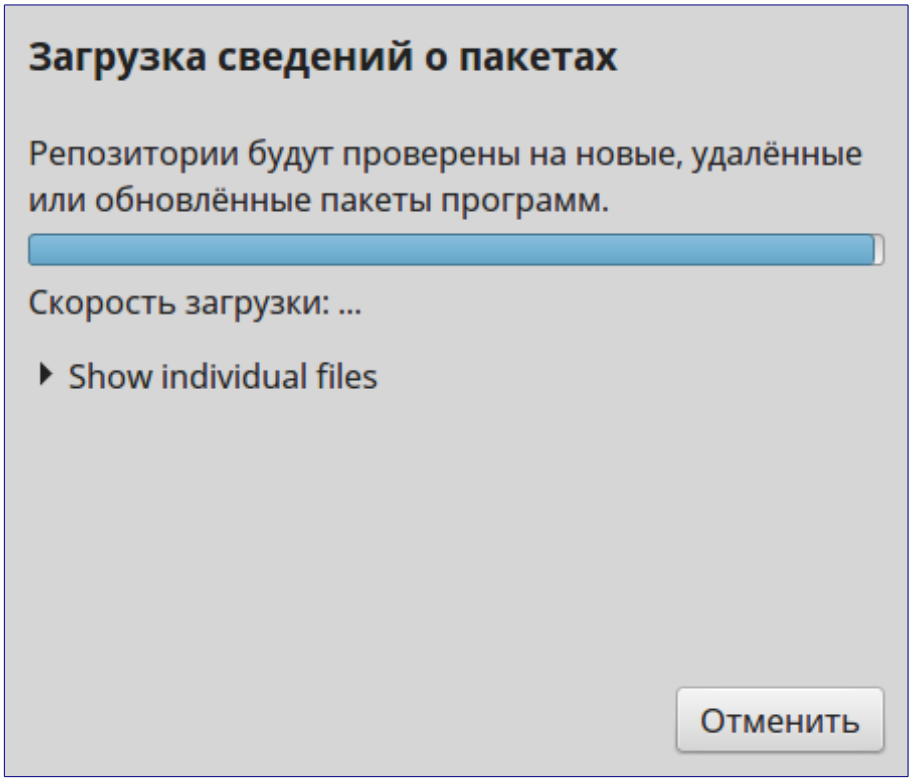
А теперь вернёмся к контекстному меню. Из приведённого скриншота можно видеть, что для установленного пакета активизированы пункты:

- Отметить для повторной установки — то есть реинсталляции, аналог команды apt reinstall;
- Отметить для удаления — удаление данного пакета, без конфигурационных файлов, аналог команды apt remove;
- Отметить для полного удаления — удаление данного пакета вместе с его конфигами, но не затрагивая зависимостей, аналог команды apt purge;
- свойства — его мы уже рассмотрели.

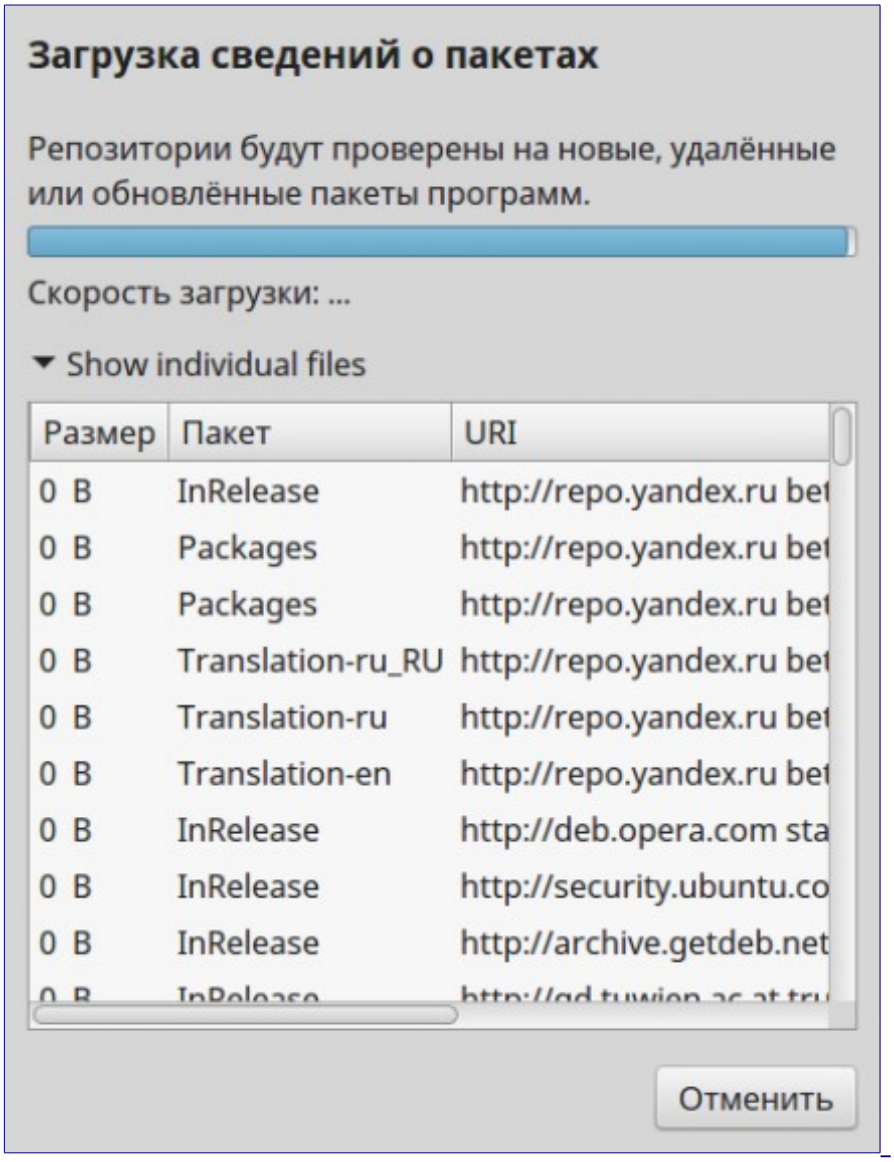
Для пакета не установленного по умолчанию доступны два пункта — Отметить для установки (аналог команды apt install) и всё те же Свойства. Не активизированы пункты Отметить для установки рекомендуемые (recommended) и предлагаемые (suggest) пакеты — они зависят от общих настроек Synaptic’a, и мы к ним ещё вернёмся.

Подчеркну, что все отметки через контекстное меню не влекут за собой немедленной установки, обновления или удаления пакетов — эти действия будут выполнены только после надатия кнопки Применить.

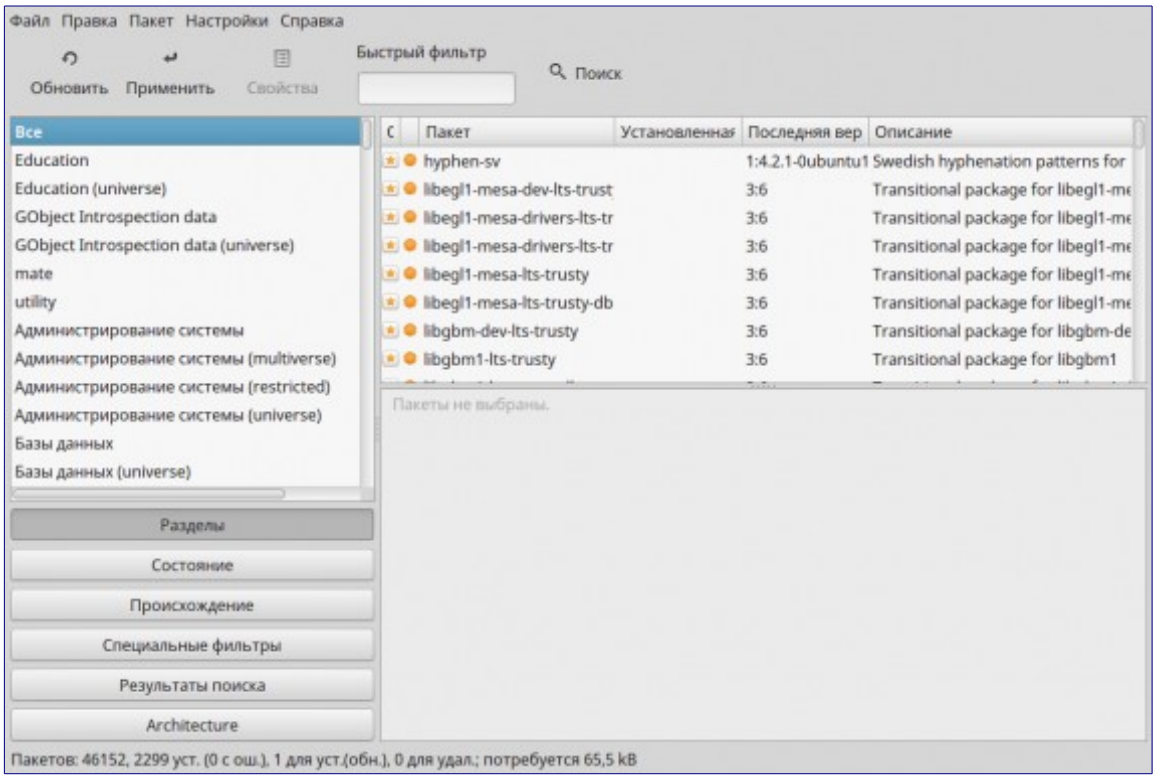
Теперь двинемся вверх по основным элементам интерфейса главного окна synapeic’a. Как уже говорилось, выше двух главных фреймов обнаруживается инструментальная панель, а на ней кнопки. Первой из них идёт кнопка Обновить — это ни что иное, как перечитывание базы данных репозиториев пакетов, тех, которые были определены в настройках (о чем будет говориться далее):



Здесь же можно наблюдать за ходом процесса попакетно:



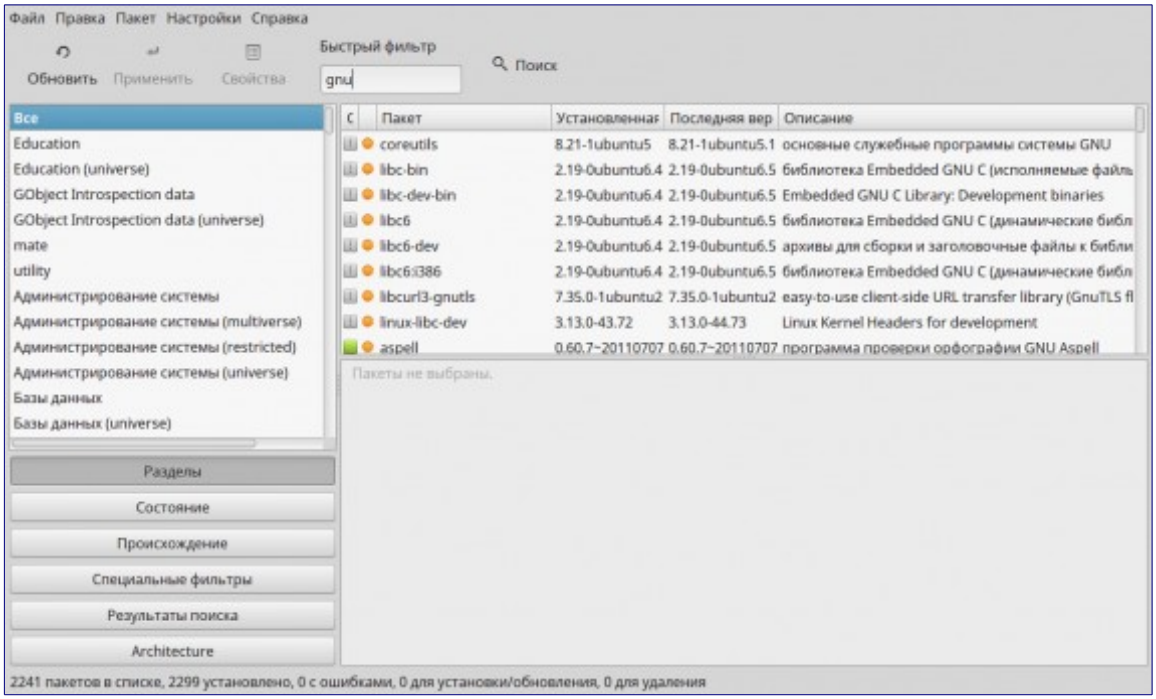
А по его завершении пакеты, для которых доступны обновления, автоматически помечаются жёлтыми звёздочками в первой колонке:



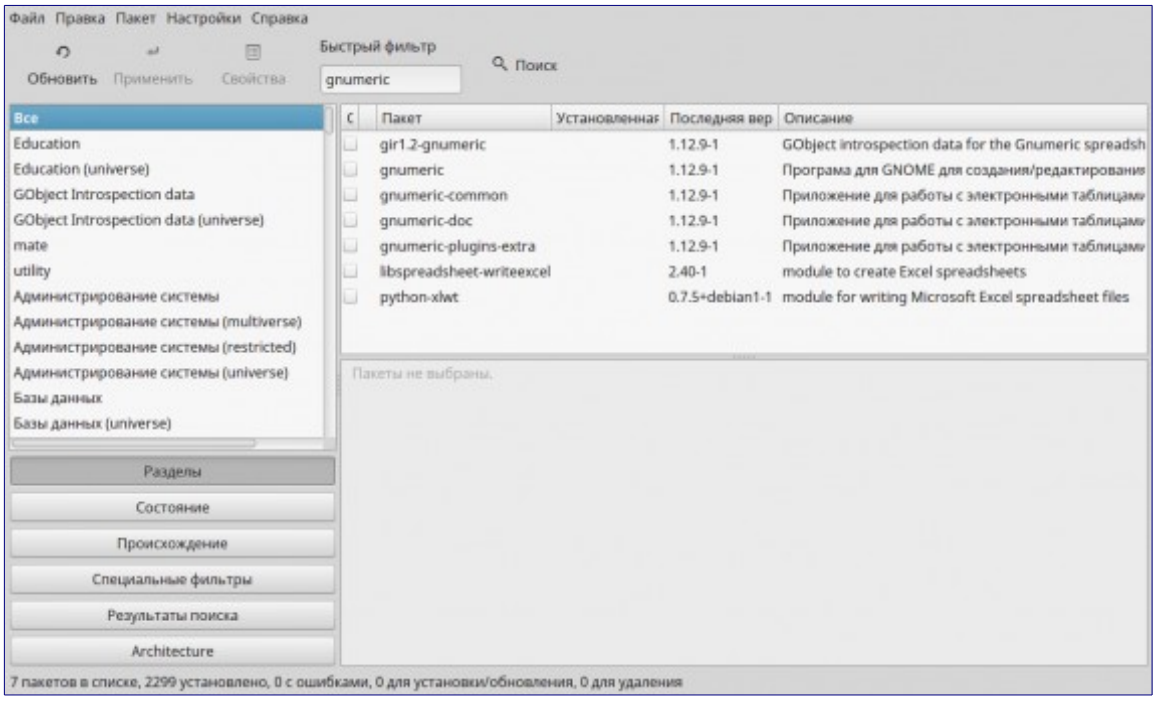
Для претворения помеченного в действительность предназначена следующая кнопка, Применить — предложение выполнить над пакетами, отмеченными для установки, обновления или удаления, соответствующие действия.

Кнопка Свойства вызывает ту же самую панель, что и пункт Свойства контекстного меню.

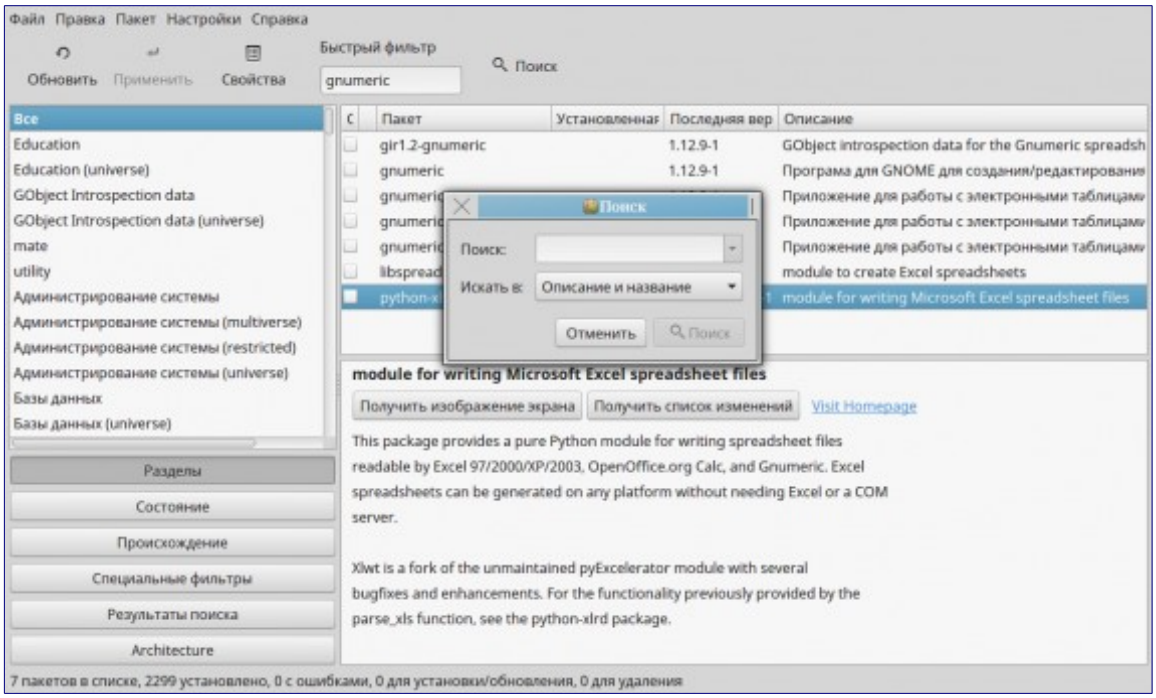
О поиске стоит поговорить отдельно. Поле Быстрый поиск предназначено для обычного наращиваемого поиска в списке правого главного фрейма в соответствие с разделами, выбранными во фрейме левом. То есть, если в последнем выбрать раздел Все, а в поле ввести gnu, мы получим список всех пакетов, содержащих эти символы в имени, в резюме или в описании:



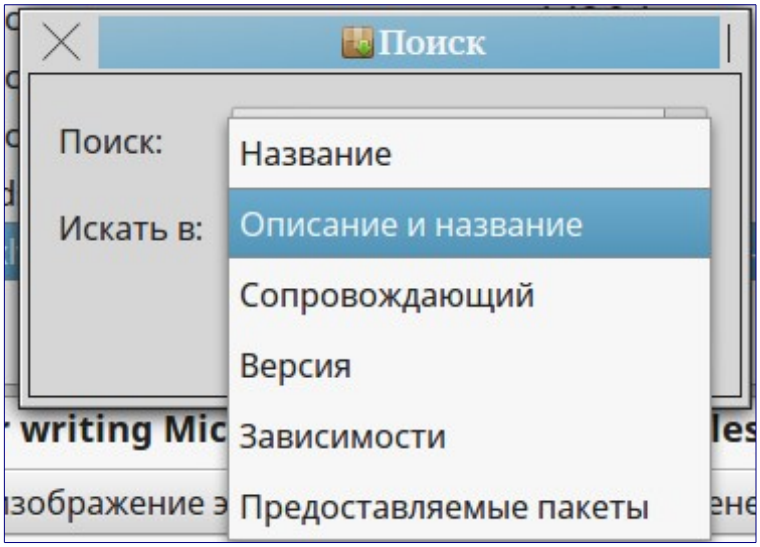
Если же мы укажем точное (или предполагаемое) имя пакета (например, gnumeric), то получим список всех пакетов, непосредственно с ним связанных:



Обращаю внимание на последнюю строку в выводе результатов поиска на скриншоте: ни в имени этого пакета, ни в его кратком описании слова gnumeric мы не увидим — это как ра пример поиска в полных описаниях — тех самых, которые выводятся в нижнем правом фрейме (или в закладке Общие панели Свойства). А вот кнопка Найти как раз и позволяет варьировать область поиска и его критерии:



Как видно из скриншота, по умолчанию поиск проводится в именах пакетов и их описаниях. Однако область поиска можно ограничить только именами. Кроме того, поиск можно выполнять по имени майнтайнера пакета, номеру версии, зависимым или предоставляемым (suggest) пакетам:



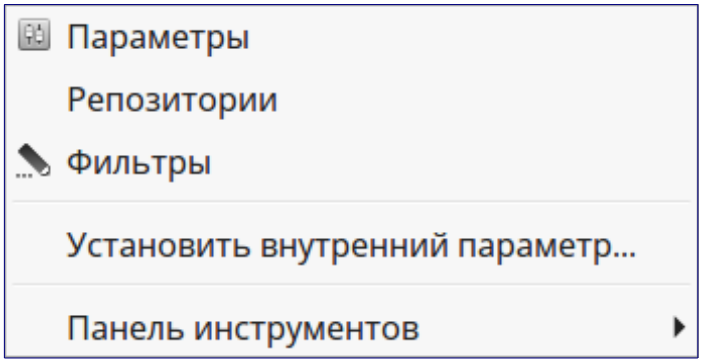
Наконец, самый верхний интерфейсный элемент окна — строка главного меню. Однако на как раз на меню мы останавливаться не будем: смысл его пунктов в общих чертах понятен из названий, И к тому же по большей части они дублируются меню контекстным и кнопками инструментальной панели. Так что скажу-ка я лучше пару слов о настройке Synaptic'а.

Настройка

Как легко догадаться, за настройки Synaptic’а отвечает одноимённый пункт

главного меню, содержащий подпункты:

- Параметры;
- Репозитории;
- Фильтры;
- Установить внутренний параметр;
- Панель инструментов.

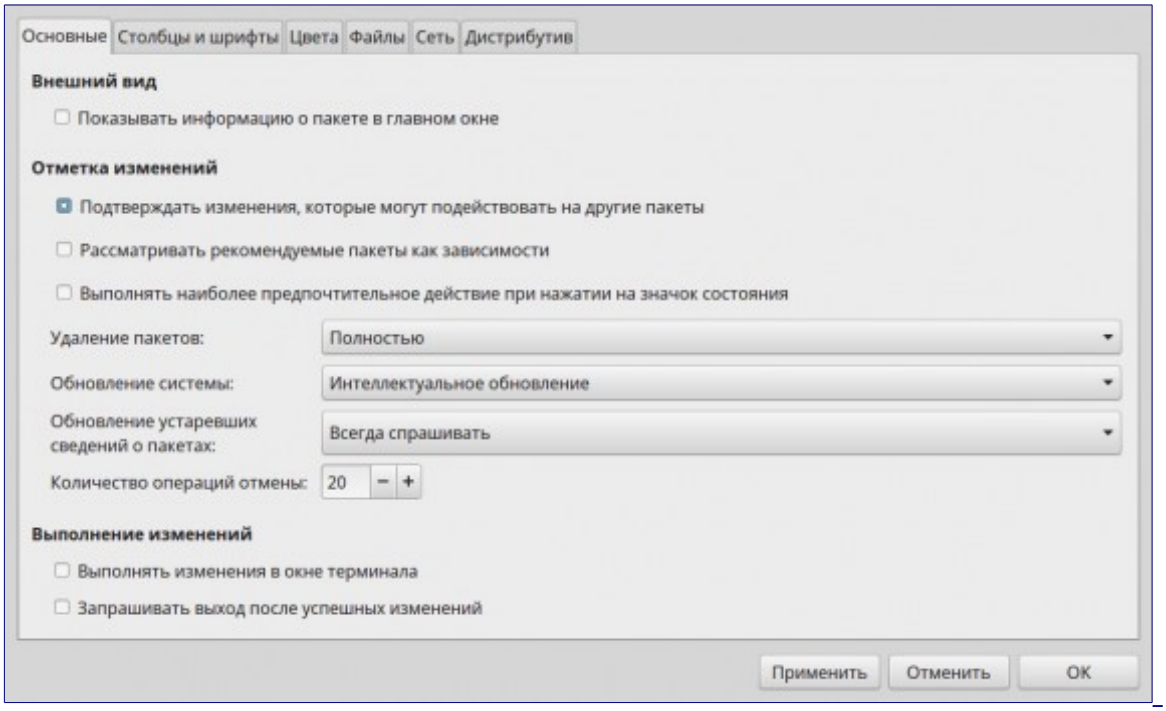


Рассмотрим их последовательно.

Пункт Параметры вызывает панель со множеством вкладок, позволяющих настроить общее поведение Synaptic’a:

- Основное;
- Столбцы и шрифты;
- Цвета;
- Файлы;
- Сеть;
- Дистрибутив.

Вкладка Основное, кроме внешнего вида (включение или выключение чекбокса о показе информации в главном окне у меня не влечёт никаких последствий для оного), позволяет установить ряд очень важных параметров:



Например, выводить ли запрос на изменение пакетов, от которых зависят другие пакеты — разумеется, лучше держать эту опцию включённой.

Важно также определить, следует ли рассматривать рекомендуемые пакеты как зависимости, обязательные к установке — в умолчальной конфигурации Mint'овской реализации apt, (как и в обычном apt, надстройкой над которым является Synaptic), пакеты из категории recommended автоматически не устанавливаются. Так что если использовать apt и Sinaptic попеременно, лучше эту опцию не трогать.

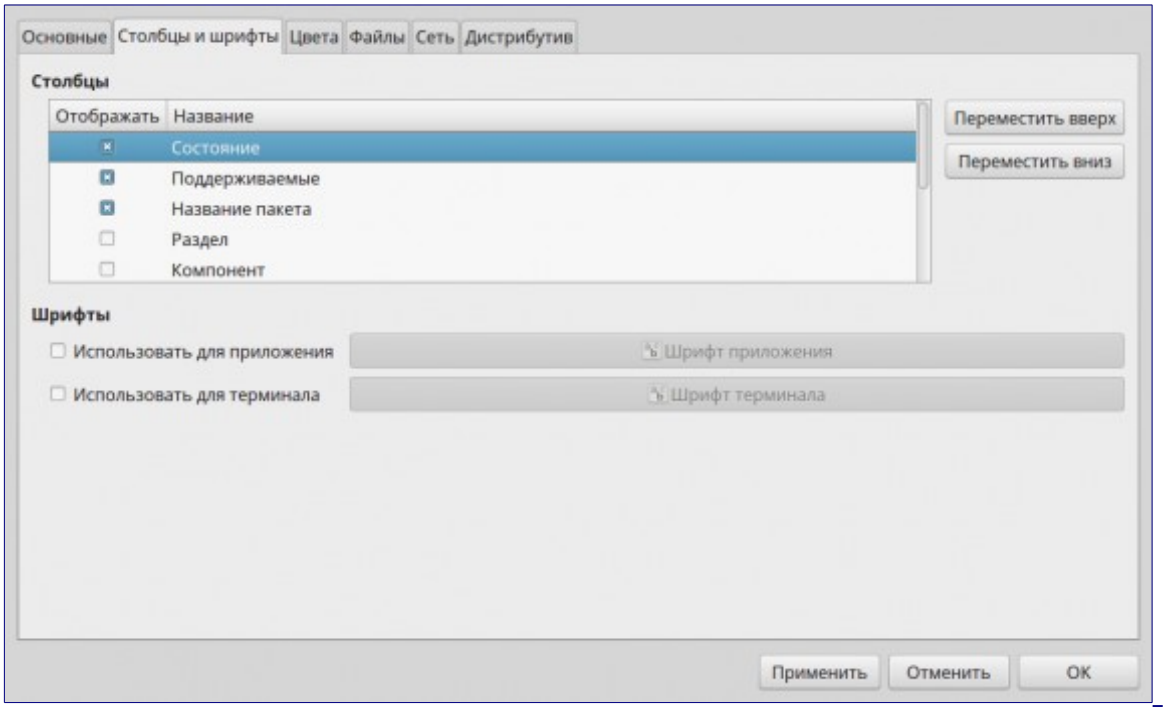
Выпадающее меню Удаление пакетов определяет, удалять ли их полностью (аналог команды apt purge, отмечено по умолчанию), или сохранять конфигурационные файлы (аналог apt remove).

Меню пункта Обновить систему позволяет установить, использовать ли по умолчанию стандартное обновление, интеллектуальное (то есть с попыткой разрешения противоречий зависимостей) или выбирать метод обновления по запросу. Последнее время проблем с интеллектуальным обновлением, вроде, не отмечалось, так что можно остановиться на этом методе (тем более что он отмечен по умолчанию).

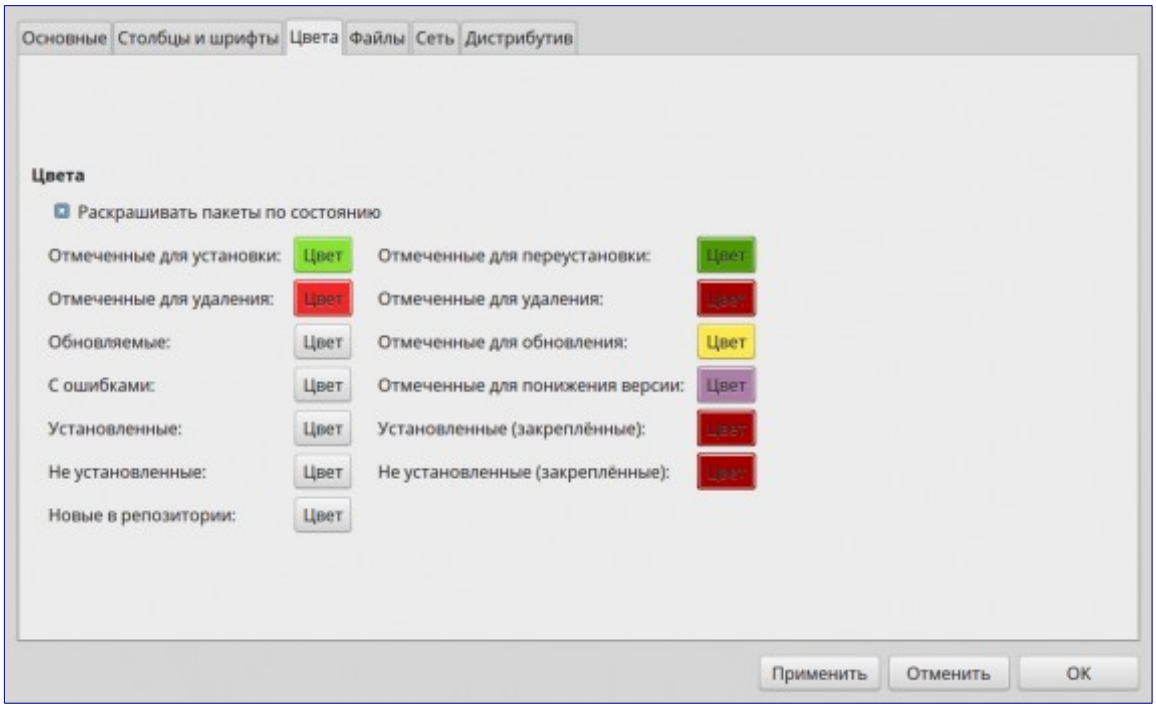
В выпадающем меню Обновление устаревших сведений о пакетах можно выбрать опции — Всегда спрашивать, Автоматически или Игнорировать. Представляется, что первая, умолчальная, будет лучшим выбором.

Прочие опции данной вкладки ясны из приведённого выше скриншота.

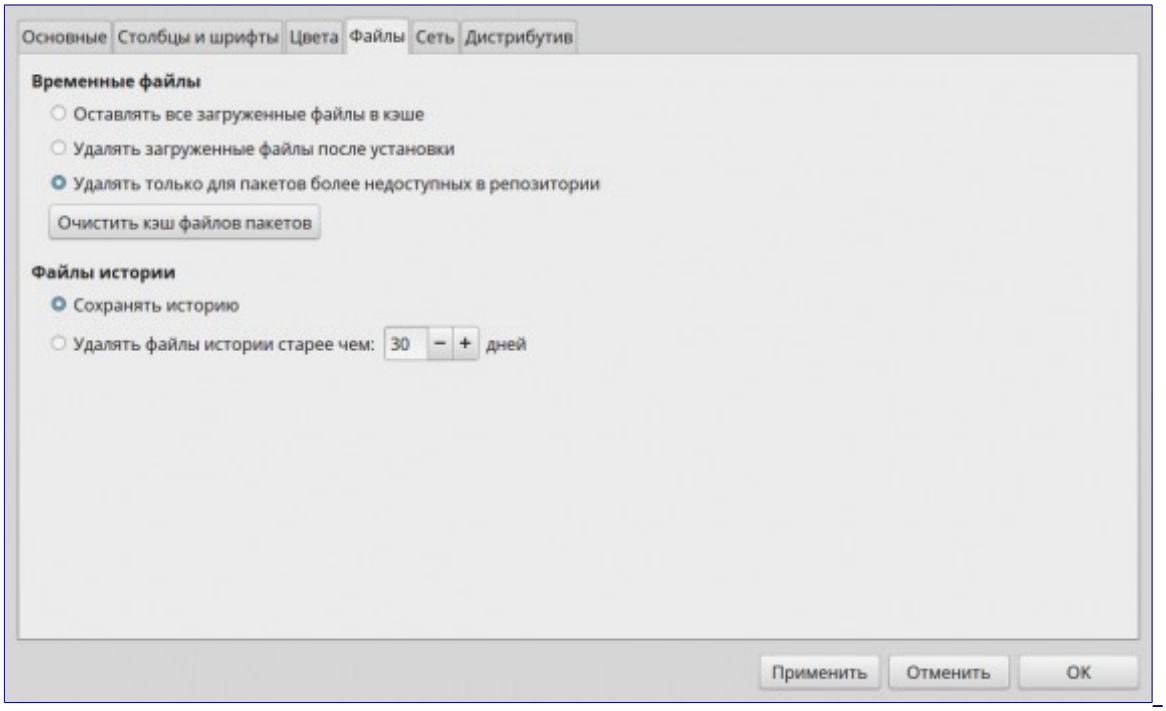
Во вкладке Столбцы и шрифты определяется набор колонок вывода информации о пакетах и их порядок. Ну а со шрифтами всё ясно — можно использовать общесистемные шрифты, заданные глобально для всех приложений среды (в данном случае Cinnamon), или задать собственные, специально для Synaptic’а.



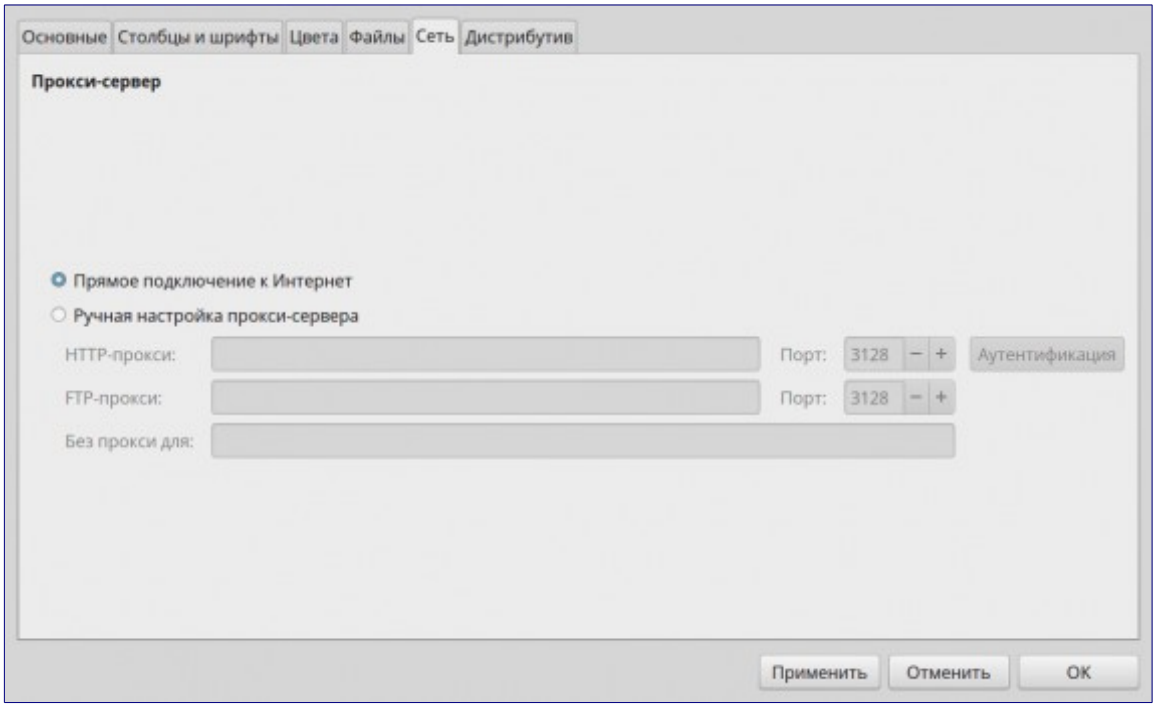
Столь же очевиден смысл установок во вкладке Цвета:



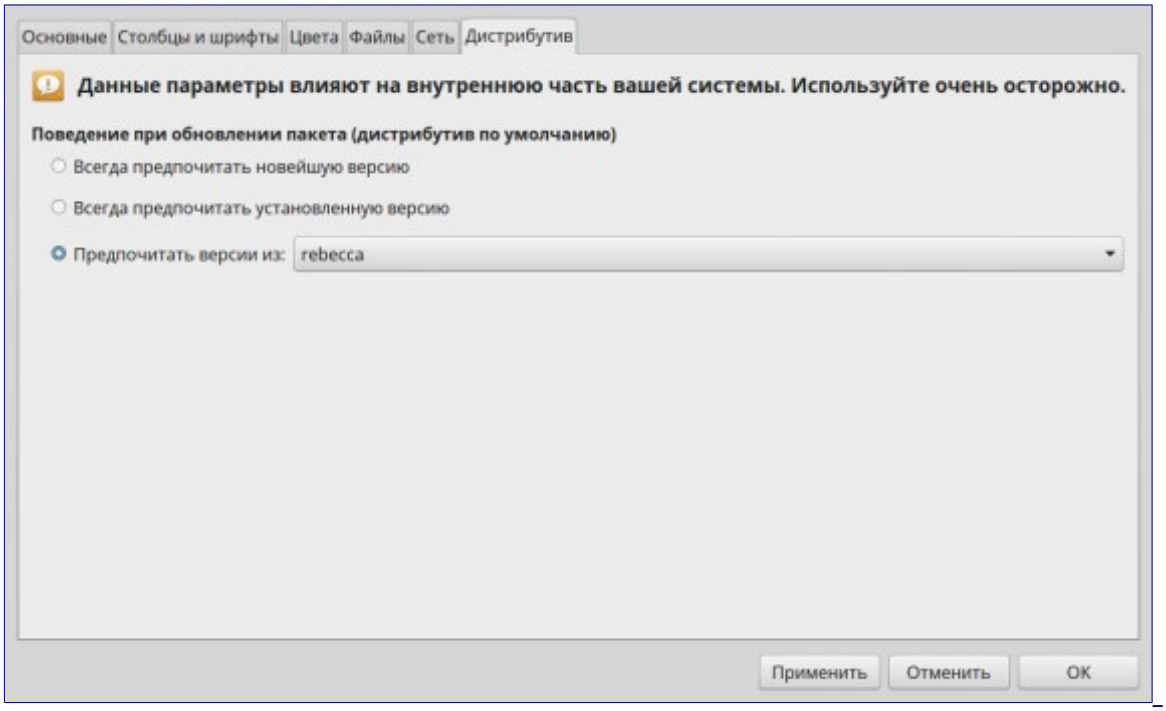
Во вкладке Файлы определяется, надо ли хранить в локальном кэше скачанные файлы пакетов, сохранять ли историю установок, а также задаётся время хранения файлов истории:



Во вкладке Сеть при необходимости можно задать адреса прокси-серверов http и ftp, буде таковые используются:

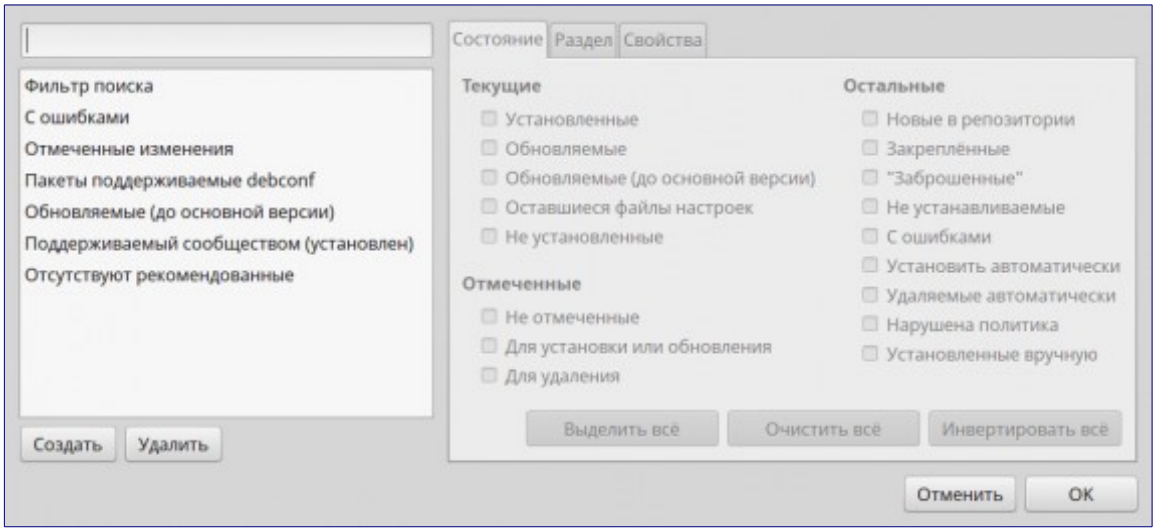


И, наконец, во вкладке Дистрибутив указывается режим обновления дистрибутива в рамках текущей версии — здесь я поменял умолчание (Всегда предпочитать новейшую версию) на :Предпочитать версии из rebecca

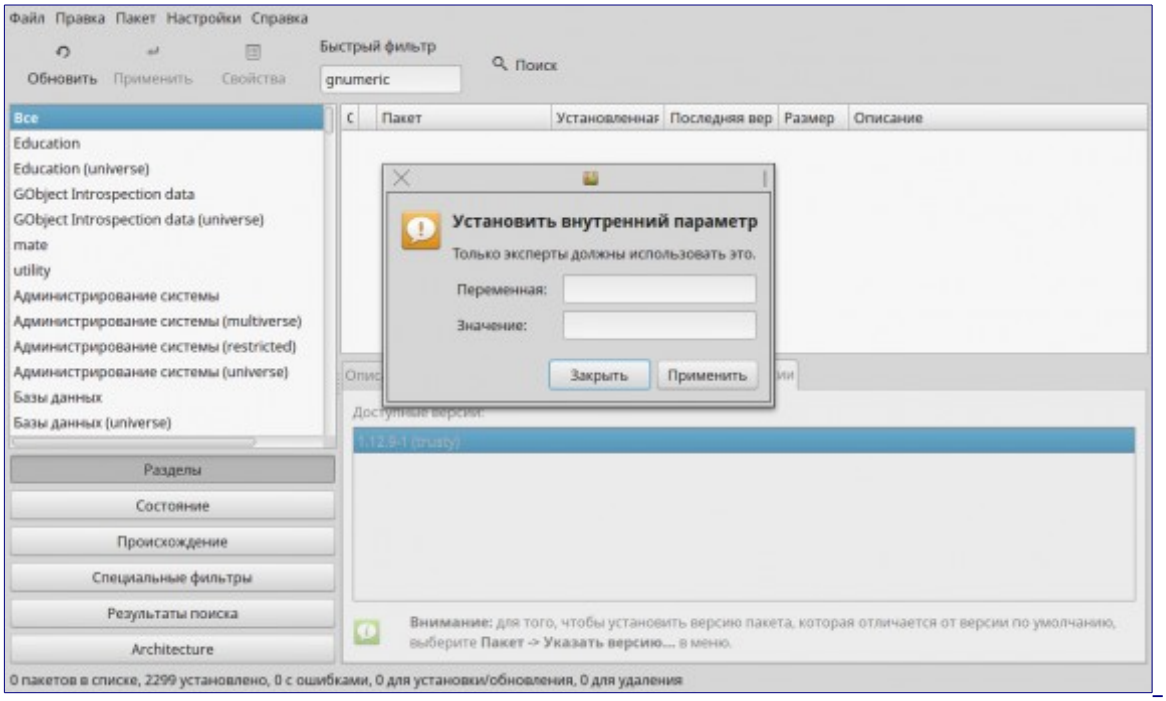


Пункт меню Репозитории, как уже упоминалось, самостоятельного значения не имеет — через него просто вызывается фирменная утилита mintsource.

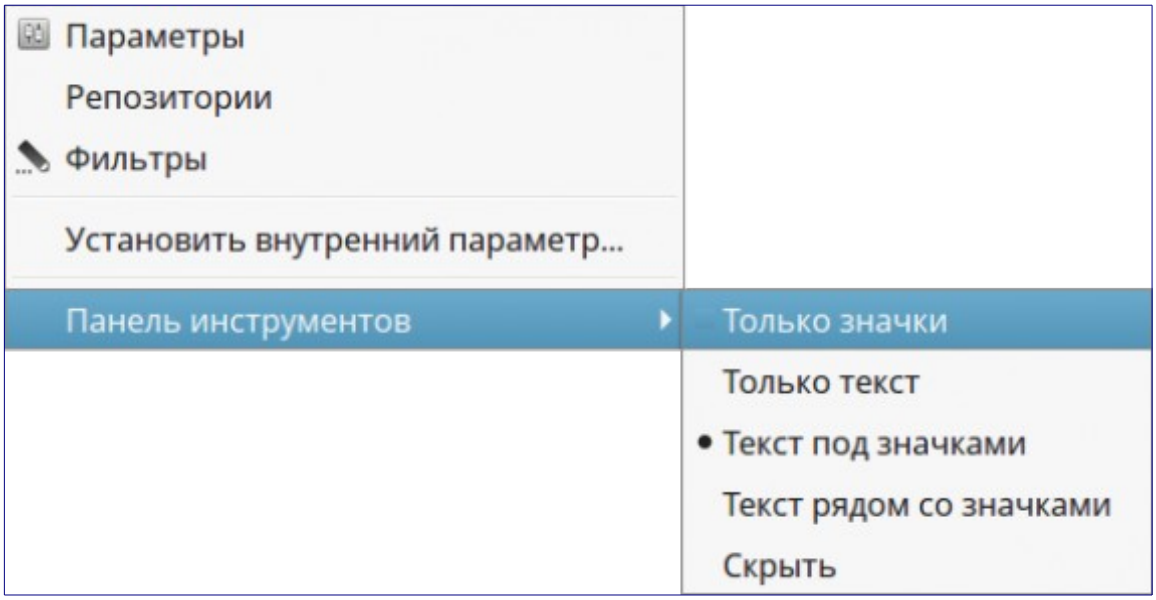
Смысл пункта Фильтры поиска (вспомним, что они фигурируют у нас среди кнопок левого нижнего фрейма главного окна Synaptic’a) в том, чтобы включить (или выключить) те или иные критерии поиска. Детально я с этим не разбирался, оставив на всякий случай всё так, как было по умолчанию:



В пункте Установить внутренний параметр можно задать некие переменные для Synaptic'a, и определить их значения (необходимости в чём, впрочем, я до сих пор не испытывал):



Ну а с пунктом Панель инструментов всё проще некуда — здесь устанавливается вид её кнопок: в виде только значков, только текста или их комбинации; можно также скрыть инструментальную панель вообще:



На этом настройки Synaptic’а можно считать законченными. Как, впрочем, и вообще разговор о нём. Ибо, на мой взгляд, практическое применение этой программы в Mint весьма ограничено: для манипуляций с единичными пакетами эффективней apt в его «фирменной» реализации, для обыденных обновлений проще использовать mintupdate, а для глобального обновления при смене версий дистрибутива — опять же обратиться к apt dist-upgrade. Единственное, для чего я иногда использую Synaptic — это для удаления пакетов, и исключительно в силу большей наглядности процесса. Хотя и здесь есть альтернатива, о которой сейчас расскажу.

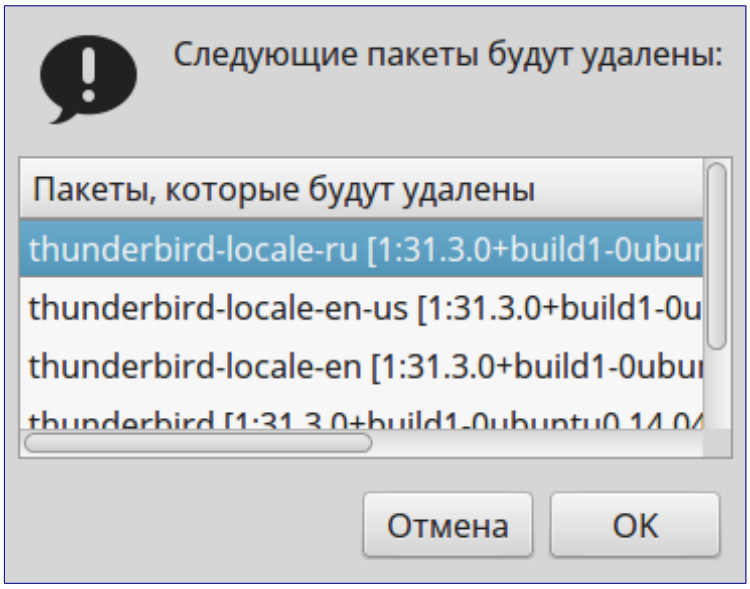
Удаление пакетов: нетрадиционный метод

Среда Cinnamon в Mint предлагает несколько неожиданный метод удаления пакетов — не проверял, имеет ли он место быть в других средах и дистрибутивах. А именно — правым кликом на имени программы вызывается

контекстное меню:



В котором легко видеть пункт Удалить. И это не удаление пункта из меню, что можно сделать в редакторе последнего, а именно удаление пакета (после запроса пароля), вместе со всеми теми, что от него зависят:



Однако пакеты, от которых зависит удаляемый пакет, остаются в неприкосновенности, даже если никем более не используются. Так что после удаления пакетов описанным способом не лишним будет выполнить команду

```
$ apt autoremove
```

Описанный метод по наглядности, как мне кажется, превосходит удаление через Synaptic. Хотя он не очень удобен для массового удаления пакетов (например, после стандартной инсталляции), так как каждый пакет надо удалять индивидуально, да ещё и вводя пароль на каждый чих. Так что в этой ситуации лучше воспользоваться «традиционным» методом — командой apt purge. Однако для удаления единичных пакетов, тем более поставленных «на посмотреть», он подходит как нельзя лучше. Таким образом я удаляю пакеты, устанавливавшиеся в экспериментальных целях и не оправдавшие ожиданий. А также просто те, на которые упал глаз как ненужные после стандартной

инсталляции.

Пользовательские приложения

Всё, что было описано в предыдущих очерках — и установка системы, и знакомство с его графической рабочей средой, и получение представлений о работе в командной оболочке, в том числе в лице лучшей их представительницы, Zsh, и овладение средствами управления пакетами — в конечном счёте преследовало одну цель: оптимальным образом применять Linux Mint для решения своих практических задач. Что осуществляется посредством пользовательских приложений, обозрение которых и будет предметом следующей серии очерков.

Рабочие среды, они же десктопы, не случайно называются также средами интегрированными: кроме средств самообеспечения (оконный менеджер, менеджер сессий и так далее) и самоконфигурирования, они в обязательном порядке включают в себя более или менее обширный набор пользовательских приложений. Из которых важнейшими являются файловый менеджер, эмулятор терминала и текстовый редактор.

Специфика среды Cinnamon — почти полное отсутствие собственных приложений. Что, тем не менее, не значит, что таковых нет после установки соответствующей редакции дистрибутива: «непременный джентльменский набор» применителя, включающий файловый менеджер Nemo, эмулятор терминала GNOME Terminal и текстовый редактор Gedit здесь присутствует. Правда, Nemo представляет собой клон GNOME'вского Nautilus'a, а остальные просто заимствованы из GNOME 3. Однако первый не просто далеко отошёл от прототипа, но и далеко превзошёл его по функционалу и настраиваемости, а терминал и редактор включены в Cinnamon в адаптированном к нему виде. И потому эти три кита интегрированного десктопа вполне могут считаться родными для нашей среды.

Кроме того, в состав Cinnamon-редакции дистрибутива включён обширный набор пользовательских приложений, основанных на библиотеках Gtk и рассчитанных почти на все случаи жизни. Однако давать их подробное описание мне показалось скучным, тем более что большую их часть я не использую и, следовательно, знаю плохо. Поэтому, за редким исключением, они будут лишь кратко охарактеризованы (или вообще просто упомянуты). Основное же внимание я уделю альтернативным приложениям, которые постоянно применяю, люблю и знаю.

Файловый менеджер Nemo

Файловый менеджер среди базовых приложений занимает центральное положение: он является сердцем интегрированной рабочей среды, причём любой (даже Windows — кое-кому памятны разборки о неразрывной связи её Windows Explorer'ом). Без него она хотя и может существовать (как показали примеры «выпиливания» того же Explorer'a), но существование это лишается смысла (что, собственно, и продемонстрировали некогда «выпиливатели», правда, вопреки своей воле).

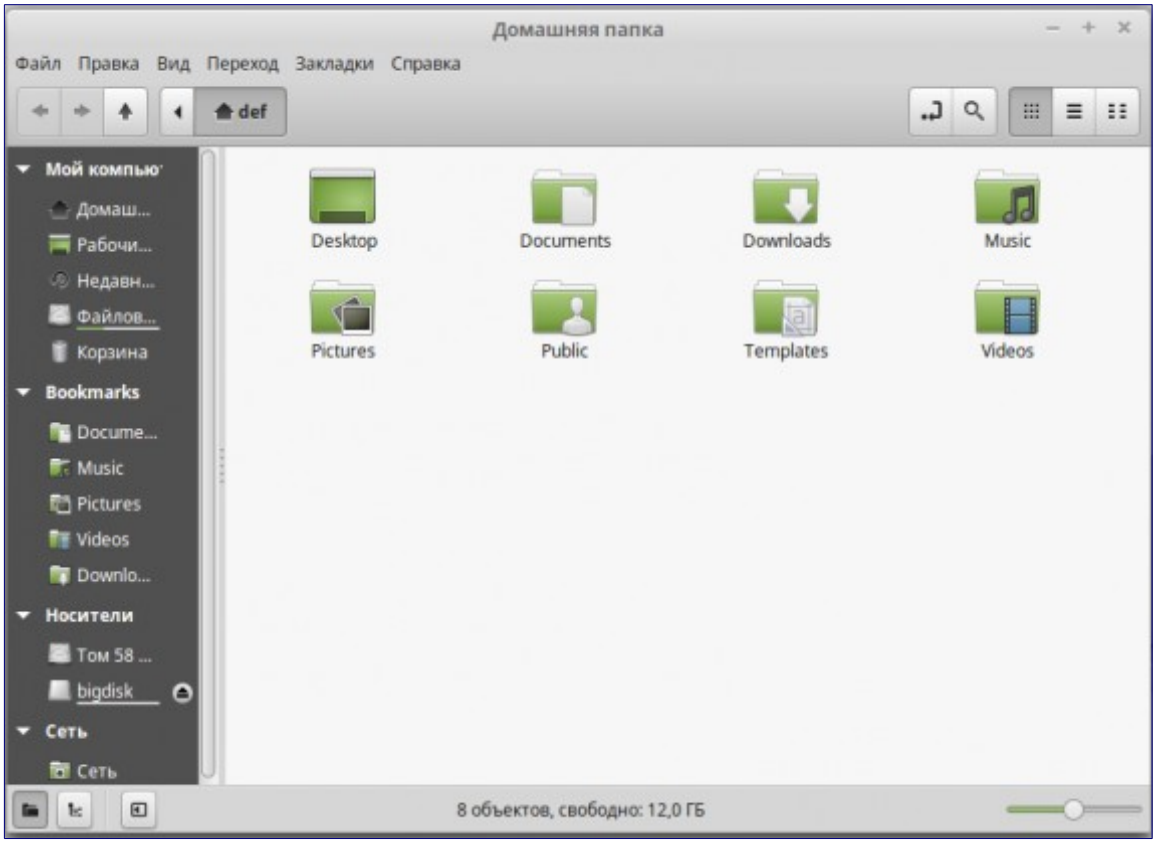
Сказанное применимо и к Nemo в Cinnamon, причём в превосходной степени. Ибо он, в сущности, является единственным штатным приложением этой среды: прочие представители «малого джентльменского набора», из которых в ней присутствуют GNOME-терминал и текстовый редактор Gedit,

выдернуты из GNOME и легко заменяются любыми аналогами, основанными на Gtk, что я покажу в следующих очерках. И только Nemo стоит свою вахту бессменно, потому что заменить его некем. Да и незачем — это уже давно очень хороший файловый менеджер, а в последней своей версии, 2.4 (той самой, что входит в состав Mint Rebecca) он стал ещё лучше.

Обзор возможностей

Итак, представляю героя нынешнего очерка: файловый менеджер Nemo. Вместе со всей средой Cinnamon он ответвился от GNOME с его Nautilus'ом на стадии версии 3.4, до того, как последний стал стремительно терять свою функциональность и настраиваемость. Поэтому Nemo сохранил исходные возможности Nautilus'a, а после отказа Cinnamon (в версии 2.0) от связи с кодовой базой GNOME 3, ещё и приумножил их.

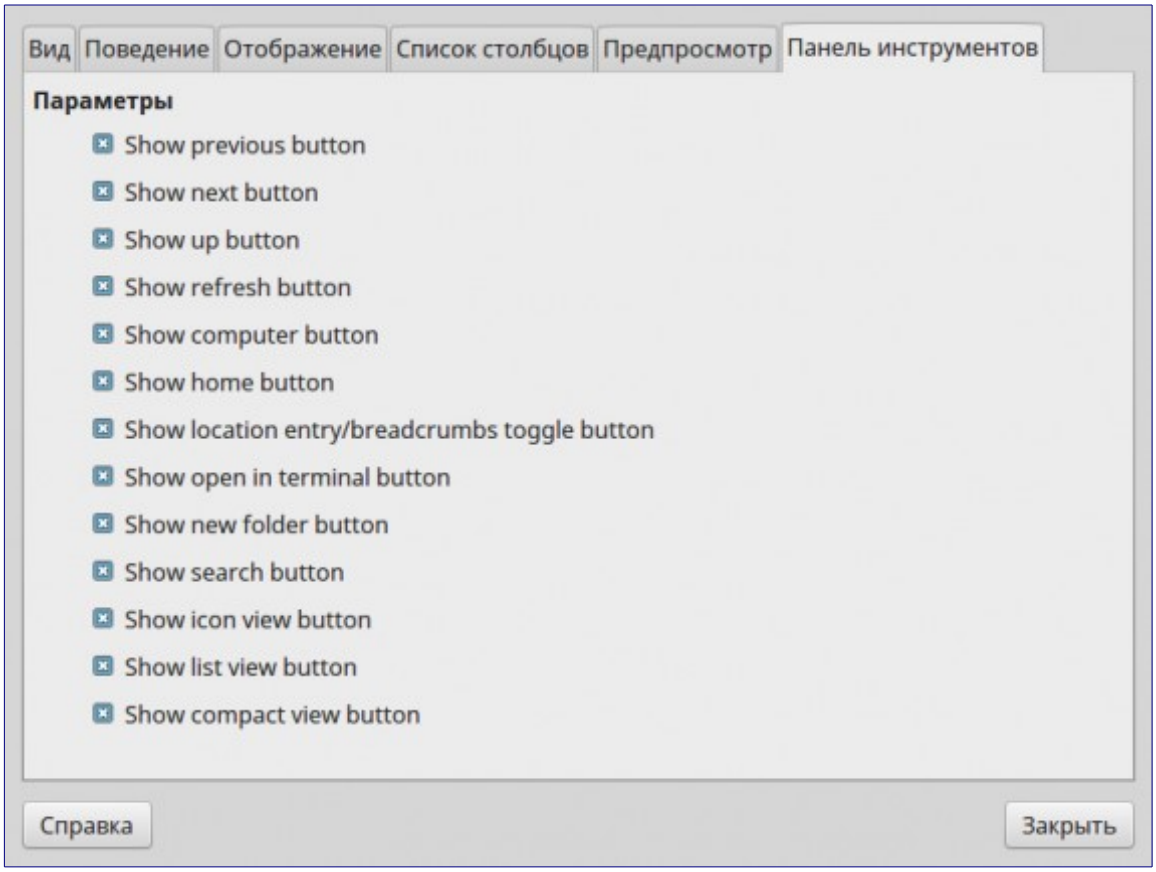
По умолчанию, при первом запуске, Nemo выглядит весьма непритязательно — примерно так:



То есть, казалось бы, Nautilus как Nautilus — визуальное отличие разве что в эмблемах на пиктограммах каталогов (которых раньше не было). Кстати, вплоть до версии 2.2 включительно Nemo под этим именем и фигурировал — и в главном меню Cinnamon, и во всплывающей подсказке при наведении на пиктограмму панели управления. Лишь в версии 2.4 он освободился от тяжкого наследния, и нынче и там, и там написано просто Файлы (Files в оригинальной локализации). Одновременно с этим появились и упомянутые только что эмблемы.

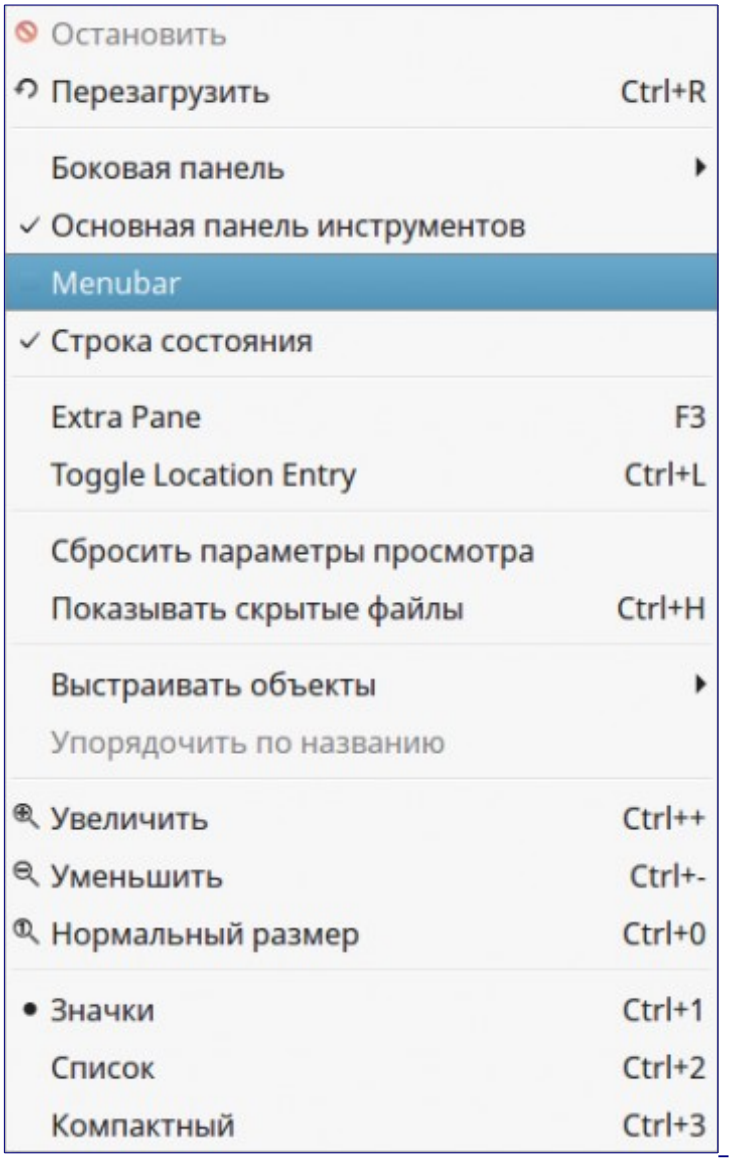
Однако на деле Nemo оказывается не так прост. То, что он поддерживает вкладки — само собой разумеется, кто их нынче не поддерживает, даже Thunar. Однако далее: графическая строка адреса лёгким нажатием на загогулину справа от неё превращается в текстовую — и остаётся таковой в последующих сеансах, если не «опиктограммить» её обратно.

Инструментальную панель легко пополнить пиктограммами перезагрузки, быстрого перехода в корневой и домашний каталоги, открытия терминала в текущем каталоге и создания нового каталога. Делается это через главное меню: Правка -> Параметры -> Панель инструментов:



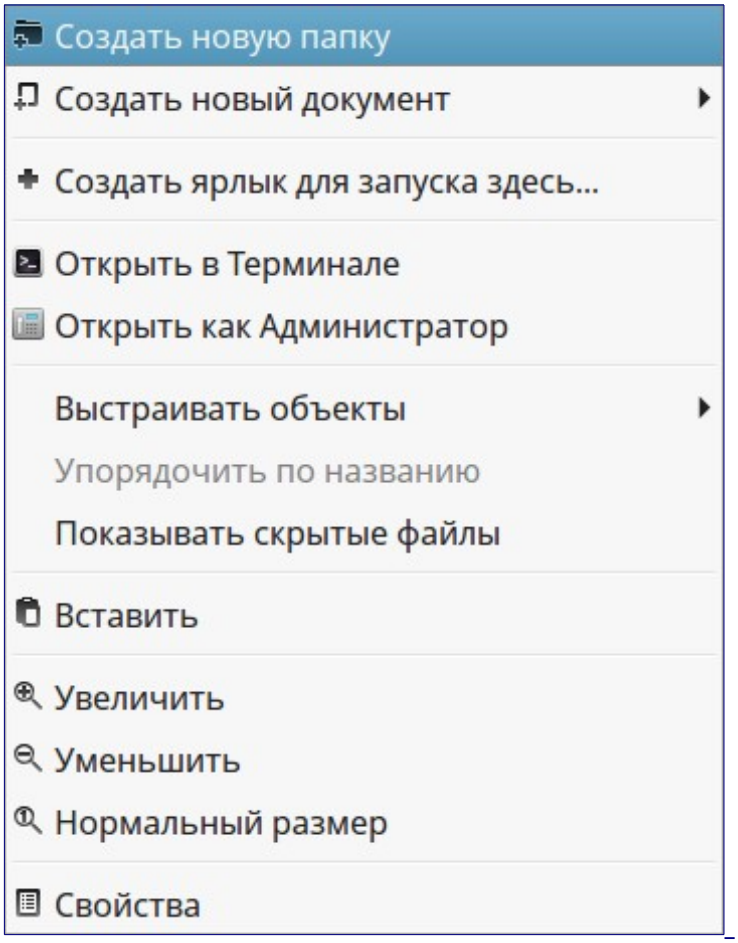
Назначение любой пиктограммы легко определяется по всплывающей подсказке.

Саму строку главного меню можно скрыть через меню же: в пункте Вид снять отметку с подпункта Menubar. После этого строку меню можно быстро делать видимой и скрывать заново либо нажатием клавиши Alt, либо правым кликом мыши на панели инструментов или строке состояния. И так — до тех пор, пока не сделать строку меню видимой постоянно — тем же образом, что она была скрыта, то есть через меню:



Впрочем, я в постоянно видимом меню необходимости не вижу: большую часть обыденных действий можно (и проще) выполнять через пиктограммы панели инструментов или через контекстное меню по правому клику мыши. А меню вызывать только при необходимости — например, для пополнения списка закладок боковой панели (см. ниже).

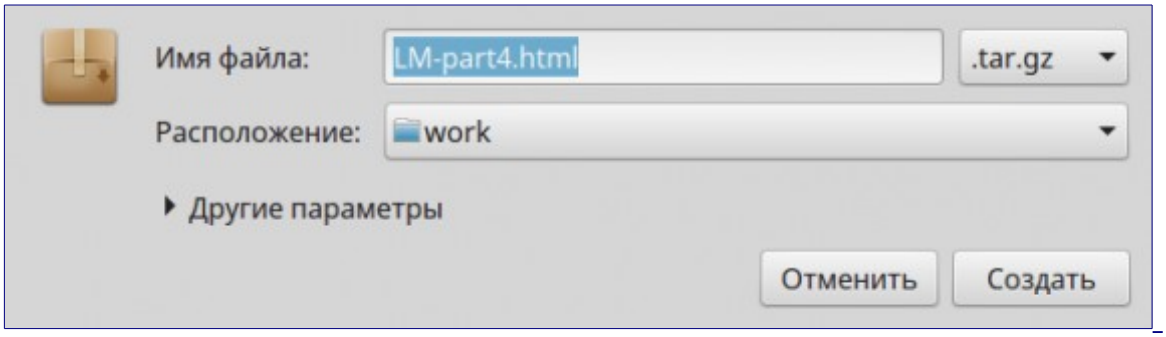
Вид контекстного меню различается в зависимости от того, где именно кликнуть. Если на пустом поле основного окна Nemo — в нём будут пункты создания каталога, файла или ярлыка запуска приложения, открытия в терминале, открытия Nemo с правами администратора, сортировки, показа и скрытия dot-файлов, вставки из буфера, масштабирования пиктограмм; последнее можно сделать и ползунком на строке состояния.



При щелчке на пиктограмме одиночного файла пункты Открыть в Терминале и Открыть как Администратор, естественно, пропадают. Зато появляются пункты открытия с помощью приложения по умолчанию и «запасных» приложений для данного типа файлов (в примере на скриншоте ниже — для HTML-файлов), вырезания, «дублирования», создания симлинка и так далее:

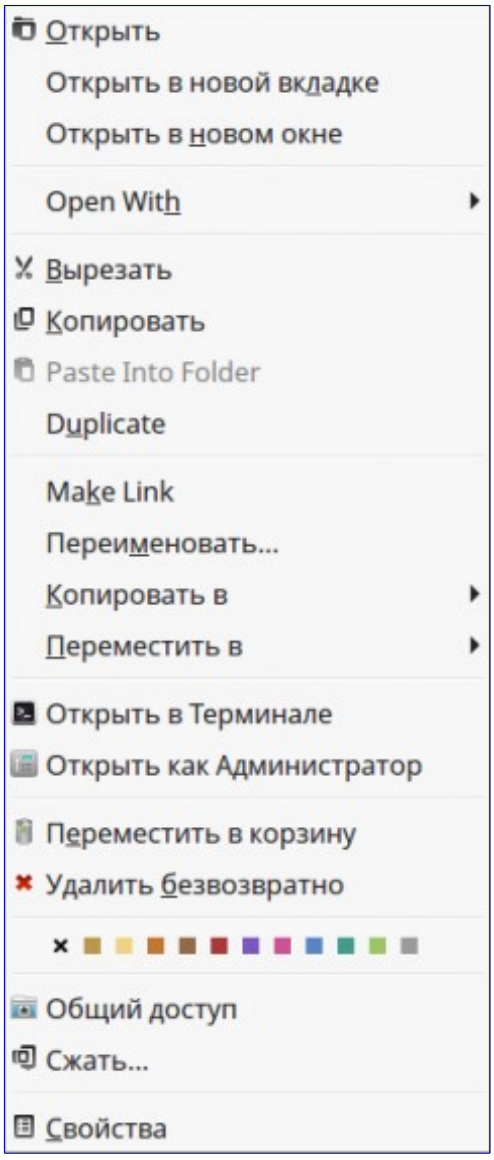


Среди «далее» отмечу безвозвратное удаление и сжатие, что для единичного файла означает именно сжатие каким-либо компрессором из доступных в системе, понятие тут архивирования смысла не имеет, хотя по умолчанию предлагается именно архив:

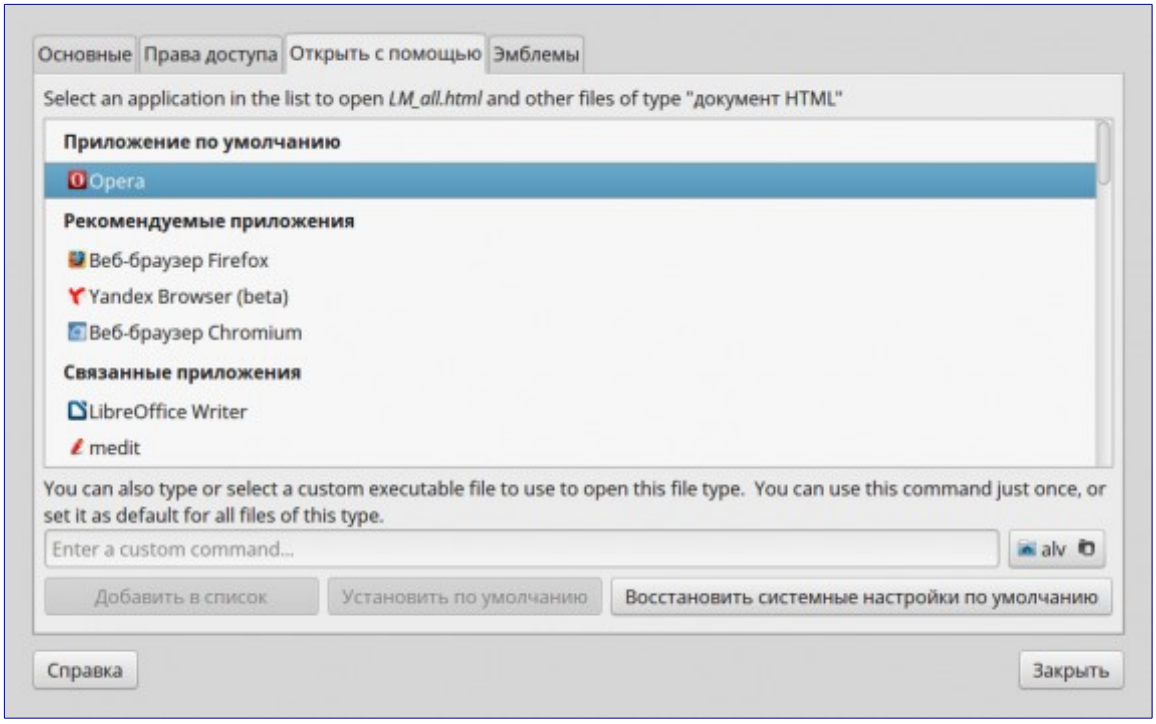


Во избежание недоразумений следует подчеркнуть, что сам по себе Nemo ничего не сжимает и не архивирует. Он лишь вызывает менеджер архивов — nemo-fileroller. Который, впрочем, тоже ничего не делает, а обращается к низкоуровневым утилитам архивации (tar) и компрессии (gzip, bzip2, xz и так далее); наличием последних и определяется число доступных архивных форматов. При щелчке на пиктограмме архивного файла в контекстном меню появится пункт Распаковать сюда, который всё той же цепочкой (nemo-fileroller -> архиватор/компрессор) развернёт архив в текущем каталоге.

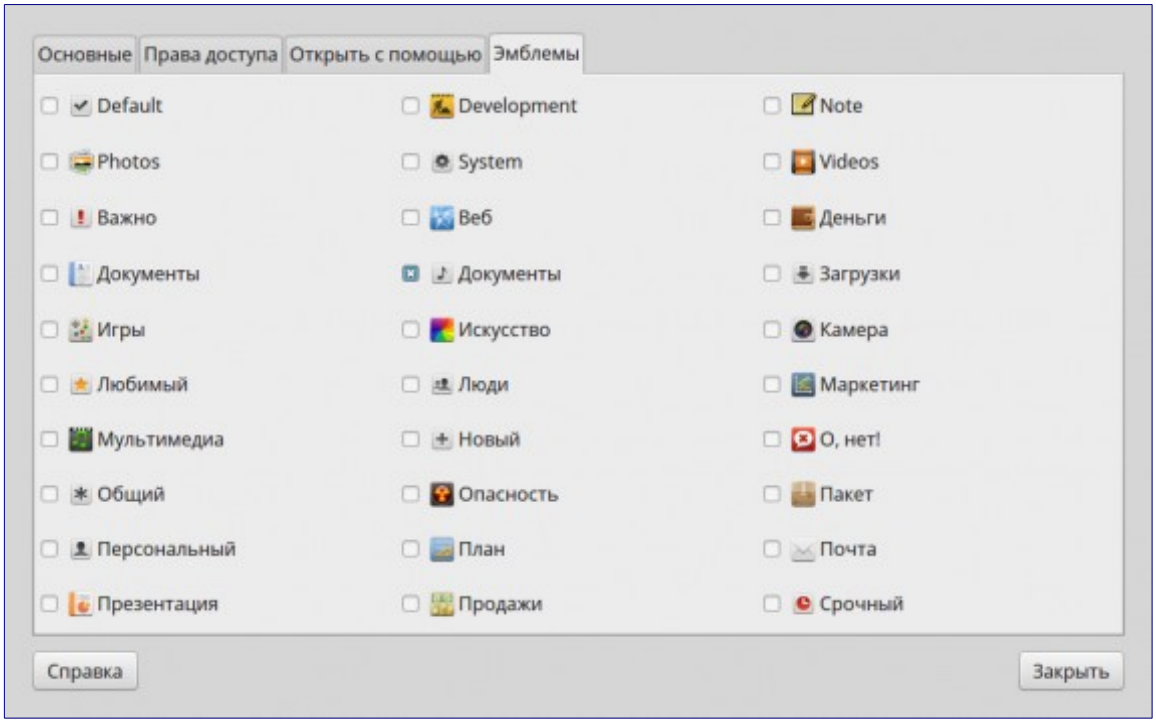
При щелчке на пиктограмме каталога пункты контекстного меню первого и второго случая как бы суммируются. Но к ним ещё присоединяются пункты открытия (на месте, в новом окне, в новой вкладке), настройки общего доступа, а также цветовая палитра, позволяющая окрасить пиктограмму каждого каталога в свой цвет (из числа предопределённых темой).



Во всех трёх случаях контекстное меню завершается пунктом Свойства, содежимое которого тоже различается в зависимости от места клика. В частности, для одиночного файла имеется вкладка Открыть с помощью, в которой можно переопределить приложение по умолчанию, связанное с данным типом файлов:

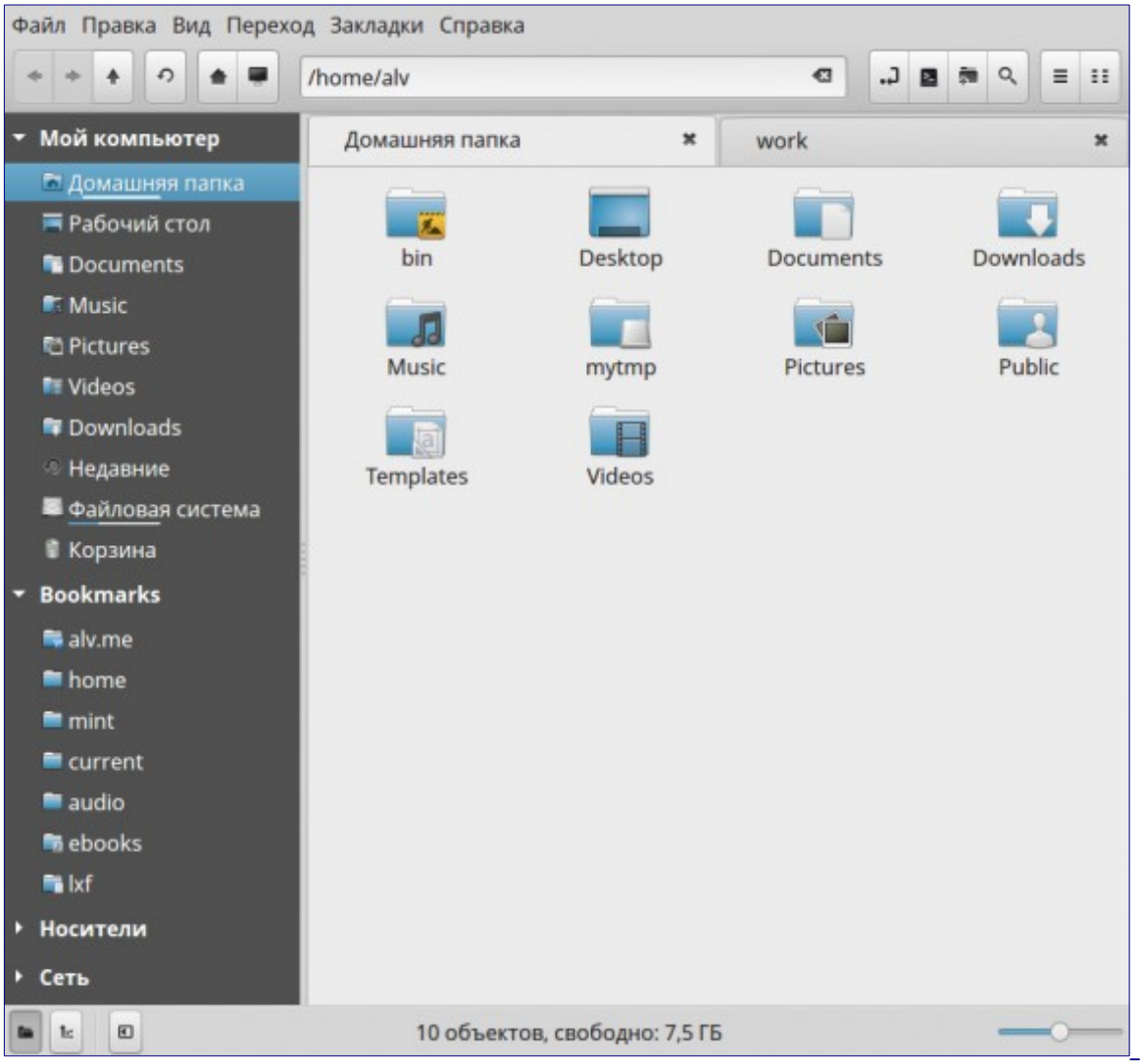


Любопытна вкладка Эмблемы, впервые появившаяся в версии Nemo 2.4. Именно с её помощью можно к пиктограмме каждого каталога и файла, зависимости от их содержимого, миниатюрное изображение из заданного набора:

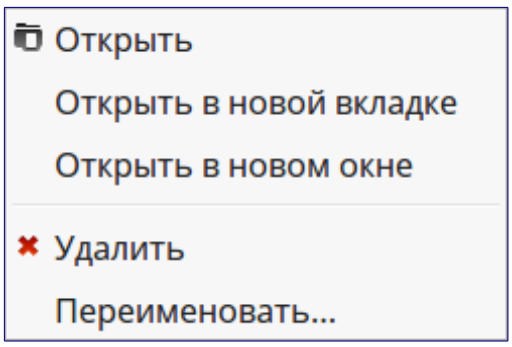


Правда, делать это придётся вручную и по одному объекту.

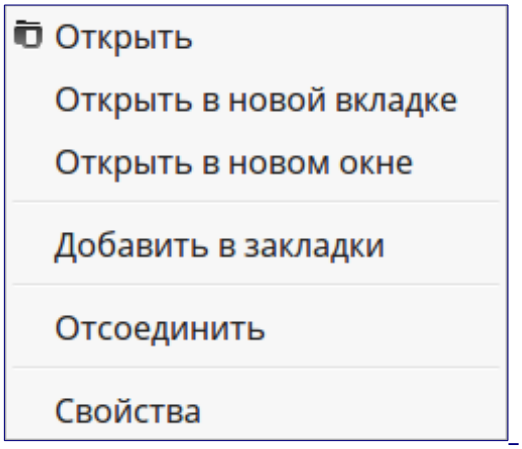
В итоге описанных выше действий по модификации внешности Nemo, в моей системе он приобрёл следующий вид:



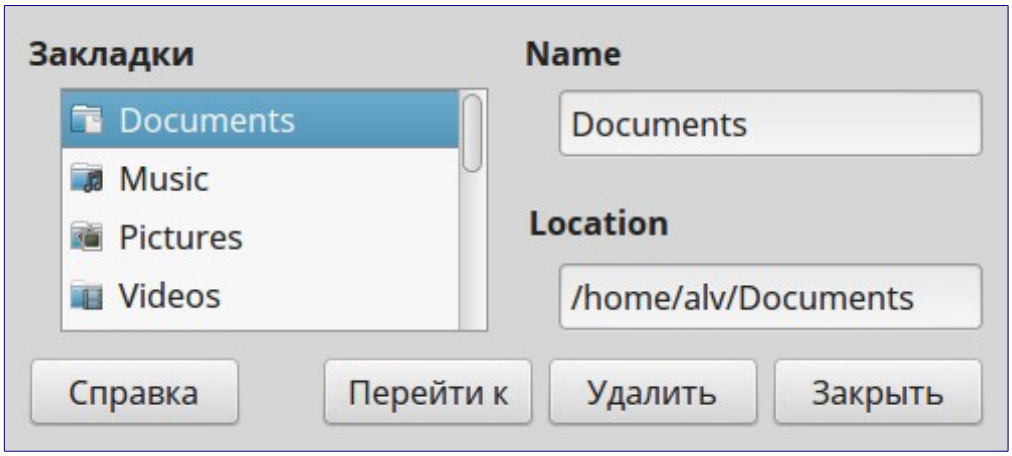
В боковой панели окна Nemo выводится список закладок — каталогов файловой системы с быстрым доступом, который можно пополнять произвольным образом, и «посторонних» (то есть автоматически не монтируемых) носителей, как внутренних, так и внешних. Из контекстного меню по правому клику они могут быть открыты в текущей вкладке, новой вкладке или новом окне:



Для «сторонних» носителей предусмотрены также пункты монтирования (без открытия) и отмонтирования:

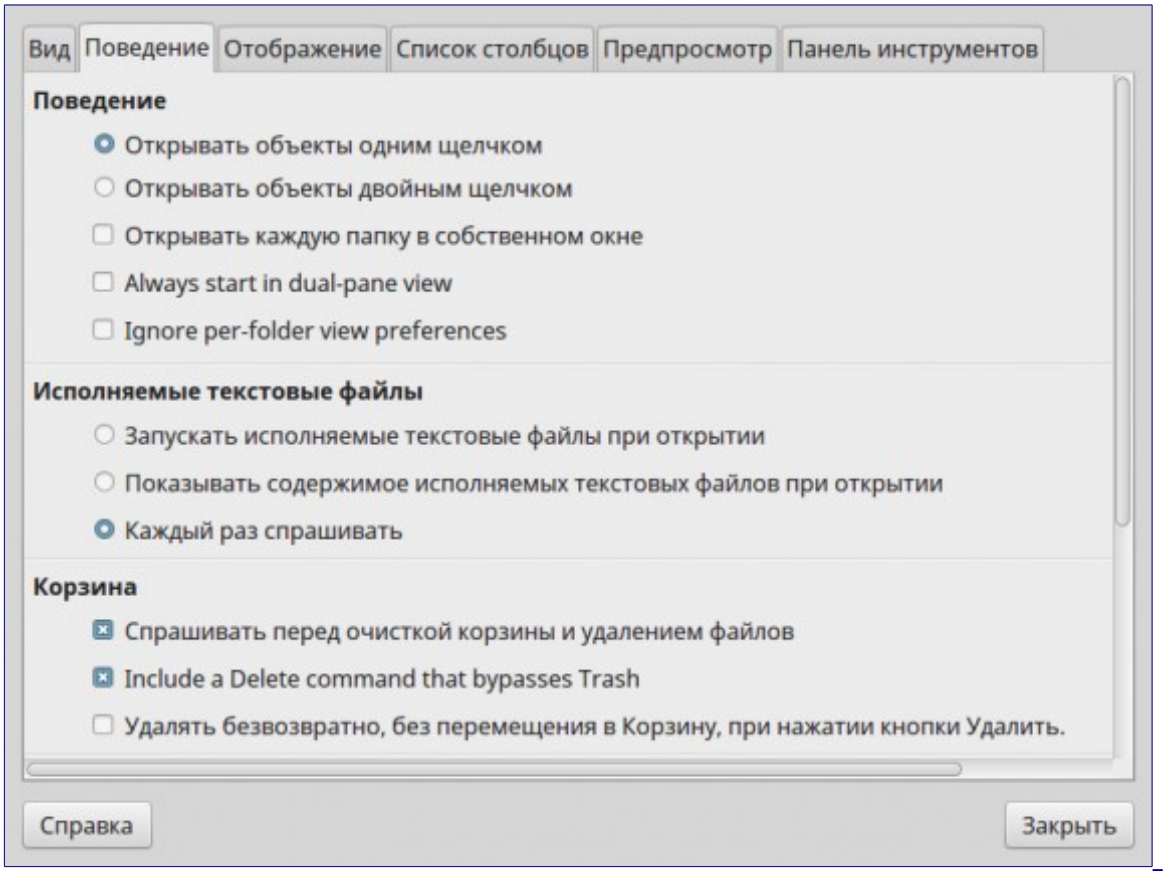


Пользовательские закладки пополняются через главное меню: Закладки -> Добавить в закладки. При этом они не сваливаются в одну кучу с предопределённым набором закладок (такими, как Desktop, Documents, etc.), а размещаются в специальной секции Bookmarks. Впрочем, перетаскиванием мышью их можно тасовать, как угодно. Кроме того, их можно удалять и переименовывать — и из контекстного меню, и из главного, через пункты Закладки -> Изменить закладки:

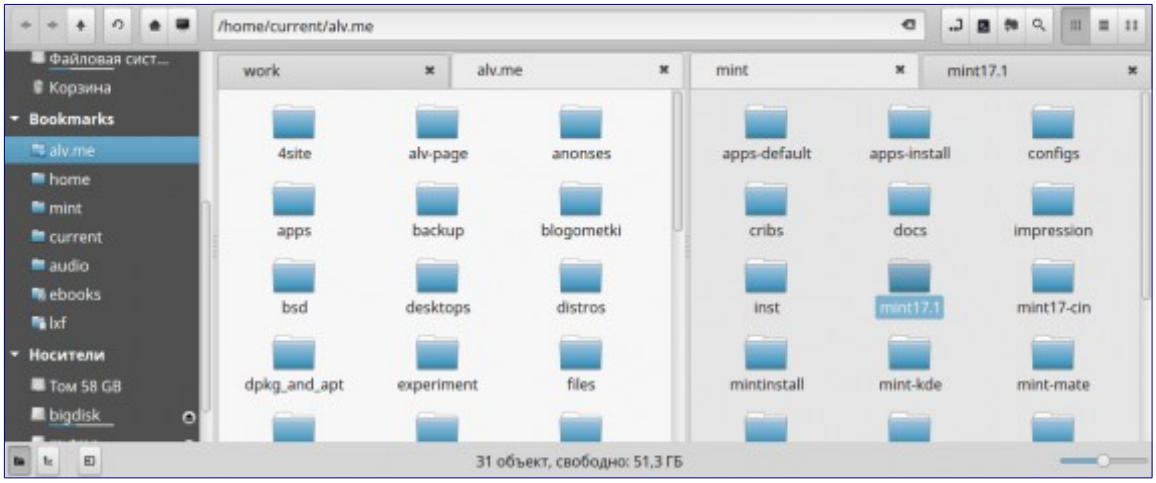


Исключение — «квазисистемные» закладки (Файловая система, Домашний каталог, Рабочий стол, Корзина) — для них эти функции недоступны.

Кстати, закладка открывается одинарным щелчком: левой кнопкой мыши — в текущей вкладке, средней кнопкой — в новой вкладке. Для открытия же каталога из пиктограммы во вкладке по умолчанию требуется двойной щелчок левой кнопкой. Что, однако, легко изменить через меню: Правка -> Параметры -> Поведение:

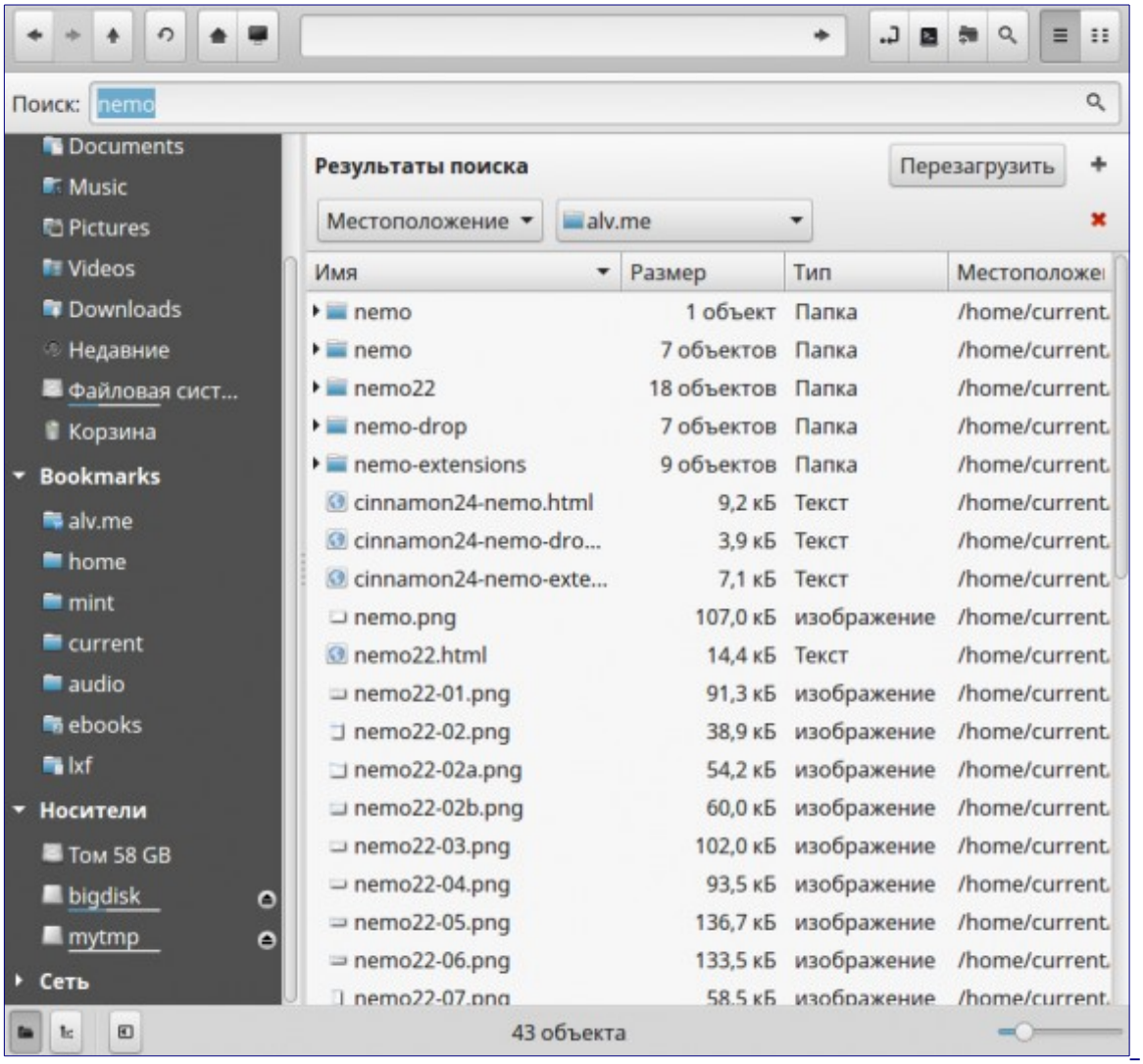


Наконец, в Nemo имеется двухпанельный режим, включаемый через меню Вид -> Extra Pane. Само собой, в каждой панели можно вывести содержимое разных каталогов, да ещё и в нескольких независимых вкладках:

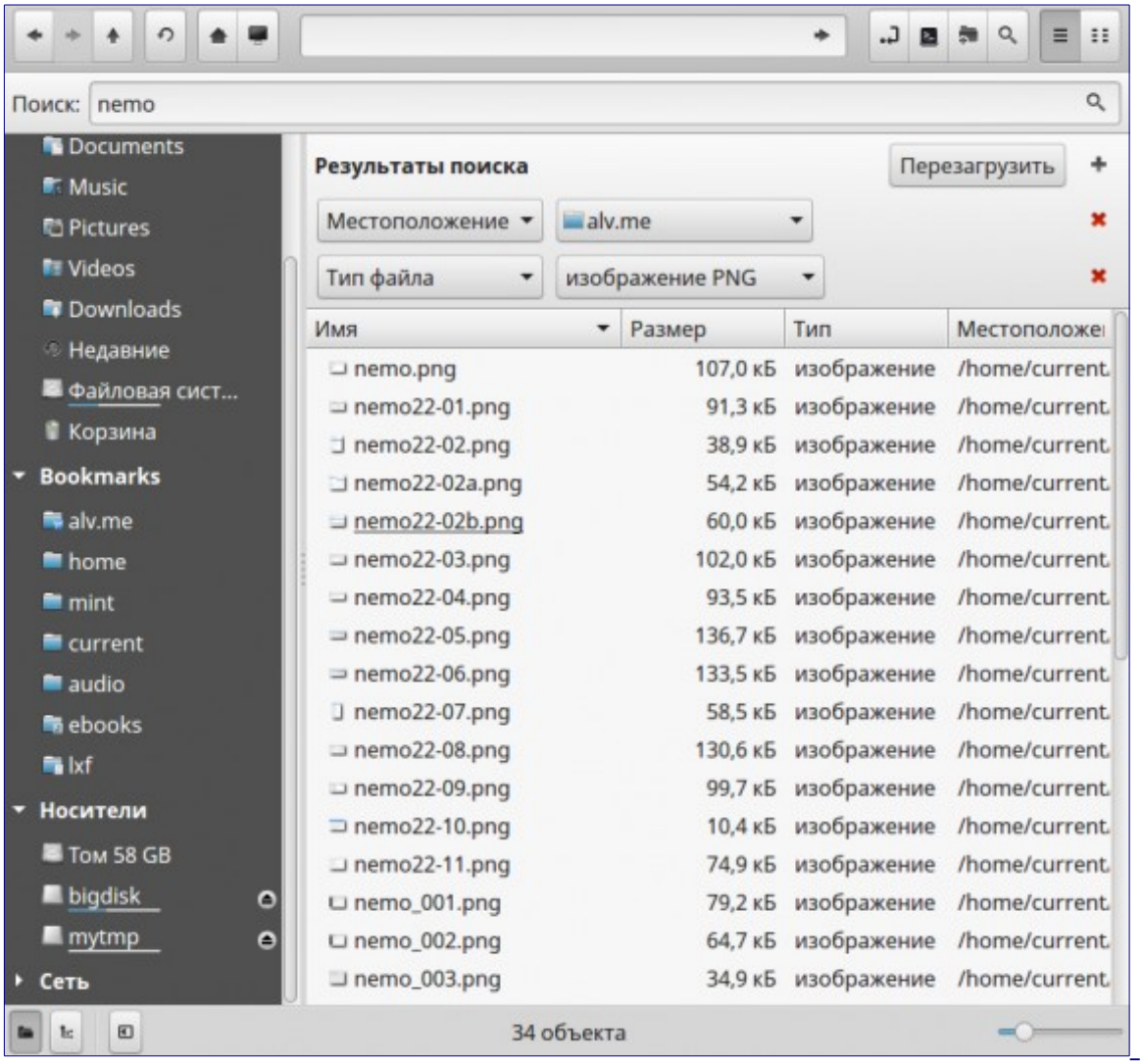


На мой взгляд, совмещёние «многовкладочности» и «двухпанельности» — явный перебор. Но в ряде случаев временное включение второй панели (а это можно сделать быстро — клавишей F3) бывает полезным — например, при работе с облачными хранилищами.

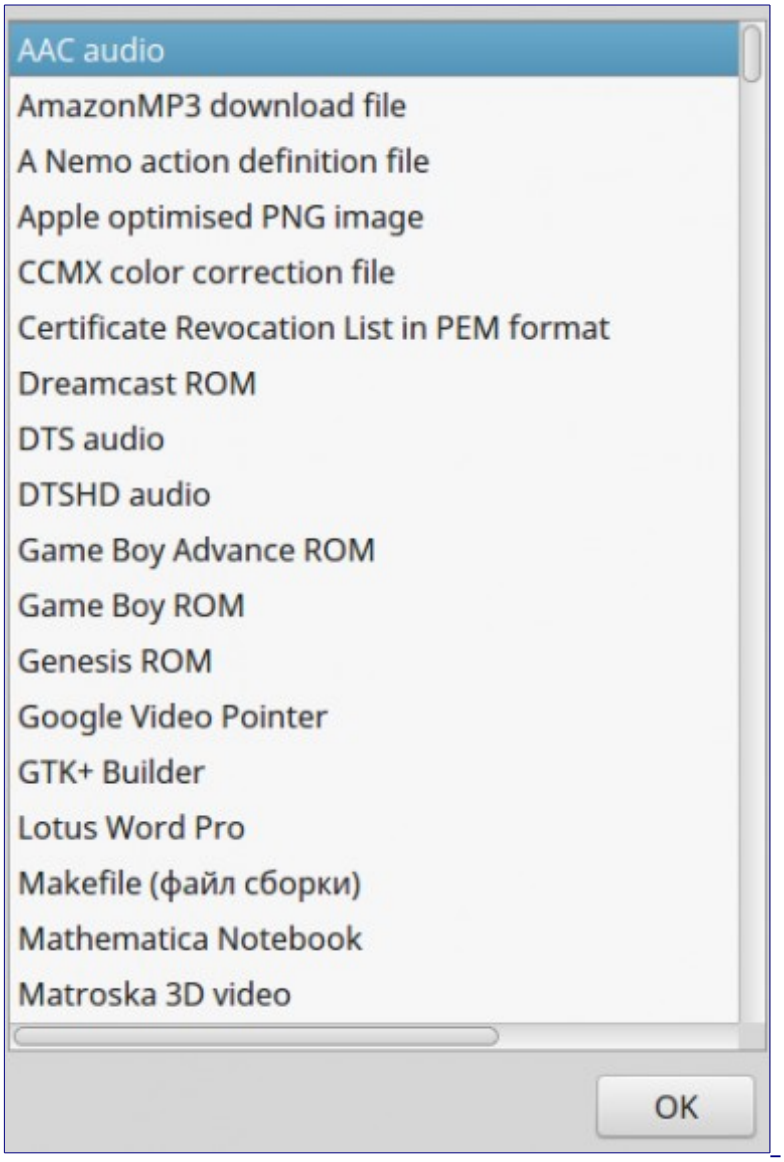
В Nemo имеются весьма богатые возможности поиска файлов, доступные из меню Переход -> Поиск файлов или по нажатии на соответствующую кнопку инструментальной панели. Для начала поиск выполняется по одному критерию — местоположению:



Круг поисков можно сузить, задав второй критерий — тип файла (в примере изображение PNG):



Кроме таких абстрактных типов, как документ, музыка, презентация и так далее, более конкретно тип файла можно выбрать из длиннющего списка, вызываемого выбором пункта Другой тип:



Теоретически критериев поиска можно задать сколько угодно, только комбинировать можно только значения двух их вариантов — местоположения и типа файлов, так что больше двух критериев практического смысла не имеют.

Иначе говоря, Nemo предоставляет большинство возможностей, которые мы вправе ожидать от современного «продвинутого» файлового менеджера. Если, конечно, вслед за разработчиками GNOME не считать признаком «современности» и «продвинутости» отсутствие возможностей...

Перепробовав немалое число программ этого рода, могу со всей ответственностью утверждать, что по функциональности и настраиваемости Nemo уступает только старому Konqueror'у и современному Dolphin'у из KDE, да и то немного. В частности, в нём (мне) очень не хватает встроенного терминального окна — но это, пожалуй, единственное, чего на самом деле недостаёт. Тем более, что в принципе эта проблема решаема, как будет показано в следующем разделе.

Nemo и его терминал

Как только что было сказано, единственное, чего не хватает в Nemo по настоящему (для меня) — это встроенного терминала. Что, однако, решается установкой одного из «расширителей» этого файлового менеджера (nemo-extensions), именуемого nemo-terminal. Он происходит от некогда

существовавшего, но потом заброшенного плагина к Nautilus'у, который, как ни странно, назывался nautilus-terminal. Который, в свою очередь, был придуман в незапамятные времена, когда Nautilus утратил терминальное окно как свою встроенную функцию.

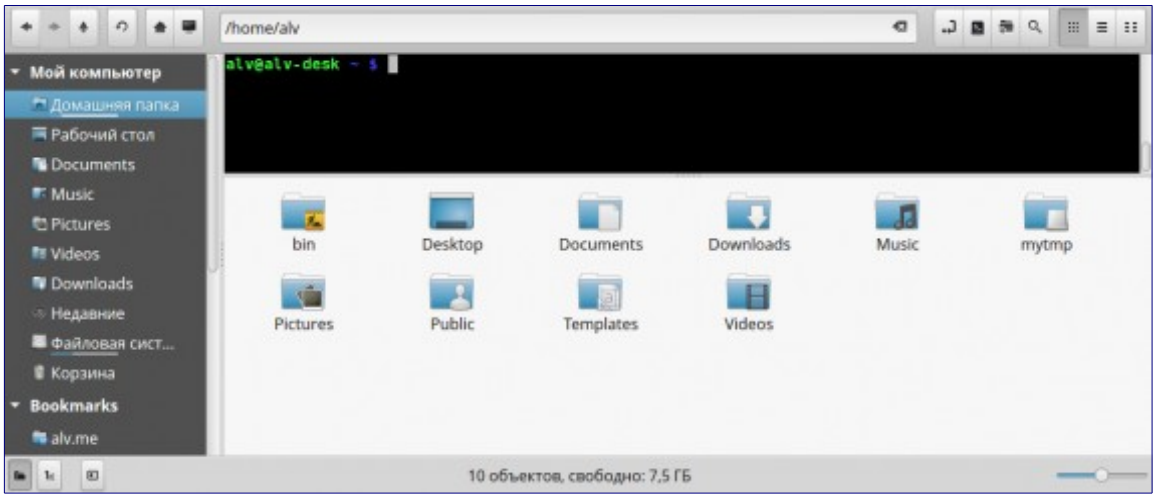
Пакет плагина nemo-terminal находится в официальном репозитории Mint, и потому ныне устанавливается стандартным образом, без всяких неожиданностей:

```
$ apt install nemo-terminal
```

После чего требуется «жёсткий» выход из Nemo, например, командой в терминале:

```
$ nemo -q
```

Запущенный в следующий раз, Nemo будет уже с терминальным окошком в верхней части рабочей области вполне уродливого вида:



Горячей клавишей F4 его можно скрыть с глаз долой и вызывать по необходимости. А чтобы терминальное окно не мозолило глаза при каждом запуске, достаточно убрать его клавишей F4 и повторить команду

```
$ nemo -q
```

И при следующем запуске Nemo окно его будет девственно чисто — о наличии терминала можно узнать, только опять нажав клавишу F4.

Никаких настроек для терминала не обнаруживается. Можно только мышью изменить высоту терминального окна — но лишь для запущенного экземпляра Nemo, при повторном его запуске оно опять будет восстановлено в исходном размере.

Теоретически конфиг nemo-terminal находится в каталоге /usr/share/glib-2.0/schemas/ и носит имя org.nemo.extensions.nemo-terminal.gschema.xml. Однако мои попытки изменить в нём что-либо (например, высоту окна по умолчанию) успехом не увенчались.

Поскольку «расширитель» nemo-terminal — это скрипт на Python'e, вероятно, всякие настройки по умолчанию можно изменить прямой правкой соответствующего файла — /usr/share/nemo-python/extensions/nemo_terminal.py, о чем будет сказано чуть

позже.

Командная оболочка в окне nemo-terminal — теоретически login shell данного пользователя, то есть в моём случае Zsh. По крайней мере, об этом говорил вывод команды

```
$ echo $SHELL

/bin/zsh
```

Но это был очень странный Zsh. В частности, он игнорировал все настройки в ~/.zshrc. Более того, в ответ на прямую команду

```
$ source ~/.zshrc
```

он выдавал ошибки буквально в каждой строке.

А в остальном, прекрасная маркиза, все функции терминала выполнялись исправно — то есть в нём можно было вводить всякие разные команды. При смене каталога в основной панели Nemo происходила смена его и в окне терминала:



В терминальное окно можно было перетаскивать мышью каталоги и файлы. В первом случае это было эквивалентом команды cd — и тут уже с синхронизацией пути в командой строке и основной панели. Файлы же открывались в той программе, которая закреплена за ними по умолчанию: текстовые файлы — в текстовом редакторе, html-файлы — в браузере, файлы изображений — в графическом вьювере, и так далее.

Проблема же с неправильным поведением командной оболочки была решена Станиславом Шрамко aka stanis. Да, действительно, оказалось, что нужно чуток отредактировать файл /usr/share/nemo-python/extensions/nemo_terminal.py, а конкретно — вот эту его секцию

```
def terminal_or_default():
    """Enforce a default value for terminal from GSettings"""
    terminalcmd = settings.get_string("terminal-shell")
    if (terminalcmd == "") or (terminalcmd is None):
```

```
        terminalcmd = Vte.get_user_shell()

    return terminalcmd
```

Вписав туда (в любимом текстовом редакторе от лица администратора) после строки

```
    terminalcmd = settings.get_string("terminal-shell")
```

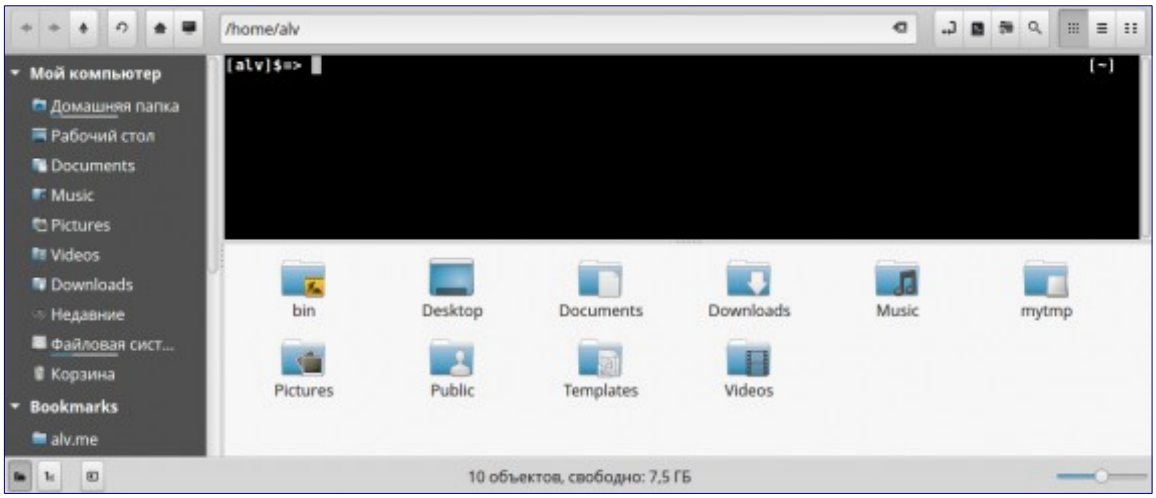
вот это:

```
    terminalcmd = ""
```

Затем — «жёсткое» завершение работы Nemo:

```
$ nemo -q
```

И при следующем запуске этого файлового менеджера в его терминальном окне красуется Zsh именно в том виде, до которого я его доводил годами. Что любопытно — после описанной процедуры nemo-terminal стал реагировать и на ручные изменения своего конфига. В частности, высота окна его увеличилась с пяти умолчальных строк до десяти, которые я раньше тщетно пытался ему внушить:



В общем, nemo-terminal не превращает Nemo в Dolphin, но в любом случае лучше хоть какой-то терминал, чем вообще никакого. Тем более, что работа над его совершенствованием будет продолжена. А пока его далёкий от эстетического совершенства вид можно скрывать, вызывая терминальное окно только при необходимости.

Некоторые расширения Nemo

Пакет nemo-terminal — не единственный из «расширителей» этого файлового менеджера (nemo-extensions). С полным их списком можно ознакомиться, например, с помошью конструкции примерно такого вида:

```
$ apt search nemo | grep " nemo-"
```

В которой следует не забыть про пробел после открывающей кавычки — иначе в выводе будет много лишнего. А так он сведётся к списку из примерно 30 строк:

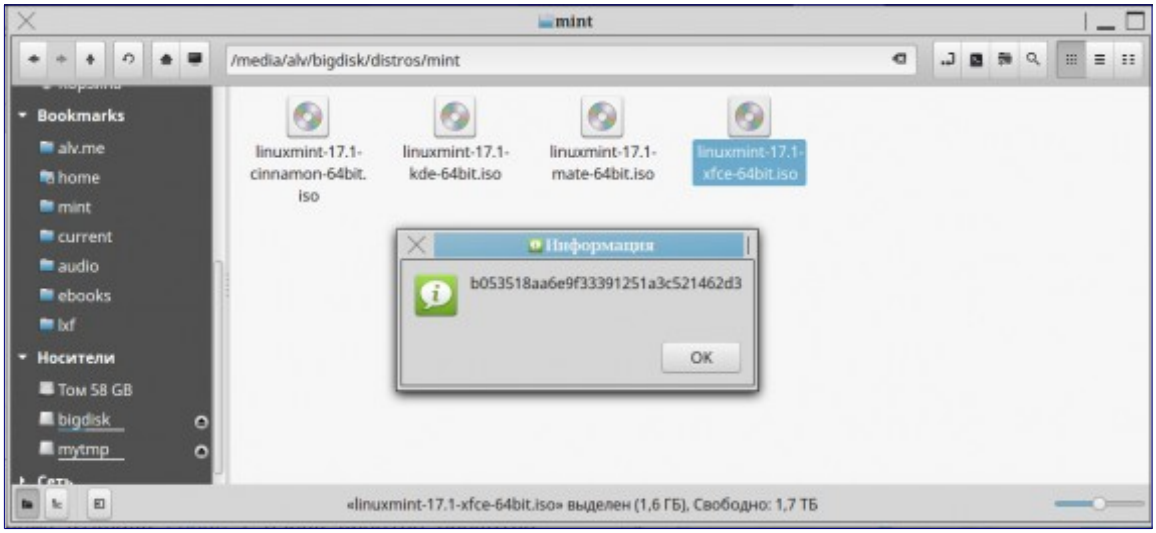
```
p   nemo-compare                - Context menu comparison extension for Nemo
i   nemo-data                   - data files for nemo
p   nemo-dbg                    - file manager and graphical shell for Cinna
p   nemo-dbg:i386               - file manager and graphical shell for Cinna
...
i   nemo-terminal               - Nemo extension to enable an embedded termi
p   nemo-terminal:i386          - Nemo extension to enable an embedded termi
```

Который, кстати, можно ещё сократить, отсортировав пакеты для ненужной архитектуры (в моём случае — для i386) довольно неуклюжей (лучше не придумал) конструкцией:

```
$ apt search nemo | grep " nemo-" | grep -v i386
p   nemo-compare                - Context menu comparison extension for Nemo
i   nemo-data                   - data files for nemo
p   nemo-dbg                    - file manager and graphical shell for Cinna
p   nemo-dropbox                - Dropbox integration for Nemo
i   nemo-emblems                - Change a folder or file emblem
p   nemo-filename-repairer      - Nemo extension for filename encoding repai
i   nemo-fileroller             - File Roller integration for Nemo
i   nemo-folder-color-switcher  - Change a folder color
p   nemo-gtkhash                - nemo extension for computing checksums and
p   nemo-image-converter        - nemo extension to mass resize or rotate im
p   nemo-keyboard               - pure QML keyboard for the Maliit framework
p   nemo-media-columns          - Nemo Extension
p   nemo-pastebin               - Nemo extension to send files to a pastebin
p   nemo-preview                - nemo-preview is a quick previewer for nemo
p   nemo-rabbitvcs              - Nemo extension for RabbitVCS
p   nemo-seahorse               - seahorse plugins and utilities for encrypt
i   nemo-share                  - Nemo extension to share folder using Samba
i   nemo-terminal               - Nemo extension to enable an embedded termi
```

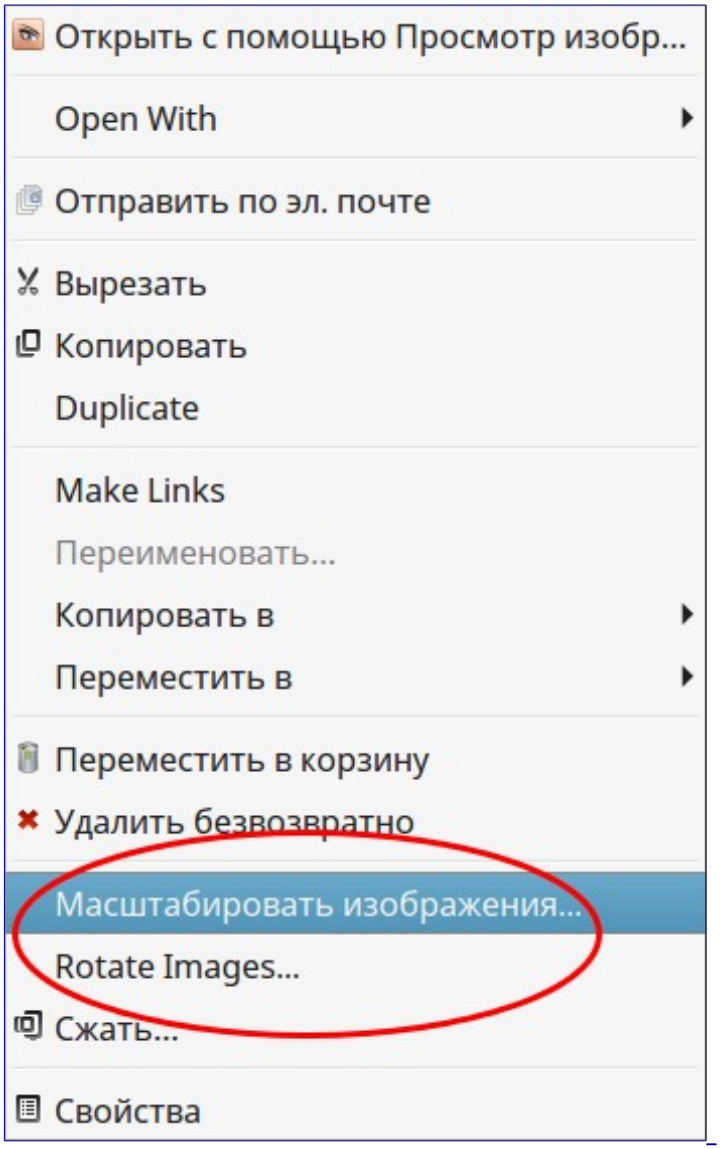
Большинство «расширителей», не установленных по умолчанию, как зависимости пакета nemo (например, nemo-emblems — это тоже «расширитель»), относятся ко всяким средствам разработки, а nemo-terminal мы только что установили собственноручно. Однако и среди оставшихся простой советский применитель может выискать кое-что для себя полезное.

В этом массиве полезностей — nemo-gtkhash, очень простое средство вычисления check-сумм, добавляющее соответствующий пункт в контекстное меню Nemo. Вроде бы ничего особенного — руки не отваляться дать соответствующую команду в CLI. Однако есть ситуации, когда этот «расширитель» оказывается удобней. Вот одна из них:

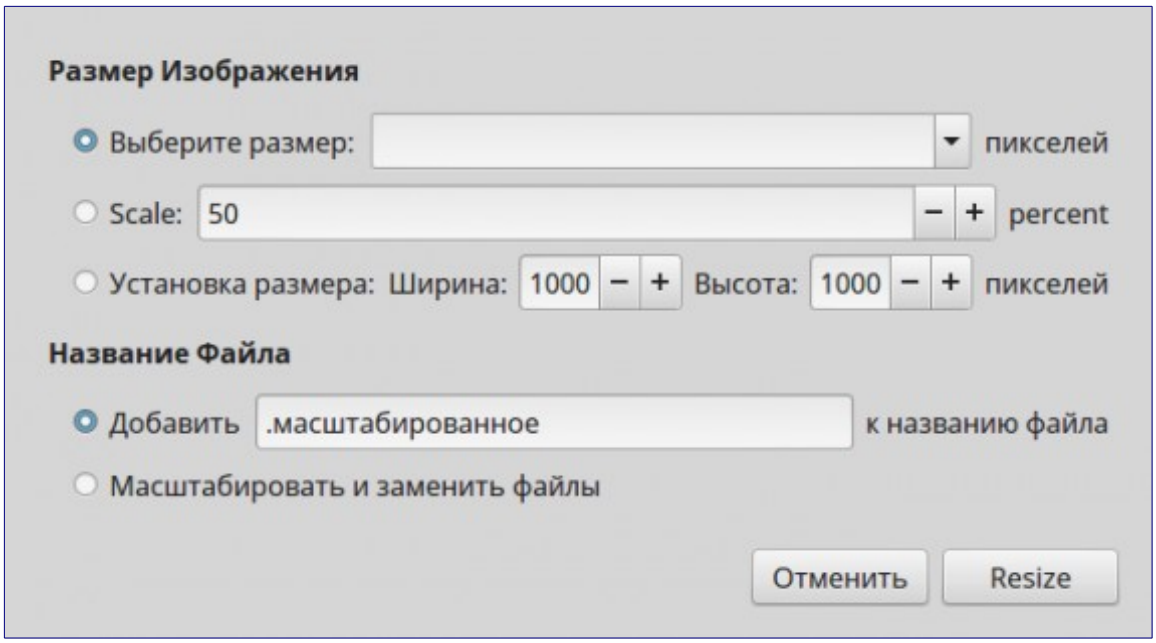


Ну вы меня поняли, ага?

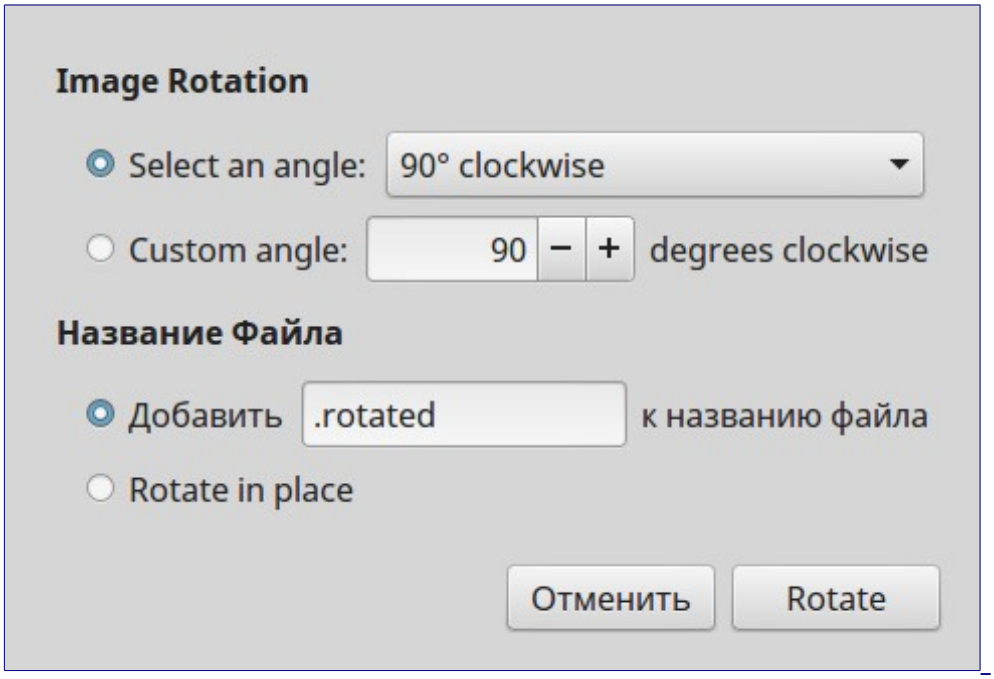
Далее, полезным может оказаться «расширитель» nemo-image-converter, предназначенный для массовой обработки графических файлов — их масштабирования и вращения. Он также встраивается в контекстное меню Nemo. Например, если выделить в текущем каталоге несколько PNG-файлов (или любых других файлов растровых изображений), и затем щелкнуть правой кнопкой мыши, то в появившемся контекстном меню можно будет увидеть пункты Масштабировать изображения... и Rotate Images...:



Первый, как это ни парадоксально, обеспечивает именно масштабирование картинок. А каким образом это может происходить — становится понятно при беглом вгляде на скриншот вызываемой им панели:



Точно так же, и столь же прозрачно, действует ротация, что видно на соответствующем скриншоте:



К которому остаётся разве что добавить, что вращать изображения можно на 90 градусов посолонь и противусолонь, на 180 градусов, а также на произвольные углы с шагом в один градус.

И ещё: разумеется, масштабирование и вращение применимы и к единичному изображению. Однако наибольшую пользу они принесут в случае, когда надо сотни скриншотов вписать в формат web-страницы. Или массив отснятых фотографий перевести из портретной ориентации в альбомную (или наоборот). А для этих целей данный «расширитель» кажется мне очень востребованным.

А вот действие nemo-filename-repairer, напротив, распространяется только на каталоги — лишь при правом клике на имени каталога в контекстном меню появляется пункт Repair filename.... Вот только, увы, у меня в системе нет ни одного каталога с именем в неправильной кодировке, и вообще практически нет файлов с именами, отличными от чистого ASCII, так что проверить, как это всё действует, не могу. А потому оставляю освещёние этого вопроса заинтересованным лицам. Как и знакомство с прочими «расширителями», не окученными ранее. Хотя об одном из них, nemo-dropbox, расскажу в следующем разделе.

Nemo и Dropbox

Назначение «расширителя» nemo-dropbox, как следует из его имени — обеспечить интеграцию Nemo с соответствующим облачным хранилищем. И делает он это так. Сразу после завершения установки, например, командой

```
$ apt install nemo-dropbox
```

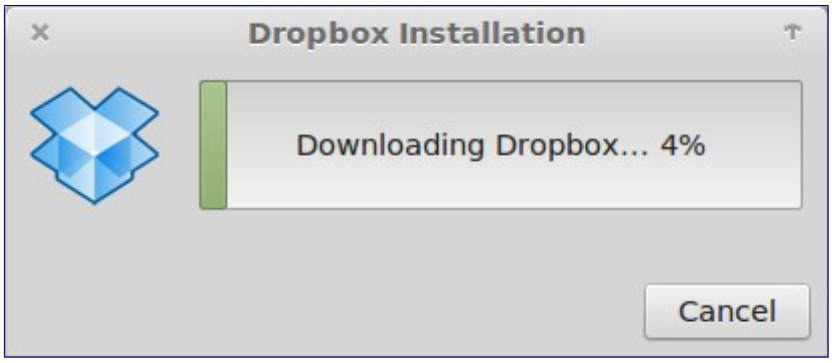
предлагается «вчистую» закрыть все, возможно, открытые экземпляры Nemo:

```
$ nemo -q
```

А вслед за тем запустить из главного меню Cinnamon программу, которая так и называется — Dropbox, находится в секции Интернет и вызывает для начала свой собственный инсталлятор:



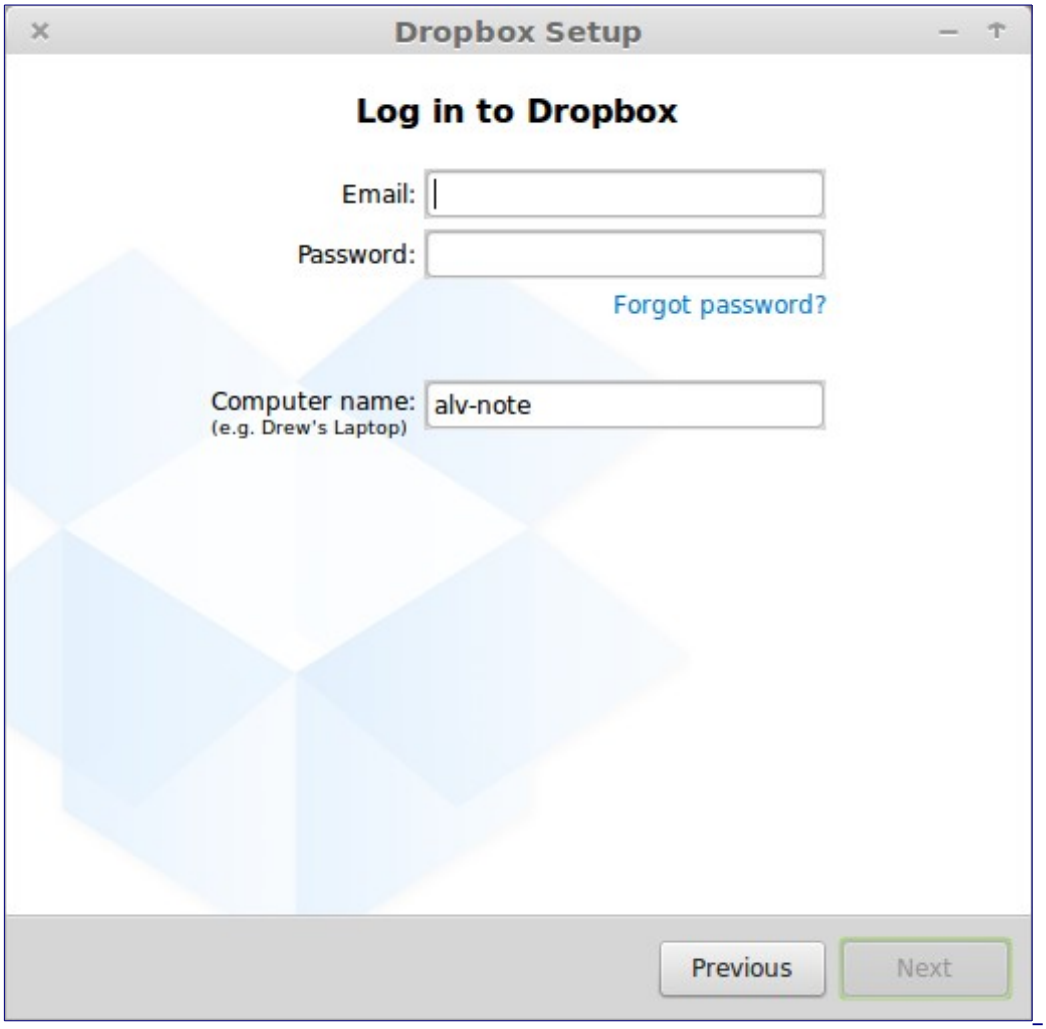
Программа скачивается:



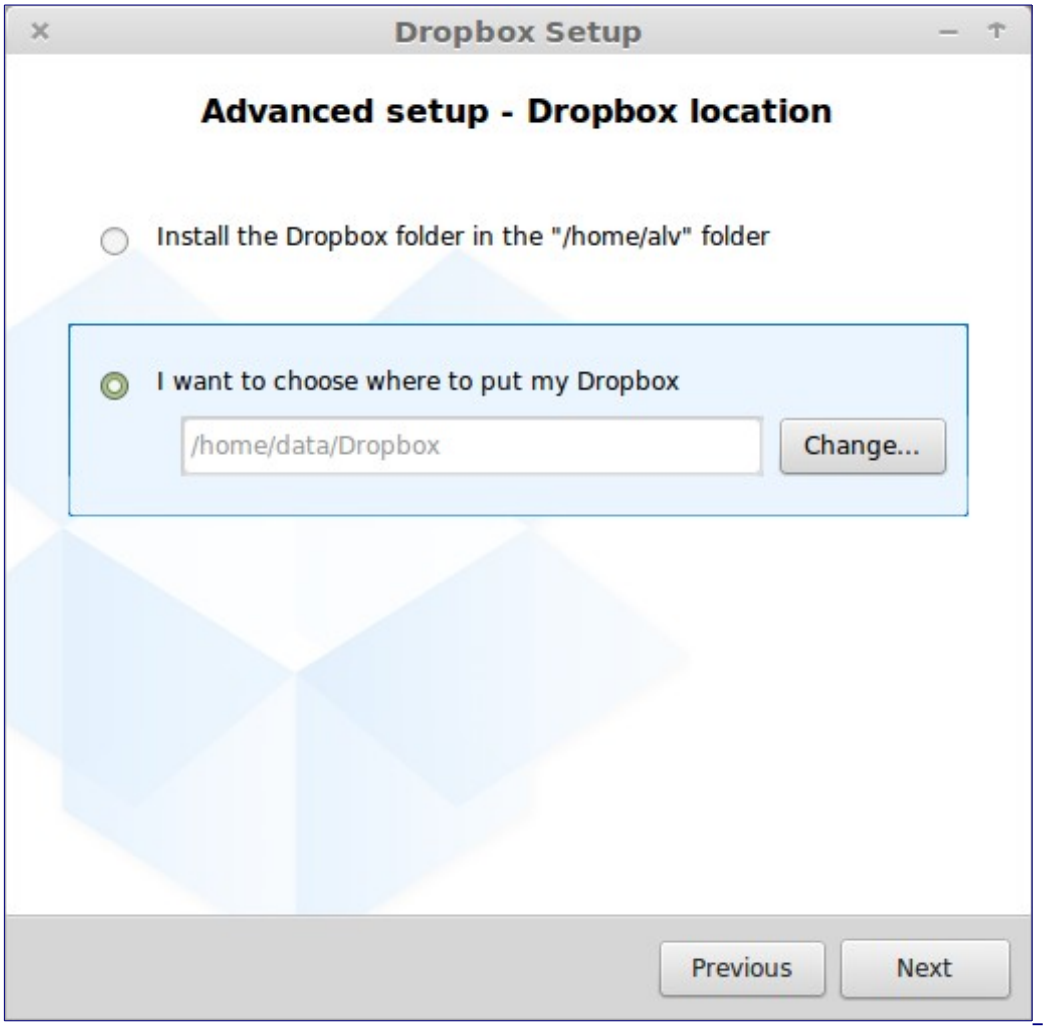
А затем вызывается панель её установщика:



Поскольку аккаунт Dropbox'а у меня существует с незапамятных времён (хотя я им почти не пользуюсь), ввожу свои учётные данные



Жму кнопку Next и жду установления коннекта. После чего совершаю выбор типа установки. Рекомендуемый Typical создаст каталог для синхронизации с Dropbox'ом в моём домашнем. Это меня по некоторым причинам ни в коем случае не устраивает, поэтому приходится выбирать Advanced, хотя ничего такого авантажного мне не требуется. Так что следующим шагом выбираю подходящее место для каталога Dropbox:



Далее по умолчанию предлагается синхронизировать все каталоги Dropbox'а. Это мне тоже ни к чему, поэтому меняю на каталоги по собственному выбору. В ответ программа хочет провести среди меня разъяснительную работу, объяснив, что такое каталоги вообще и каталог Dropbox в частности:



Отказываюсь, нажав кнопку Skip Tour. В ответ предлагается финишировать, открыв при это каталог Dropbox'а:

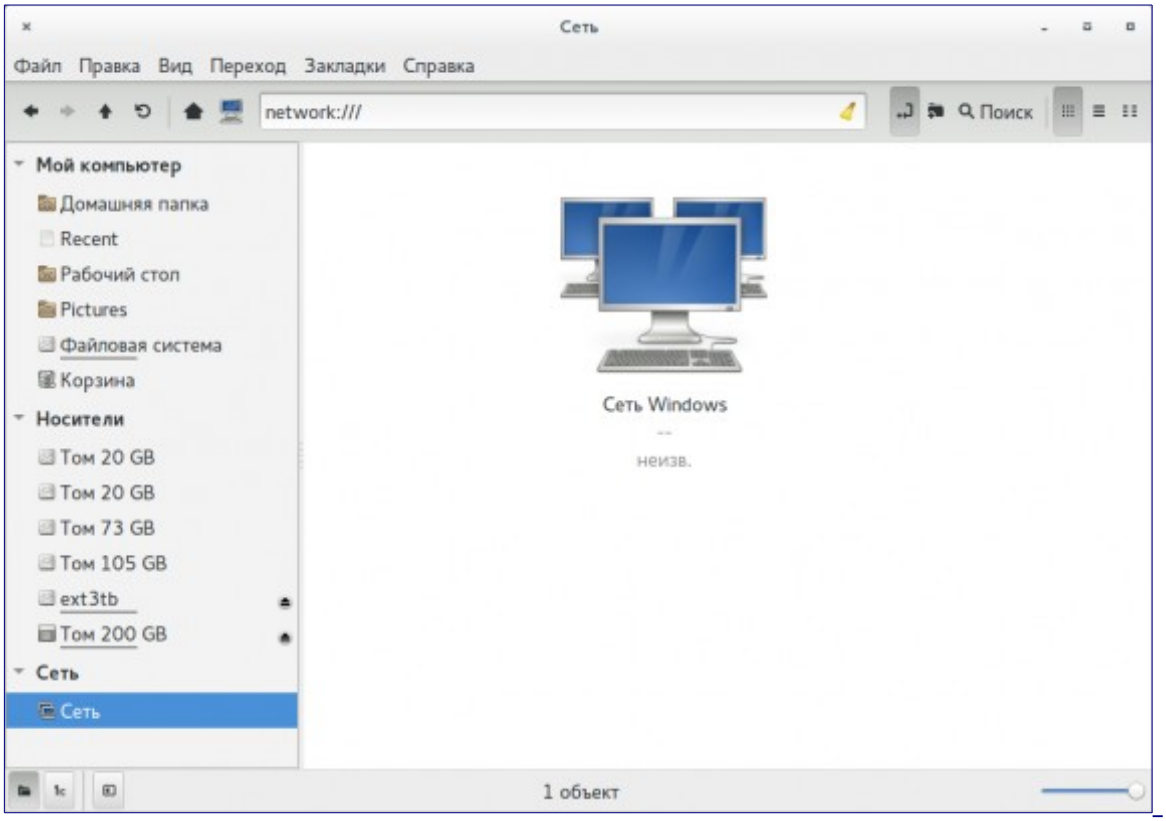


Запускается Nemo, в нём открывается каталог /home/data/Dropbox, а в него очень быстро помещается то, что лежало у меня в Drop'локе (это я только что неологизм придумал).

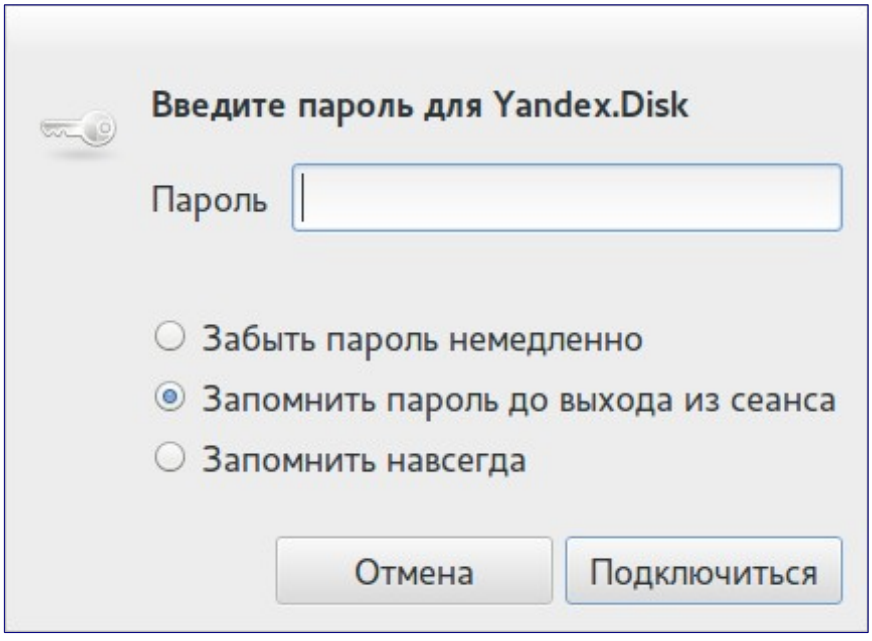
Все описанные события происходили на Ноутбучке, а настоящий текст сочинялся параллельно на большой машине. Так что собираю все сделанные скриншоты (те, что были приведены выше) и отправляю их в Drop'лако. Поскольку на большой машине описанныя процедура была проделана ещё раньше, скриншоты через посредство Drop'лака уже там. Дописываю последний абзац этого мини-очерка — и занимаюсь вставкой в него иллюстраций. Результат перед вами.

Nemo и Яндекс.Диск

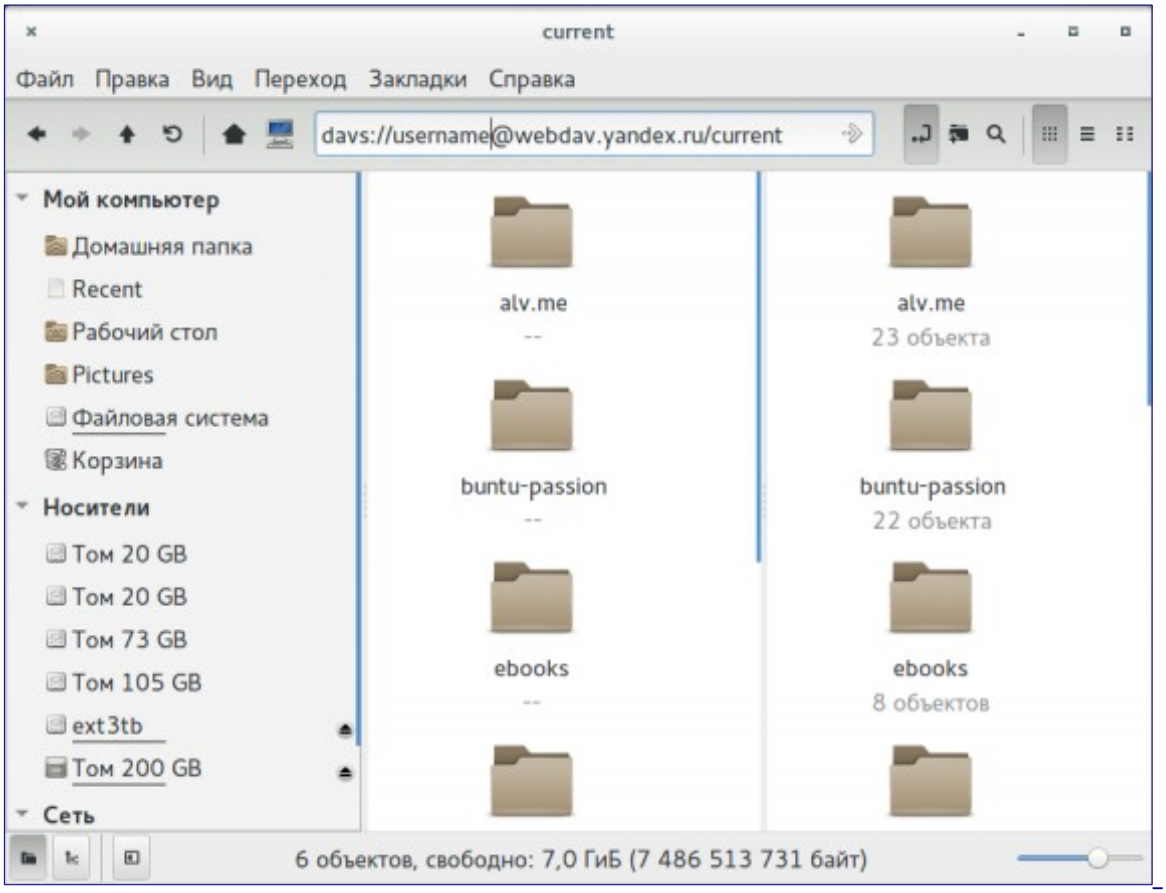
Только что мы разобрались с ихним буржуазным облачным сервисом Dropbox'ом. Но ведь и мы не лыком шиты — у нас есть его собрат по поднебесному ремеслу, Яндекс.Диск, который подключается ничуть не сложнее. И даже проще, поскольку задача сводится к настройке подключения по WebDAV. Для чего нужно отправиться в боковую панель, найти там раздел Сеть, а в нём — закладку Сеть, на которой и щёлкнуть, чтобы получить вот такую картину:



Нажимается Enter. В появившейся панельке авторизации вводится пароль доступа к сервисам Яндекса:



Всё — подключение свершилось. Теперь между локальными дисками и Яндекс.Диском можно взаимодействовать через Copy&Paste. А можно клавишей F3 включить двухпанельный режим:



И проникнуться его полезностью — файлы и каталоги можно просто таскать мышью туда и сюда.

Справедливости ради следует сказать, что у Яндекса имеется и свой клиент для работы с ихним диском, причём и в версии под абстрактный Linux. Однако по ряду причин я его не использую, и потому ничего о нём сказать не могу.

Программы эмуляции терминала

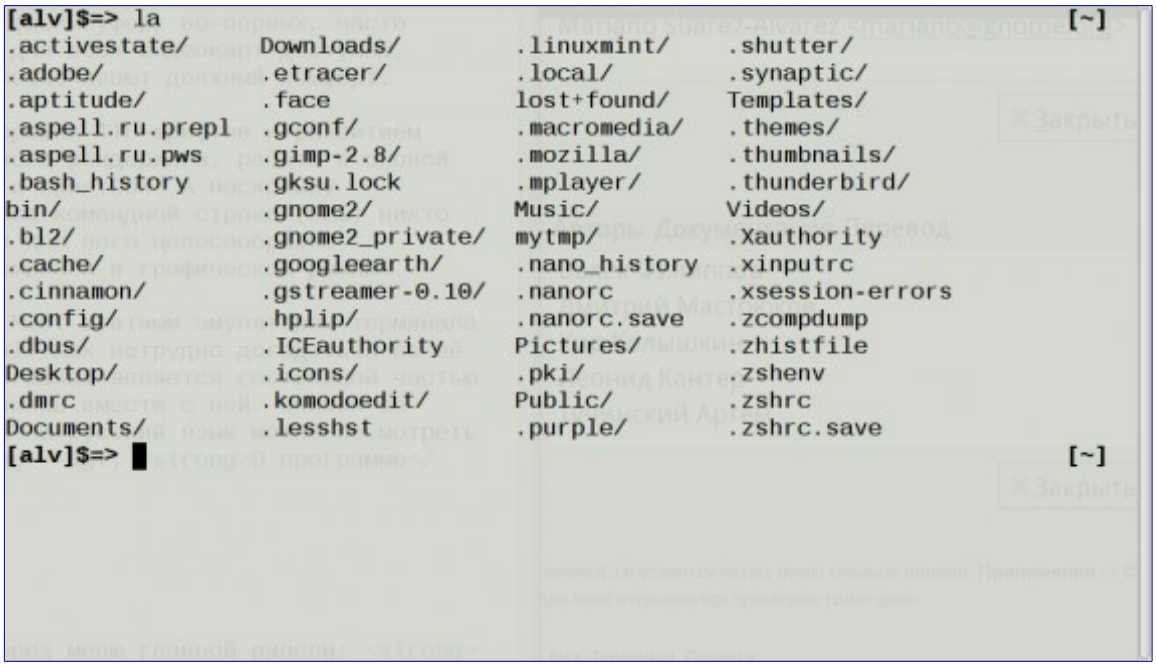
Роль терминальных программ в жизни современного применителя Linux переоценить трудно. Это связано с постепенным отмиранием чисто текстовой консоли — ведь давно минули времена, когда она обеспечивала больший комфорт для глаз, нежели любой графический режим. На нынешних LCD-мониторах, да ещё и широкоформатных, стандартный текстовый режим 80x25 способен лишь вышибить скупую мужскую слезу (иногда — в буквальном смысле слова). А режимы нестандартные, реализуемые через фреймбуфер, во-первых, часто требуют настройки, во-вторых, не для всех видеокарт доступны, в-третьих, всё равно не всегда обеспечивают должный комфорт.

С другой стороны, с появлением хороших TTF-шрифтов и развитием методов обеспечения их качественного рендеринга, работа в оконой системе X стала очень комфортной и приятной. А поскольку эффективность применения интерфейса командной строки (CLI) никто ещё не отменил — в качестве поля для него целесообразно использовать именно эмуляторы терминала в графическом режиме.

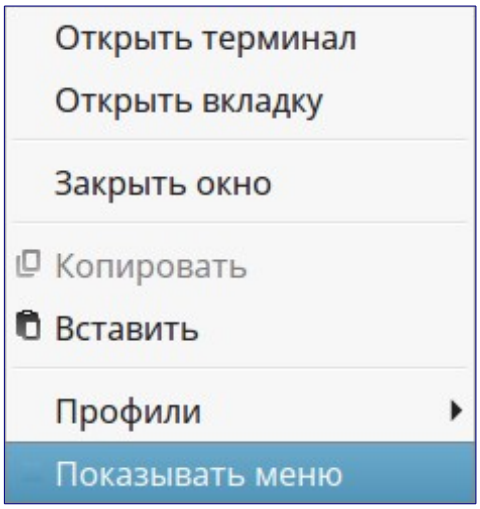
GNOME Terminal

В Cinnamon-редакции Mint штатным эмулятором терминала выступает программа GNOME Terminal. Как нетрудно догадаться по её имени, она заимствована из среды GNOME (с адаптацией для Cinnamon). И в виде пакета

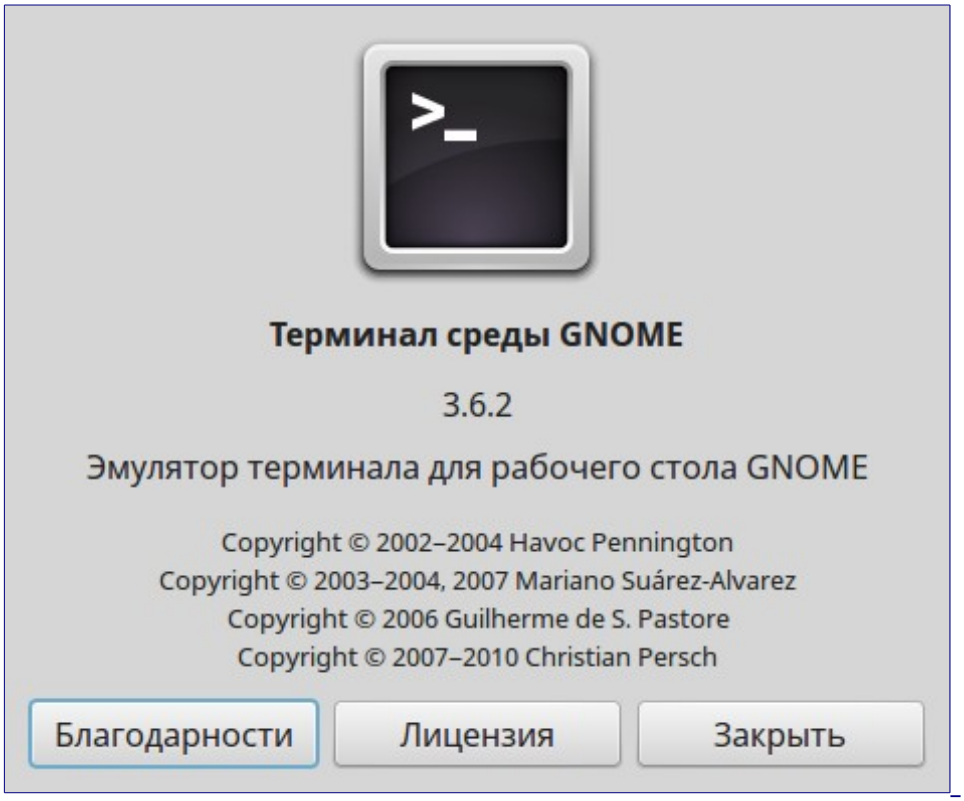
gnome-terminal устанавливается при стандартной инсталляции дистрибутива. После чего запускается через главное меню: Стандартные -> Терминал, открываясь примерно в таком виде:



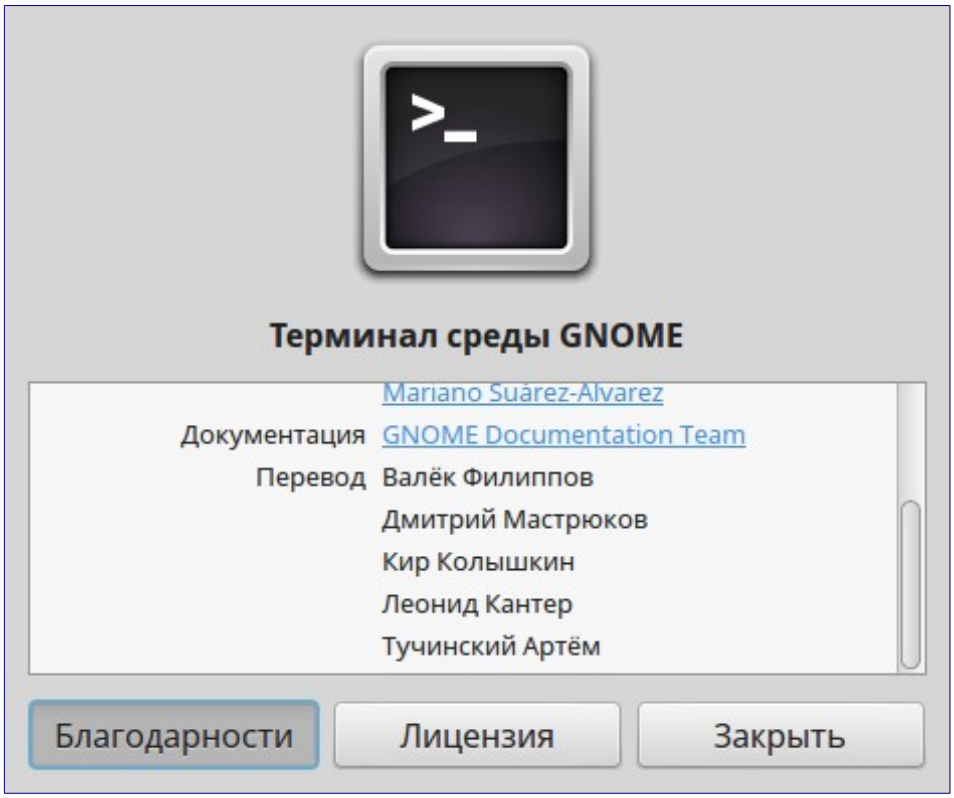
Обращаю внимание на отсутствие строки главного меню по умолчанию. Включить показ его можно через меню контекстное, по правому клику мыши:



Оно включает следующие пункты: Файл, Правка, Вид, Терминал и упомянутая ранее Справка. Через последний пункт, в подпункте О программе, можно посмотреть список её авторов:



И переводчиков интерфейса на русский язык:



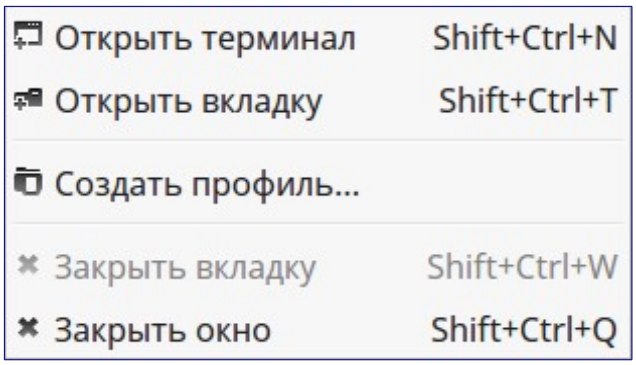
А по остальным пунктам можно оценить возможности GNOME Terminal.

В меню Файл присутствуют следующие пункты:

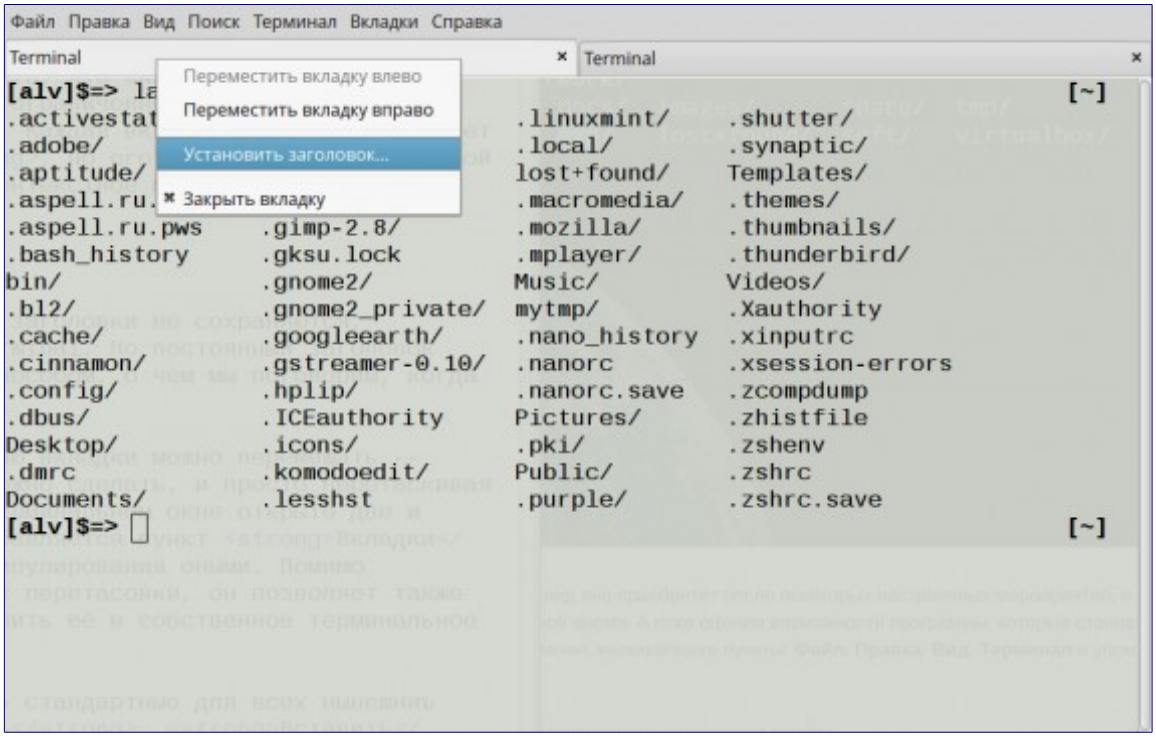
- Открыть терминал — создание нового терминального окна;
- Открыть вкладку — создание вкладки (tab) в текущем окне, в которой запускается собственный экземпляр командной оболочки пользователя;

• Создать профиль — об этом мы поговорим, когда займёмся настройками терминала;

• Закрыть вкладку и Закрыть окно, смысл которых очевиден.

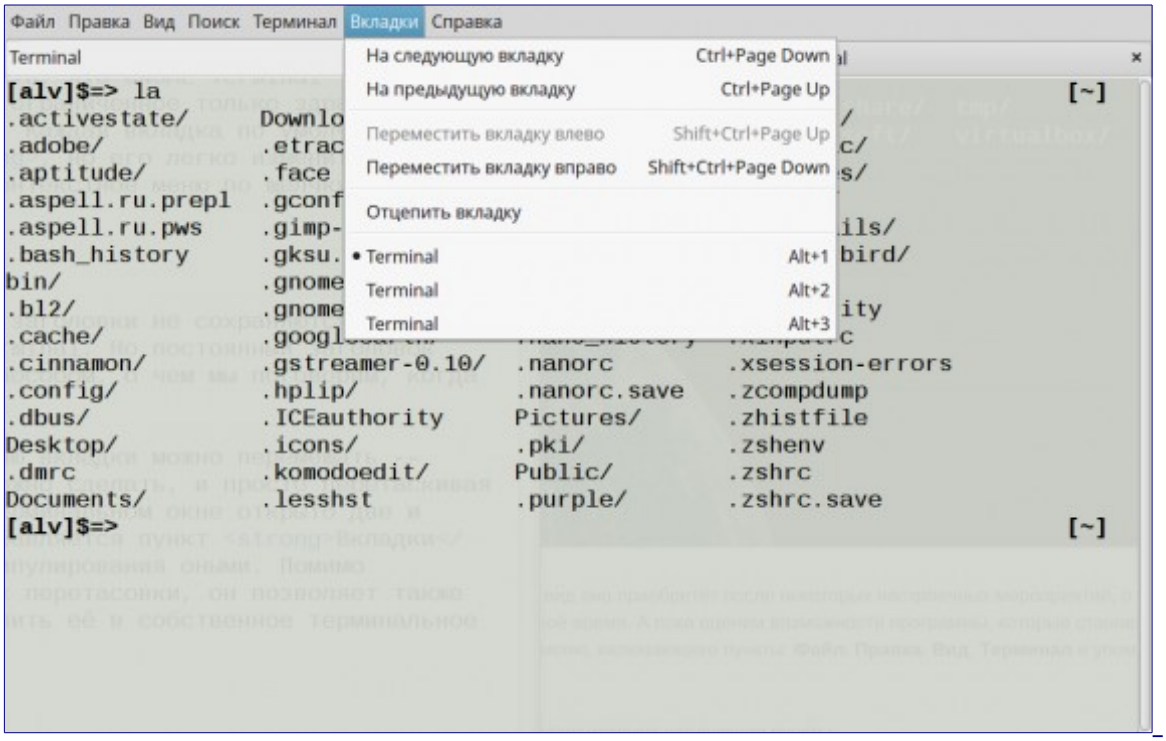


Здесь пока остаётся только добавить, что GNOME Terminal поддерживает количество вкладок, ограниченное только здравым смыслом и соображениями удобства. Каждая вкладка по умолчанию имеет заголовок Terminal, но его легко изменить на любой мнемонически осмысленный через контекстное меню по щелчку правой кнопкой мыши на табе:

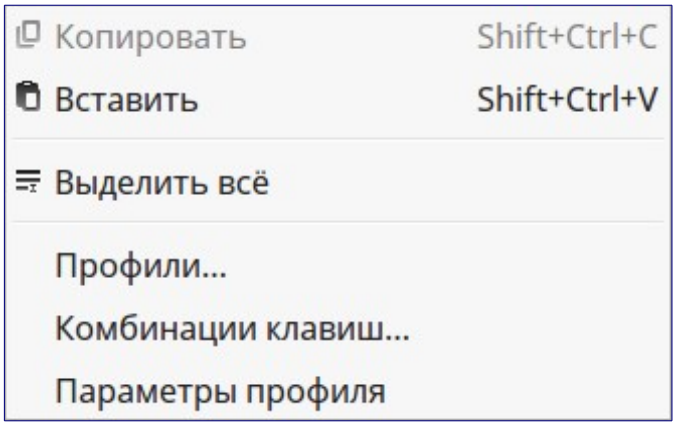


Правда, при перезапуске заданные заголовки не сохраняются, восстанавливаясь как безликий Terminal. Но постоянный заголовок можно приписать вкладке другим способом, о чем мы поговорим, когда займёмся настройками.

Через то же самое контекстное меню вкладки можно перемещать — влево или вправо. Впрочем, это можно сделать, и просто перетаскивая любой таб мышью. Далее, если в терминальном окне открыто две и более вкладок, в главное меню добавляется пункт Вкладки, предназначенный для манипулирования оными. Помимо переключения между вкладками и их перетасовки, он позволяет также отцепить вкладку — то есть выделить её в собственное терминальное окно:

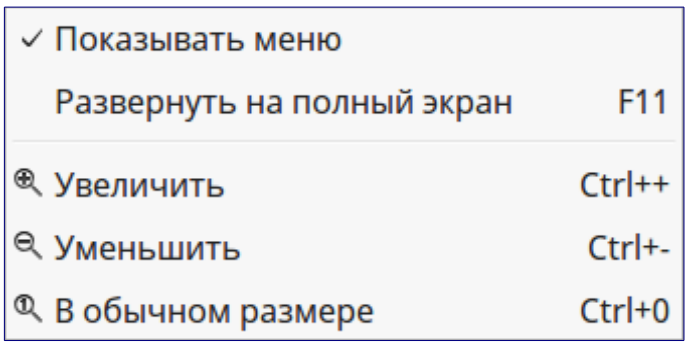


В меню Правка — стандартные для всех нынешних GUI'ёв пункты: Копировать, Вставить, Выделить всё. Назначение их своеобычное, так что и задерживаться на них не будем:



А о пунктах Профили, Комбинации клавиш, Настройки профиля поговорим, когда дело дойдёт до триариев... то есть до конфигурирования.

В меню Вид — пункты показа/скрытия меню, переключения в полноэкранный режим и обратно (по клавише F11) и масштабирования:



Масштабирование выполняется также обычными комбинациями клавиш — Control++, Control+- и Control+0 (увеличение, уменьшение и приведение к исходному размеру, соответственно).

В меню Терминал — такие пункты:

- Изменить профиль, рассмотрение которого пока отложим, тем более что он ещё не активизирован;

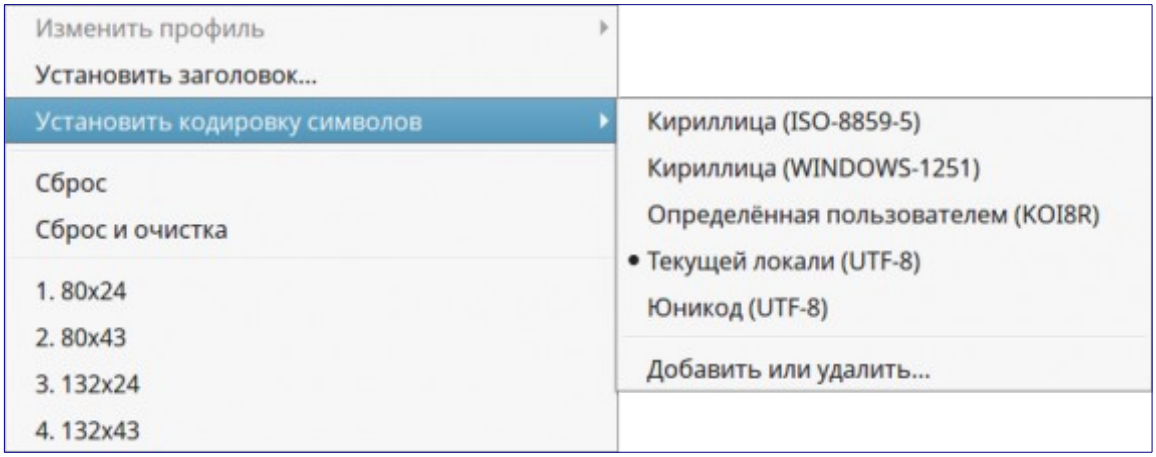
- Установить заголовок — действие, аналогичное таковому через контекстное меню вкладки;

- Установить кодировку символов — изменить чарсет вывода вместо определённого общесистемной локалью; чрезвычайно полезная опция, когда при юникодовской кодировке приходится читать старые тексты в KOI8 или CP1251;

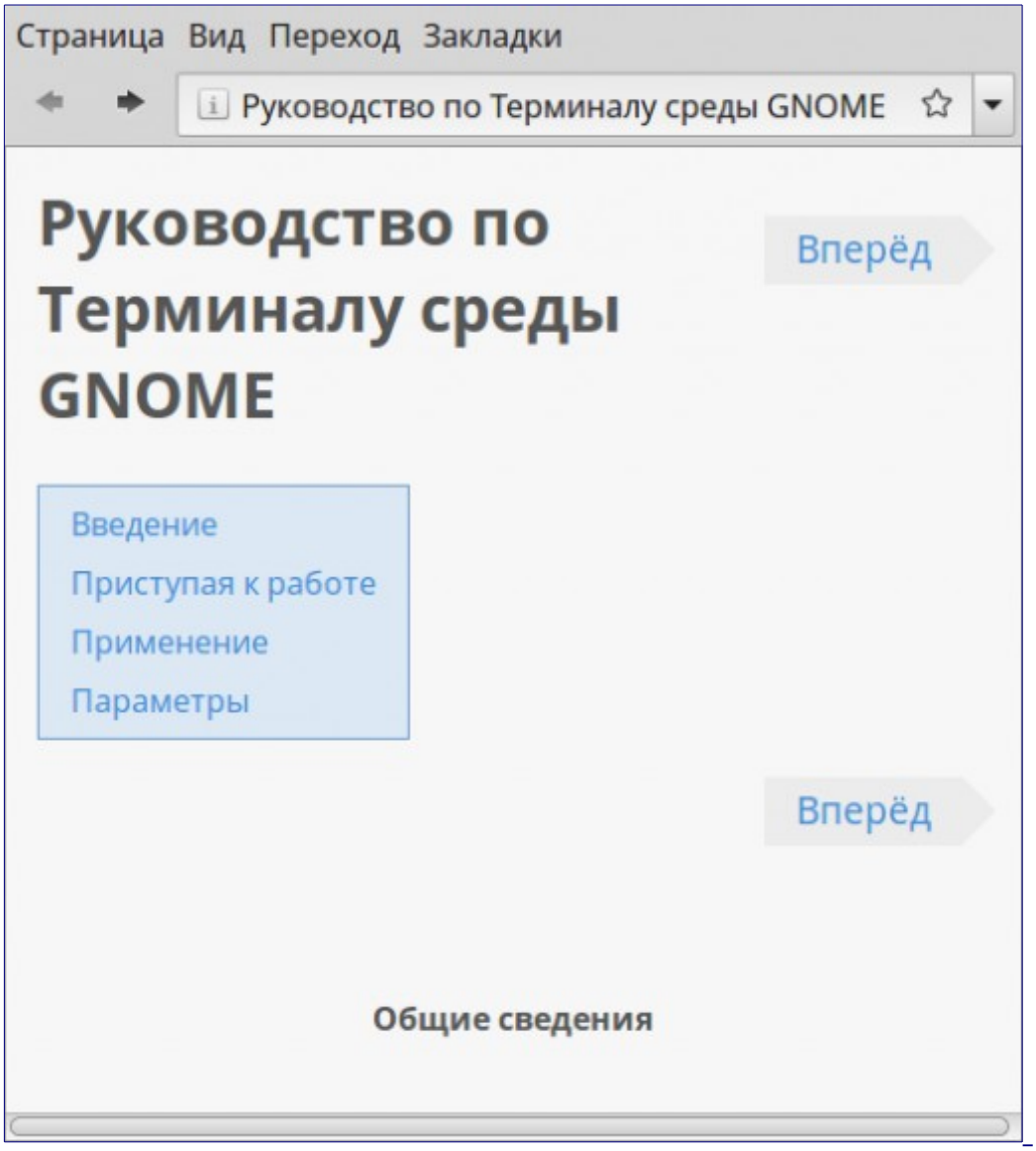
- Сброс — теоретически удаление текущего сождержимого командной строки, аналогично комбинации Control+C; практически никакого эффекта не оказывает;

- Сброс и очистка — кроме удаления содержимого командной строки (действительно работает), ещё и очищает экран от вывода предыдущих команд, аналогично действию команды clear.

Кроме того, в меню Терминал можно изменить размеры окна, выбрав одно из четырёх фиксированных значений (в символах): 80x24, 80x43, 132x24, 132x43. Впрочем, перемасштабировать терминальное окно произвольным образом с помощью мыши тоже не запрещается.

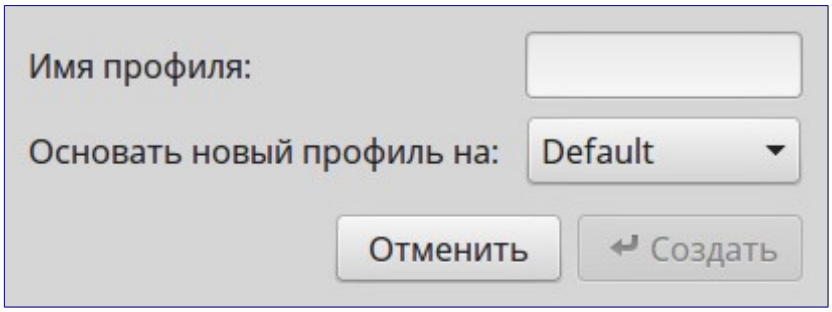


Ну а через меню Справка, кроме упомянутых ранее сведений о программе, можно вызвать и собственно руководство по программе:

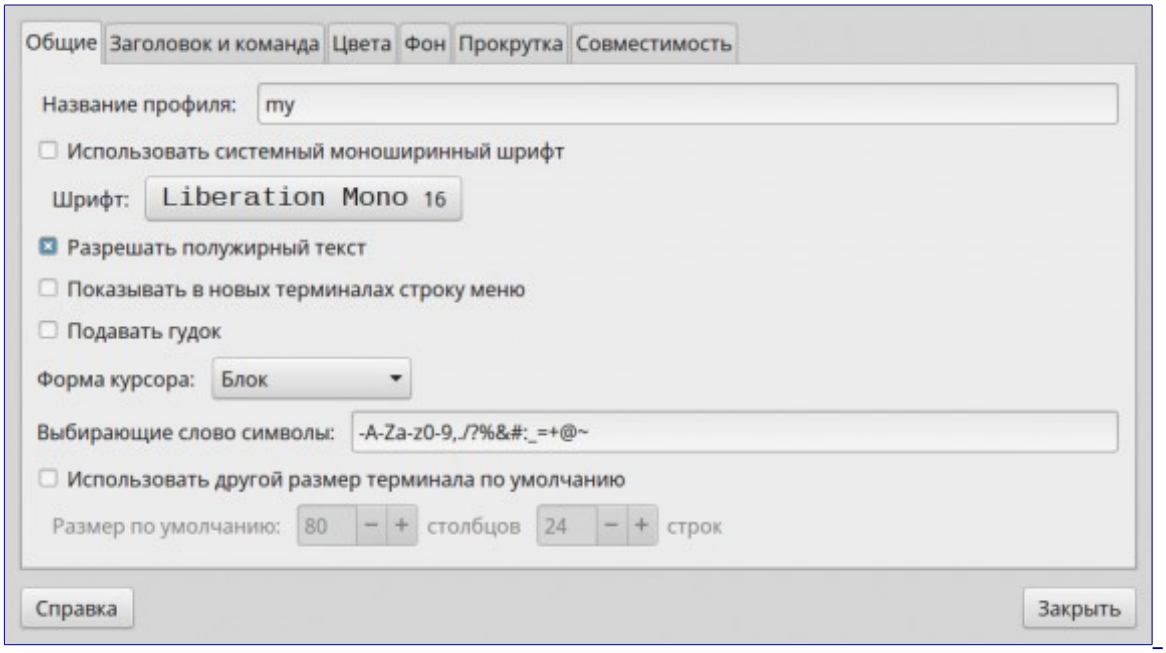


Удобство использования любой терминальной программы во многом определяется гибкостью и простотой её настроек. Посмотрим, что нам в этом отношении может предложить GNOME Terminal.

Все настройки терминала осуществляются через модификацию профилей в соответствующем пункте меню Правка. По умолчанию имеется всего один профиль, который так и называется - Default. Однако через меню Файл -> Создать профиль их можно определить сколько угодно:



После задания имени профиля, определения, на каком из существующих он будет основываться (а пока, кроме профиля Default, основываться больше не на чем), и нажатия кнопки Создать появляется собственно панель настройки, содержащая серию вкладок:



Как видно из скриншота, во вкладке Общие можно изменить имя профиля (если это необходимо), задать шрифт — системный, определённый в общих настройках Cinnamon, или любой произвольный, разрешив или запретив, заодно, полужирное его начертание, включить или выключить строку меню и звуковые сигналы (например, при ошибках), изменить форму курсора и задать символы — ограничители командного «слова», выделяемого двойным щелчком мыши, наконец, размер терминального окна.

Во вкладке Заголовок и команда определяются:

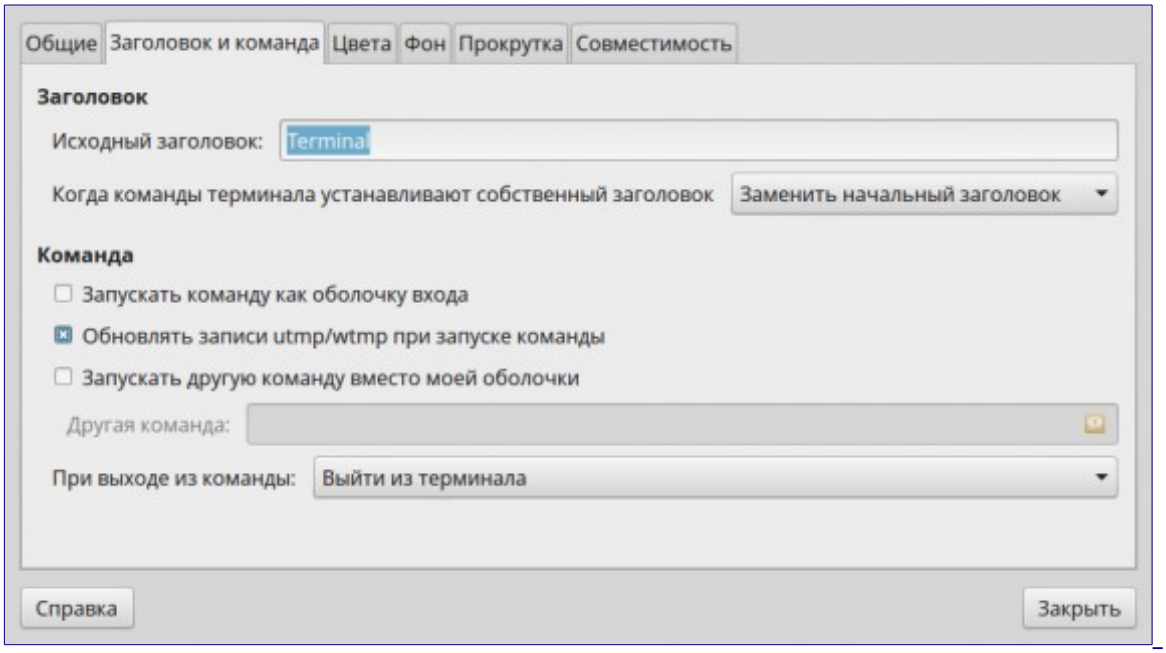
- исходный заголовок терминала;

- поведение заголовка в зависимости от устанавливаемого исполняемой программой — замещёние исходного, присоединение к нему, и так далее;

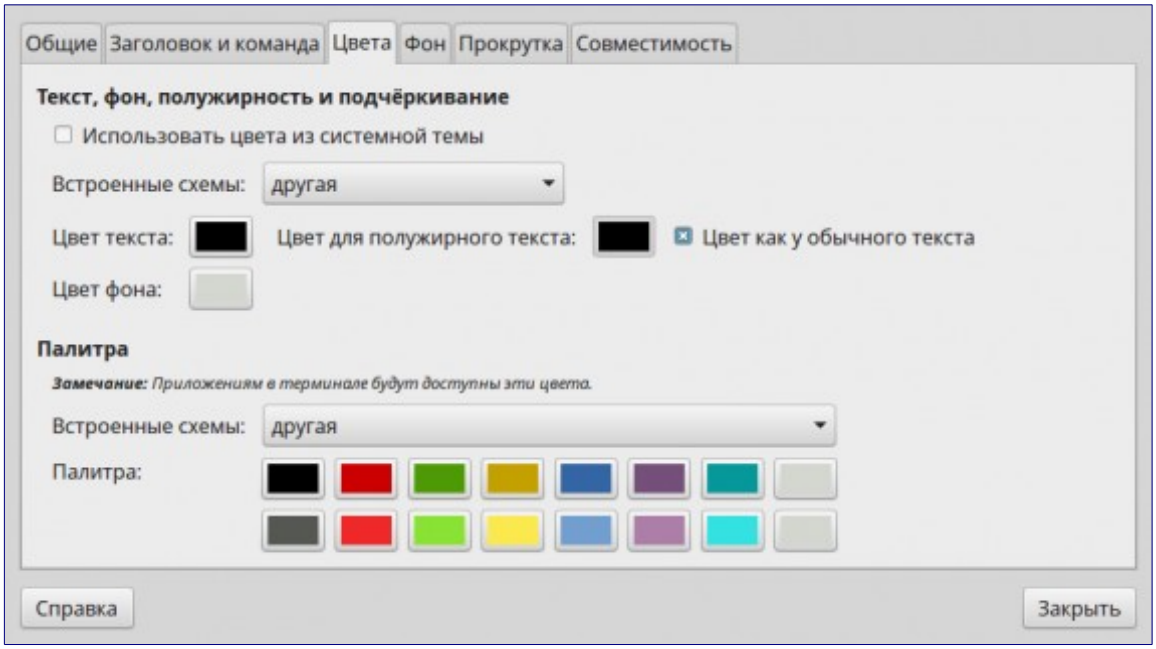
- запуск командной оболочки как обычной интерактивной или регистрационной оболочки пользователя (login shell); при обычных настройках bash разницы между ними почти нет (у меня так нет вообще), но при желании это положение можно изменить, особенно, если использовать не bash, а zsh;

- запуск иной команды вместо пользовательского шелла: так, для разработчиков это может быть интерпретатор любимого языка программирования, а для обычных пользователей — например, Midnight Commander;

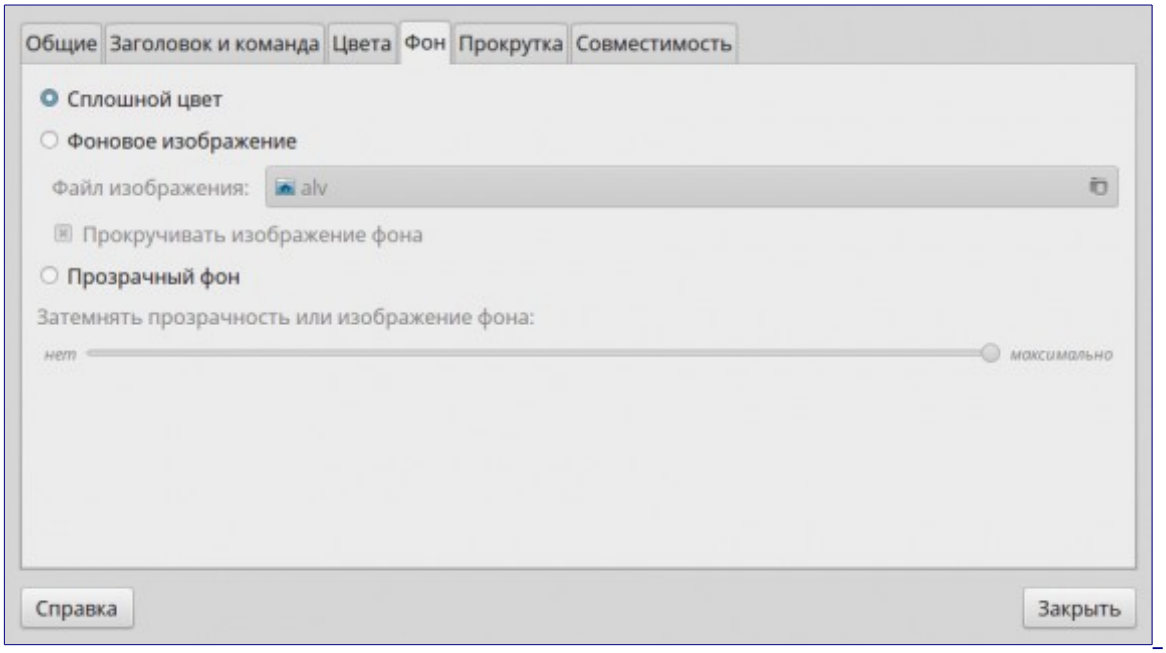
- поведение терминального окна по завершении исполняемой в нём команды — закрытие терминала или перезапуск команды.



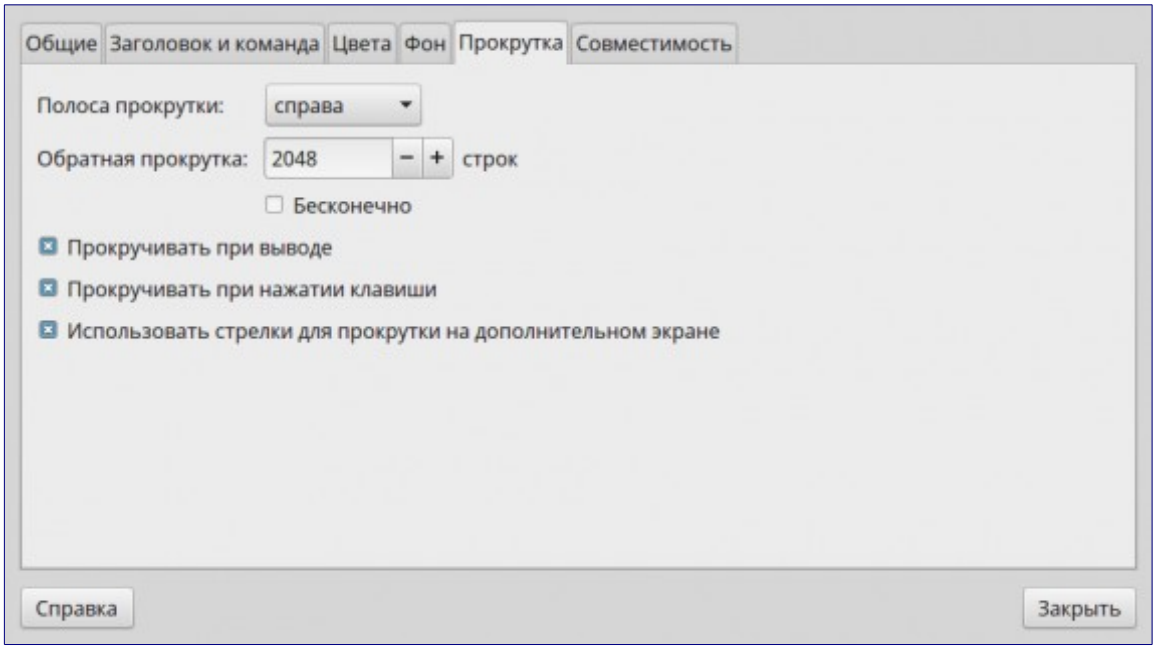
Во вкладке Цвета определяются цвет текста и фона — приводимый скриншот в комментариях не нуждается:



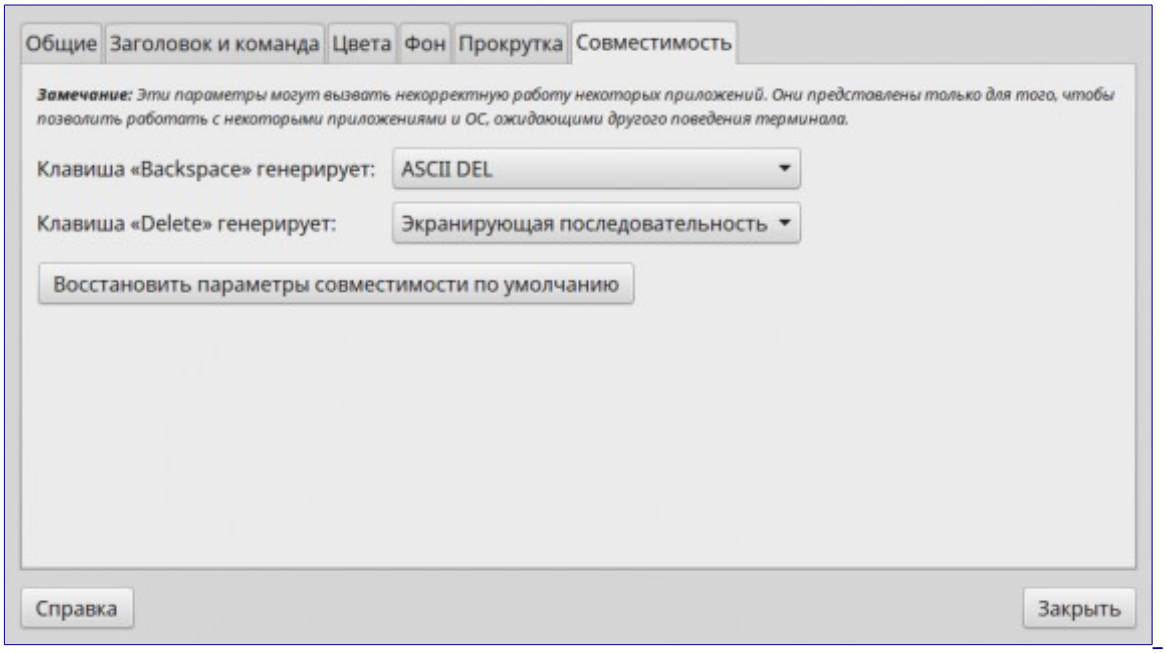
С вкладкой Тип фона также всё ясно — его можно сделать сплошным (тем, что был определён в предыдущей вкладке), задать фоновое изображение (в том числе и с прокруткой оного) или установить прозрачность — в этом случае сквозь терминальное окно будут просвечивать обои рабочего стола:



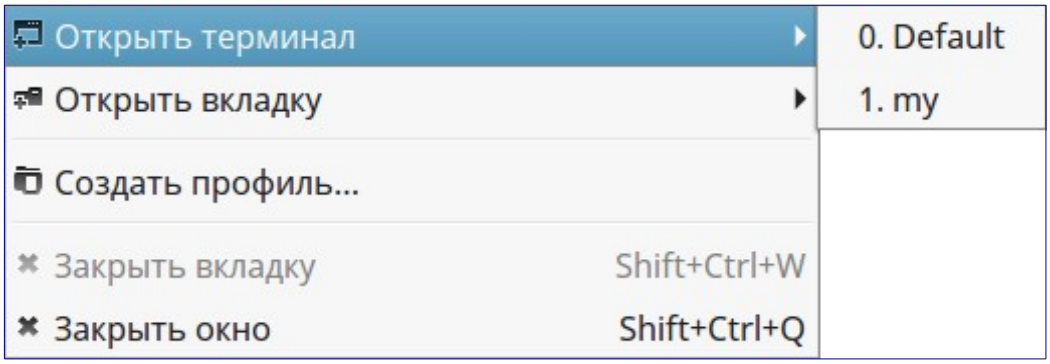
Во вкладке Прокрутка устанавливается положение полоски скроллинга и задаётся (в строках) длина прокручиваемой истории команд:



Наконец, во вкладке Совместимость можно переопределить назначение клавиш Backspace и Delete. Зачем это может понадобиться — исчерпывающе сказано в комментарии (как правило, не за чем):



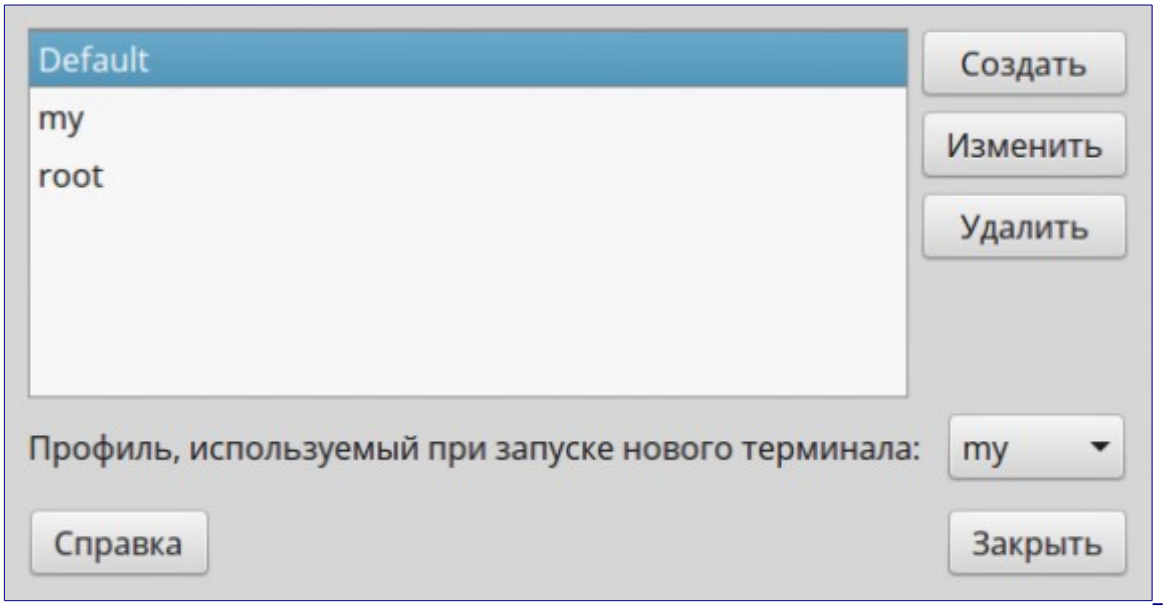
После создания дополнительных профилей их имена появляются как альтернативы выбора в меню Файл -> Открыть терминал — в новом терминальном окне будет запущен выбранный профиль:



Однако, как явствует из меню Файл -> Открыть вкладку, и в каждой вкладке одного и того же окна теперь может быть выбран тот или иной профиль, со всеми своими атрибутами — шрифтом, расцветкой, заголовком, типом фона и так далее:

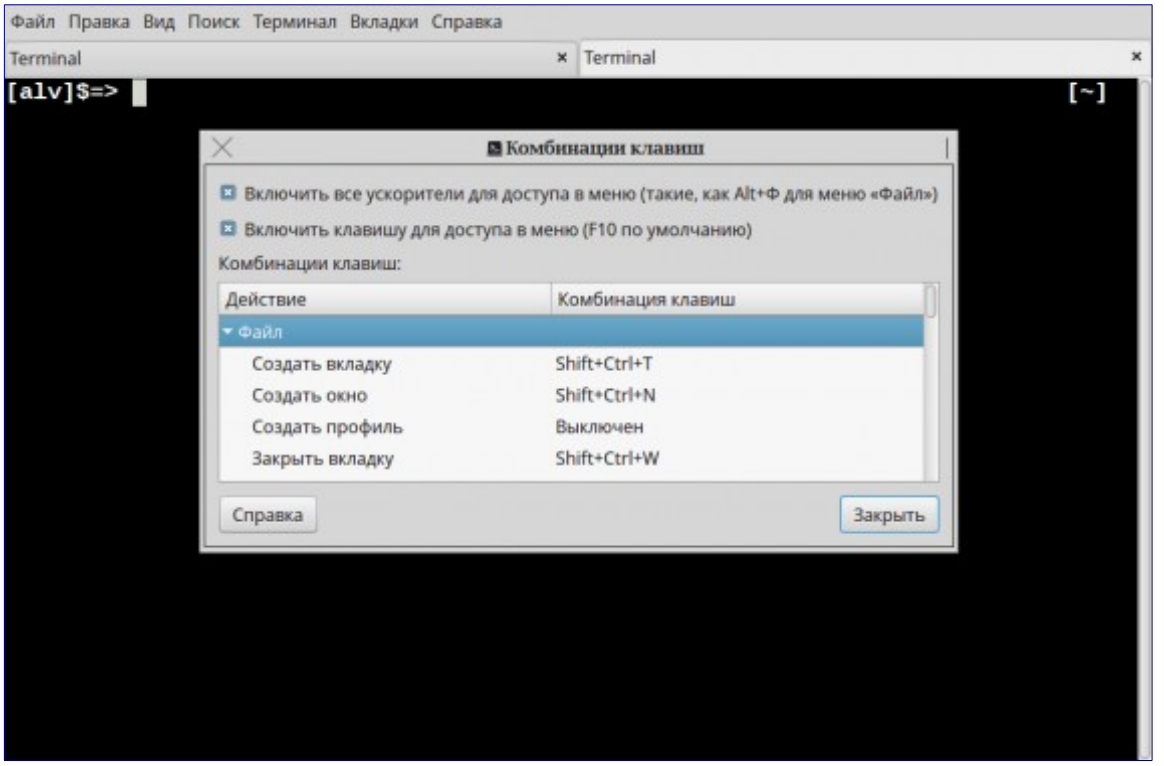


Это удобно для различения вкладок разного назначения. Так, я определяю разные параметры для обычного пользовательского профиля и профиля вкладки, в которой я обычно получаю права root'a (командой sudo -i). Список заданых профилей можно просмотреть через меню Правка -> Профили:



И, к слову сказать, выбрать профиль можно не только при открытии окна или вкладки, но и для уже открытых: это делается через главное меню Терминал -> Использовать профиль или меню контекстное. Перенастройка любого профиля также может быть выполнена в любой момент после его создания: через меню Правка -> Настроить профиль или через кнопку Изменить на панели списка профилей вызывается та же самая настроечная панель, что и при создании профиля.

Завершая разговор о настройках, обратимся к пропущенному нами ранее пункту меню Правка -> Комбинации клавиш. Как явствует из названия и скриншота, здесь можно переопределить сочетания клавиш, привязанные к любому из определённых по умолчанию действий:



Для этого достаточно дважды щёлкнуть по строке нужного действия и нажать новую комбинацию. Правда, определить собственное действие, отсутствующее в списке, не получится.

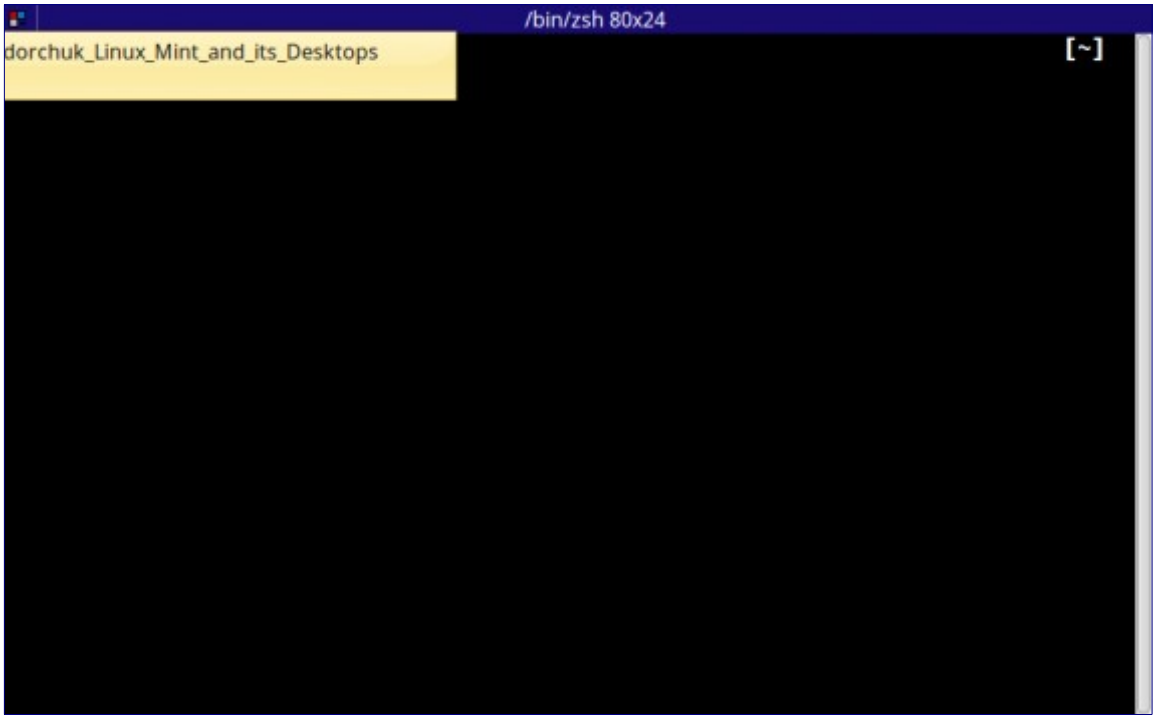
Таким образом, можно видеть, что по своей функциональности GNOME Terminal вполне соответствует своему назначению. И к нему можно подбирать не столько альтернативы, сколько дополнения.

Terminator

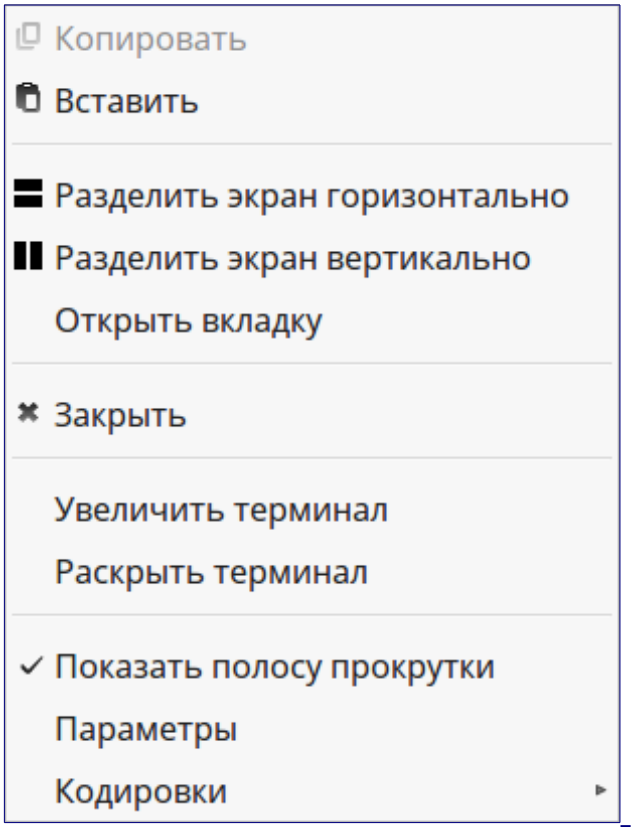
Таким дополнением к GNOME Terminal может стать терминальная программа Terminator. Она имеется в официальном репозитории в виде одноимённого пакета, который устанавливается стандартным образом:

```
$ apt install terminator
```

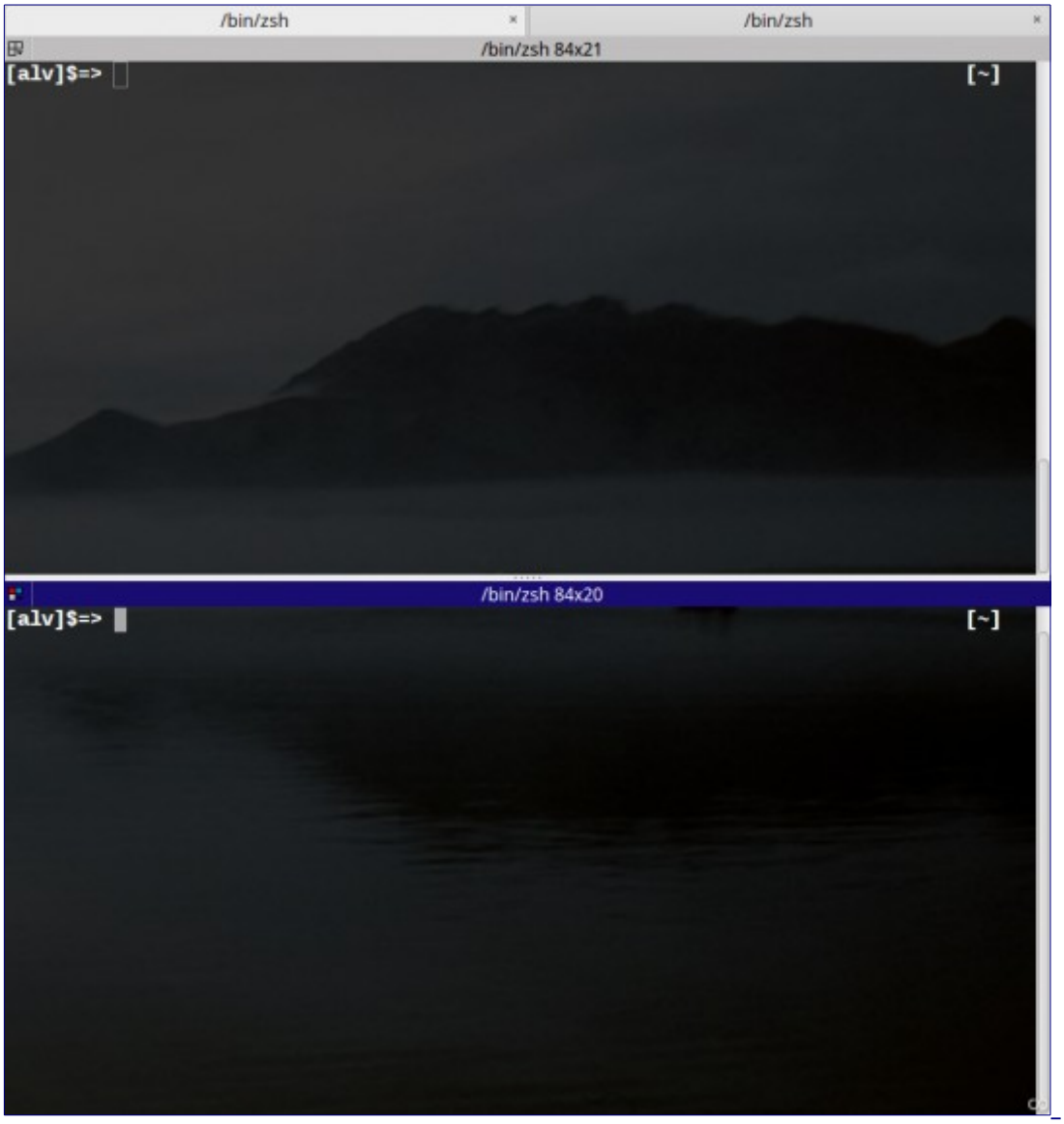
После этого программа обнаруживается в секции Администрирование главного меню Cinnamon, где она носит имя Терминатор. И после запуска выглядит следующим образом:



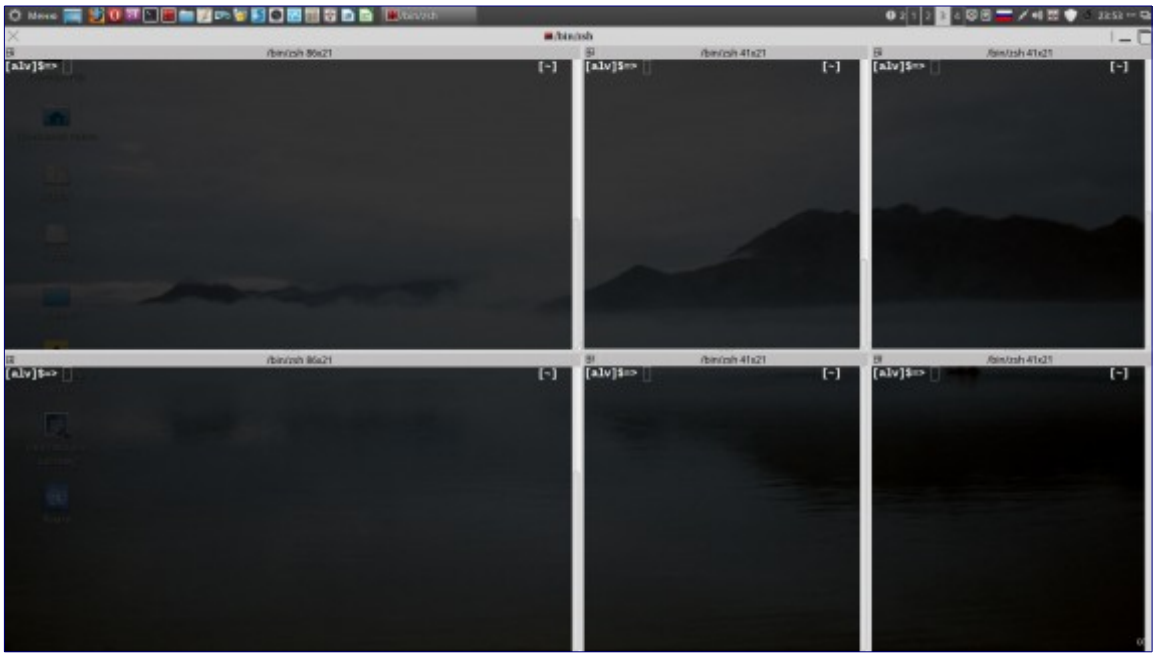
Как и в GNOME Terminal, в Terminator'е строки меню нет — но не по умолчанию, а от слова «вообще»: все действия выполняются через контекстное меню по правому клику мыши:



И первый же взгляд на контекстное меню выявляет главную (и убийственную) фичу Terminator'a — возможность разбиения терминального окна на произвольное количество субтерминалов, каждый из которых имеет свою титульную строку, и в каждом запущен независимый экземпляр командной оболочки:

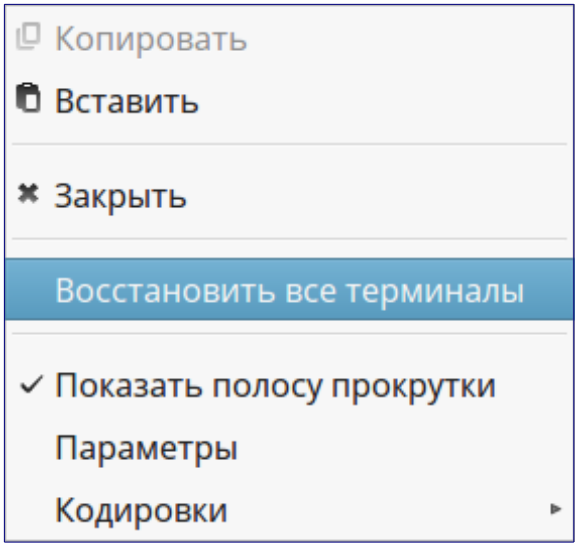


Число субтерминалов ограничено только целесообразностью и здравым смыслом:



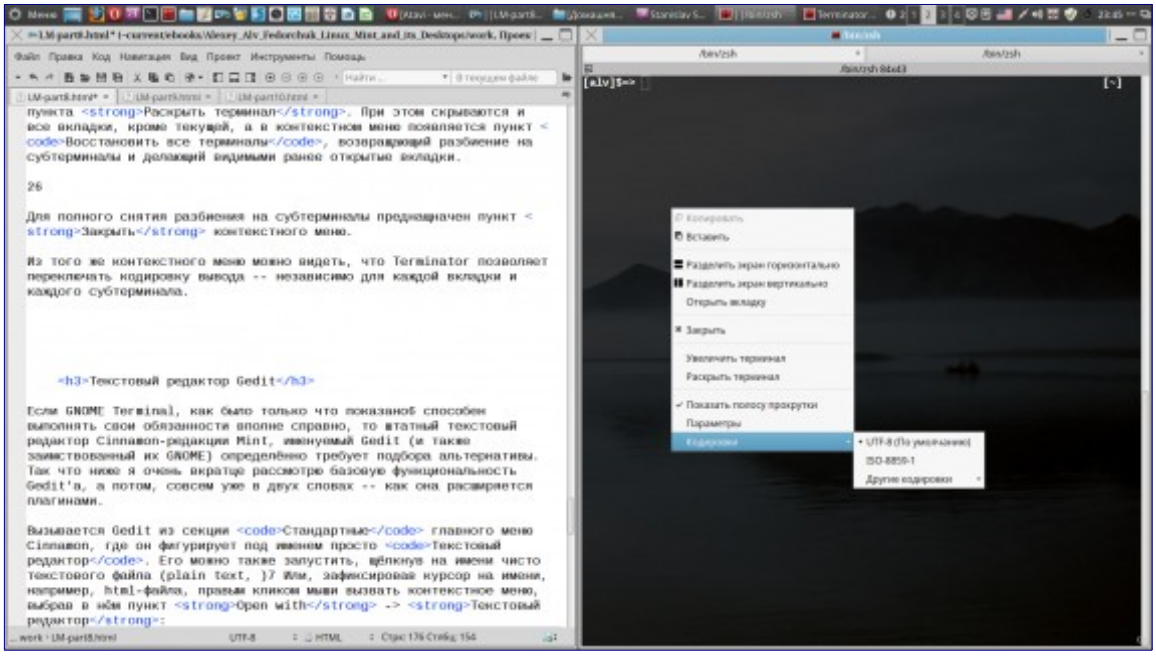
Не запрещается и создание вкладок, причём каждая вкладка может быть разбита независимо от других.

Временно развернуть один из субтерминалов на всё окно можно выбором пункта Раскрыть терминал. При этом скрываются и все вкладки, кроме текущей, а в контекстном меню появляется пункт Восстановить все терминалы, возвращающий разбиение на субтерминалы и делающий видимыми ранее открытые вкладки.

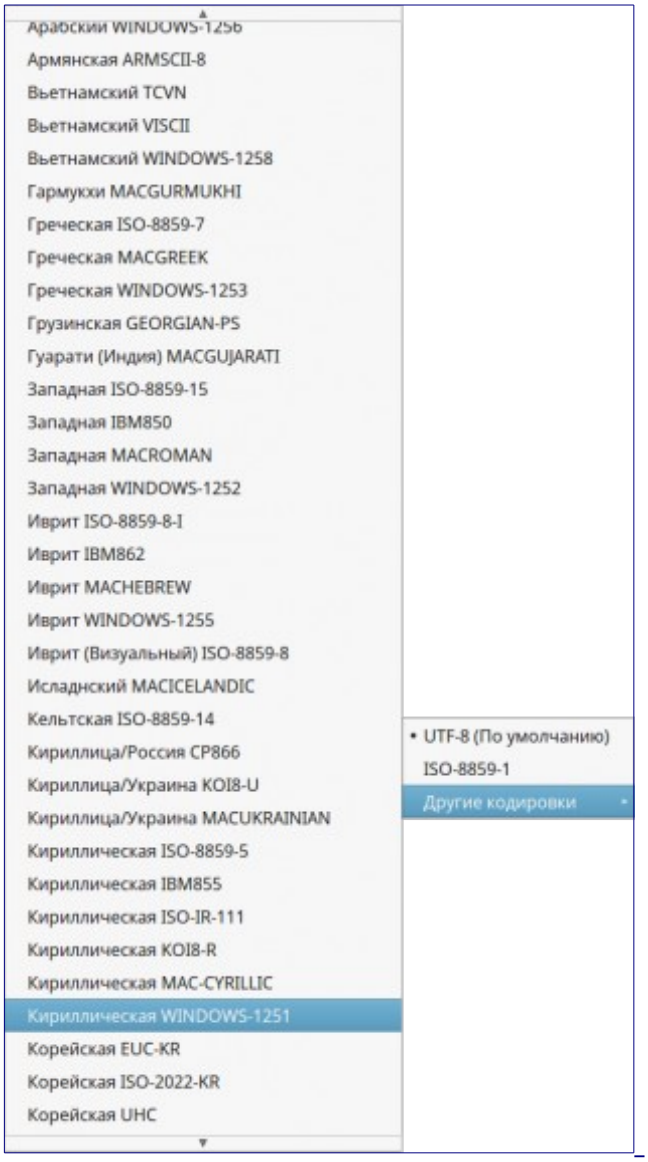


Для полного снятия разбиения на субтерминалы предназначен пункт Закрыть контекстного меню.

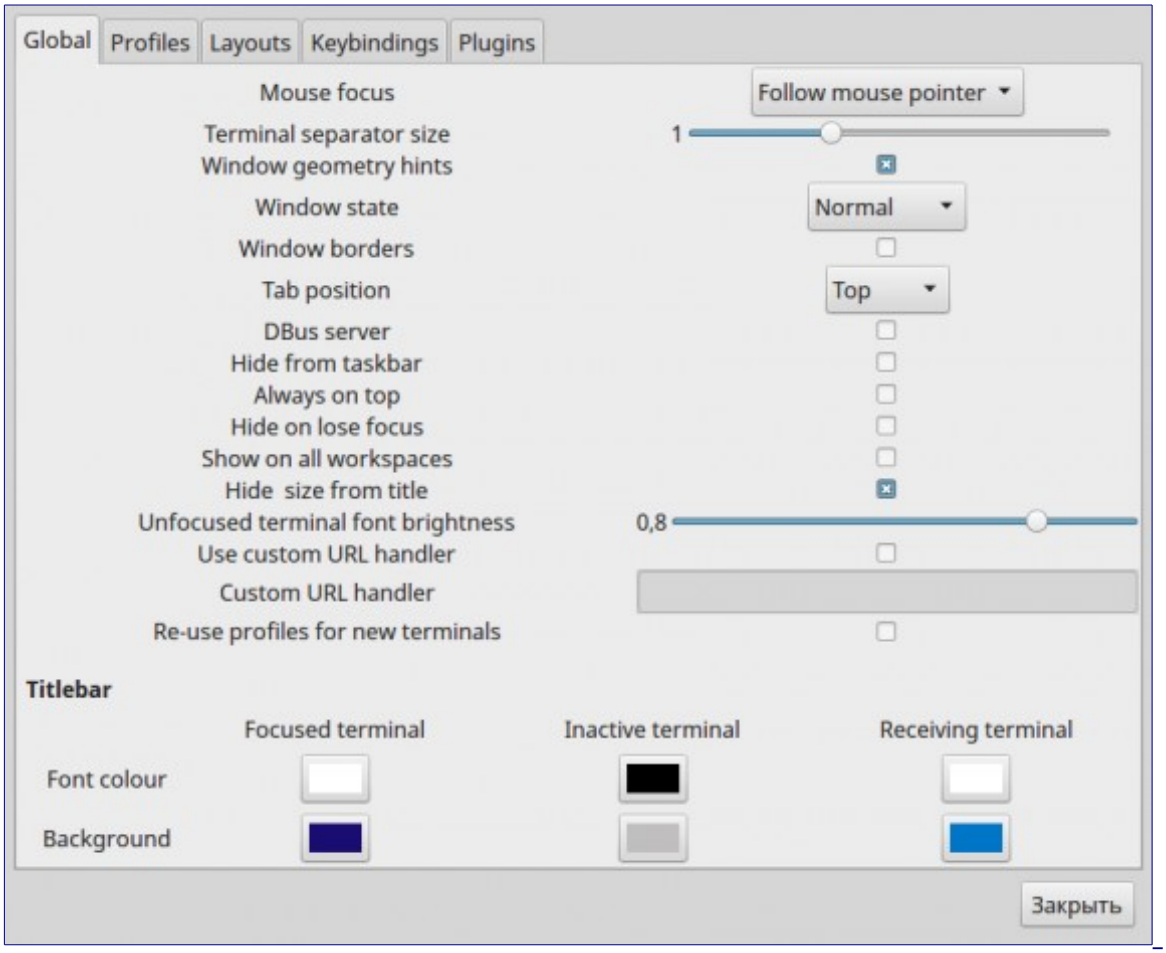
Из того же контекстного меню можно видеть, что Terminator позволяет переключать кодировку вывода — независимо для каждой вкладки и каждого субтерминала.



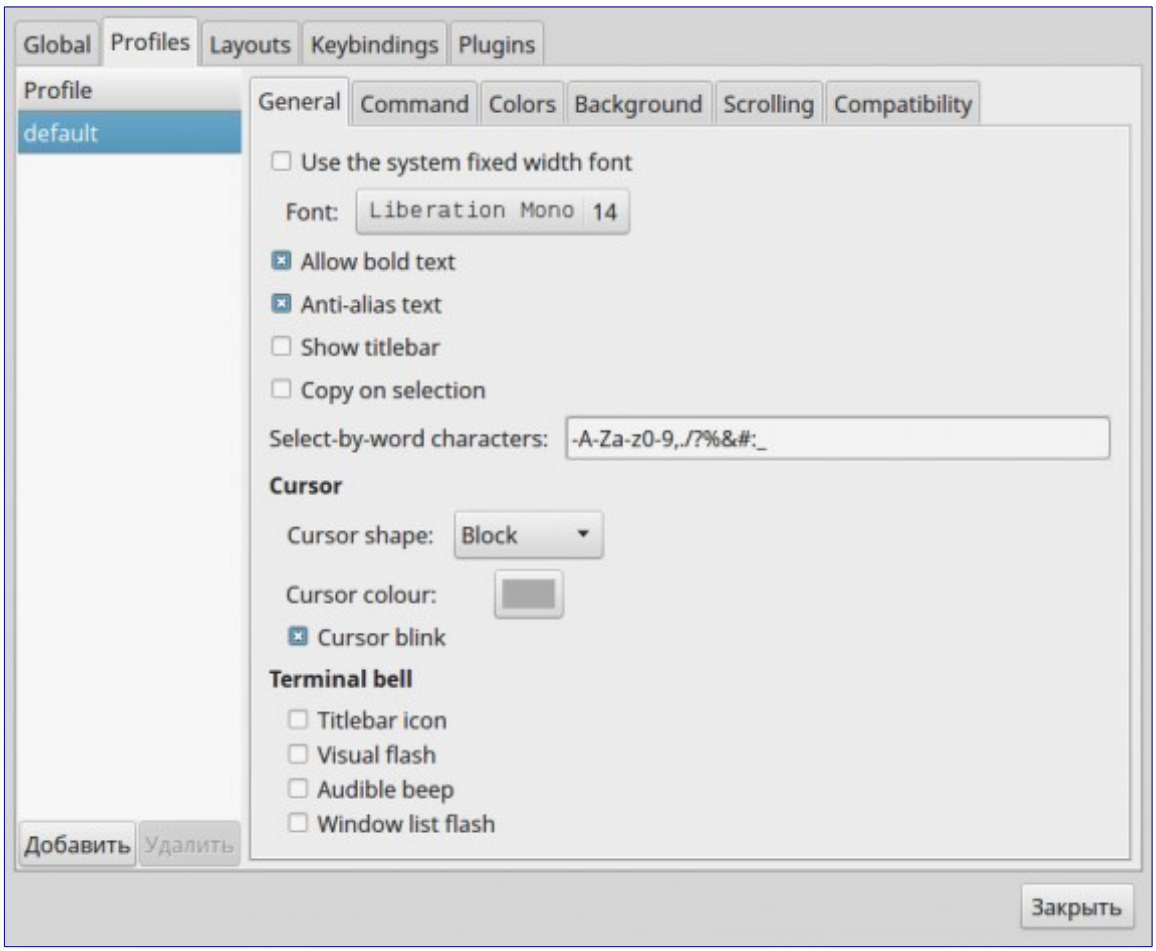
Причём список доступных кодировок трудно обозрим, и включает все кодировки кириллицы, о которых я только слышал:

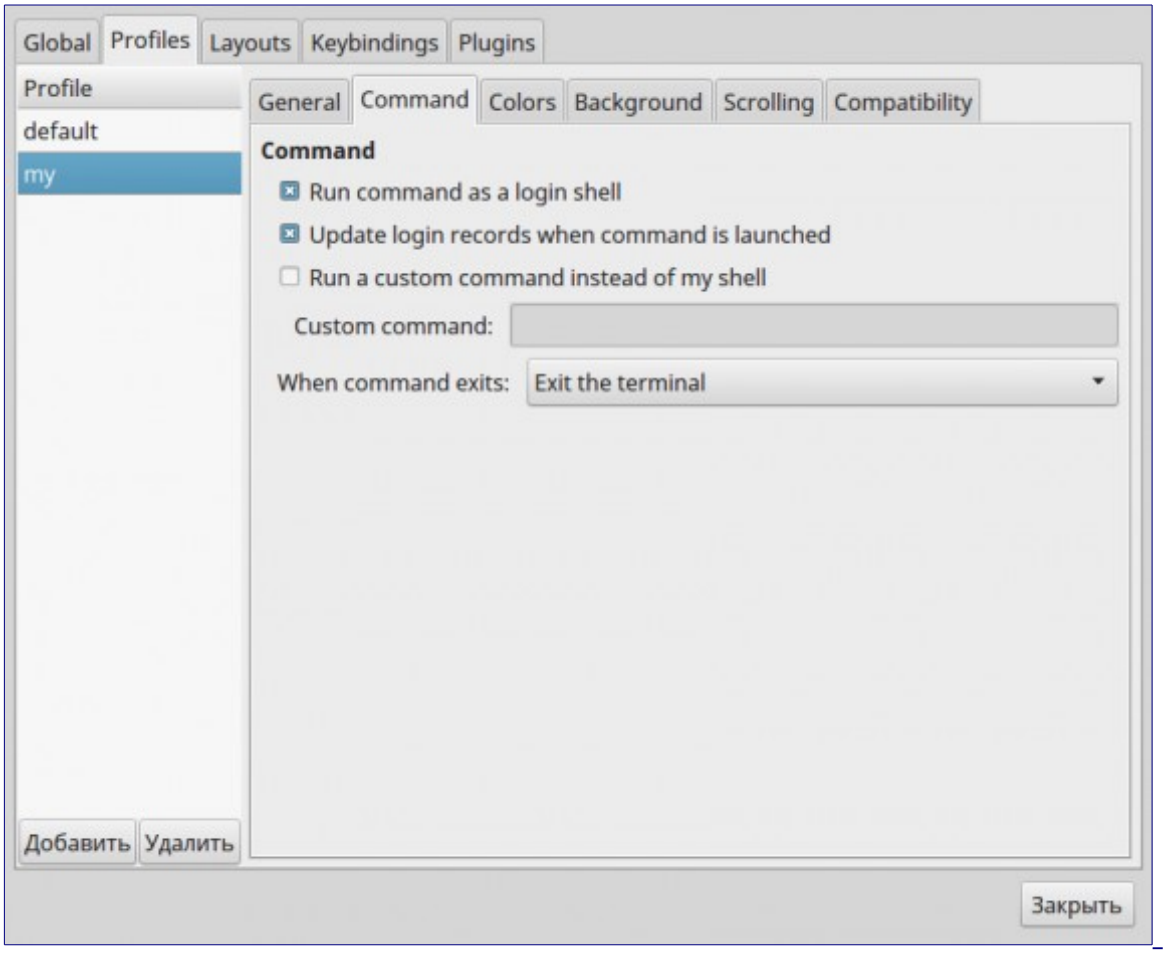


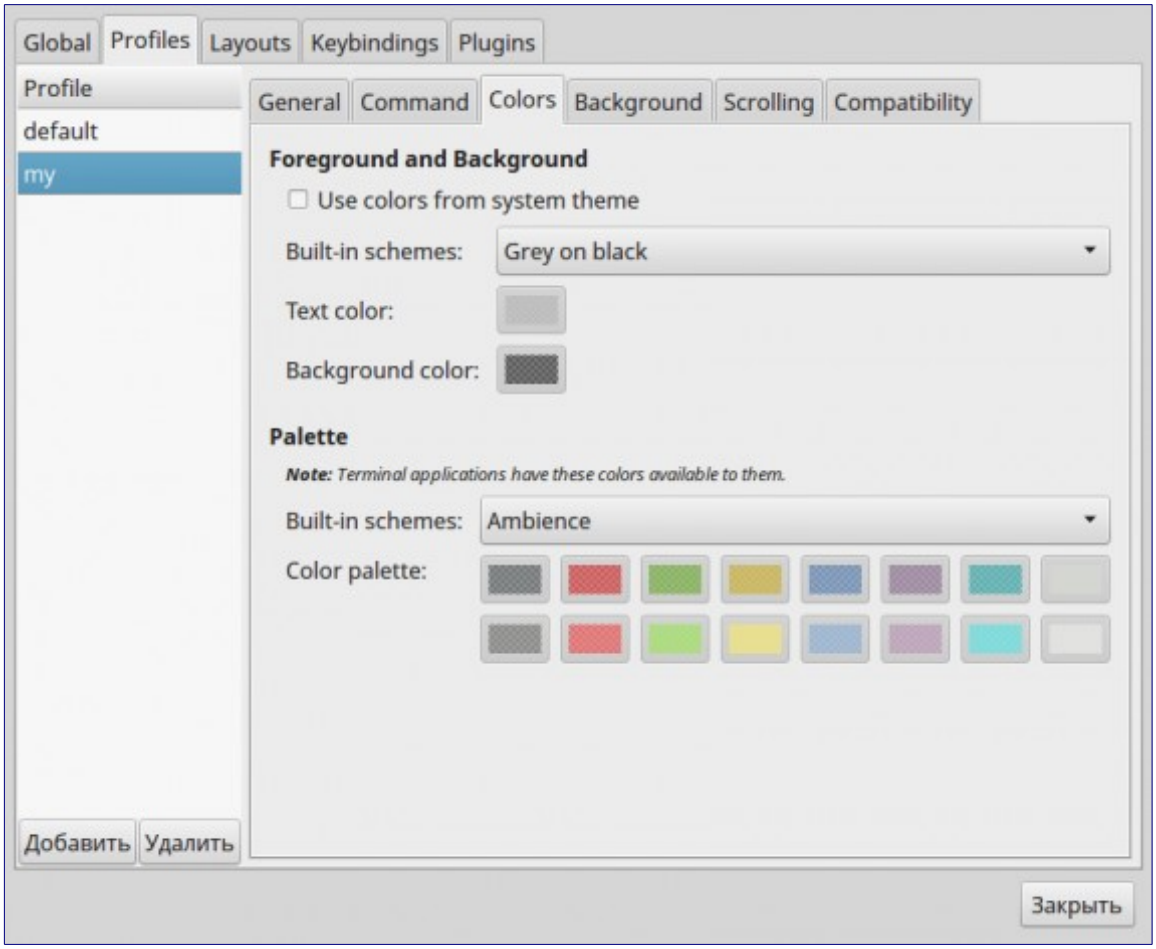
В пункте Параметры, как обычно, вызывает панель настройки программы о пяти вкладках. В первой, Global, настраивается фокусировка, положение вкладок (они могут располагаться с любой стороны окна), расцвета титульных строк субтерминалов и вкладок, и так далее:

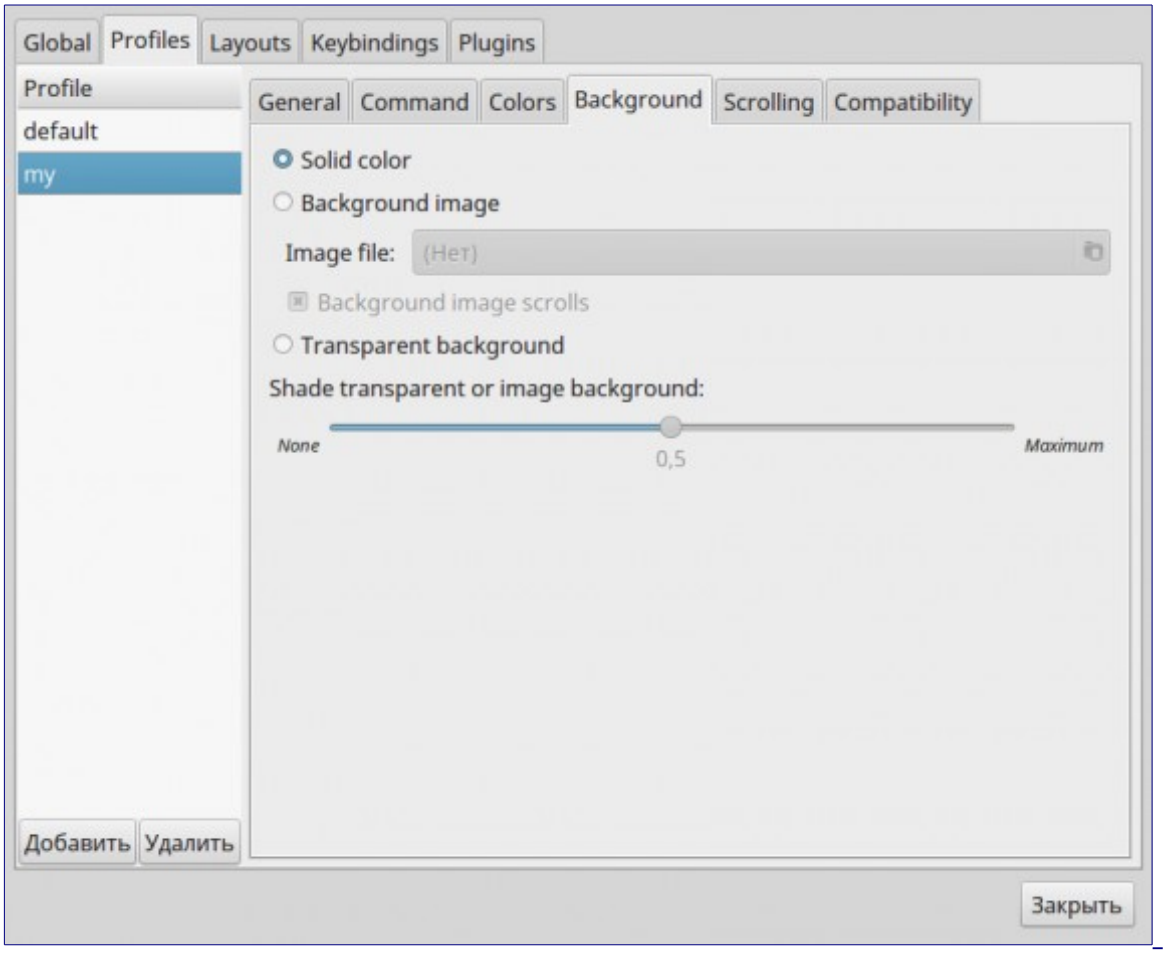


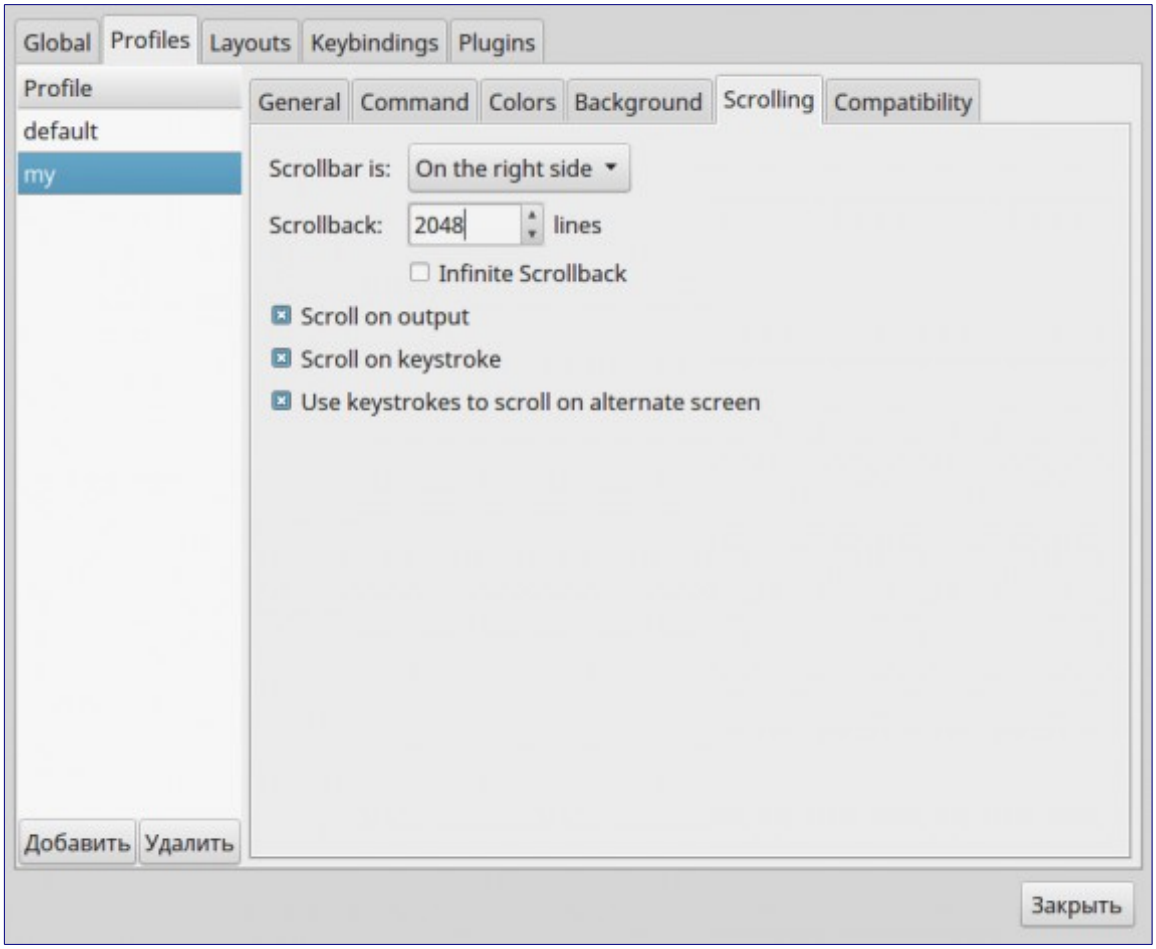
Во вкладке Profiles — шесть субвкладок, смысл которых понятен из скриншотов или по аналогии с настройкой профилей GNOME Terminal:

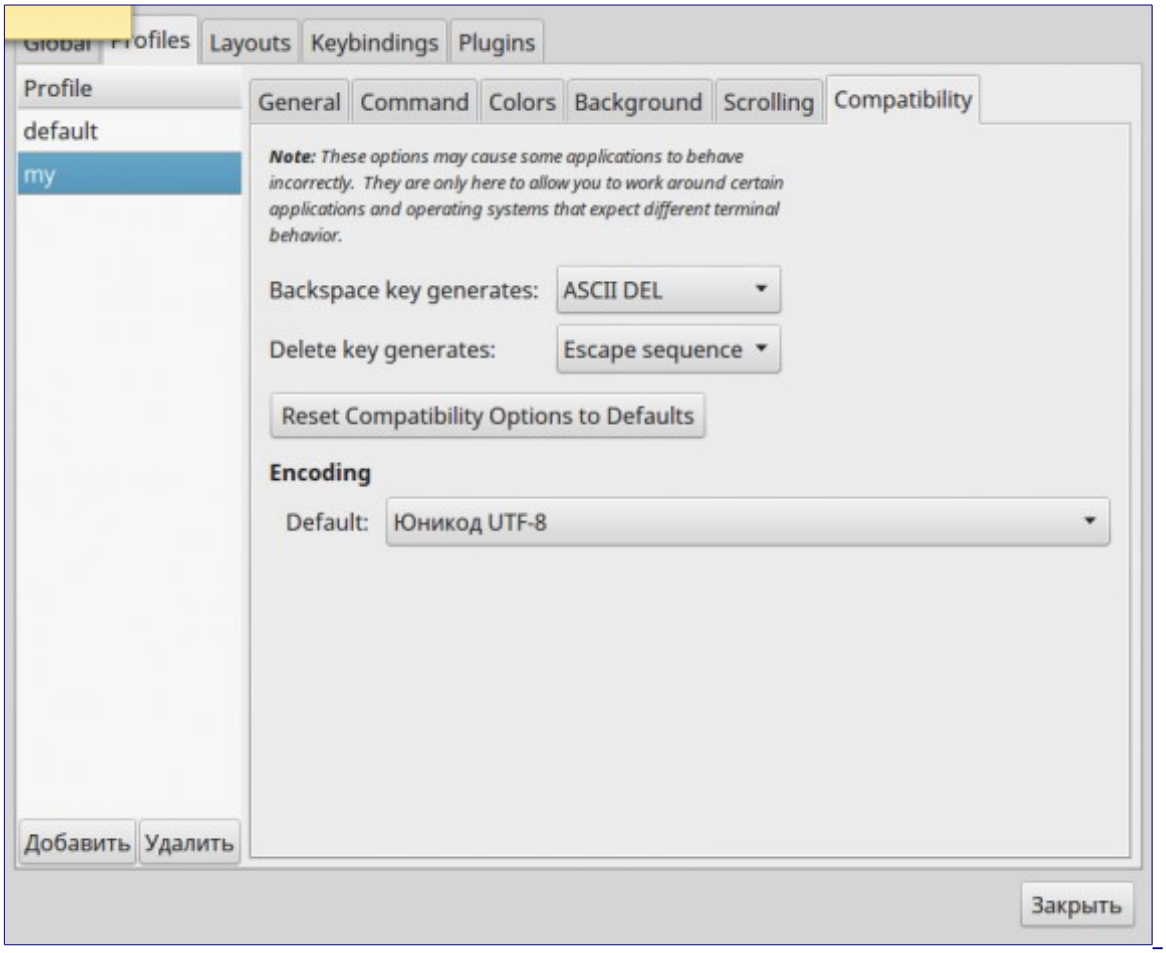



Global
Profiles
Layouts
Keybindings
Plugins
Profile
default
my
General
Command
Colors
Background
Scrolling
Compatibility
Command
Run command as a login shell
Update login records when command is launched
Run a custom command instead of my shell
Custom command:
When command exits:
Exit the terminal
Добавить
Удалить
Закрыть


Global
Profiles
Layouts
Keybindings
Plugins
Profile
default
my
General
Command
Colors
Background
Scrolling
Compatibility
Foreground and Background
Use colors from system theme
Built-in schemes:
Grey on black
Text color:
Background color:
Palette
Note: Terminal applications have these colors available to them.
Built-in schemes:
Ambience
Color palette:
Добавить
Удалить
Закрыть

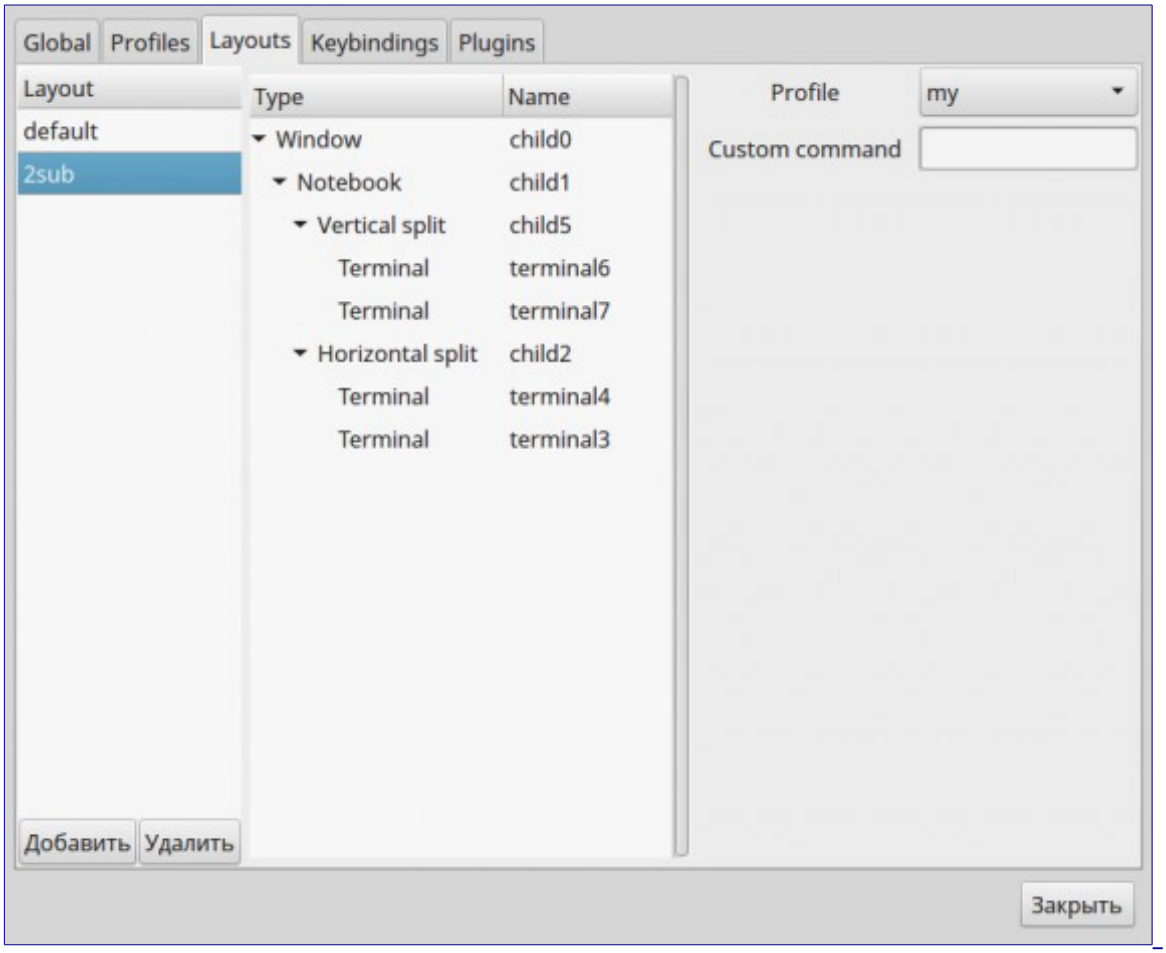

Global
Profiles
Layouts
Keybindings
Plugins
Profile
default
my
General
Command
Colors
Background
Scrolling
Compatibility
Solid color
Background image
Image file:
(Нет)
Background image scrolls
Transparent background
Shade transparent or image background:
None
0,5
Maximum
Добавить
Удалить
Закрыть


Global
Profiles
Layouts
Keybindings
Plugins
Profile
default
my
General
Command
Colors
Background
Scrolling
Compatibility
Scrollbar is:
On the right side
Scrollback:
2048
lines
Infinite Scrollback
Scroll on output
Scroll on keystroke
Use keystrokes to scroll on alternate screen
Добавить
Удалить
Закрыть

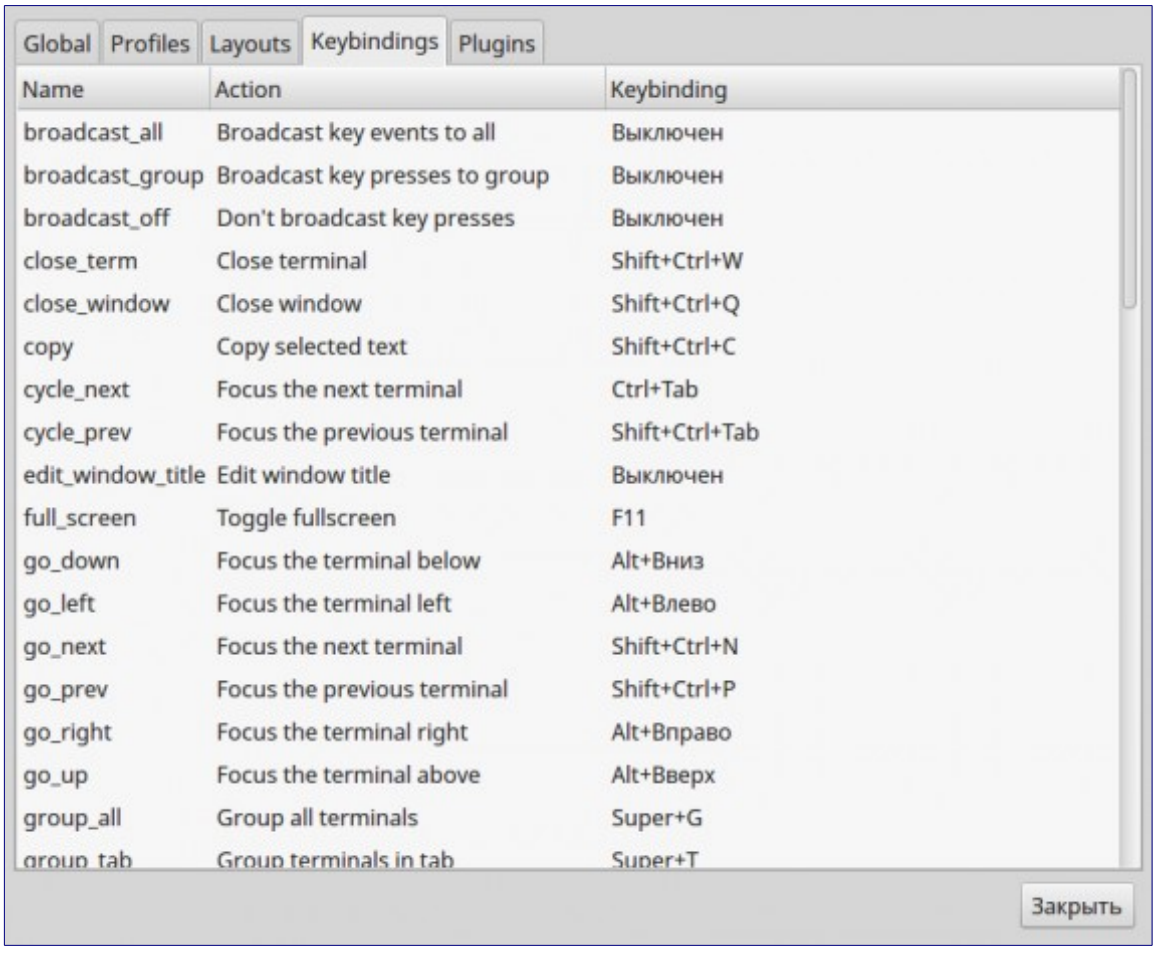


Профиль может быть определён для каждого субтерминала и каждой вкладки независимо друг от друга.

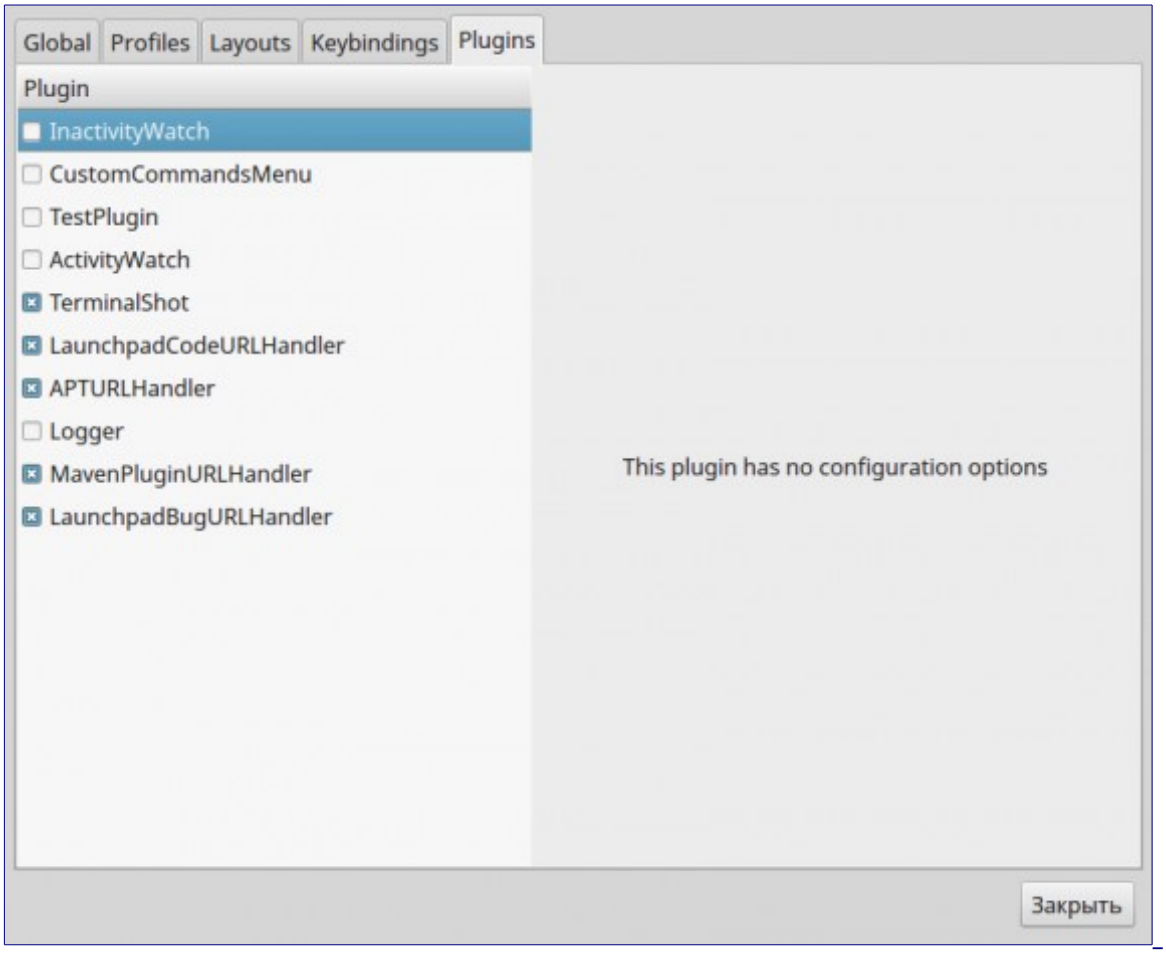
Вкладка Layouts позволяет создать разбиение окна на субтерминалы и привязать его к определённому профилю:



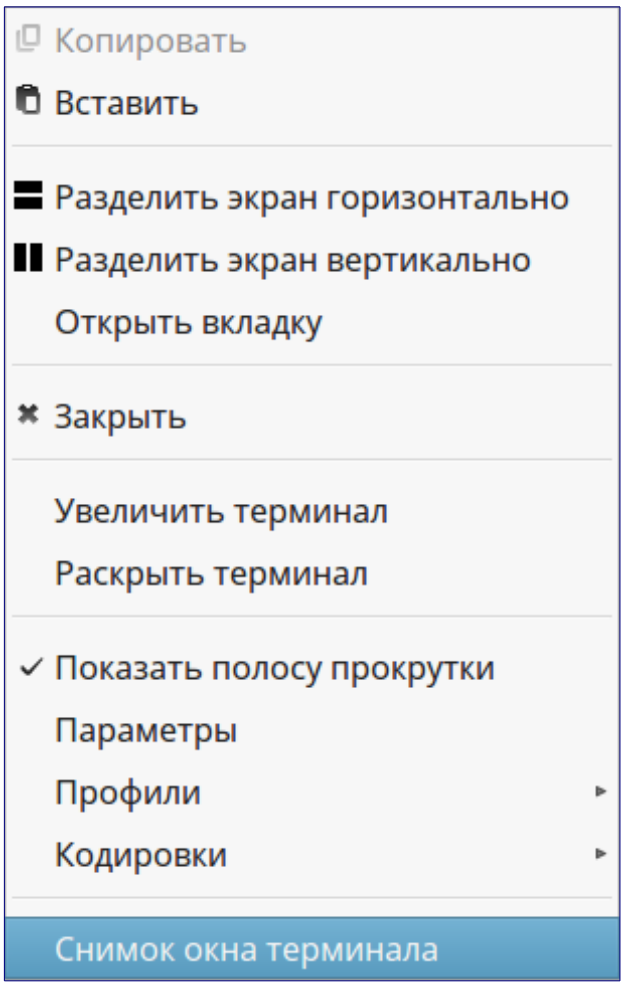
Во вкладке Keybindings настраиваются горячие клавиши:



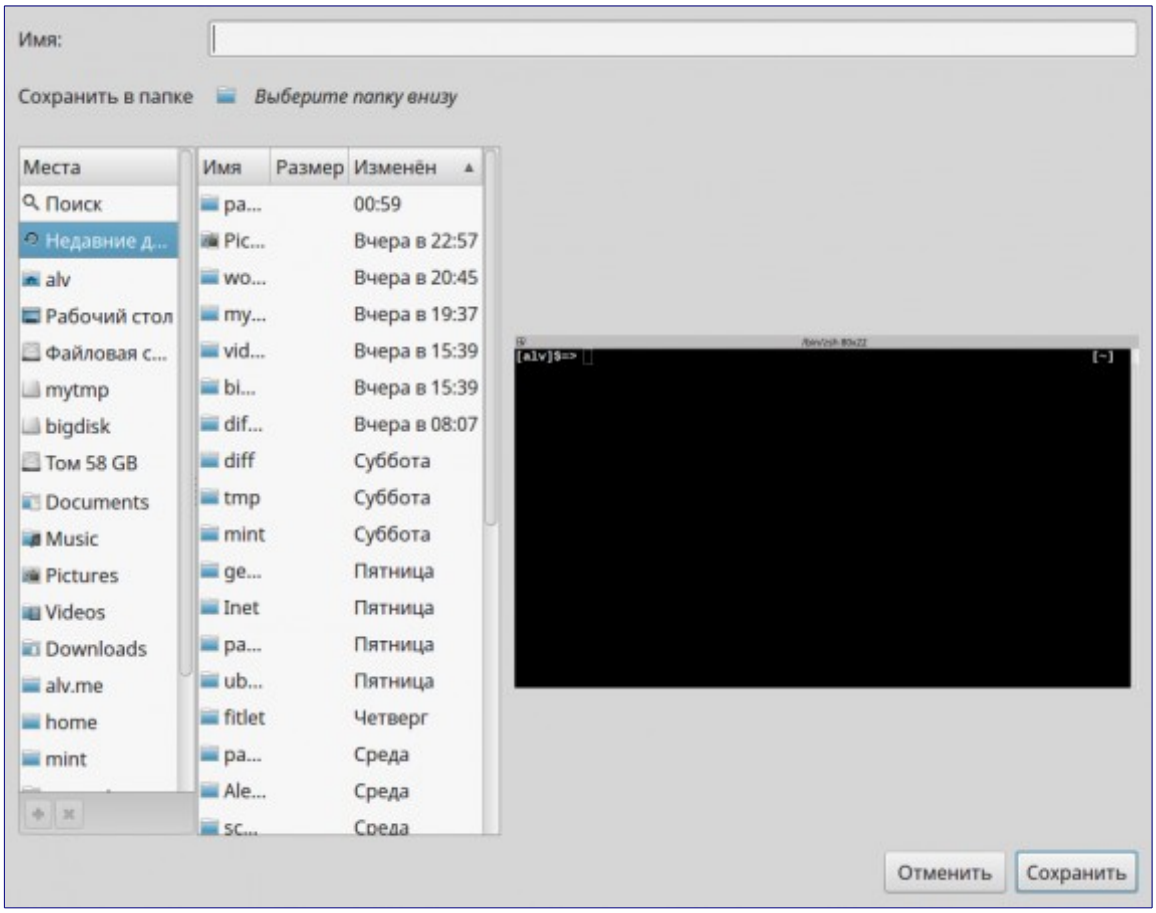
Во вкладке Plugins включаются и выключаются дополнительные модули:



Они отражаются в контекстном меню. Например, включение модуля TerminalSot добаляет в него пункт Снимок окна терминала:



Который предлагается сохранить в виде файла:



В отличие от GNOME Terminal, где все изменения вступают в силу немедленно, после включения или отключения любой опции, в Terminator'е они претворяются в действительность в момент закрытия панели настроек. Аналога кнопки Применить, характерной для KDE-приложений, также не имеется.

В общем, функционал Terminator'а может быть востребован в ряде случаев. Однако настройки его не вполне прозначны, и потому в повседневной жизни проще применять GNOME Terminal.

Выпадающий терминал Tilda

Некогда, с подачи Сергея Голубева, проникся я идеей выпадающих (drop-down) терминалов — тогда в виде Yakuake, ибо работал преимущественно в среде KDE. Проникся настолько, что почти перестал применять обычный эмулятор терминала, в те времена Konsole: практически во всех случаях удобней оказывалось прибегнуть либо к терминальному окну, встроенному в файловый менеджер (будь то Konqueror или Dolphin) или текстовый редактор (сиречь Kate), либо вызвать терминал выпадающий.

Переключившись на рабочие среды, основанные на Gtk (Xfce, Unity, Cinnamon), я начал подыскивать аналогичные средства эмуляции терминального режима. Как было сказано в очерке про Nemo, с терминалом, встраиваемым в этот файловые менеджеры, в конце концов решилась. А по части выпадающих терминалов имелся изрядный выбор: Terra Terminal, Guake и Tilda.

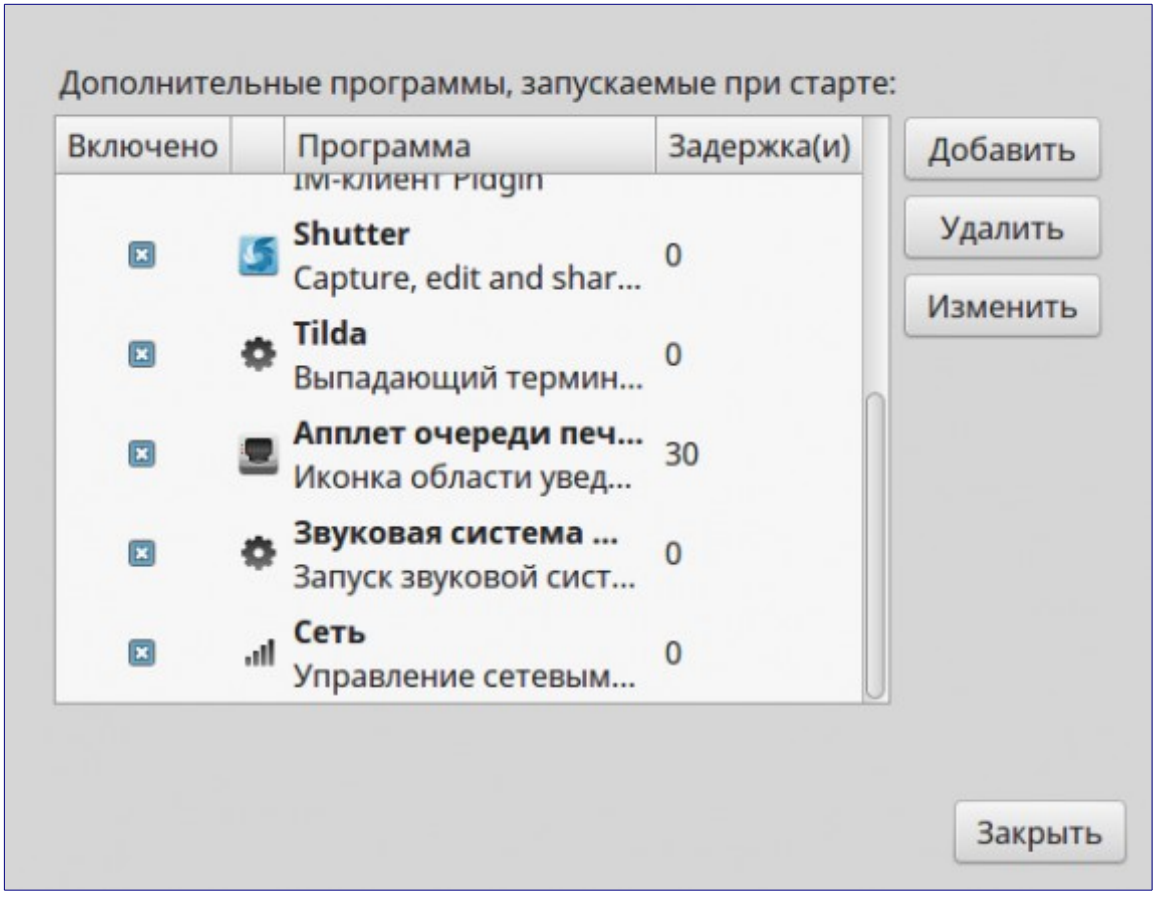
К сожалению, первая из названных программ прекратила своё развитие, а

две остальные я применял попеременно, пока в итоге не остановился последней: основанная на Gtk 3, Tilda, как мне кажется, лучше вписывается в окружение Cinnamon, базирующееся на тех же библиотеках, нежели Guake, в основе которой лежит Gtk 2. Впрочем, с практической точки зрения, разница между этими двумя программами не велика. И по описанию Tilda легко понять, как работать с Guake, буде такая необходимость возникнет.

Пакет Tilda входит в официальный репозиторий Mint (точнее, в ту его часть, которая напрямую заимствована из Ubuntu), и потому устанавливается стандартно:

```
$ apt install tilda
```

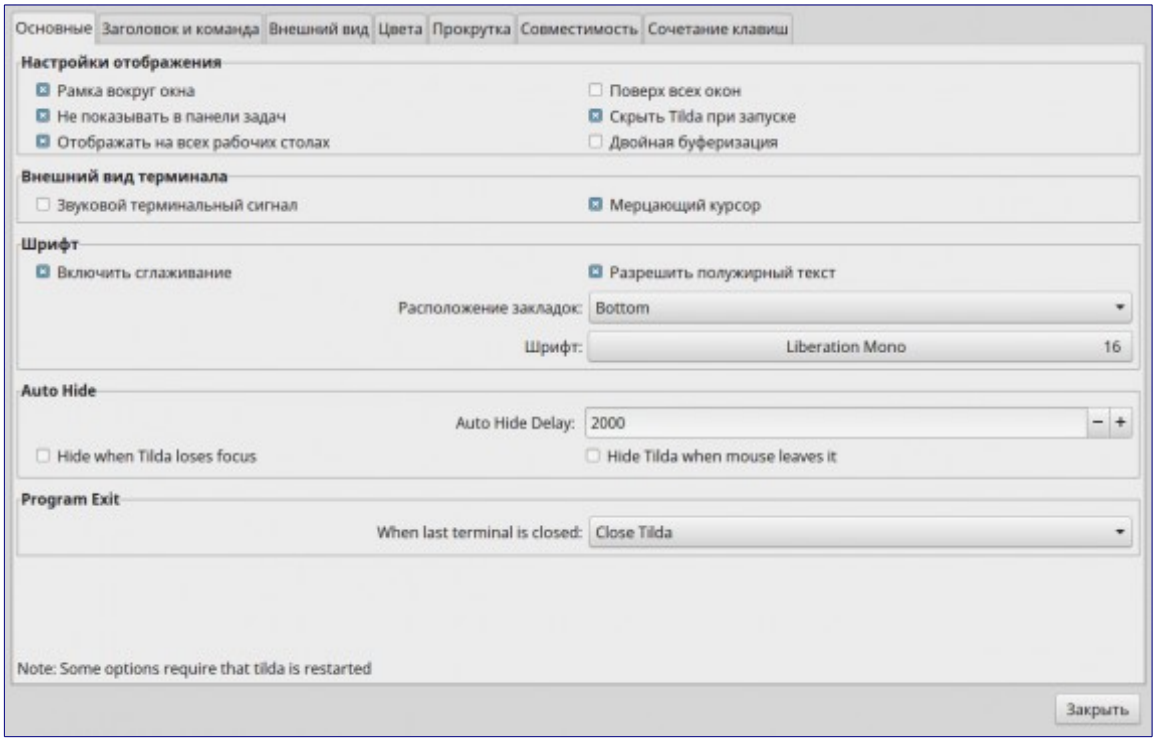
После чего Tilda может быть запущена из одноимённого пункта секции Администрирование главного меню Cinnamon. Однако для любого выпадающего терминала такой метод запуска имеет не много смысла — он всегда должен быть под рукой. И потому надо обеспечить Tilda постоянное присутствие посредством Системных настроек и их пункта Автозагрузка:



Однако первый раз имеет смысл запустить Tilda из главного меню:

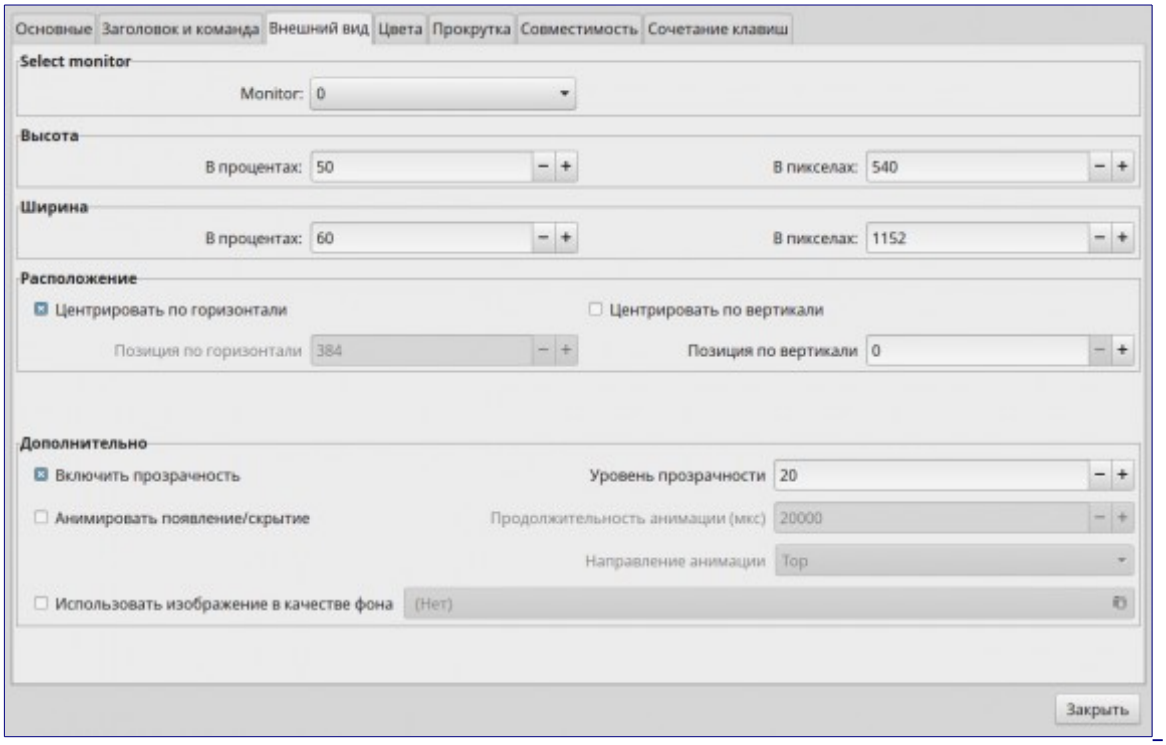


И заняться её настройками: соответствующая панель при первом запуске вызывается автоматически:

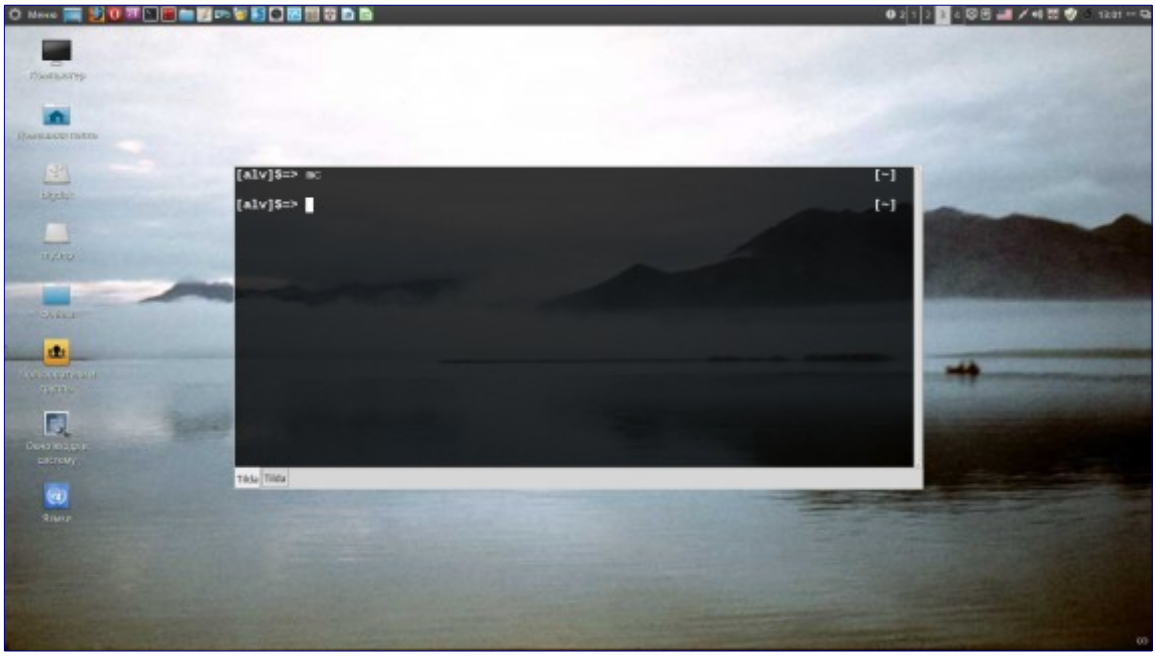


Правда, на скриншоте дан вид с уже сделанными мной настройками, но смысл их, я думаю, понятен без комментариев — как и настроек в остальных вкладках панели. Остановлюсь только на трёх моментах.

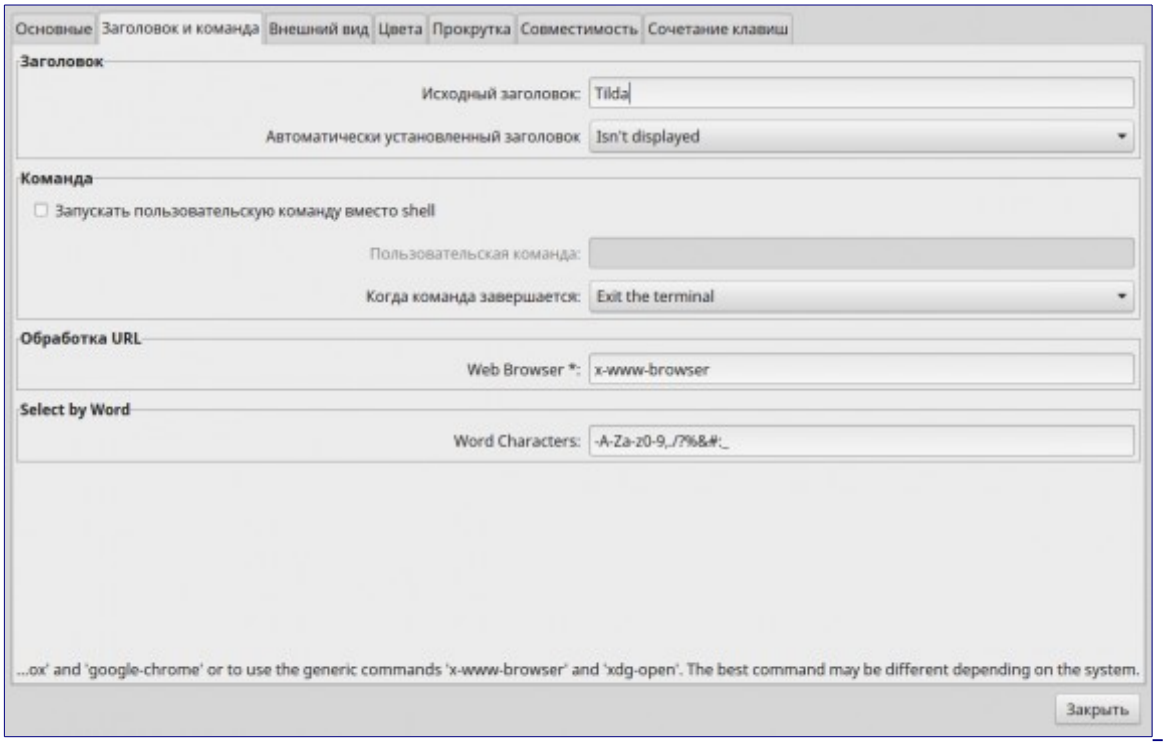
Во вкладке Внешний вид можно не только задать размеры терминального окна, но и его центрирование — не только по горизонтали, но и по вертикали, включить анимацию и её направление:



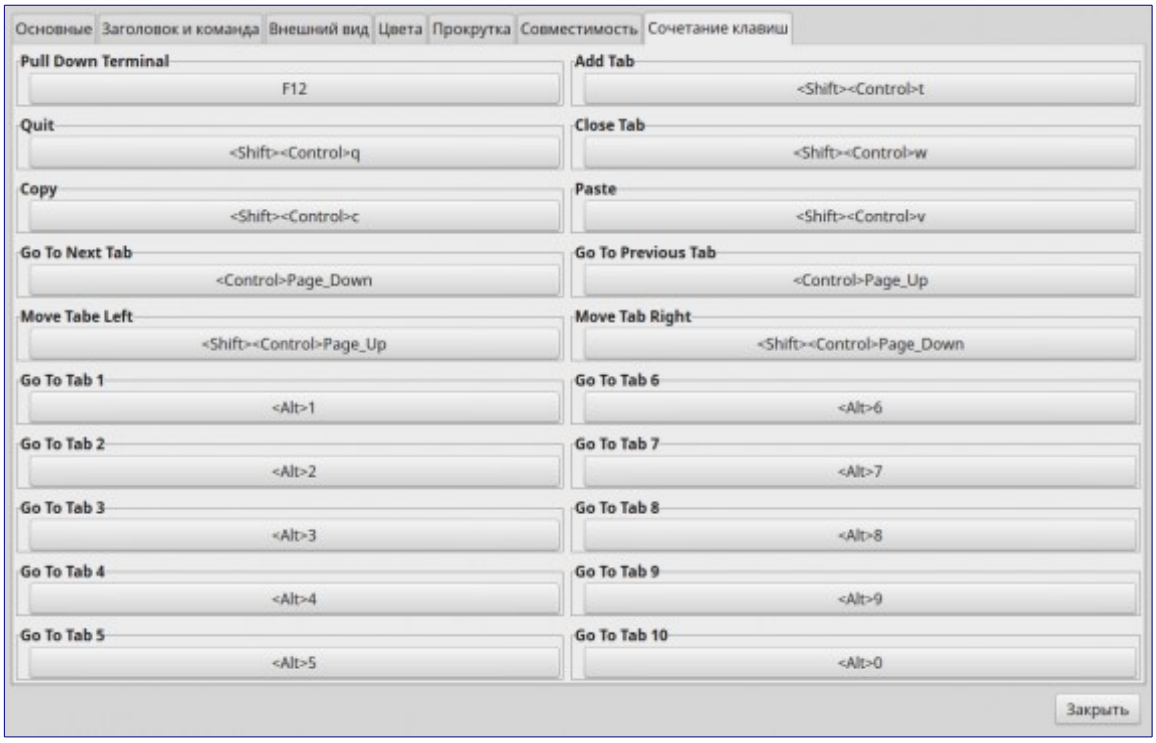
При центрировании по обеим осям и включённой анимации выпадающий терминал можно при желании превратить в терминал, «всплывающий» посреди экрана:



Во вкладке Заголовок и команда можно изменить начальный заголовок терминала и расположение заголовка, автоматически присваиваемого запущенной в нём командой (например, Midnight Commander):



Во вкладке Сочетание клавиш устанавливается способ вызова терминала — по умолчанию почему-то это клавиша F1. Что я немедленно заменил на стандартную для программ такого рода клавишу F12:



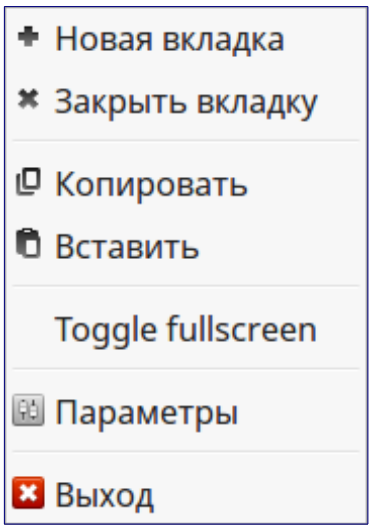
Повторное нажатие той же клавиши убирает выпадающий терминал с глаз долой.

Можно переопределить и комбинации клавиш для выполнения других действий. А по умолчанию работают все стандартные для большинства терминальных программ хоткеи:

- Shift+Control+T — создание новой вкладки;
- Shift+Control+W — закрытие текущей вкладки;

- Control+PageUp — переход на предыдущую вкладку;
- Control+PageDown — переход на следующую вкладку;
- Shift+Control+C — копирование выделенного фрагмента в буфер;
- Shift+Control+V — вставка содержимого буфера позицию курсора;
- Shift+Control+Q — выход из Tilda.

Из контекстного меню по правому мышиному клику можно открыть новую вкладку и закрыть существующую, копировать и вставлять выделенные мышью блоки, переключиться в полноэкранный режим и вызвать панель настроек:



Кроме контекстного меню, новую вкладку, как только что было сказано, можно создать и обычными для большинства терминальных программ хоткеями — Shift+Control+T. Каждой новой вкладке присваивается заголовок, установленный в панели настроек по умолчанию. Вкладки можно перемещать, просто перетаскивая их мышью.

На этом функционал программы исчерпывается — однако больше от неё ничего и не нужно. А свои непосредственные функции — представить интерфейс командной строки в нужном месте и в нужное время, Tilda выполняет исправно и к тому же быстро — заметно быстрее, чем Guake.

Текстовые редакторы

Текстовый редактор — третья из важнейших программ «джентльменского набора применителя». И потому этот очерк будет посвящён их рассмотрению — как вообще, так и на конкретных примерах.

Введение

Текстовый редактор в равной степени необходим программисту и сисадмину, веб-мастеру и блогеру, а также массе трудящихся, чья сфера деятельности с IT никак не связана: «традиционным» журналистам, писателям и поэтам, редакторам и переводчикам, научным работникам... короче говоря, всем работникам профессий, которые в советское время было принято называть творческими. И всем, чьё творчество связано со СЛОВОМ.

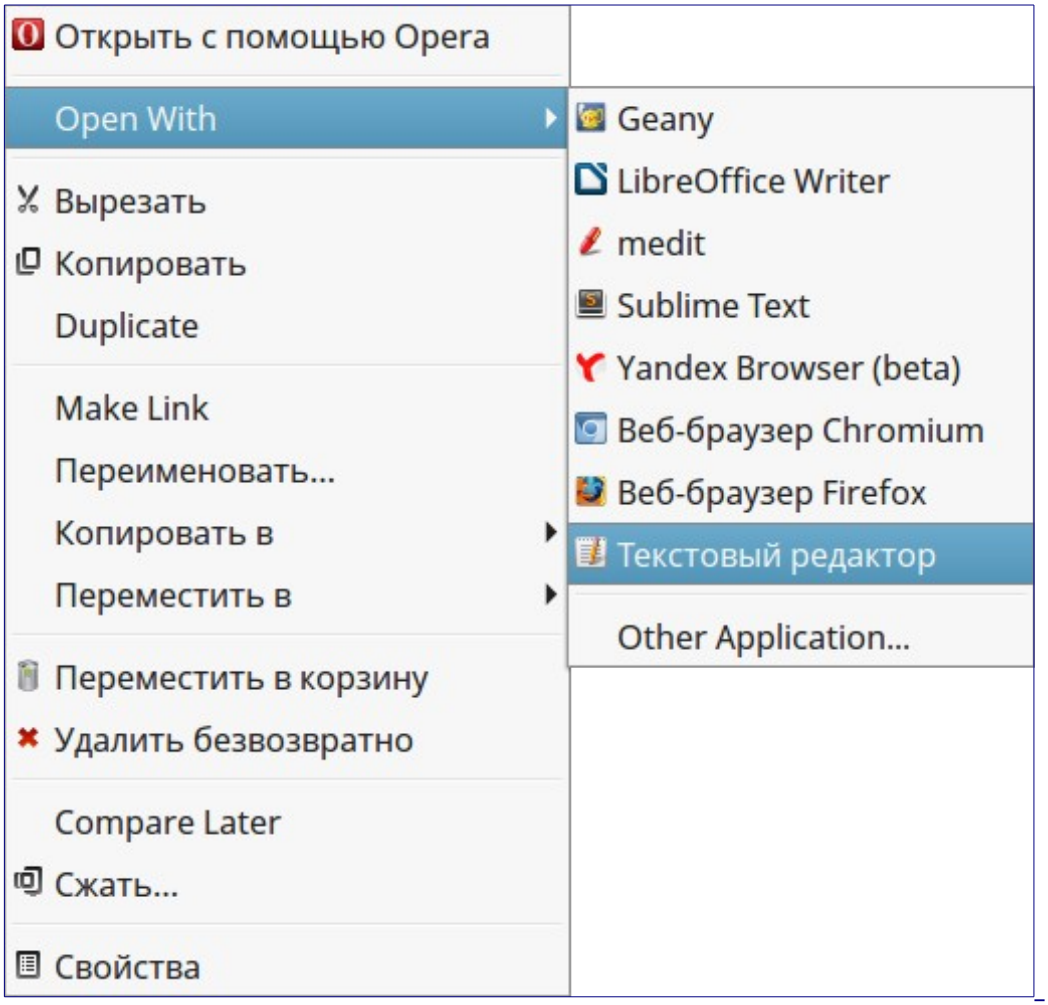
Правда, за исключением первых трёх категорий, большинство творческих работников об этом и не подозревают — за исключением, конечно, тех немногих из них, кто в одной из своих прошлых жизней каким-то боком не пересекался с IT-сферой. А потому, сочиняя свою нетленку или прелагая чужую, они обычно пользуются таким неуклюжим инструментам, как word processor (которые по русски почему-то называются текстовыми процессорами, хотя на самом деле text processor — это нечто совсем иное). И неважно, как этот инструмент называется — MS Writer'ом, AOo Writer'ом, Libre Writer'ом или даже AbiWord'ом — важно, что все они разрабатывались для изготовления бюрократических циркуляров, а не для творческой работы.

Разумеется, представители клана творческо-технологического, в частности, линуксописатели (в том числе и автор этих строк), предпринимали многочисленные попытки изменить существующее положение дел, открыв глаза собратьям по клану творческо-гуманитарному на мощь и величие текстовых редакторов вкупе с набором несложных утилит командной строки, таких, как cat, split, find, grep etc., или их графических фронт-эндов. Попытки эти, за небольшими частными исключениями, терпели фетяску. В том числе и потому, что многие из них предпринимались с негодными средствами, но к этому вопросу я ещё вернусь в заключительных строках.

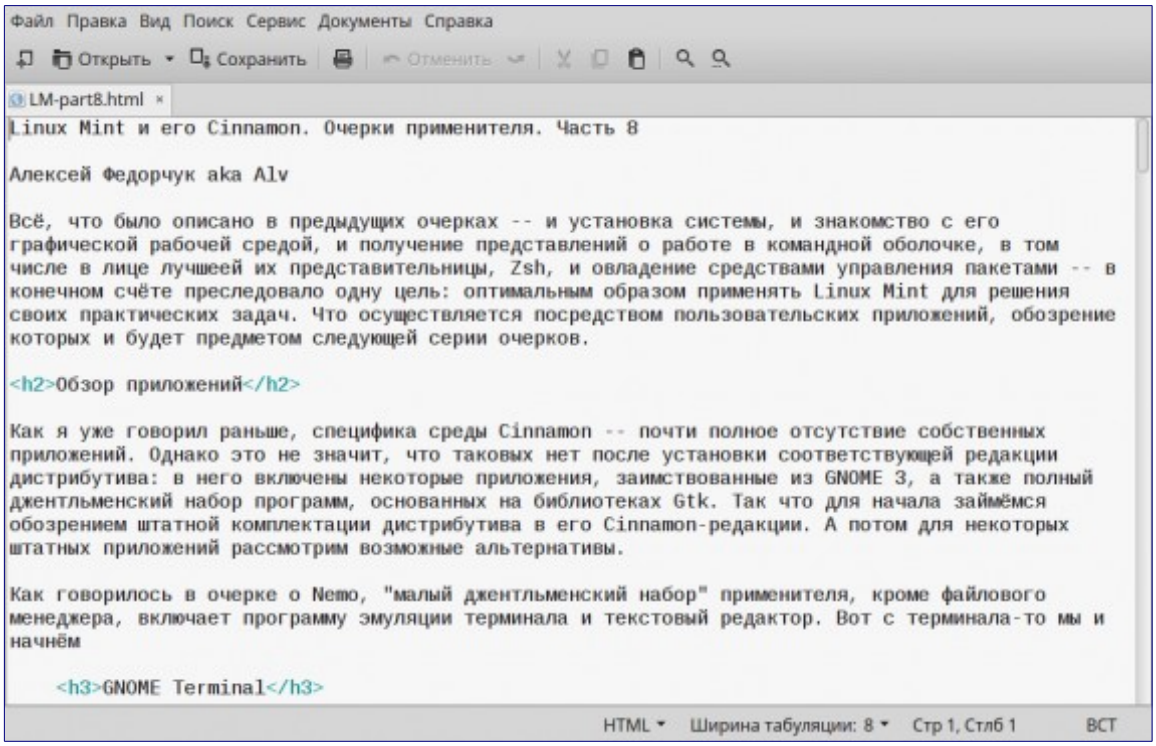
В Cinnamon-редакции Mint в роли штатного текстового редактора выступает Gedit. Однако, если GNOME Terminal, как было только что показано, способен выполнять свои обязанности вполне справно, то Gedit определённо требует подбора альтернативы. Так что ниже я очень вкратце рассмотрю базовую его функциональность. А потом перейду к описанию альтернатив, которых оказывается целых две — текстовые редакторы (или лёгкие IDE) Geany и Komodo Edit.

Текстовый редактор Gedit

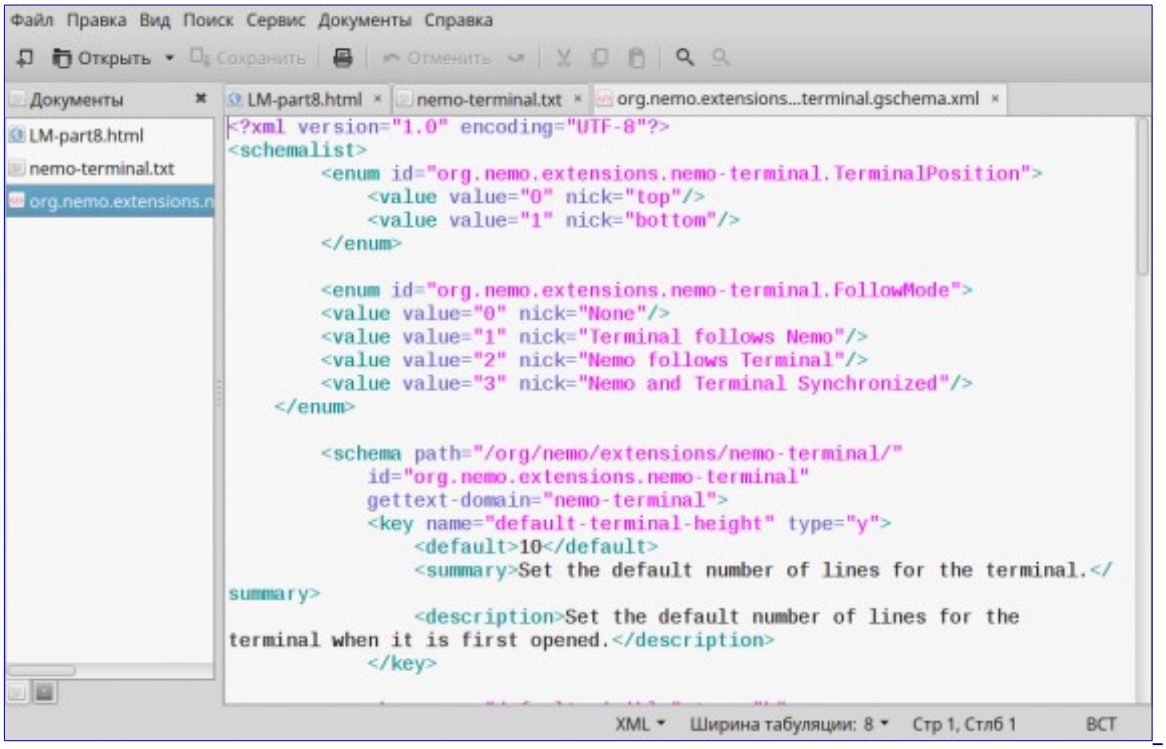
Текстовый редактор Gedit, подобно GNOME Terminal, заимствован из среды GNOME 3. Устанавливаясь по умолчанию при стандартной инсталляции, он вызывается из секции Стандартные главного меню Cinnamon, где фигурирует под именем просто Текстовый редактор. Его можно также запустить, щёлкнув на имени чисто текстового файла (plain text,)7 Или, зафиксировав курсор на имени, например, html-файла, правым кликом мыши вызвать контекстное меню, выбрав в нём пункт Open with -> Текстовый редактор:



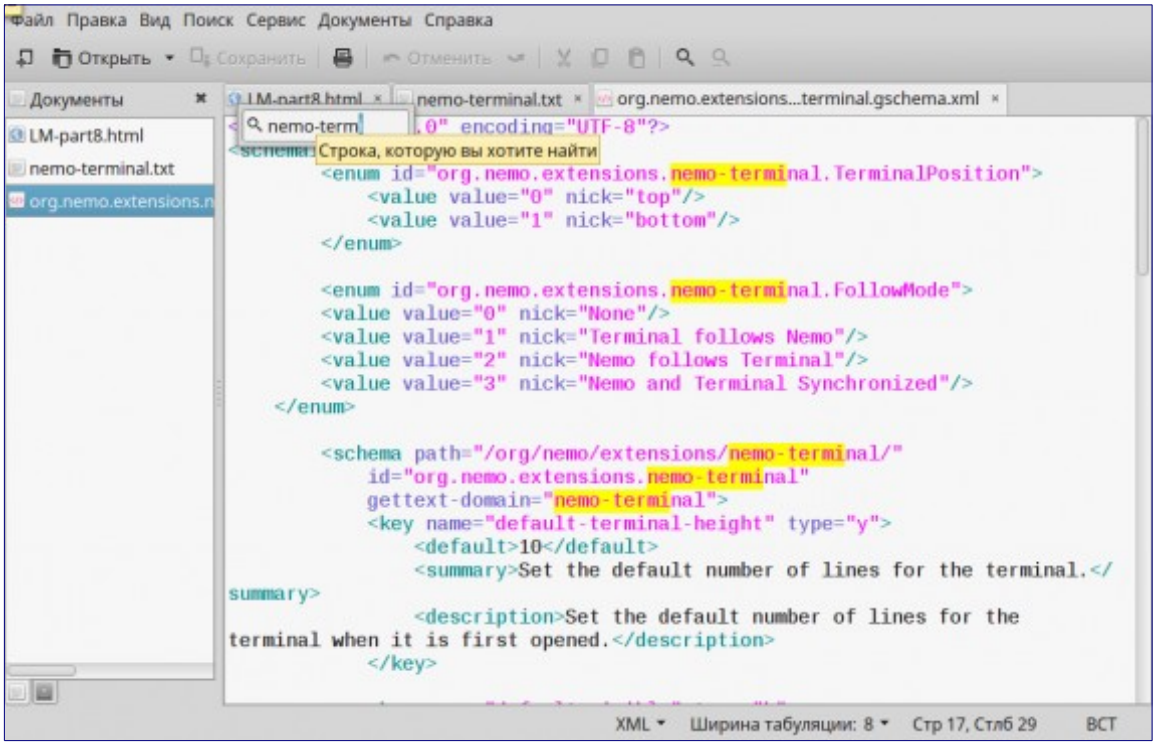
После чего он предстаёт примерно в таком виде:



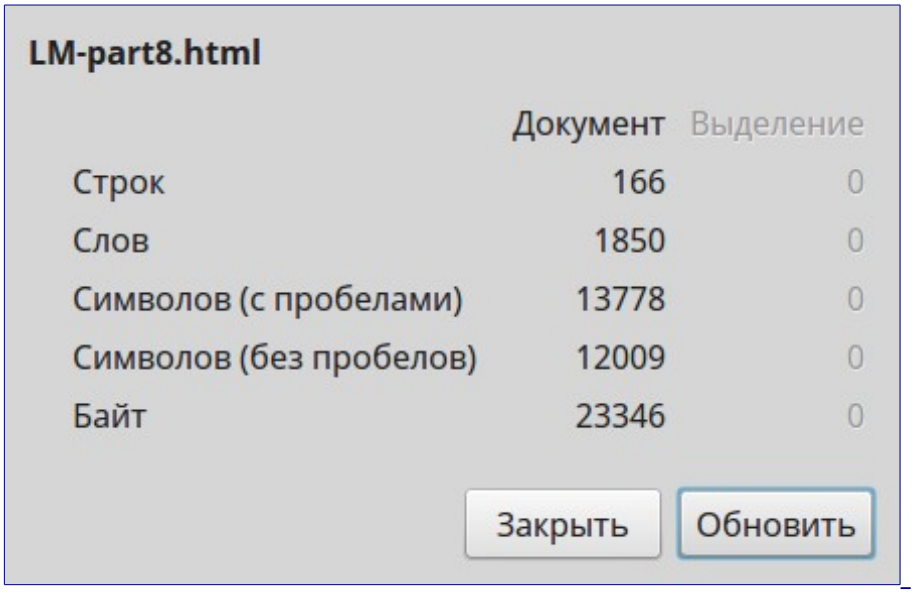
Последующие файлы будут открываться в том же окне — в новых его вкладках. Для ориентации в которых можно включить боковую панель (через меню Вид или клавишей F9):



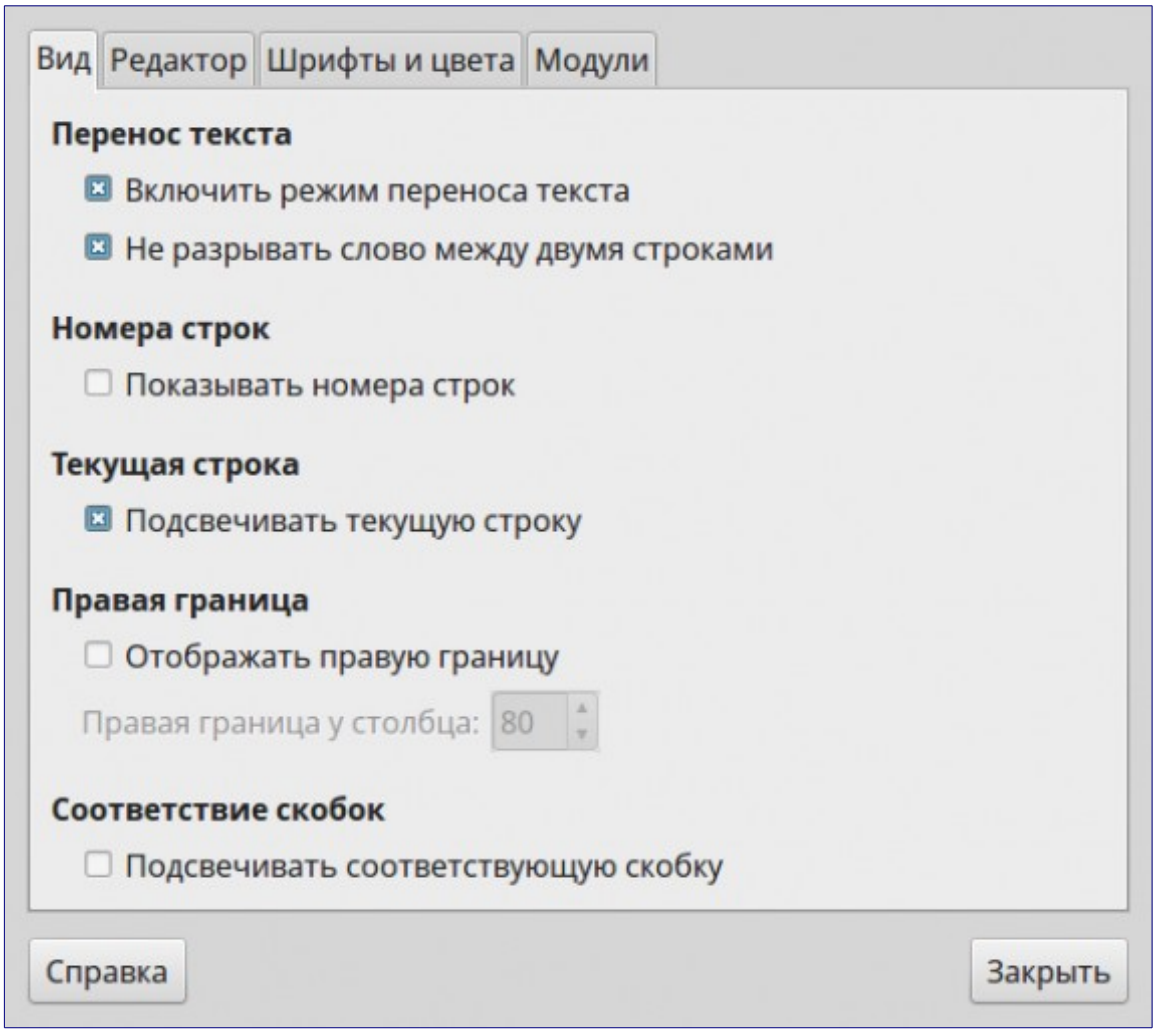
Как видно на скриншоте, Gedit обеспечивает подсветку синтаксиса разных языков программирования, а также языков разметки. Из других особенностей можно отметить наращиваемый поиск (здесь он называется последовательным), как в браузерах. Только здесь он вызывается комбинацией клавиш Control+K:



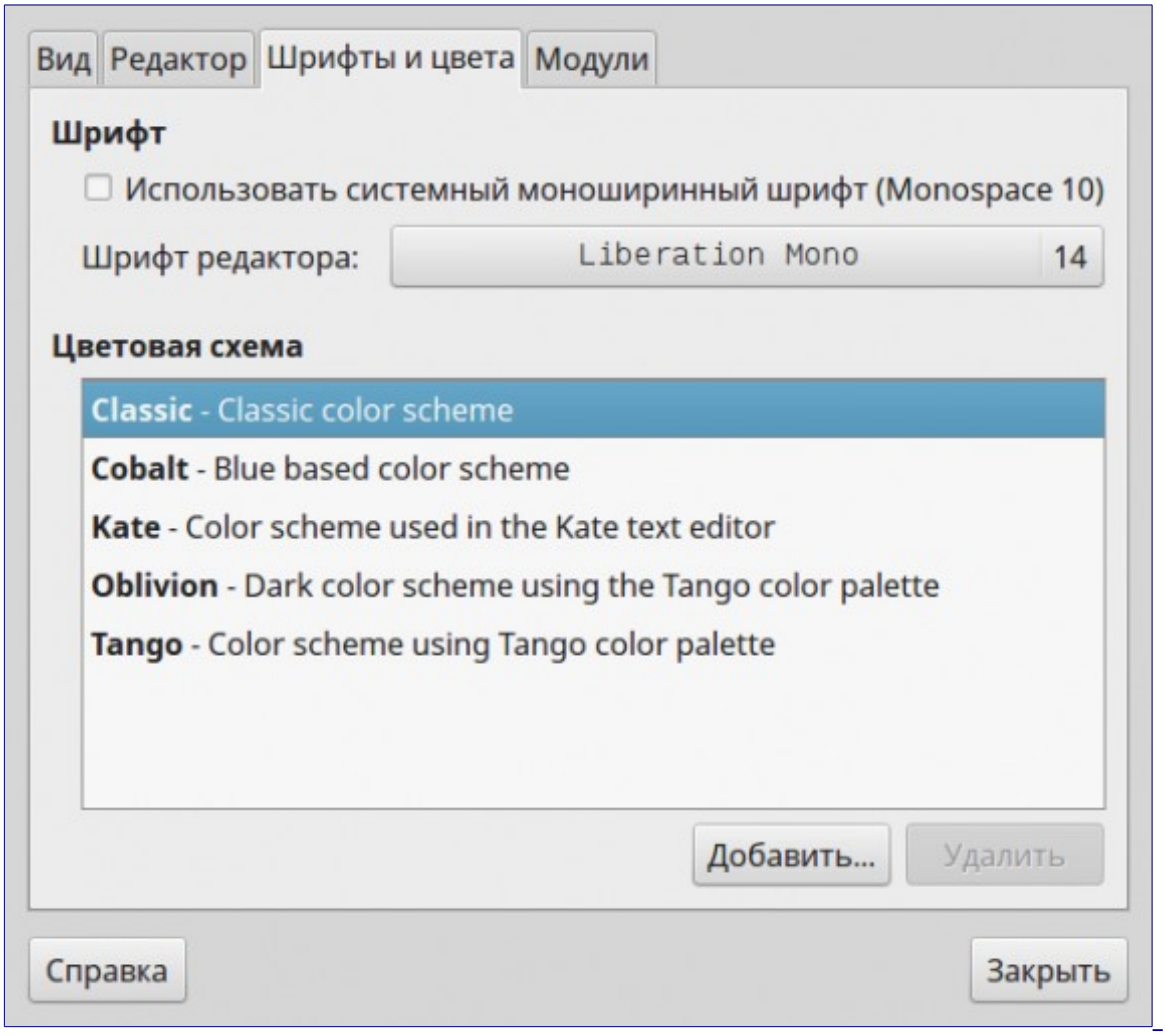
Имеется также проверка орфографии, в том числе и автоматическая, а также статистика документа (в меню Сервис):



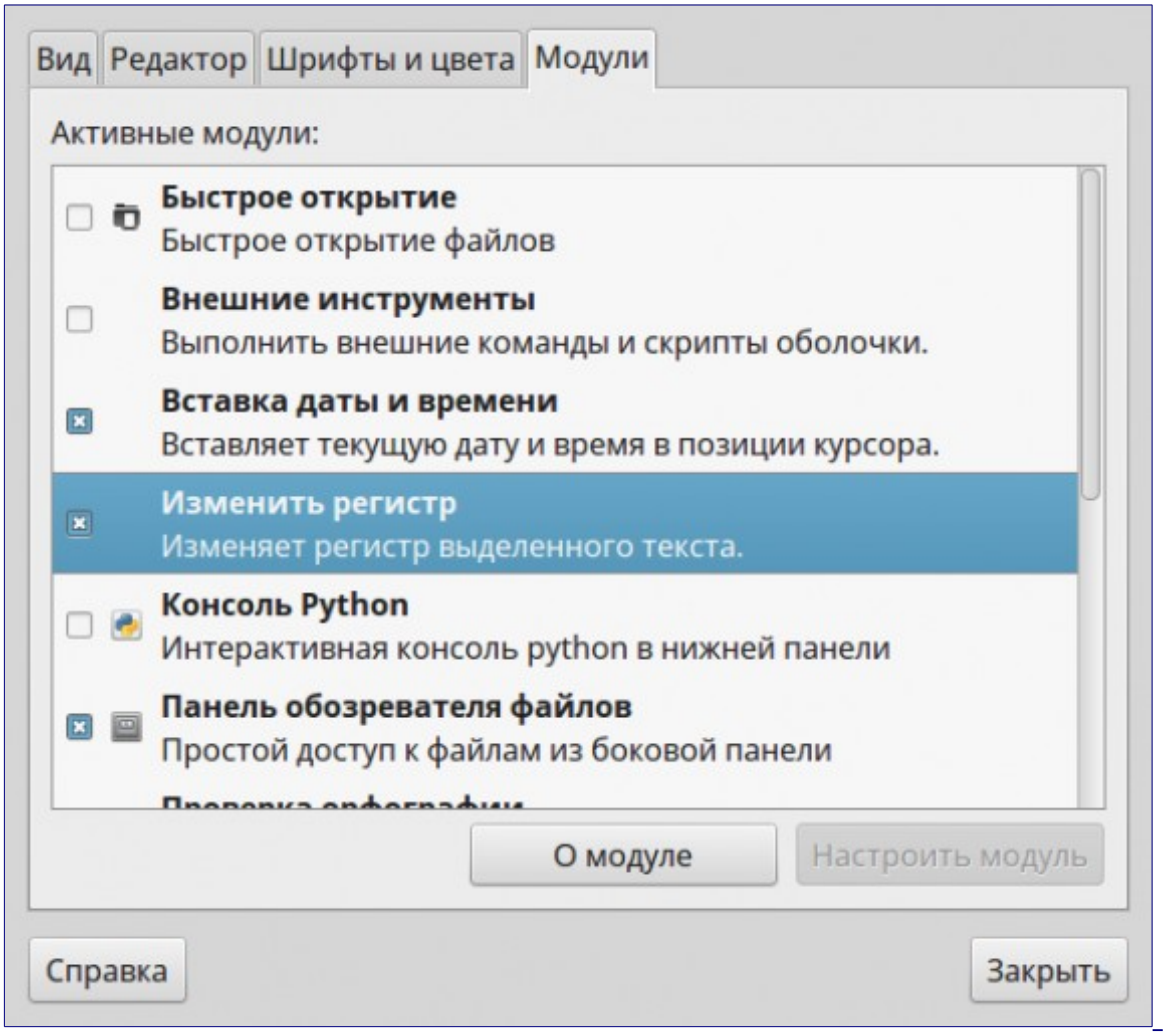
В Gedit настраиваются (Правка -> Параметры) режим «мягкого» переноса, включение/выключение нумерации строк и так далее:



Можно поменять шрифт и цветовую схему:



Во вкладке Модули включаются или отключаются дополнительные функции, такие, как изменение регистра выделенного текста и другие (ряд модулей включён по умолчанию):



В репозитории для Gedit доступно несколько плагинов, большинство из которых ориантировано на специальные задачи разработчиков или Tex-верстальщиков. Однако пакет gedit-plugins — более общего назначения, почему и подлежит и подлежит установке:

```
$ apt install gedit-plugins
```

После этого во вкладке Модули добавляется ряд новых функций, таких, как автоматическое закрытие скобок и кавычек, знакомство с которыми оставляю на усмотрение заинтересованных лиц.

Мне Gedit кажется откровенно скучным. Кроме того, не вполне понятно его позиционирование: в качестве лёгкого редактора, предназначенного для правки конфигов (подобно Mousepad или Leafpad) он явно избыточен, а для всамделишней работы с большими текстами — недостаточно функционален, даже при условии установки дополнительных плагинов. В частности, в нём напрочь отсутствует возможность работы с проектами, что обесценивало для меня любые иные его достоинства, даже если бы таковые нашлись. И потому в следующих очерках я обращусь к возможным альтернативам Gedit'а.

Альтернативные редакторы: введение

Таким образом, Gedit не подходит для сочинения объёмных материалов, хотя бы серии статей в рамках одной темы, не говоря уже о книгах, когда приходится работать со множеством взаимосвязанных текстовых документов. И тут, как ни странно, оказывается, что требования к текстовому редактору со стороны применителя-текстовика часто пересекаются с таковыми

разработчика, сочиняющего что-либо посложнее Hello, world!, с некоторым, иногда большим, числом файлов, предназначенных для реализации одной задачи. И потому обоим нашим героям от текстового редактора требуется средства управления проектами и манипуляций файлами. Причём не обязательно файлами только текстовыми — ведь некоторые авторы имеют обыкновение иллюстрировать своим материалы.

Кроме того, сочинение оригинальных материалов не сводится к их набору из головы (ну или кто чем эти материалы сочиняет). Потому что набор текста оказывается неотделимым от его редактирования. Под которым в рассматриваемом случае подразумевается не только (и не столько) исправление орфографических ошибок и опечаток, сколько стилистическая правка. Которая часто требует свободного перемещёния между строками, абзацами и даже страницами уже набранного текста. И тут очень важной оказывается развитая система кейбиндингов, позволяющая мгновенно, без нудного стучания по клавишам Left и Right, Up и Down, переместиться к участку текста, требующего стилистической правки.

Предвижу возражение: стилистическую правку можно отложить на потом — после полного набора текста. И с негодованием его отвергаю: каждый, для кого стилистика текста — не пустой звук, знает, что удачный стилистический оборот, неожиданно пришедший в голову по ассоциации, может испариться из неё на следующей странице, не оставив ничего, кроме сожаления:

Ай да я, ай да сукин сын: так здорово придумал — и забыл...

Наконец, ещё одна составляющая набора и редактирования текста — разметка. Да, я понимаю стремление отделить ~~мух от котлет~~ набор и вёрстку. Но дело в том, что при работе на онлайн (а большинство нынешних сочинителей с оффлайном завязали) разметка оказывается частью стилистики, и подлежит немедленному претворению в жизнь. Иначе ищи потом ~~ветра в поле~~ фрагмента, который хотел бы выделить перечёркнутым начертанием, тегом strong или emphasis. Не говоря уж о ссылках на использованные по ходу дела материалы, про которые тоже легко забыть, или которые потом приходится долго искать.

В штатном режиме текстовые редакторы предполагают единственный способ ввода тегов (или элементов разметки TeX'а, если речь идёт о подготовке к «бумаге»): вручную. Что а) скучно, б) долго, в) чревато ошибками. И, наконец, просто отвлекает от процесса сочинительства. Но тут на помощь приходит режим записи макрокоманд, привязываемых к хоткеям. А вот этот режим в текстовом редакторе может отсутствовать или присутствовать. А если присутствует — может быть реализованным по разному: просто, но слабо, мощно, но сложно; в идеале, конечно, мощно и просто.

Так что выбор редактора очень сильно влияет на эффективность работы с текстом на всех её стадиях — от сочинения до разметки. И эффективность эта определяется не только наличием «рюшечек и менюшечек» или их отсутствием, а зависит от многих факторов. И потому следующие несколько абзацев посвящены этому вопросу. Только сразу предупреждаю: тем, кто полагает, что Vim/Emacs наше фсио, лучше их не читать. Как, впрочем, и весь этот очерк...

Программисты находят решение своих задач в применении всякого рода IDE, то есть интегрированных сред разработки (Integrated Development

Environment). Интегрированных сред для сочинительства нарративных текстов ещё никем не придумано. Но в этом качестве отлично могут выступить программы, которые лежат там, где кончаются текстовые редакторы и начинаются IDE. А вот их список оказывается очень коротким. Если исключить практически сошедший со сцены NEdit и его клон QEdit, так на эту сцену и не вышедший (в причины обоих явлений вдаваться не буду), то он сведётся к двум с половиной позициям: Komodo Edit за номером один, Geany за номером два и Kate за номером полтора.

Почему столь унижен Kate? Да потому, что, хотя в нём собственные средства управления проектами появились едва ли впервые среди текстовых редакторов такого рода (апологеты Vim и особенно Emacs, подождите немного), по сию пору остаются в зачаточном состоянии. Хотя этот редактор несравненен по части управления файлами, но это — лишь одна из сторон проектного менеджмента. К тому же Kate, будучи приложением KDE, не очень хорошо вписывается в среду Cinnamon.

Между тем Geany, хотя по части управления файлами он и отстал от Kate, все остальные аспекты управления проектами развивались со страшной научно-фантастической силой. И ныне этот некогда просто текстовый редактор именуется обычно лёгкой IDE. Хотя именно вследствие своей лёгкости, то есть — не перегруженности чисто программистскими фичами, он отлично подходит на роль основного инструмента применителя-текстовика.

Однако оказывается, что Komodo Edit превосходит Geany и по части управления проектами (в частности, позволяя одновременно держать открытыми несколько разных), так и в плане файловых манипуляций, включая в себя полноценный файловый менеджер с функциями просмотра изображений. Средства навигации по тексту в нём чрезвычайно изобильны и настраиваемы беспредельно. А уж по части сочинения макросов с ним не может конкурировать ни один текстоый редактор из тех, что я видел.

А что же Emacs и Vim? — спросите вы меня. Отвечаю: это те самые попытки с негодными средствами, о которых я упоминал в начале этой страницы. Да, и из того, и из другого мострактора (монстроидального редактора) можно сделать интегрированную среду для разработки любого рода текстов — от исходников до «нарративников». Но её придётся делать, затрачивая силы и время. А подчас ещё и приобретая по ходу дела некие специфические навыки, например, для Emacs'a — программирования на Lisp. Навыки, которые применителю-текстовику больше никогда и нигде не понадобятся.

Поэтому меня умиляет, когда на всяких форумах на вопрос о выборе текстового редактора технологически продвинутые граждане в качестве универсального решения предлагают Vim/Emacs (в зависимости от своей религиозной ориентации). Для меня это просто показатель непонимания означенными гражданами специфики сочинительской работы. А ведь профессиональный сочинитель воспримет (и воспринимает) такой совет как форменное издевательство. Ибо в том же Geany или Komodo Edit он мог бы иметь всё ему необходимое «искаропки» — хотя он об этом пока и не подозревает.

Так что далее будут последовательно рассмотрены обе редакторские альтернативы — каждая из них заслуживает отдельного полноценного очерка.

Текстовый редактор Geany

Вступление

Текстовый редактор Geany (буду называть его так, хотя он и представляется как лёгкая IDE) разрабатывается Энрико Трёгером (Enrico Tröger) и Ником Трелевеном (Nick Treleaven), базируется на библиотеке Gtk 2, распространяется под лицензией GNU GPL v2 (по крайней мере до сих пор). Он присутствует присутствует в официальном репозитории Mint и устанавливается командой

```
$ apt install geany
```

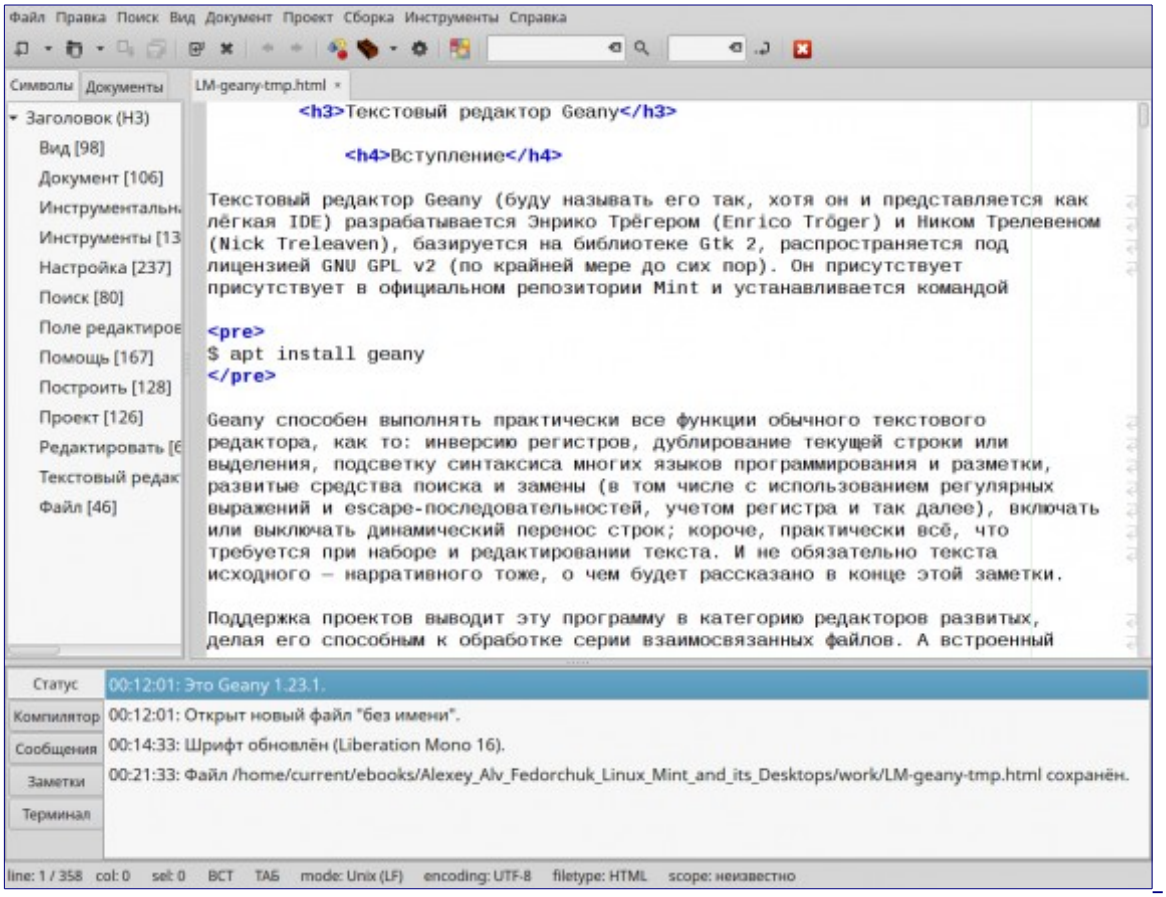
Geany способен выполнять практически все функции обычного текстового редактора, как то: инверсию регистров, дублирование текущей строки или выделения, подсветку синтаксиса многих языков программирования и разметки, развитые средства поиска и замены (в том числе с использованием регулярных выражений и escape-последовательностей, учетом регистра и так далее), включать или выключать динамический перенос строк; короче, практически всё, что требуется при наборе и редактировании текста. И не обязательно текста исходного — нарративного тоже, о чем будет рассказано в конце этой заметки.

Поддержка проектов выводит эту программу в категорию редакторов развитых, делая его способным к обработке серии взаимосвязанных файлов. А встроенный эмулятор терминала полезен не только программистам, но незаменим также для линуксописателей. Автодополнение языковых конструкций (имеются ввиду языки программирования и разметки) — также функция, подчас не лишняя для простых юзеров, имеющих дело, например, с созданием HTML-документов.

Настоящая заметка посвящена общему описанию редактора Geany и методам его использования при работе с обычными текстами и HTML-документами. Не будучи программистами, авторы не затрагивают вопросы применения этой программы в качестве собственно IDE.

Запускается Geany из главного меню панели задач или рабочего стола (Разработка -> Geany), после чего в открытом окне программы можно видеть следующие интерфейсные элементы (рис. 1):

- заголовок с именем текущего открытого файла и указанием полного пути к нему;
- строку главного меню;
- панель инструментов;
- боковую панель;
- окно ввода и редактирования текста с вкладками открытых документов по верхнему краю;
- окно сообщений;
- статусную строку.



Вид главного меню предопределён используемой в Geany библиотекой Gtk+, остальные же элементы, в терминологии программы именуемые виджетами, настраиваются внутренними её средствами.

Основные элементы интерфейса редактора — окно ввода, боковая панель и окно сообщений — масштабируемы, как, разумеется, и главное окно; выполненные в сеансе изменения размеров можно сохранить навсегда, о чем будет сказано в разделе про настройку программы.

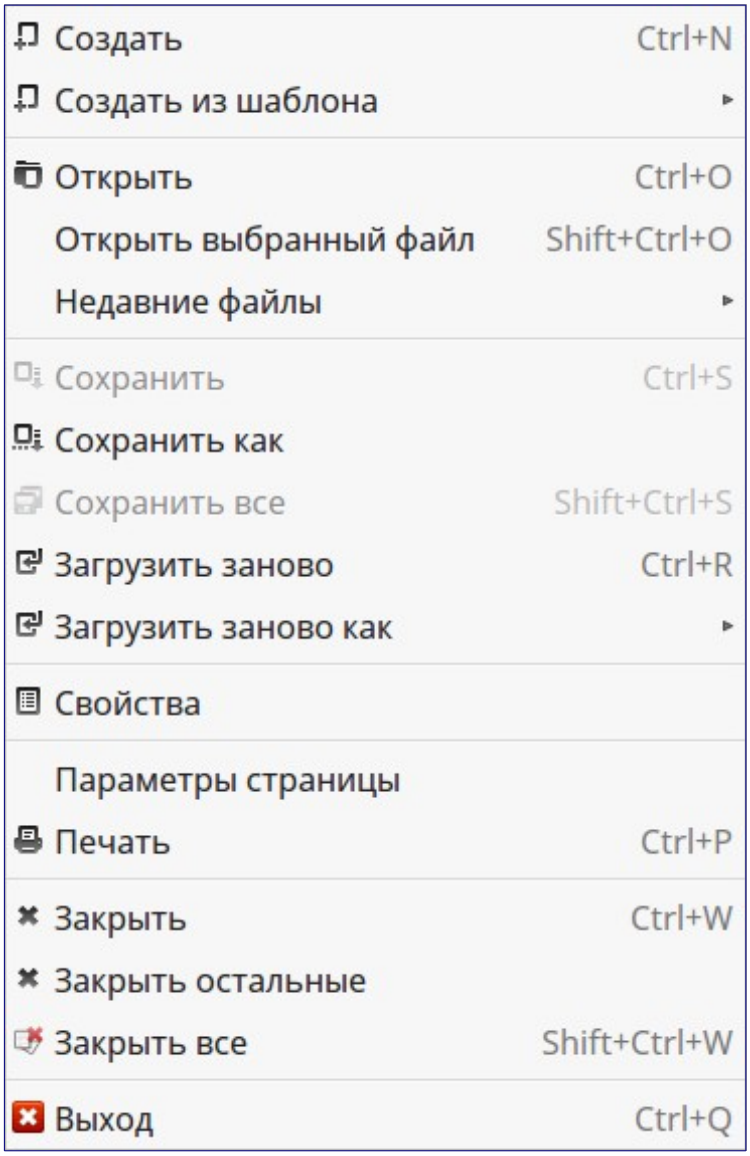
Главное меню программы включает следующие пункты:

- Файл;
- Правка;
- Поиск;
- Вид;
- Документ;
- Проект;
- Сборка;
- Инструменты;
- Справка.

Рассмотрим эти пункты последовательно.

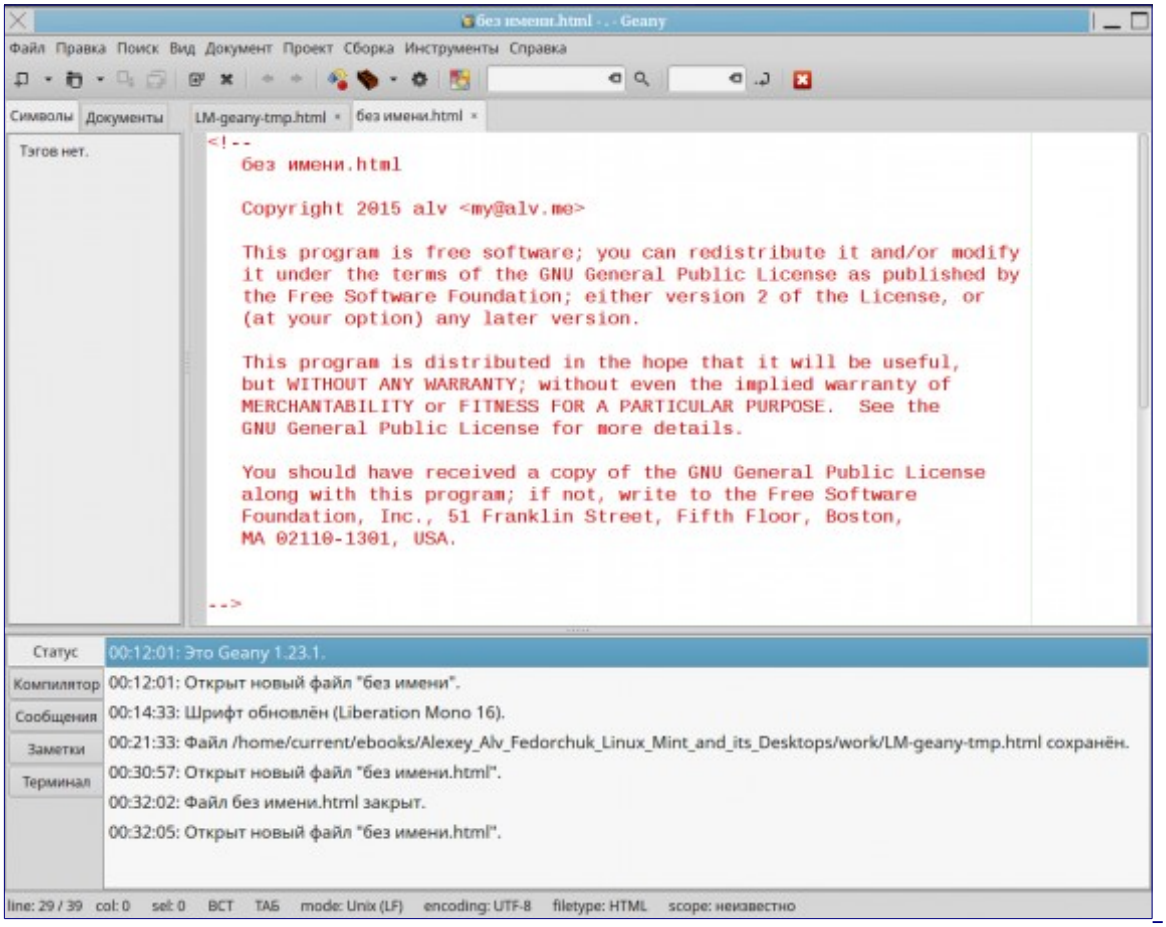
Файл

Пункты меню Файл сгруппированы в несколько блоков:

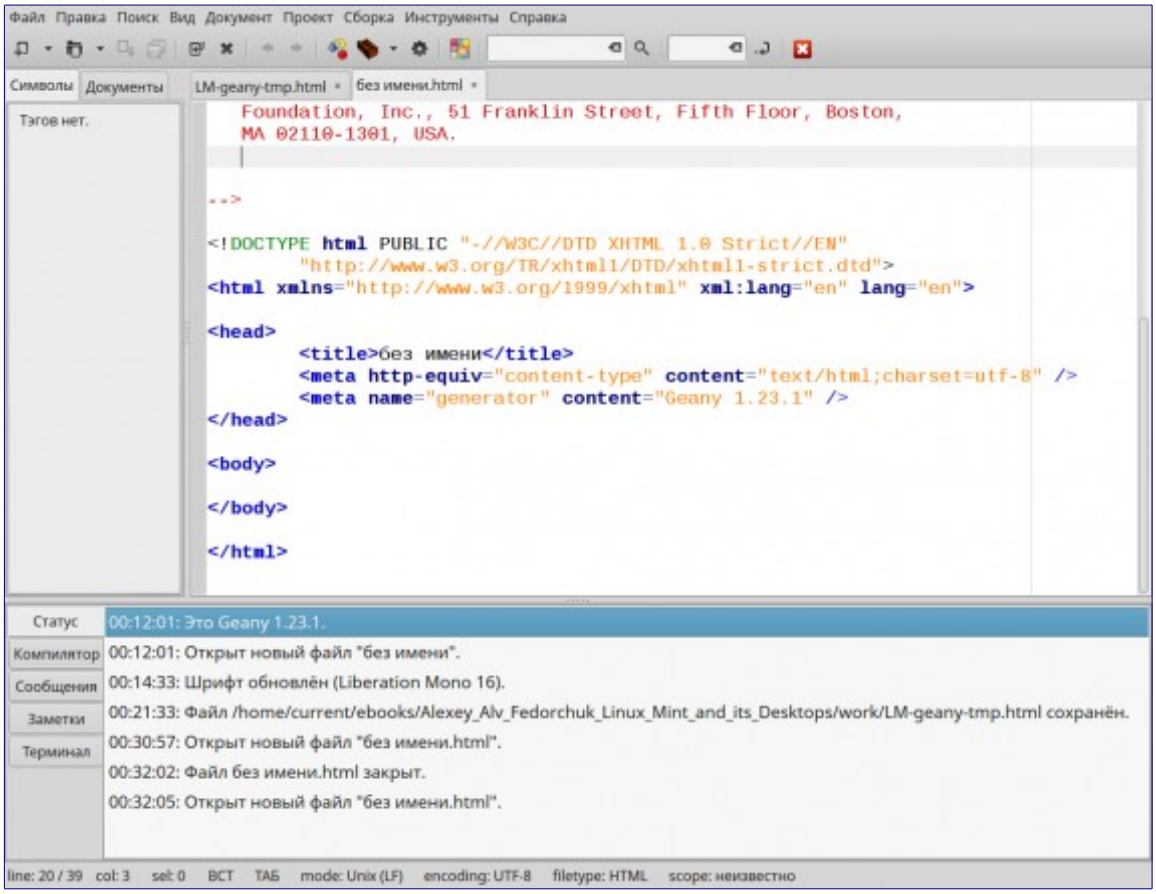


Первый из них посвящен созданию новых файлов. Пункт Создать предполагает открытие в окне редактирования пустого документа. Пункт Создать из шаблона предоставляет на выбор с десяток вариантов, позволяющих создать исходный файл с предопределённым шаблоном для нескольких языков программирования (Си, Си++, PHP, Python, Ruby и так далее) и разметки (html, tex).

В начале каждого шаблона содержится комментарий (обозначенный в соответствие с синтаксисом выбранного языка), включающий имя файла, указание на копирайт создателя (откуда оно берётся — мы увидим позднее) и традиционный для программ Open Source отказ от гарантий:



Далее следует «скелет», типичный для данного языка. Например, для HTML-файла он выглядит следующим образом:



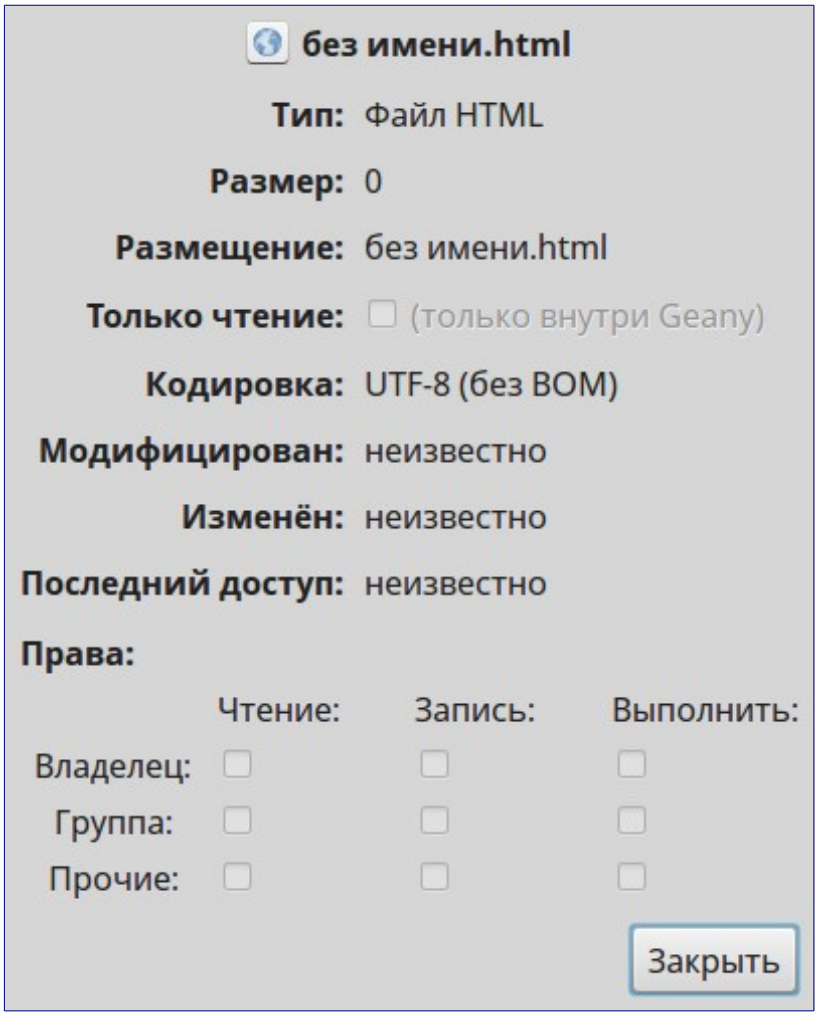
Сначала идет определение типа документа (!DOCTYPE) и тег html с соответствующими атрибутами. Затем — заголовочный блок с титулом HTML-

страницы, указанием набора символов (соответствующим по умолчанию текущей локали) и программы-генератора (то есть самой Geany), открывающий и закрывающий теги body и закрывающий тег html. Рассмотрение шаблона показывает, что он соответствует спецификации XHTML, поэтому при создании чистого HTML-документа (pure html) лучше начинать это дело с «чистого листа».

Следующий блок пунктов меню Файл касается открытия существующих документов, том числе выбранного файла и одного из списка недавно открывавшихся документов (по умолчанию в списке десять позиций).

Блок сохранения файлов включает пункты: Сохранить (текущий файл), Сохранить как, то есть под другим именем (если файл был создан из шаблона — это единственно доступный вариант, причём соответствующий суффикс, например .html, выводится автоматически), Сохранить все (открытые документы), Загрузить заново, то есть считать документ заново, например, если он был изменён внешней программой (с потерей несохранённых результатов текущего редактирования) и Загрузить заново как, что предоставляет возможность сменить текущий набор символов (по-простому говоря, изменить кодировку документа).

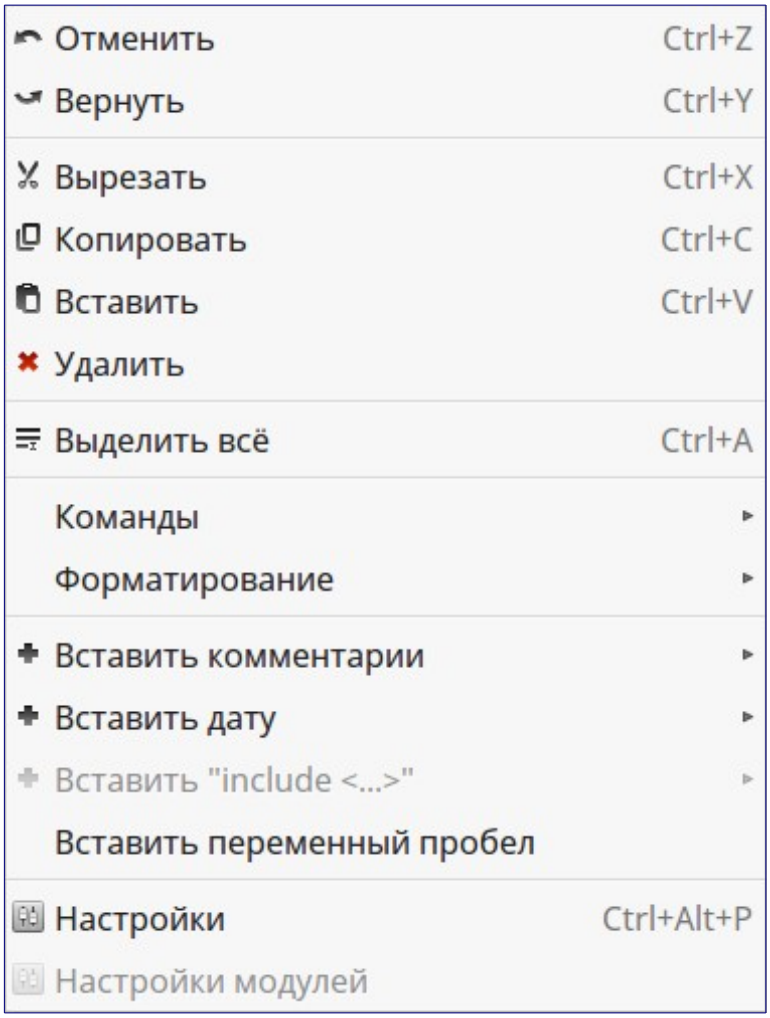
Пункт Свойства вызывает панель с указанием типа файла, его размера и полного пути к нему, кодировки, атрибутов времени (модификации, изменения статуса, последнего доступа), принадлежности и прав доступа:



Далее следуют пункты, относящиеся к печати, закрытию (текущего документа или всех открытых) и, наконец, выход из программы.

Правка

В меню Правка также имеет блочную структуру:



Сначала следует блок пунктов для обычных действий над текстом — Отменить и Вернуть (последние изменения, откат и возврат многоступенчатые), Вырезать, Копировать, Вставить и Удалить, а также Выделить всё.

Все эти операции дублируются стандартными для современных GUI комбинациями клавиш, типа Control+X, Control+C и Control+V для вырезания, копирования и вставки выделенного фрагмента соответственно. Причём ныне комбинации эти работают и при русской раскладке клавиатуры.

Пункт Форматирование позволяет:

- переключить регистр выделенного текста;
- закомментировать строку или снять с неё комментарий;
- продублировать строку или выделенный фрагмент текста;
- увеличить или уменьшить отступ текущей строки или выделенной группы строк;
- отправить выделенный текст на обработку какой-либо команде, после чего вывод этой команды заменит исходное выделение; правда, команды обработки предварительно нужно определить — это делается в

этом же подпункте; данная возможность неоценима для линуксописателя (и не только для «линуксо-»).

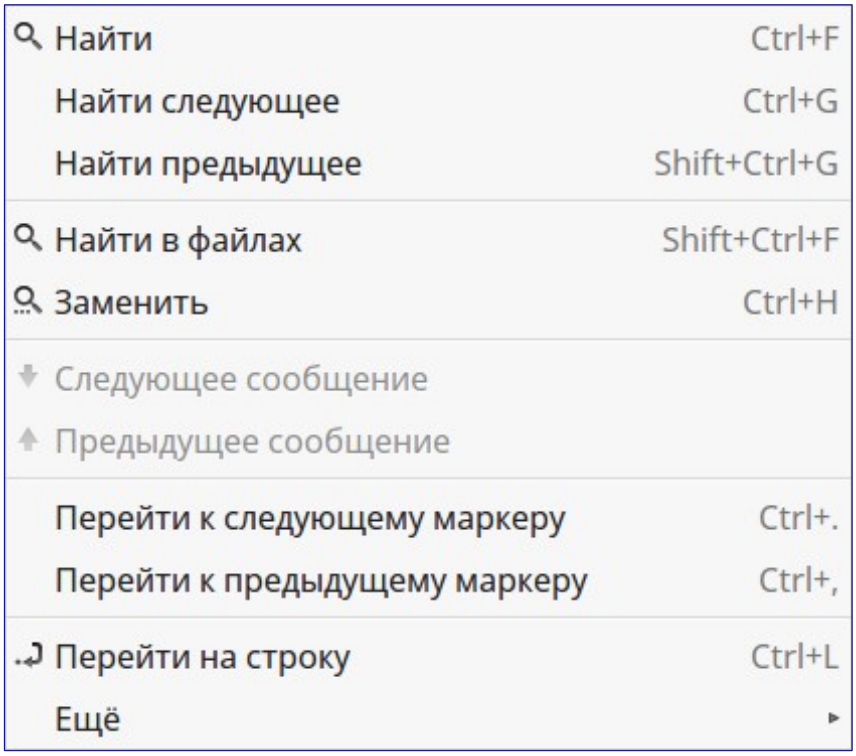
Пункт Вставить комментарии подразумевает нечто совсем иное, нежели близкий по звучанию подпункт из Форматирования. Он предлагает на выбор включить в документ такие фиксированные фрагменты, как отказ от гарантий (о котором говорилось ранее), BSD- и GPL-уведомления. Хотя и просто закомментировать несколько пустых строк можно тоже.

Далее идут пункты Вставить дату (с возможностью выбора формата оной) и Вставить include (по умолчанию не активизировано).

И, наконец, пункт Параметры: в нём можно выполнить настройки программы, к которым мы вернёмся после того, как рассмотрим возможности, предоставляемые Geany без всяких настроек.

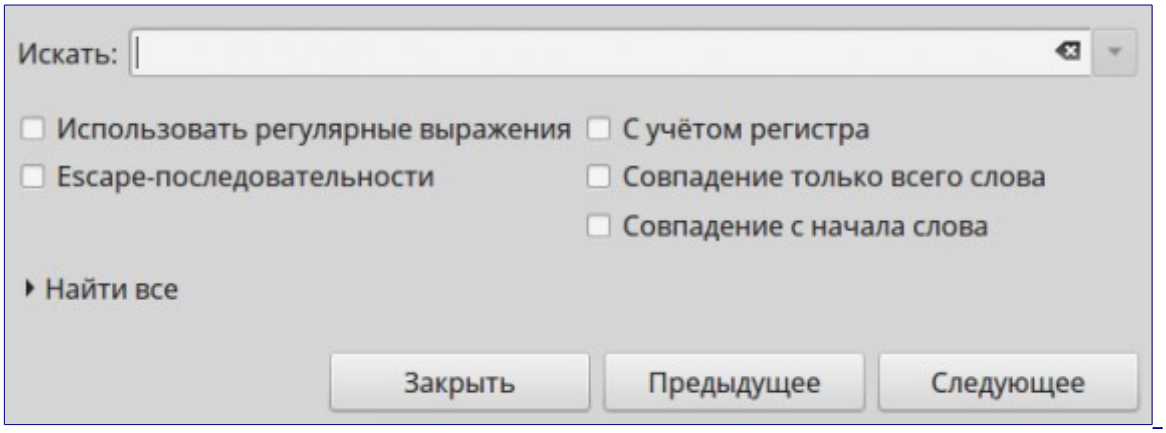
Поиск

Развитые функции поиска и замены — одно из важнейших отличий «настоящих» текстовых редакторов от «буквонабивалок» класса Notepad'a. И Geany здесь стыдиться нечего:

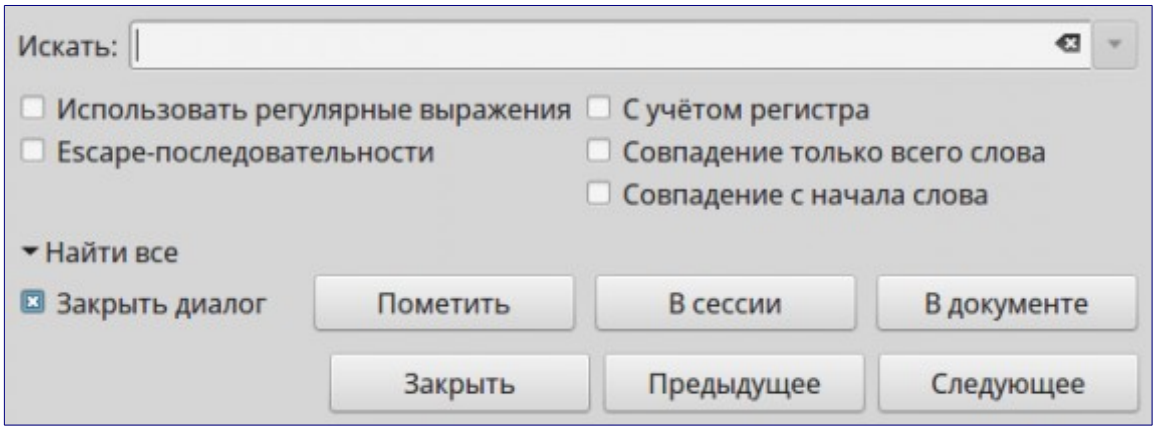


При запуске поиска на экране появляется панель со строкой, где вводится искомая последовательность символов, и серией чекбоксов, определяющих его условия. Поисковая панель не блокирует доступ к окну редактирования; в частности, в обрабатываемом тексте можно выделить фрагмент и щелчком средней кнопки мыши и операцией копирования поместить его в поисковую строку. Предусматривается поиск в двух режимах — обычном и расширенном.

Обычный режим поиска предполагает поиск последовательности символов, поиск последующего или предыдущего, осуществляемые в пределах текущего документа. Возможно использование регулярных выражений, учёт регистра и так далее:

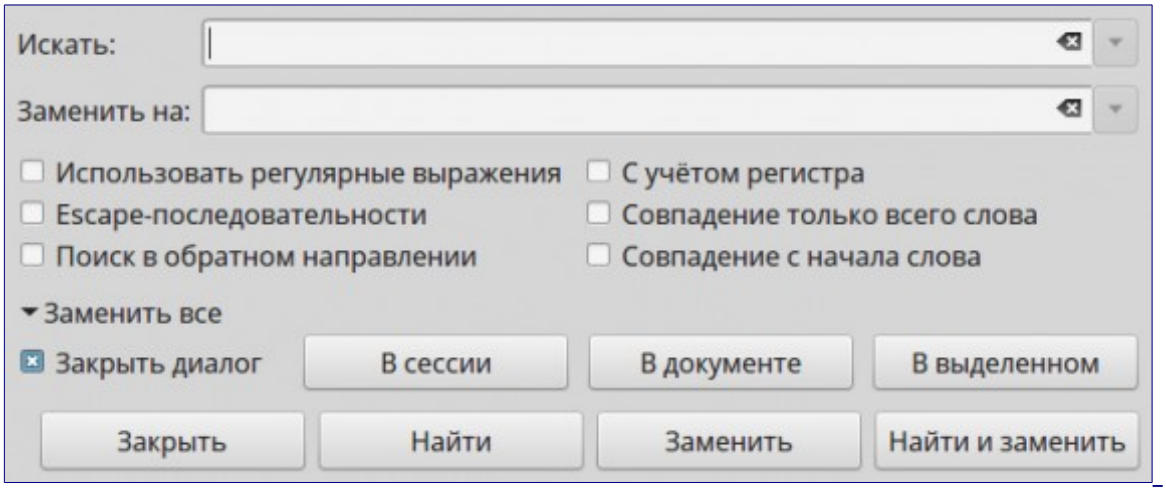


Перейдя с помощью чекера Найти всё в расширенный режим поиска, оный можно осуществить в текущем документе, во всех открытых файлах, а также установить метки на строки, содержащие искомую последовательность символов.



Пункт Найти в файлах позволяет указать каталог, во всех файлах которого следует выполнить поиск введённой последовательности символов, в том числе, при желании, и в подкаталогах любой степени вложенности. При поиске возможно использование обычных и расширенных регулярных выражений для grep, который, судя по всему, и выполняет собственно поиск в этом случае.

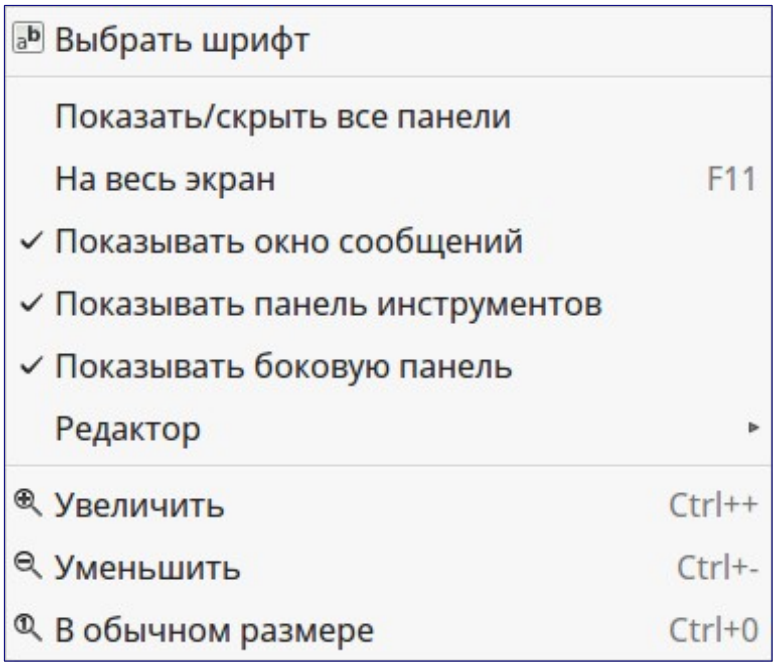
Поиск предполагает возможность замены найденного — и таковая в Geany имеется. И также осуществляется в двух режимах. В обычном режиме замена происходит последовательно, по мере нахождения очередной подлежащей замене последовательности символов. Нажав всё ту же кнопку Заменить все, можно перейти в расширенный режим, дающий возможность замены в выделенном фрагменте текста, в документе целиком, а также в сессии, то есть во всех документах, открытых в данный момент



Возможностью, предоставляемой функцией Найти выделенное нас не удивить — собственно, и при обычном поиске в строке для оного по умолчанию будет помещён выделенный фрагмент, если таковой имелся. А вот с функцией Найти предыдущее выделенное мы ещё не сталкивались — что не делает её менее полезной в ряде ситуаций. Ну и Переход на строку (по заданному её номеру) видится вполне уместным среди функций поиска и замены.

Вид

В меню Вид, как нетрудно догадаться, устанавливается визуальное представление как интерфейсных элементов редактора, так и редактируемого документа:



Так, первый его пункт, Выбрать шрифт, изменяет таковой в окне редактирования — но только на время текущего сеанса (глобальная замена шрифта выполняется через пункт Правка -> Параметры, до которого мы со временем доберёмся).

Пункт Показать/скрыть все панели убирает с экрана панель инструментов, вкладки для открытых документов, окно сообщений и статусную строку — словом, всё, кроме боковой панели (которая, как мы сейчас увидим, тоже может быть отключена) и окна редактирования, максимально расширяя

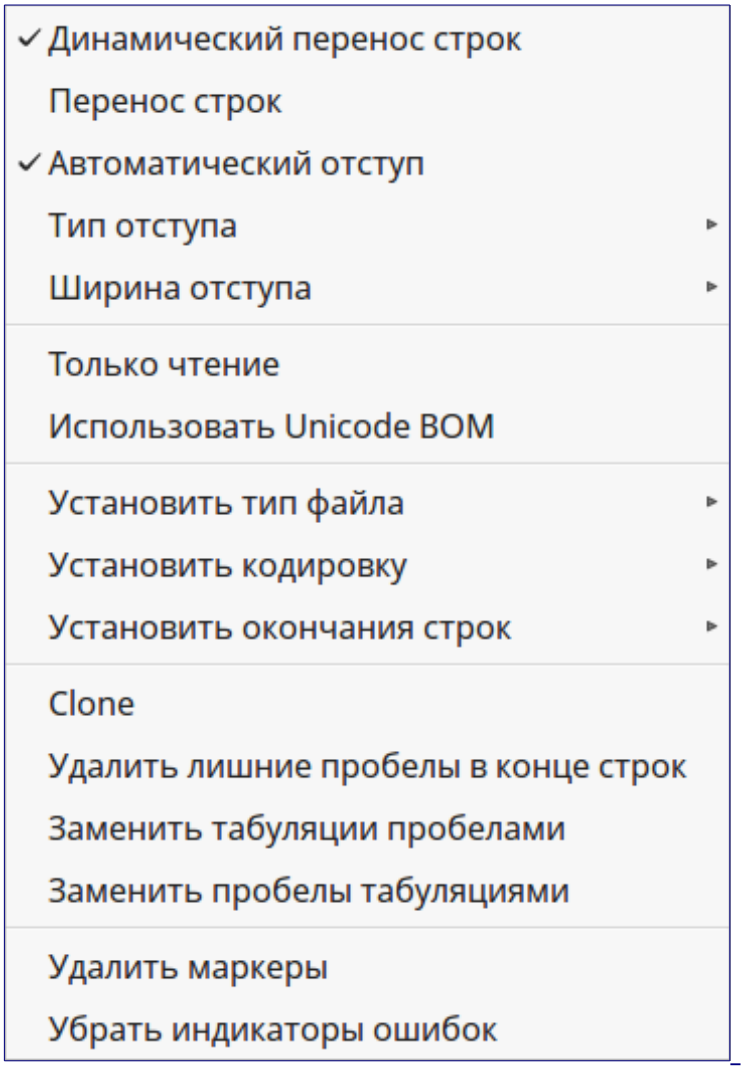
пространство для последнего. Хотя нет, максимальным поле для приложения своих писательских склонностей становится, если включить ещё и полноэкранный режим, что делается через следующий, одноименный пункт меню Вид (или достигается нажатием клавиши F11).

Далее следует блок пунктов, включающих или отключающих основные интерфейсные элементы главного окна редактора, такие как окно сообщений, панель инструментов, боковая панель, маркер строк (то есть подсветка строки, на которой расположен курсор), номера строк. В комментариях это, видимо, не нуждается.

Пункты Увеличить, Уменьшить и В обычном размере выполняют масштабирование содержимого окна редактирования, подобно тому, как это делается в браузерах со времен Netscape какой-то бородатой версии. Кстати, и выполняется масштабирование также теми самыми, вот уже много лет привычными клавишами — Control+»серый плюс», Control+»сервый минус» и Control+0, а не только через меню.

Документ

Пункты меню Документ включают или выключают динамический перенос строк и использование автоотступов (только в текущем сеансе, для увековечивания установленной ситуации нужно обратиться всё к тем же Параметрам), определяют представление отступа — символами табуляции или наборами пробелов, устанавливают для текущего документа режим «только для чтения» (лишь в пределах Geany, изменять его внешними программами никто не запрещает):



Далее следуют пункты:

- использовать кодировку Unicode с меткой порядка байтов (BOM), что, насколько нам известно, необходимо только для различения вариантов UTF-16 и UTF-32;
- установить тип файла для:
  - языка программирования (ассемблер, Си, Си++, Java и так далее),
  - языка скриптинга (shell, Perl, Python, PHP, Ruby, JavaScript и др.),
  - языка разметки (CSS, DockBook, HTML, XML), и
  - прочих языков (конфиги, diff-файлы, LaTeX и так далее),

  в соответствие с чем осуществляются подсветка синтаксиса и автодополнение языковых конструкций; например, в файле, тип которого определен как HTML, закрывающие теги будут ставиться автоматически после набора тега открывающего;
- установить кодировку — они сгруппированы по регионам (западноевропейские, восточноевропейские, восточноазиатские и прочие), внутри которых уже выбираются собственно наборы символов; кириллические кодировки, если используется не UTF-8, следует искать

среди восточноевропейских;

- установить символ конца строки в стиле Unix, DOS или Mac;
- убирать остаточные пробелы и заменять символы табуляции соответствующим количеством пробелов;
- удалить маркеры на строках, которые были помечены при поиске;

ещё раз напомним, что все установки в меню Документ действуют только для текущего файла.

Проект

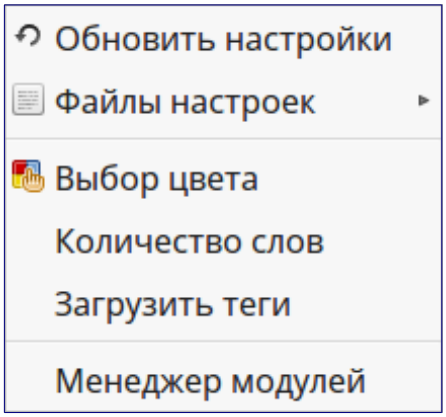
Средства управления проектами в Geany, как штатные, так и альтернативные,, реализованные в качестве плагинов, будут предметом отдельного мини-очерка.

Сборка

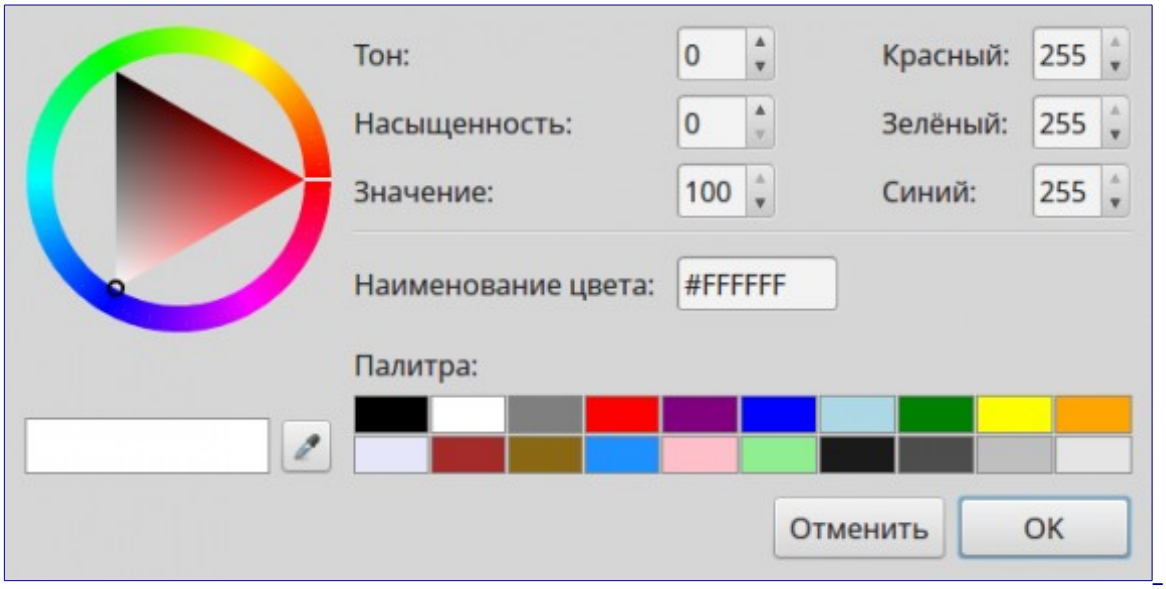
Это меню предназначено в основном для сочинителей исходных текстов. Однако один из его пунктов, а именно, Выполнить, может представлять интерес и для тех, кто сочиняет тексты просто. В частности, файл HTML при выборе этого пункта будет просто-напросто открыт в браузере. Каком — поговорим чуть позже.

Инструменты

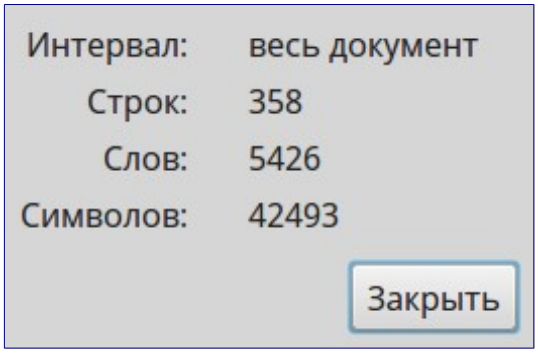
В меню Инструменты можно видеть такие пункты:



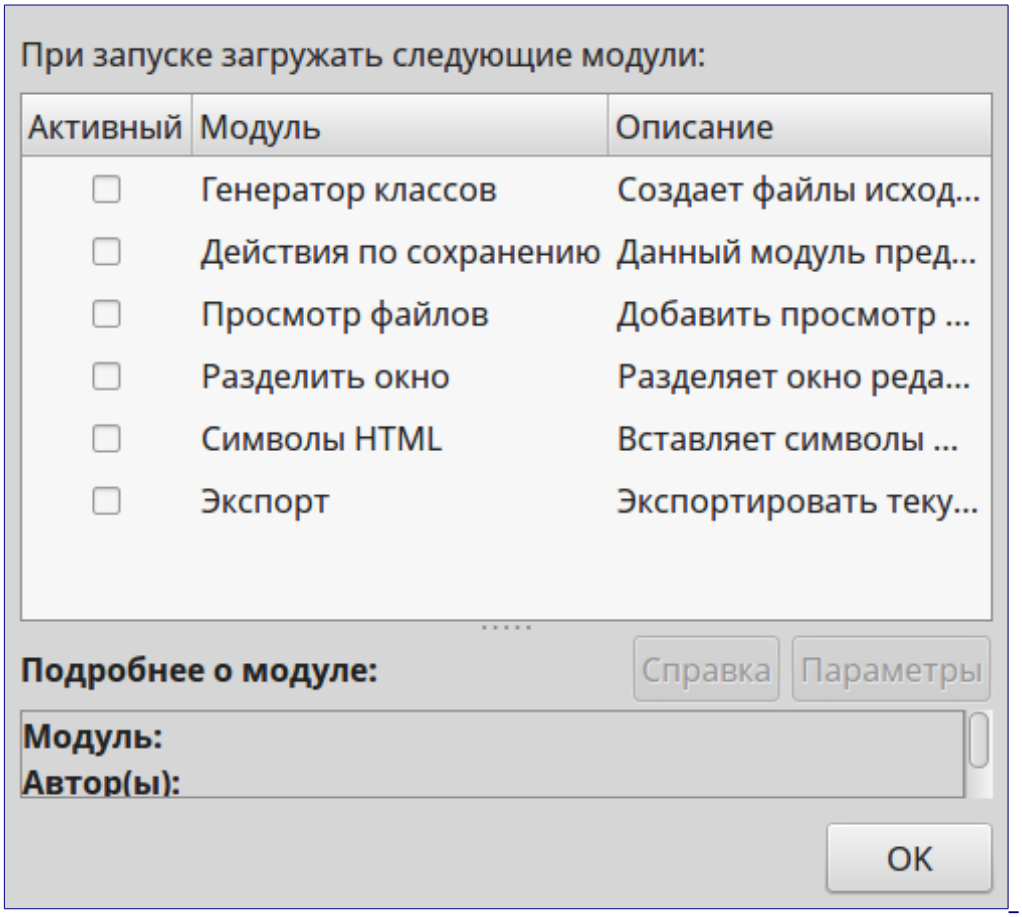
Из них интересен, во первых, пункт Выбор цвета — он выводит панель, в которой можно выбрать цвет из палитры, задать его значение численно или определить, с помощью «пипетки», по образцу, ткнув в любую область экрана (не обязательно в пределах окна Geany)



Далее, Подсчёт слов — абсолютно необходимый инстструмент всякого профессионального сочинителя (то есть зарабатывающего сочинительством на хлеб). Ибо он выводи не только количество слов, но также строк и, главное, символов (с пробелами), для всего документа или выбранной его части:



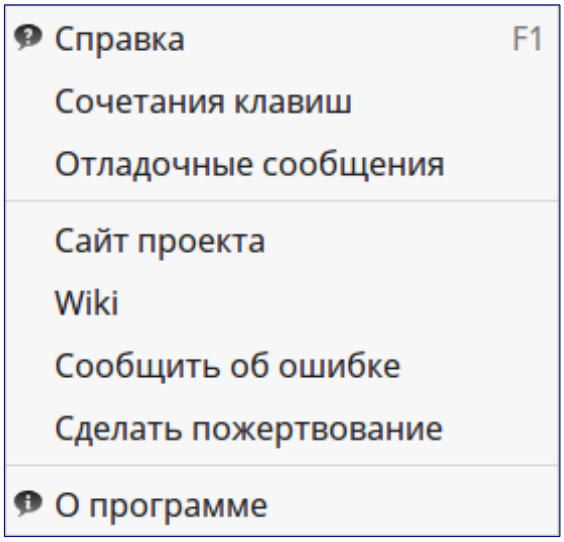
И, наконец, Менеджер модулей — в этом пункте можно включить использование различных плагинов плагинов:



Подробнее этот вопрос будет рассмотрен в соответствующем миниочерке. А разбираться с прочими пунктами меню Инструменты япредоставляю заинтересованным лицам.

Справка

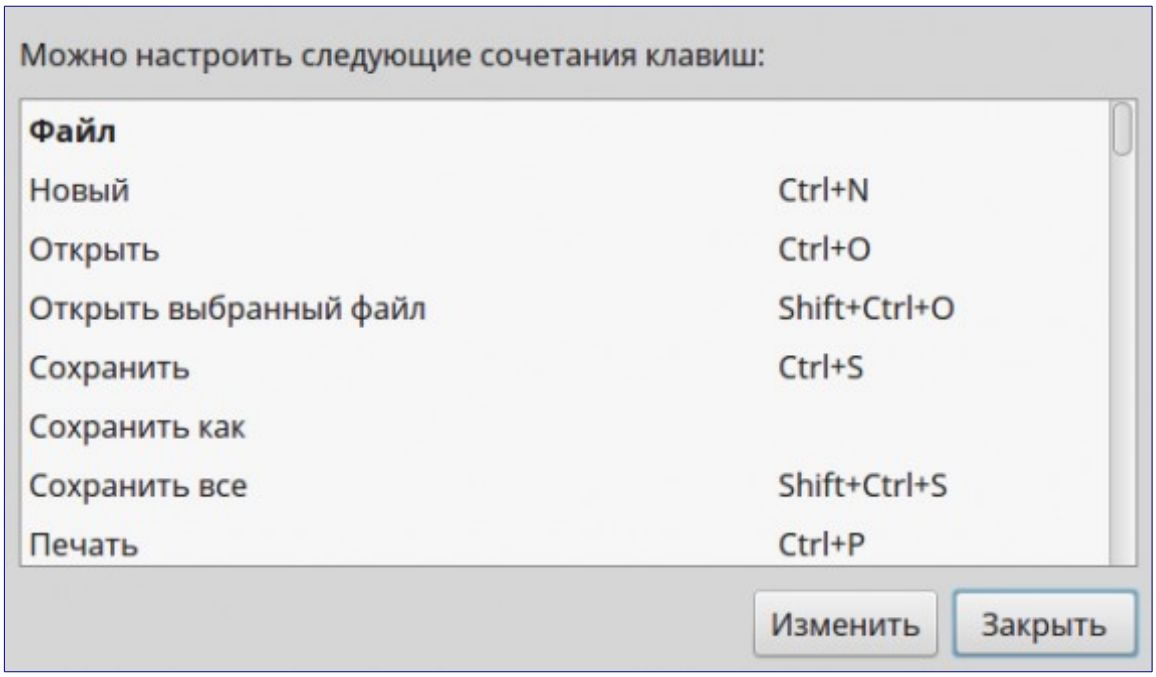
Во многих свободных программах это вполне формальный пункт главного меню, сводящийся к указанию официального сайта проекта, списка его участников, лицензии и тому подобных элементов матрицы Остапа Бендера «Азиатский орнамент». Однако Geany принадлежит к тем немногим программам Open Source, которые являют собой приятное исключение.



Первый пункт меню, собственно Справка, вызывает очень подробную (хотя и англоязычную) документацию по программе в формате HTML. Причём вызывает не из Сети, а с локальной машины, куда она была помещёна при

инсталляции. Останавливаться на её содержании я не буду, но к прочтению всячески рекомендую.

Следующий пункт — Сочетания клавиш. Это не просто справка по существующим клавишным комбинациям, а руководство к действию, о чём недвусмысленно говорит надпись сразу под заголовком панели:



И если последовать совету разработчиков и нажать кнопку Изменить, то оказываешься как раз в том месте настроек Geany, в котором горячие клавиши и можно переопределить. И где мы скоро окажемся.

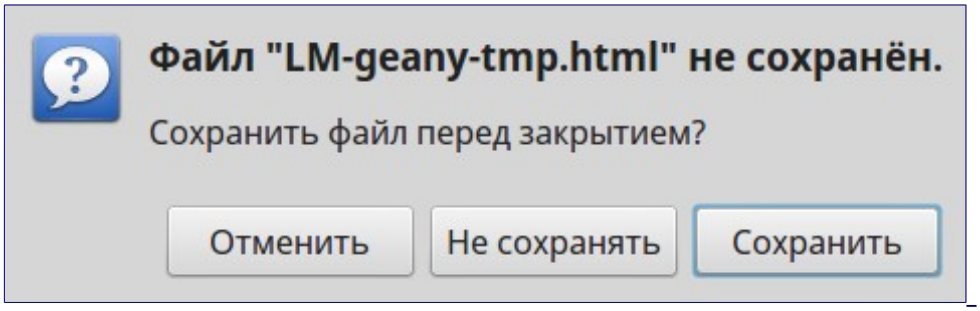
Далее можно увидеть ссылку на официальный сайт проекта — также весьма информативный. Во всяком случае, в разделы Manual и FAQ заглянуть явно стоит. Как и в следующий пункт, Wiki, который приведёт нас вот сюда.

И, наконец, в пункте О программе приведены сведения о её разработчиках и переводчиках интерфейса.

Инструментальная панель

Инструментальная панель включена в редакторе Geany по умолчанию, хотя, как мы видели, расширения рабочего пространства ради, её можно и убрать — временно, через меню Вид, или постоянно, через пункты Правка -> Параметры. Действия через пиктограммы в основном дублируют основные операции, доступные через главное меню, хотя некоторые из них и своеобразны.

Первые две пиктограммы в ряду инструментов — создание нового файла и открытие существующего. Далее следуют две пиктограммы — сохранения текущего файла и сохранения всех открытых в сеансе документов, а также иконка перечитывания текущего файла с диска, а затем косой серый крестик закрытия текущего документа; если последний содержал не сохранённые изменения, последует запрос на подтверждение действия с вариантами — Отменить, Не сохранять и Сохранить:



Стрелки Назад и Вперед подобны таковым в браузерах, только перемещают они в пределах текущего документа — на предыдущую и последующую позиции курсора.

Три следующие пиктограммы вызывают компиляцию текущего файла, его сборку (на разнице между этими понятиями останавливаться не буду) и просмотр или запуск (в зависимости от типа).

Следующая в этом ряду пиктограмма — выбор цвета, который происходит точно так же, как было описано в разделе про меню Инструменты.

Далее следует два поля ввода. В первом можно поместить текст для поиска, во втором — номер строки, на которую требуется перейти. Нажатие на сопровождающие их кнопки вызывает соответствующие действия.

И последняя на панели инструментов пиктограмма — выход из редактора с запросом на сохранение не записанных перед тем изменений.

Мы привели набор пиктограмм, имеющихся в инструментальной панели по умолчанию. В одном из следующих разделов мы увидим, что он может быть как пополнен (хотя и не в очень широких рамках), так и урезан произвольным образом.

Поле редактирования, боковая панель и окно сообщений

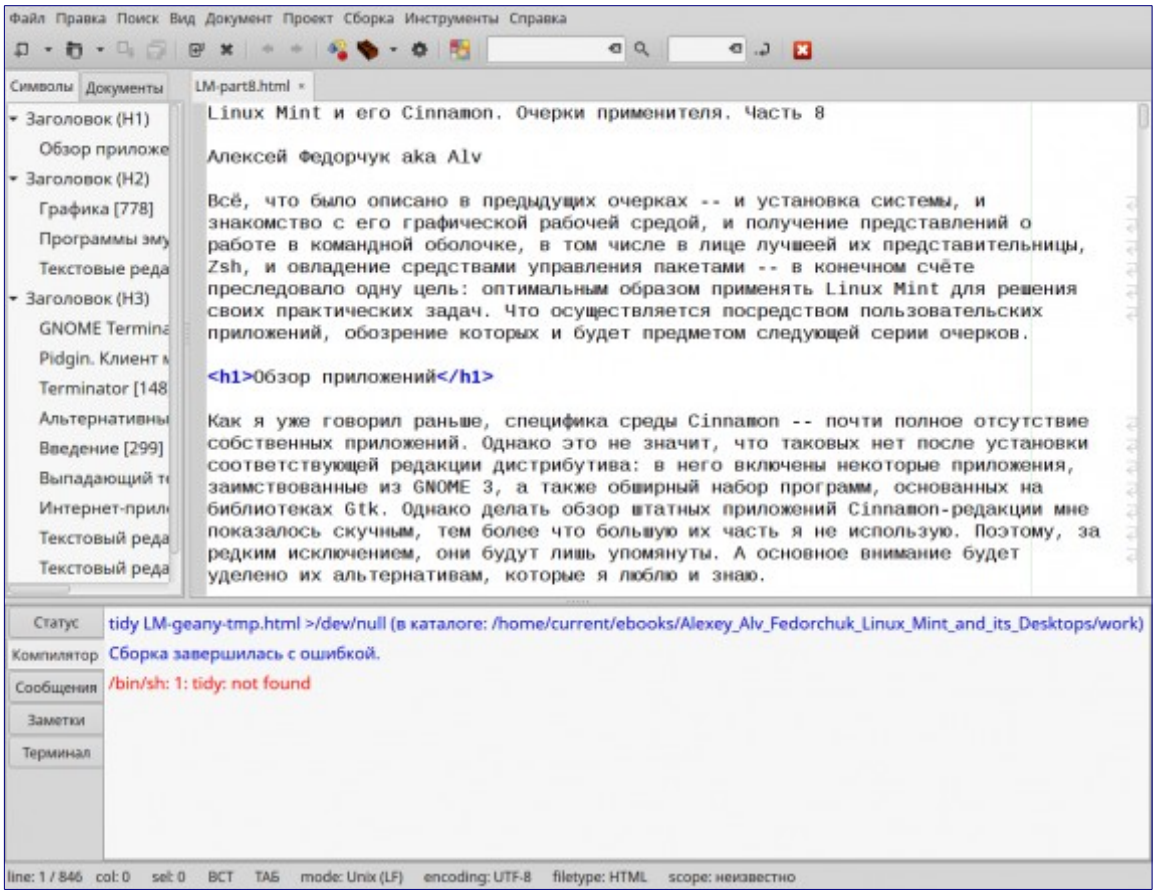
Покончив с обзором элементов управления редактором — главного меню и инструментальной панели, перейдём к основным рабочим областям его интерфейса.

Главная рабочая область текстового редактора — это, разумеется, поле ввода и редактирования текста. Но как раз про него-то можно сказать меньше всего — разве только то, что в нём действительно можно вводить и редактировать текст :), и что оно имеет полосу прокрутки оного.

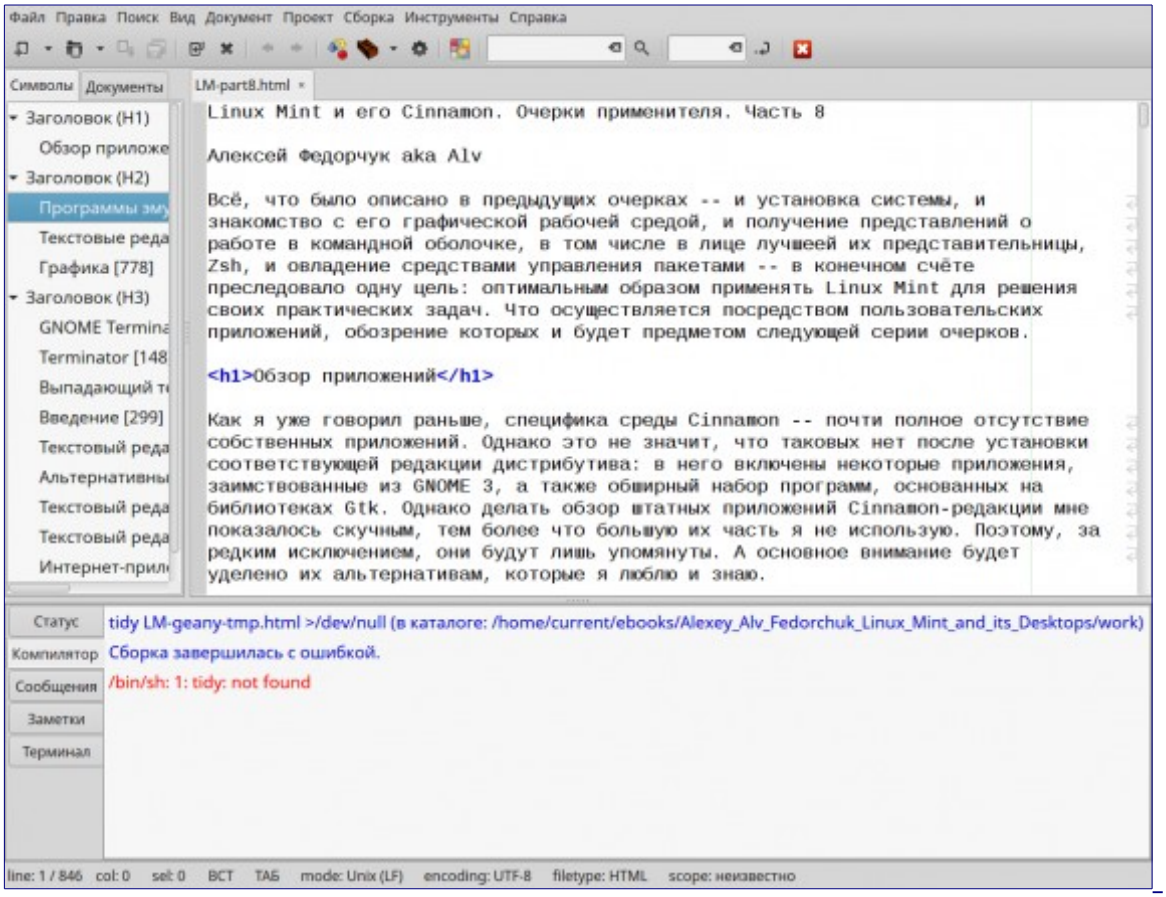
Хотя нет, самое главное: вдоль верхней границы рабочего поля идут вкладки для переключения между открытыми документами, имеющие также кнопку закрытия — такой же косой серый крестик, что и на инструментальной панели. Вкладки вновь открываемых документов по умолчанию возникают справа от существующих. Впрочем, вкладки эти можно перетасовать как угодно простым перетаскиванием мышью.

Боковая панель служит целям навигации по текущему документу, перемещёнию между документами открытыми и просмотру дерева файлов, как открытых, так и не открытых. И, соответственно этому, имеет три вкладки.

Первая из них — Символы. Для HTML-документов тут фигурируют разметочные теги, в частности, заголовков соответствующих уровней (H1, H2 и так далее) и заключённый в них текст с указанием номера строки. То есть мы можем видеть своего рода гипертекстовое оглавление: щелчок мышью на одном из заголовков во вкладке тегов приводит к перемещёнию на него в тексте рабочего поля:

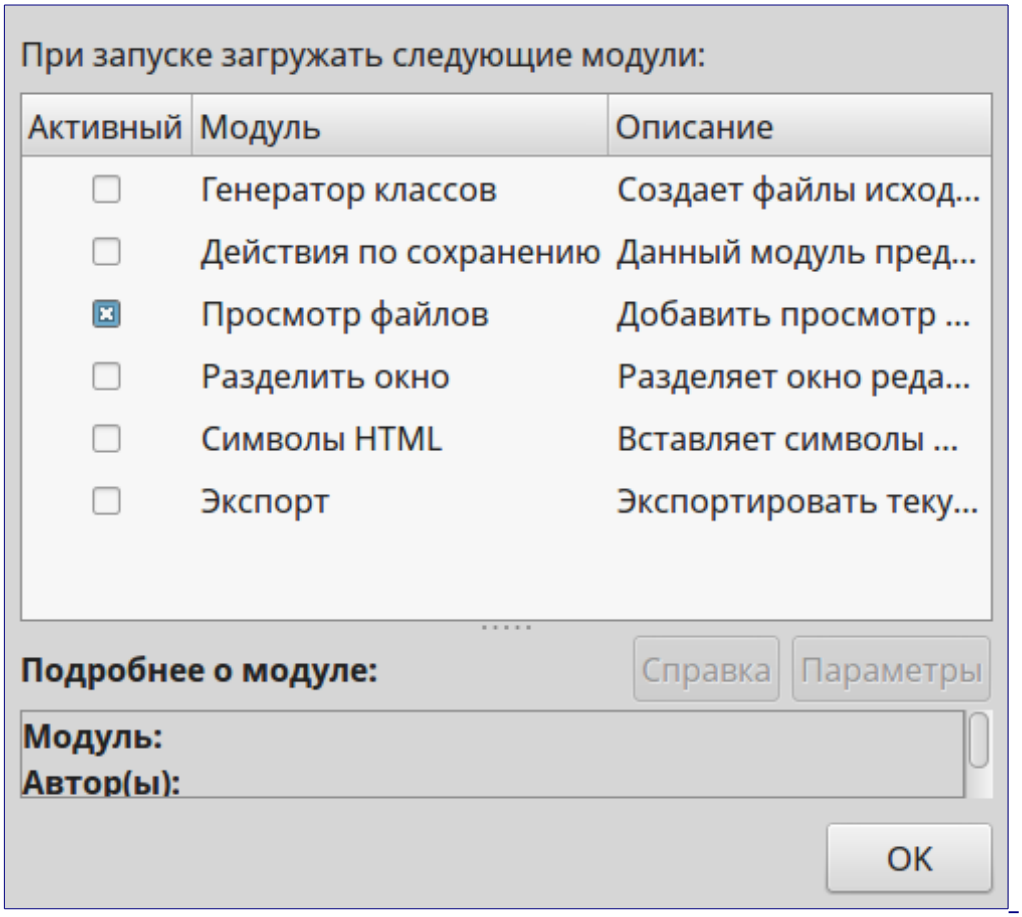


По умолчанию заголовки отсортированы по имени, то есть в алфавитном порядке. Через контекстное меню, вызываемое щелчком правой кнопки мыши в боковой панели, их можно пересортировать в порядке появления в тексте, как в обычном оглавлении:

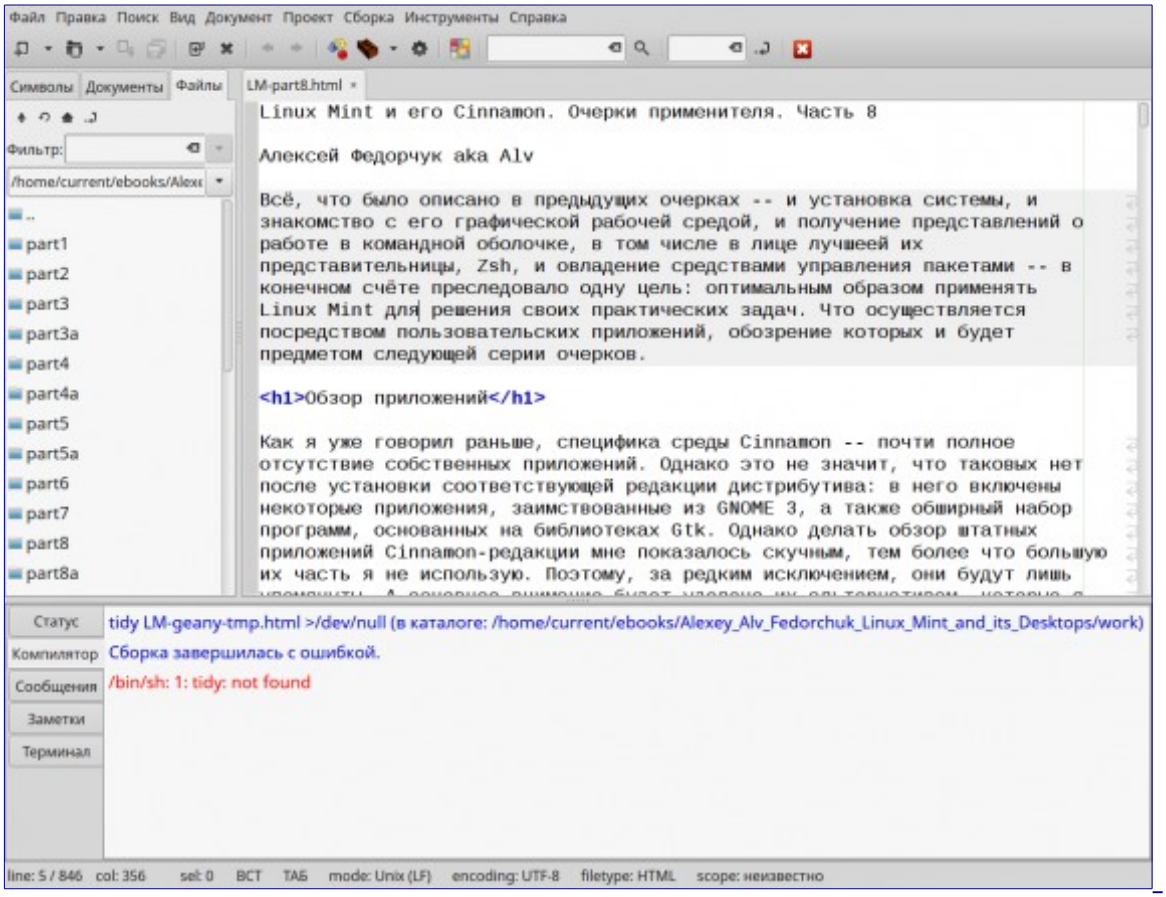


Вкладка Документы боковой панели — это просто список открытых в данный момент файлов, между которыми можно переключаться точно так же, как и по вкладкам поля редактирования. Через контекстное меню, вызываемое щелчком правой кнопки мыши, файл под курсором можно сохранить, обновить или закрыть. Действие пункта Показать полный путь будет распространено на все файлы вкладки.

Наконец, вкладка Файлы появится только после того, как через менеджер плагинов будет включён плагин Просмотр файлов:



В этой вкладке выводится содержимое текущего каталога (рис. 15), и в ней можно перемещаться, как в обычном файловом менеджере. Собственно, этот плагин и представляет собой упрощённый файловый менеджер с ограниченной функциональностью: щелчком правой кнопки мыши вызывается контекстное меню, через которое можно открыть файл в окне редактирования, открыть его во внешней программе, вызвать поиск, аналогично пункту Найти в файлах из меню Поиск. Обладает эта вкладка и собственной маленькой инструментальной панелькой с четырьмя пиктограммами, с помощью которых можно переместиться на уровень вверх, обновить содержимое вкладки, перейти в домашний каталог и в каталог, который содержит документ, являющийся текущим для поля редактирования:



Теперь окно сообщений. Оно тоже включает в себя отдельные вкладки — целых пять штук:

- Статус;
- Компилятор;
- Сообщения;
- Заметки;
- Терминал.

При вкладке Статус (она включается по умолчанию при запуске программы) в окне сообщений выводится своего рода журнал операций над текущими файлами, в котором фиксируются время открытия каждого файла, всех сохранений и закрытия. Аналогичные операции над проектами (поскольку они также представляют собой файлы, только остающиеся как бы за кадром) протоколируются тут также.

При переходе к вкладке Компилятор в окне сообщений выводится информация о ходе сборки текущего файла (запущенной через меню Построить -> Собрать. В нашем случае попытка «собрать» HTML-файл закончилась тем, чего и следовало ожидать — сообщением об ошибке.

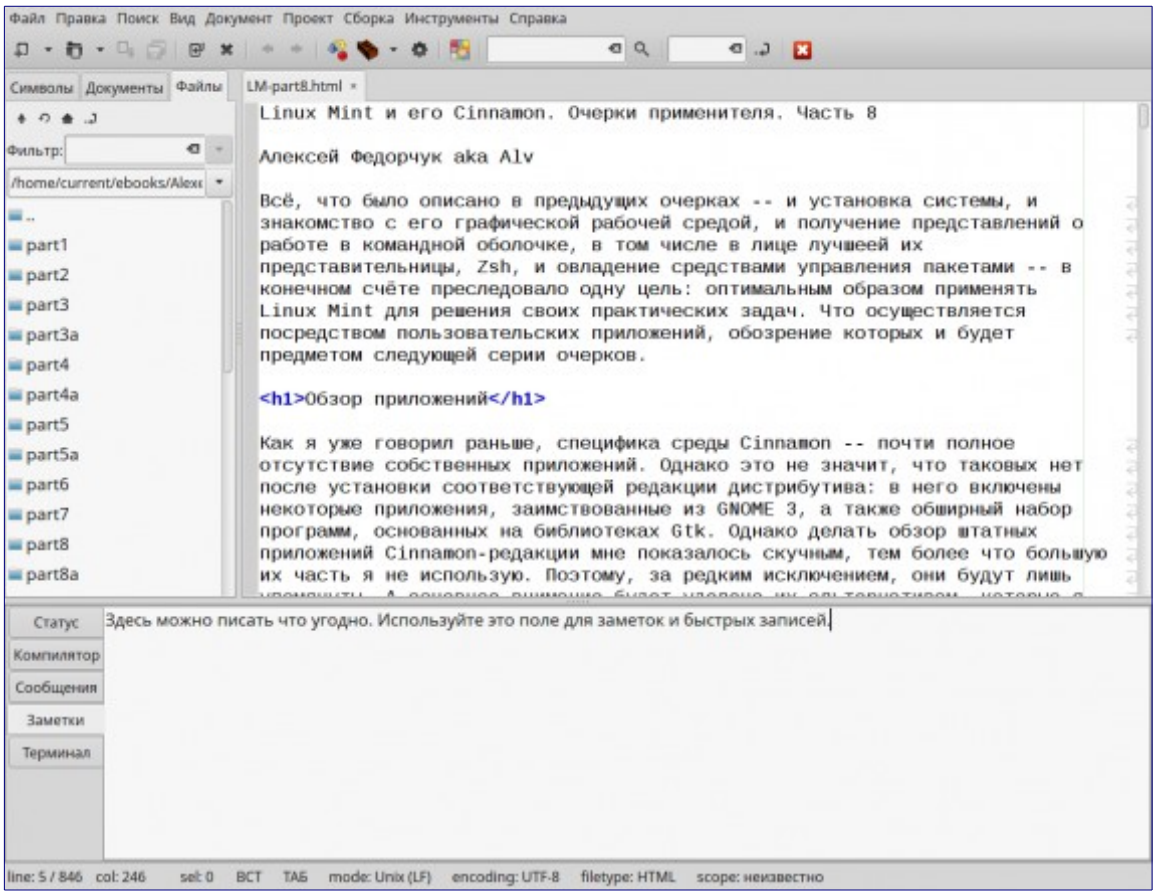
Вкладка Сообщения задействуется только при поиске сообщений — никаких других применений ей не находится.

С переходом к вкладке Заметки окно сообщений превращается в своего рода текстовый мини-редактор, о чем нас и информирует появляющаяся при

переключении надпись:

Здесь можно писать все, что угодно, используйте это для заметок и быстрых записей

Действительно, теперь в окне сообщений можно вводить текст и редактировать его как угодно. Разве что сохранить в виде файла непосредственно не получится. Но текст можно скопировать в «мышиный» или Иксовый буфер и поместить уже в окно редактирования (или в любую другую программу, способную обрабатывать тексты).



Пятая вкладка — Терминал. С переключением на неё окно сообщений становится действительно полноценным терминальным окном с командной строкой пользовательского шелла, в которой можно вводить практически любые команды, в чем и состоит её ценность для линуксописателя: результаты выполнения команды можно тут же «скопипастить» в сочиняемую статью.

Последний элемент интерфейса нашего редактора, также отключаемый — строка состояния вдоль нижнего края окна сообщений. В ней выводятся: номер строки и колонки для текущего положения курсора, режим работы редактора (вставки или замены), тип конца строки, кодировка документа и тип его файла.

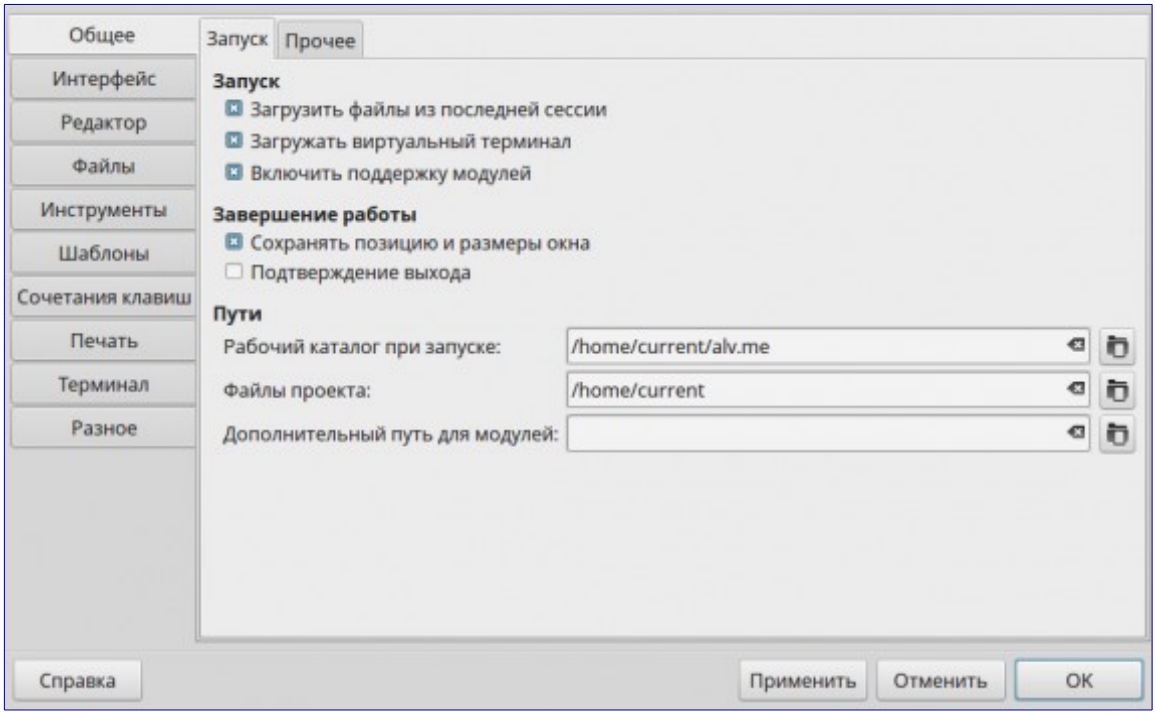
Настройка

Мы рассмотрели интерфейс и возможности Geany по умолчанию. Теперь давайте поглядим, как их можно модифицировать под свои потребности и привычки.

Как уже говорилось, практически все настройки Geany выполняются

посредством меню Правка -> Параметры, вызывающего панель с одиннадцатью вкладками:

1. Общее;
2. Интерфейс;
3. Панель инструментов;
4. Отображение;
5. Редактор;
6. Файлы;
7. Инструменты;
8. Шаблоны;
9. Привязки;
10. Печать;
11. Терминал.



Рассмотрим последовательно, какие возможности они предоставляют.

Во вкладке Общее — две секции, Запуск и Прочее, содержащих чекбоксы включения/отключения соответствующих функций. В первой из них отмечается, загружать ли при старте редактора файлы из последней сессии, включать ли виртуальный терминал (тот самый, на который можно будет переключиться в окне сообщений) и поддержку дополнительных плагинов, о которых говорилось выше.

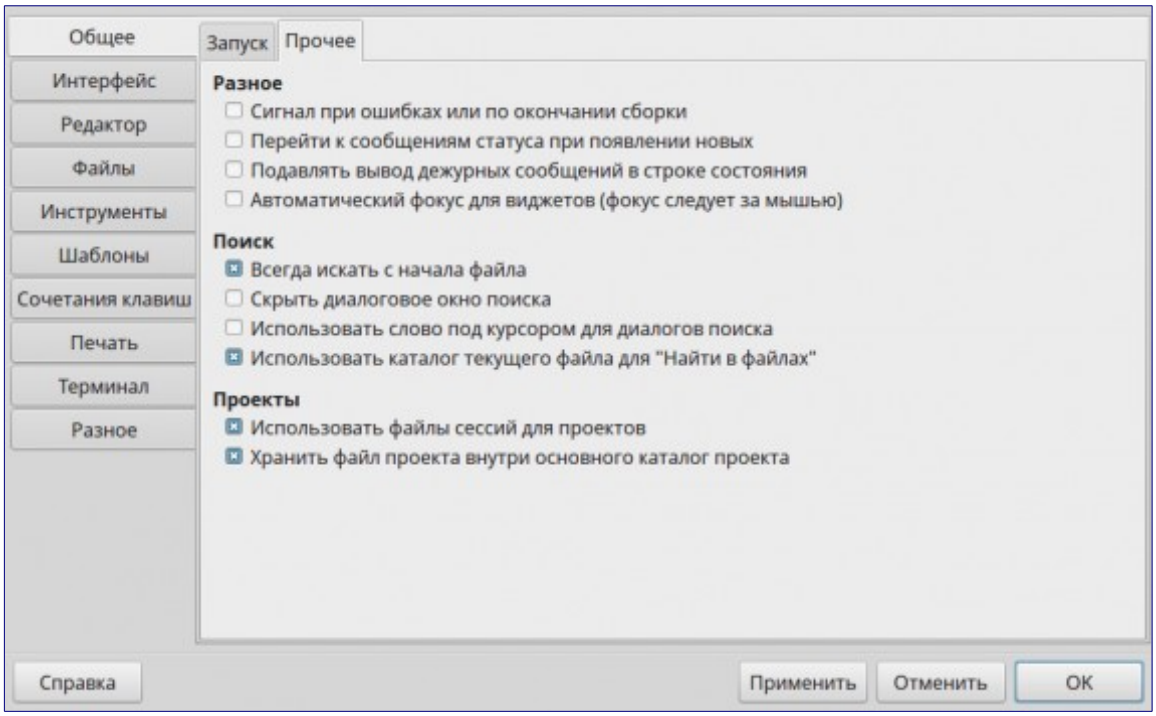
При завершении работы можно сохранить позицию и размеры главного окна программы и его составляющих — боковой панели и окна сообщений; здесь

же указывается, запрашивать ли подтверждение при выходе из редактора.

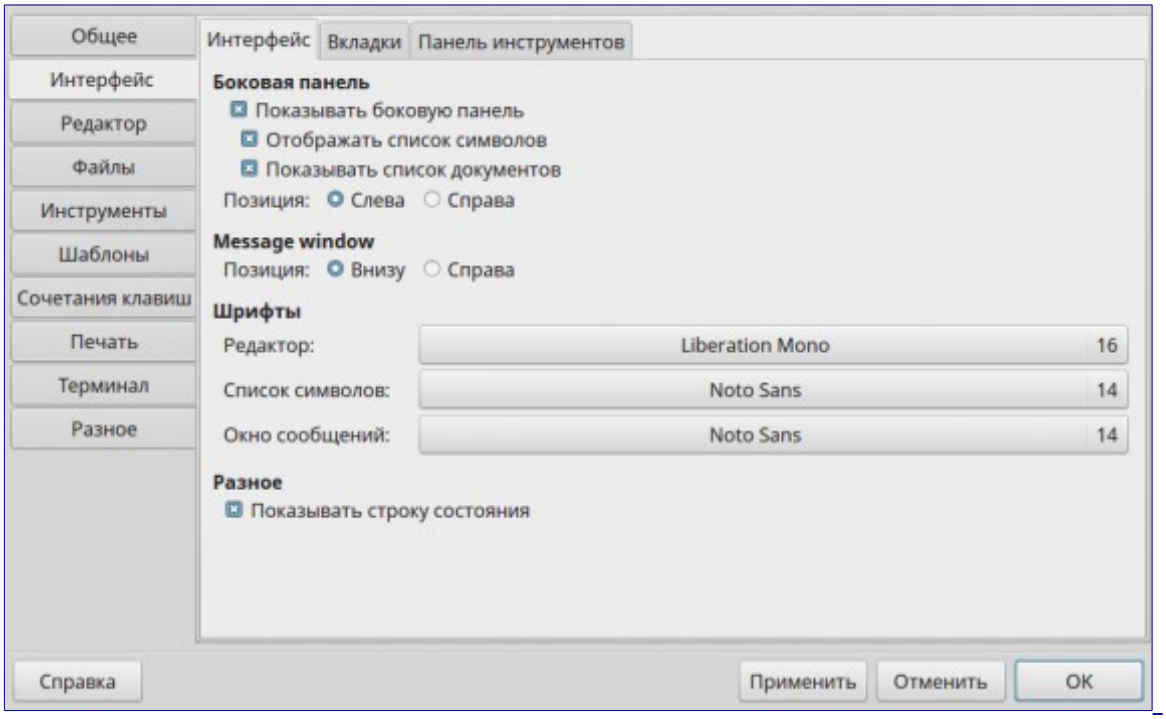
Далее можно указать пути к рабочему каталогу при запуске и к файлам проекта. Они не обязаны совпадать — в некоторых случаях удобно файлы всех проектов держать в отдельном от рабочих файлов месте.

В секции Прочее, как и положено, настраивается всякая всячина, как то:

- включение звукового сигнала при ошибках;
- переход к дежурным сообщениям при получении оных и, напротив, подавление вывода дежурных сообщений;
- включенние автофокусировки окон по перемещёнию курсора мыши, без щелчка оной, — это удобно, если надо постоянно приходится переключаться между окном редактирования и окном сообщений;
- скрытие панели поиска по его завершении;
- помещёние слова под курсором в поисковую строку при обращении к функциям поиска и замены.



Внешний вид редактора и его основных элементов определяется во вкладке Интерфейс:

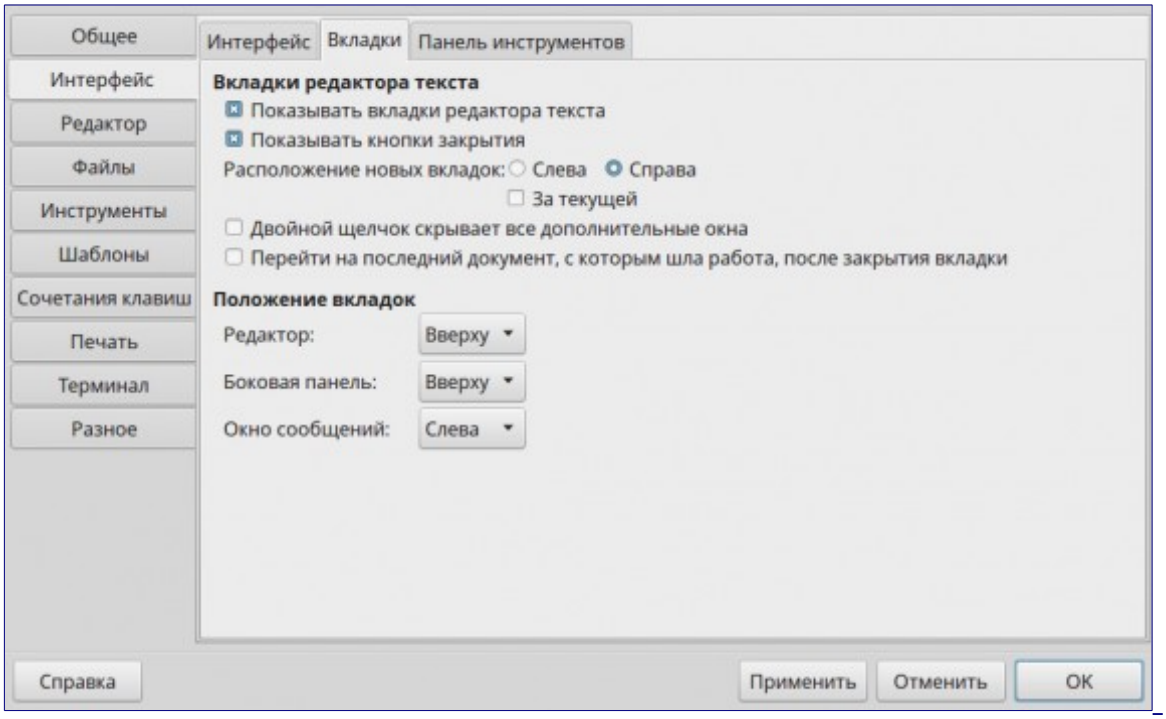


Здесь для боковой панели можно включить или выключить отображение списка символов и списка документов; отображения списка файлов здесь нет — как уже говорилось, оно определяется включением соответствующего плагина. Так что, если отключить вывод и списка символов, и списка документов, исчезнет и список файлов. Ну а с включением показа полных путей к файлам открытых документов всё ясно без комментариев.

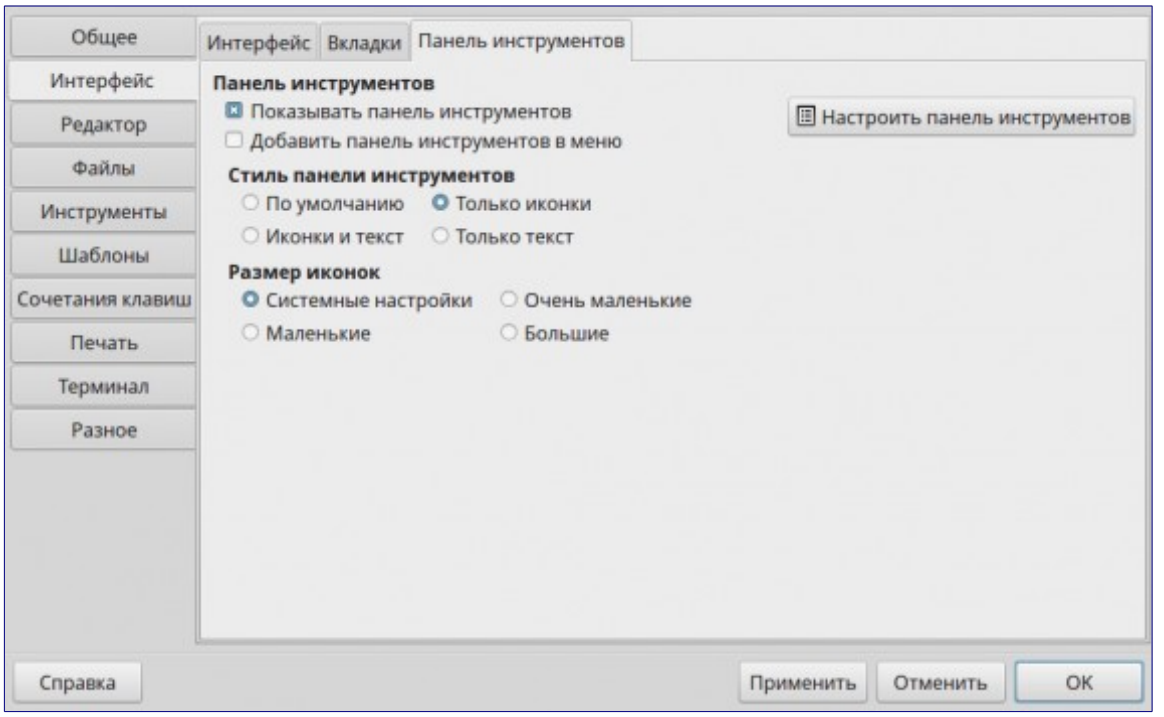
Шрифты — как их гарнитура, так и размер, — можно установить независимо для окна редактирования, для боковой панели и для окна сообщений. Забегая вперед, заметим, что терминал в окне сообщений также настраивается независимо от остальных элементов редактора.

Далее, экономии места ради, можно выключить вкладки для открытых файлов в окне редактирования. Если же их оставить, то можно отключить показ кнопки закрытия на вкладках, во избежание случайного нажатия на неё. Ну и позиция открытия новых вкладок при создании документа — слева или справа от текущей — также может быть переопределена.

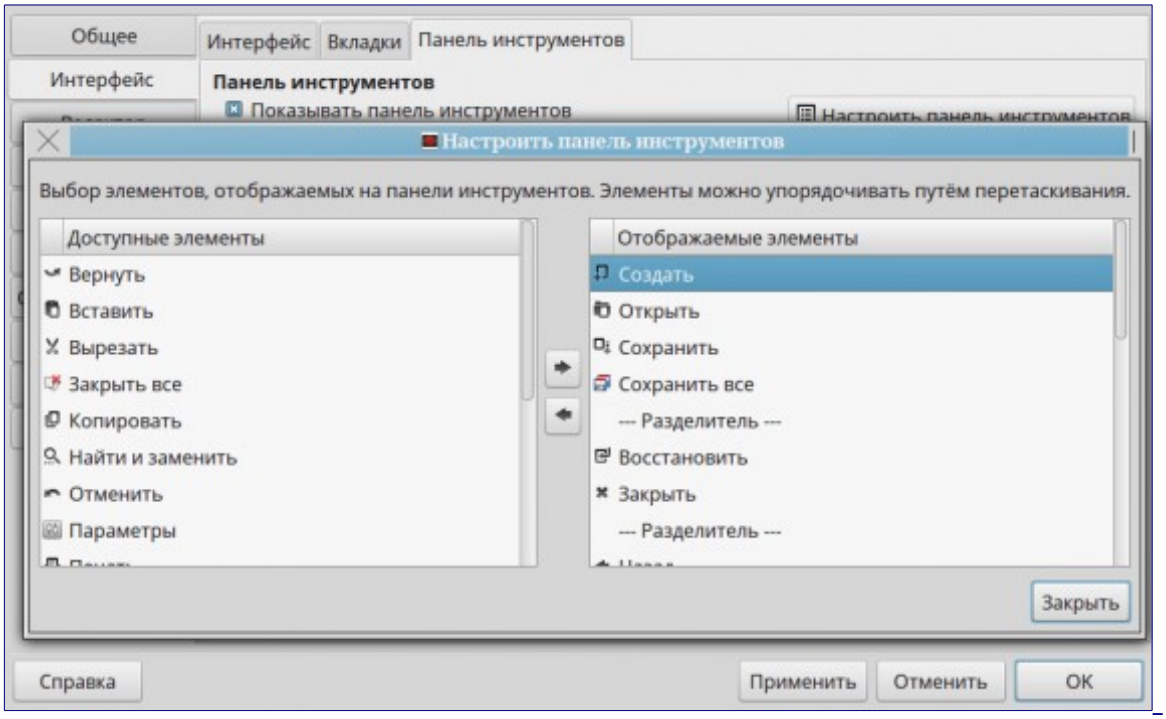
Положение вкладок задаётся для главных виджетов программы относительно их самих независимо, и они могут располагаться по любому краю панели или окна, хотя их позиция по умолчанию наиболее разумна — разве что можно было бы поспорить относительно «верха» и «низа» для окна редактирования и «права» и «лева» — для окна сообщений.



Как говорилось в разделе Панель инструментов, она может быть отключена, или набор кнопок на ней изменён. Это делается в одноименной вкладке отметками в соответствующих чекбоксах. Можно также изменить внешний вид кнопок (в виде только иконок, только текста или того и другого) и их размер (большой, как по умолчанию, или маленький).



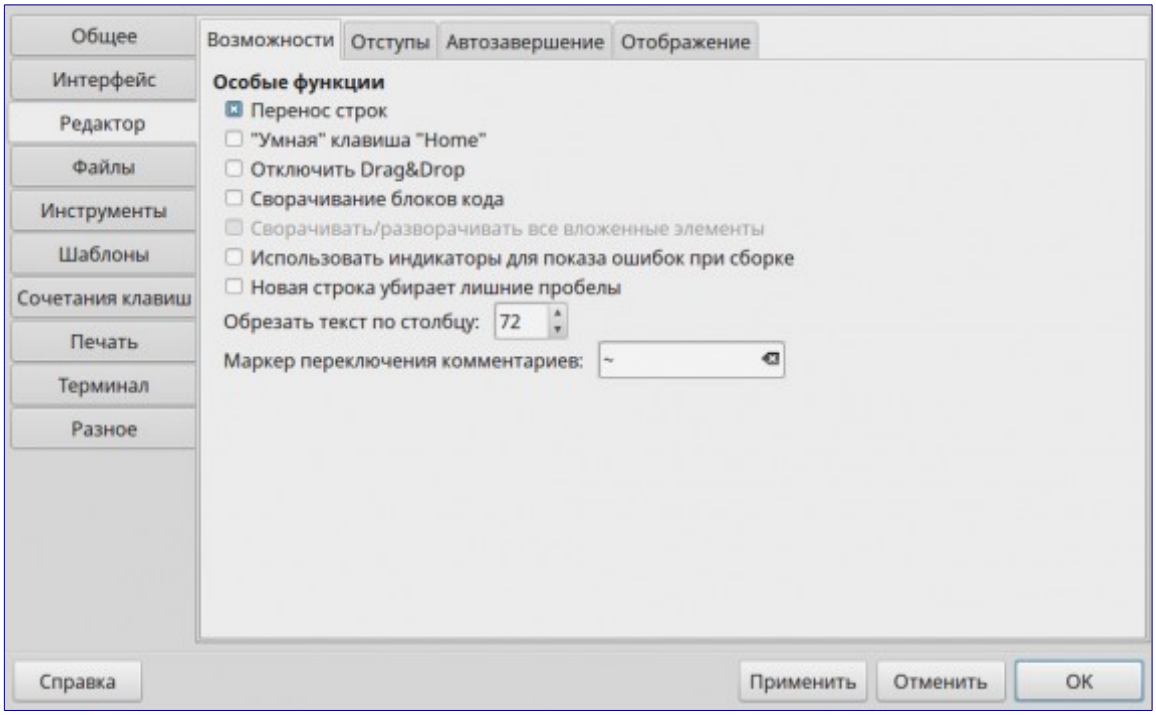
Набор пиктограмм на инструментальной панели настраивается в отдельном окошке, вызываемом нажатием соответствующей кнопки. Тут, я думаю, всё поонятно из скриншота:

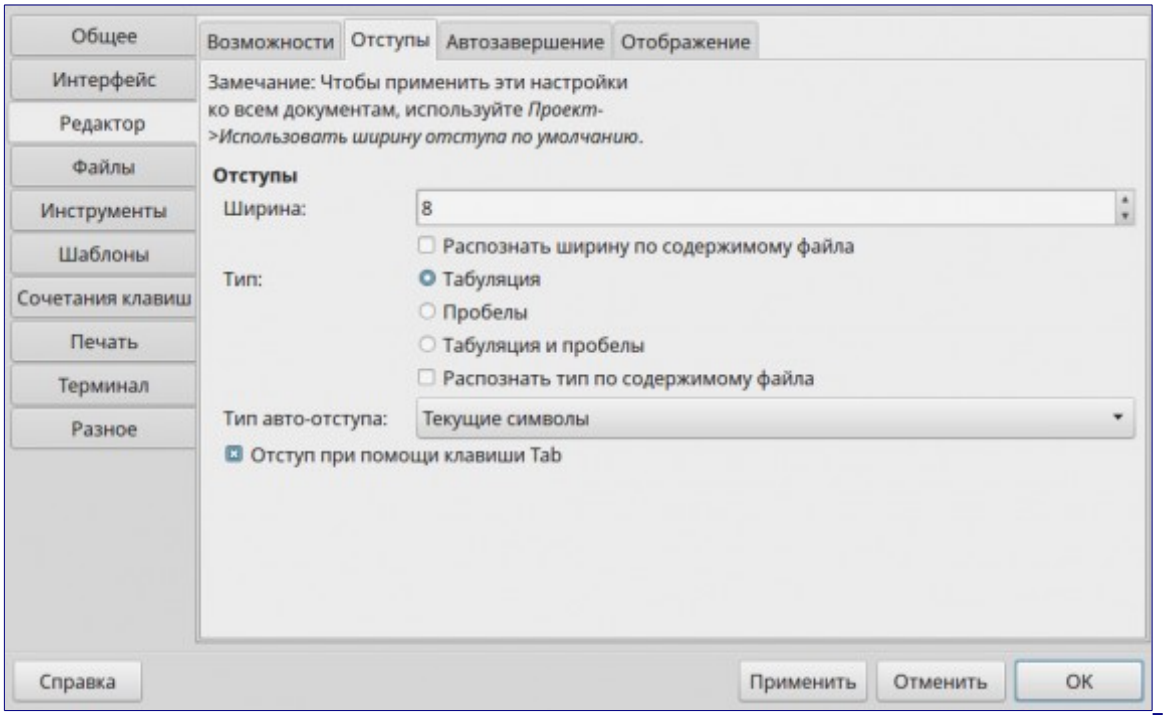


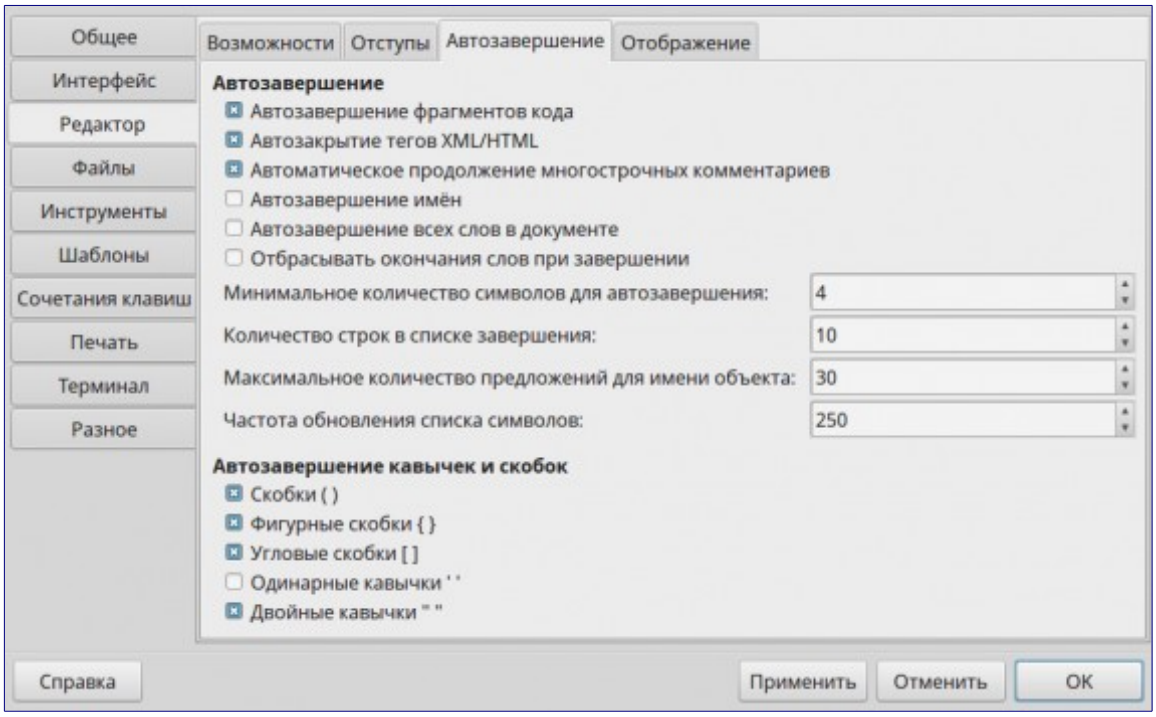
Во вкладке Редактор устанавливаются правила поведения в окне редактирования, такие как:

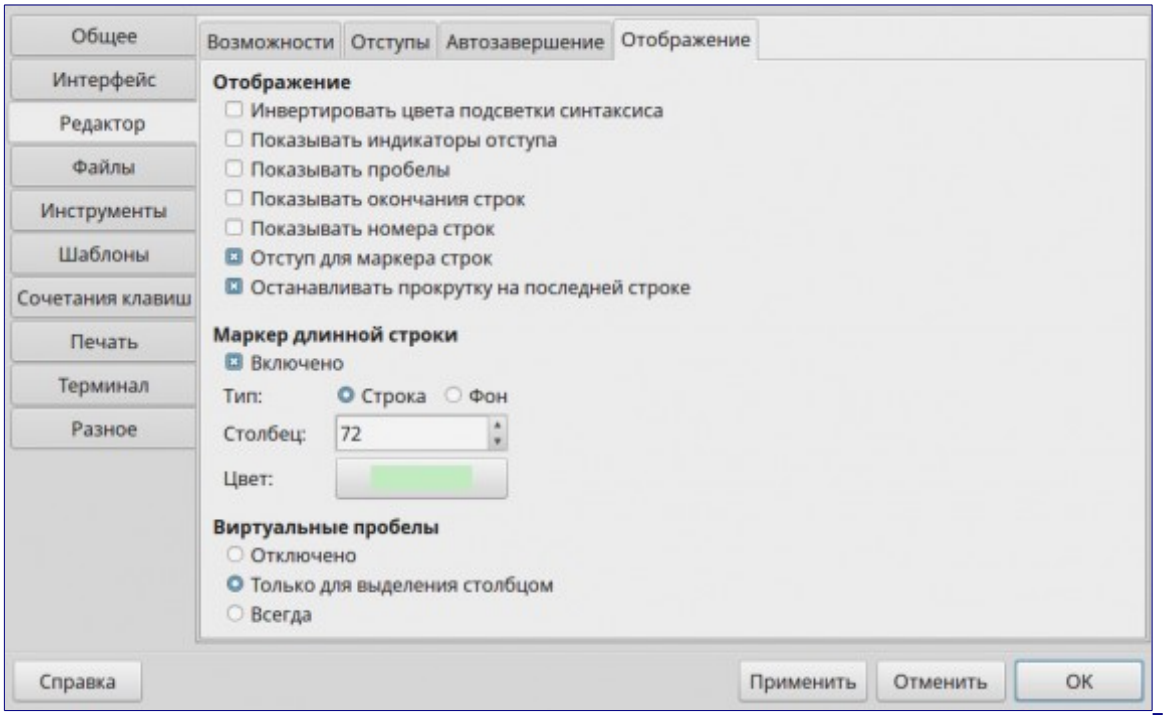
- включение и выключение режима переноса слов;
- отключение режима Drag-and-Drop;
- удаление остаточных пробелов в конце строк, перед символом её окончания;

и так далее — детали можно видеть на серии скриншотов.




Общее
Интерфейс
Редактор
Файлы
Инструменты
Шаблоны
Сочетания клавиш
Печать
Терминал
Разное
Возможности
Отступы
Автозавершение
Отображение
Замечание: Чтобы применить эти настройки ко всем документам, используйте Проект->Использовать ширину отступа по умолчанию.
Отступы
Ширина:
8
Распознать ширину по содержимому файла
Тип:
Табуляция
Пробелы
Табуляция и пробелы
Распознать тип по содержимому файла
Тип авто-отступа:
Текущие символы
Отступ при помощи клавиши Tab
Справка
Применить
Отменить
OK

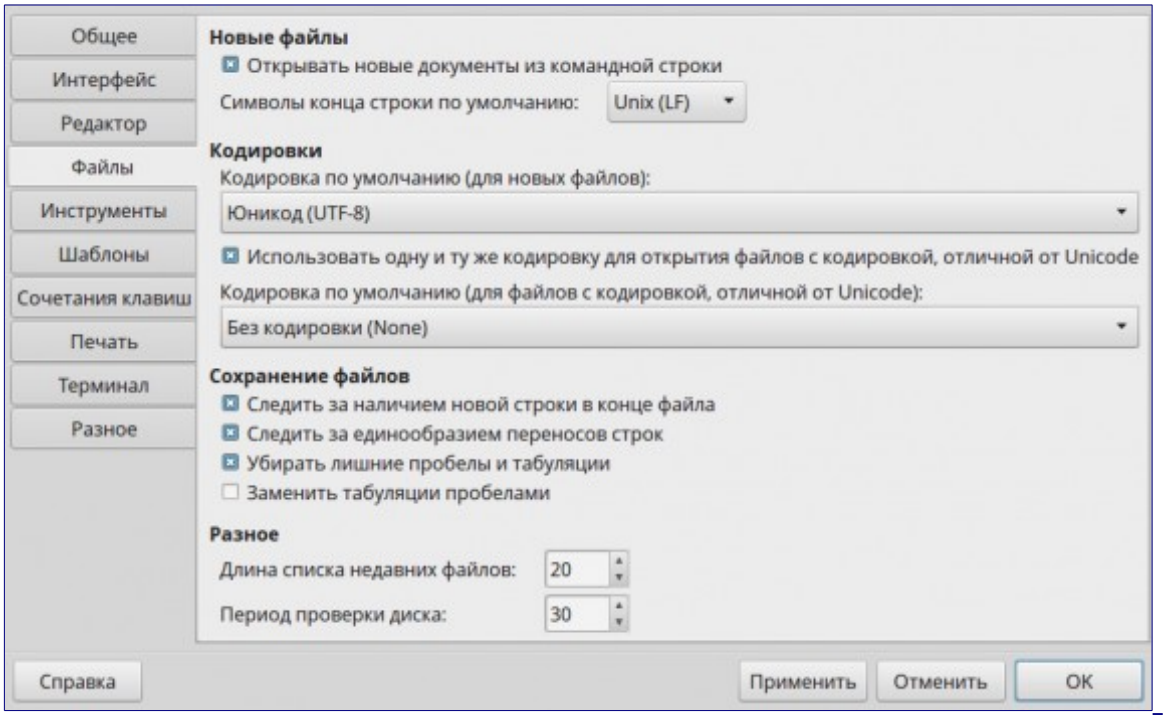

Общее
Интерфейс
Редактор
Файлы
Инструменты
Шаблоны
Сочетания клавиш
Печать
Терминал
Разное
Возможности
Отступы
Автозавершение
Отображение
Автозавершение
Автозавершение фрагментов кода
Автозакрытие тегов XML/HTML
Автоматическое продолжение многострочных комментариев
Автозавершение имён
Автозавершение всех слов в документе
Отбрасывать окончания слов при завершении
Минимальное количество символов для автозавершения:
4
Количество строк в списке завершения:
10
Максимальное количество предложений для имени объекта:
30
Частота обновления списка символов:
250
Автозавершение кавычек и скобок
Скобки ( )
Фигурные скобки { }
Угловые скобки [ ]
Одинарные кавычки ' '
Двойные кавычки " "
Справка
Применить
Отменить
OK



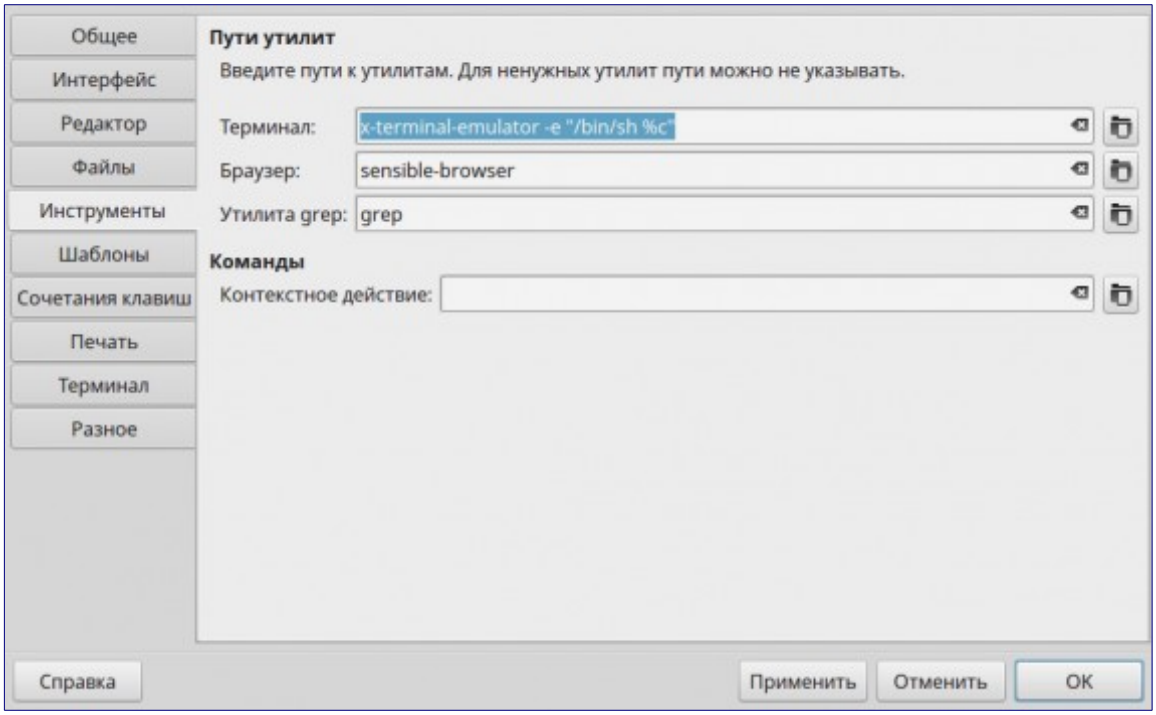
Комментарий требуентся тольк для последнего скриншота и его секции Маркер длинной строки. При включении режима динамического переноса строк служит для различения «истинных» строк (фиксируемых символами конца строки, в случае стиля Unix — LF) и строк «экранных», создаваемых за счет переноса слов по границе экрана, длина которых зависит от размера окна редактирования. Варианты выбора маркера — отмечать цветом текст строки, фон текста (цвет может быть изменен) или выключить вообще (последнее имеет смысл, если режим переноса слов не используется).

Во вкладке Файлы сначала определяется кодировка по умолчанию для вновь создаваемых файлов и устанавливается кодировка, в которой должны открываться файлы уже существующие. По умолчанию значения обоих параметров берутся из системной локали, но в общем случае совпадать они не обязаны.

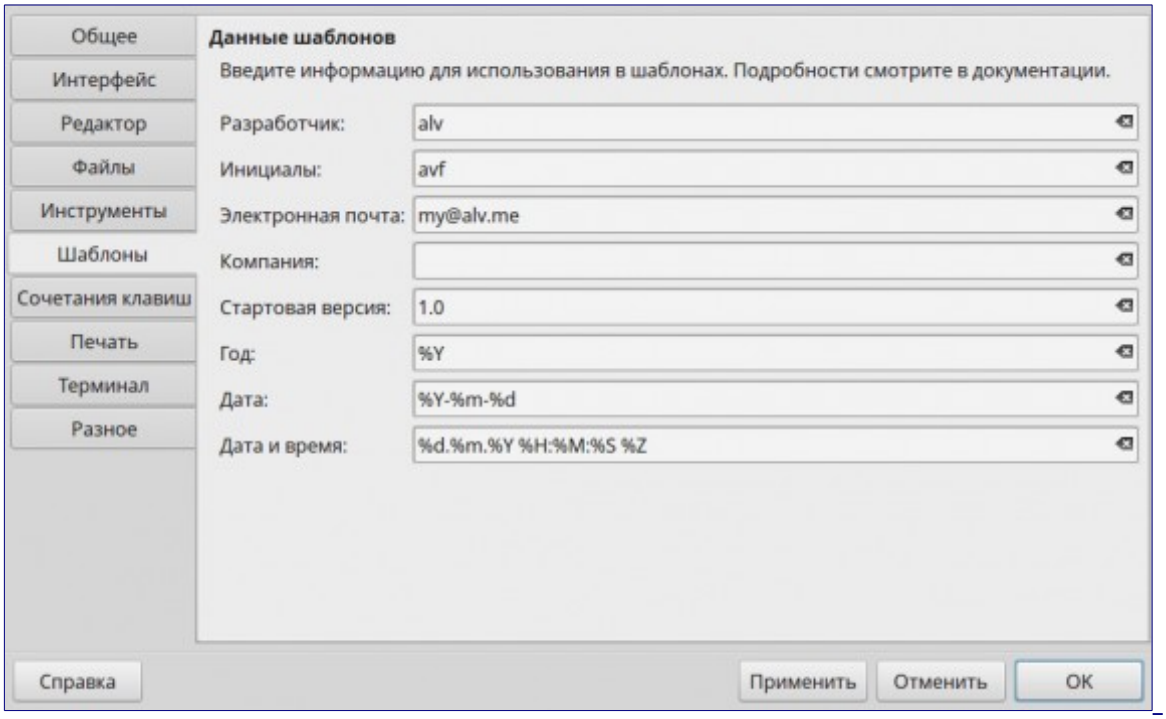
Далее включаются (или, напротив, выключаются) действия, производимые при записи файлов: удаление остаточных пробелов, обязательный ввод новой строки в конце файла (необходимо для некоторых конфигов), замена символов табуляции эквивалентным числом пробелов. Длина списка недавно открывавшихся файлов (выводимого при действиях через меню Файл -> Недавние файлы) также указывается в этой вкладке.



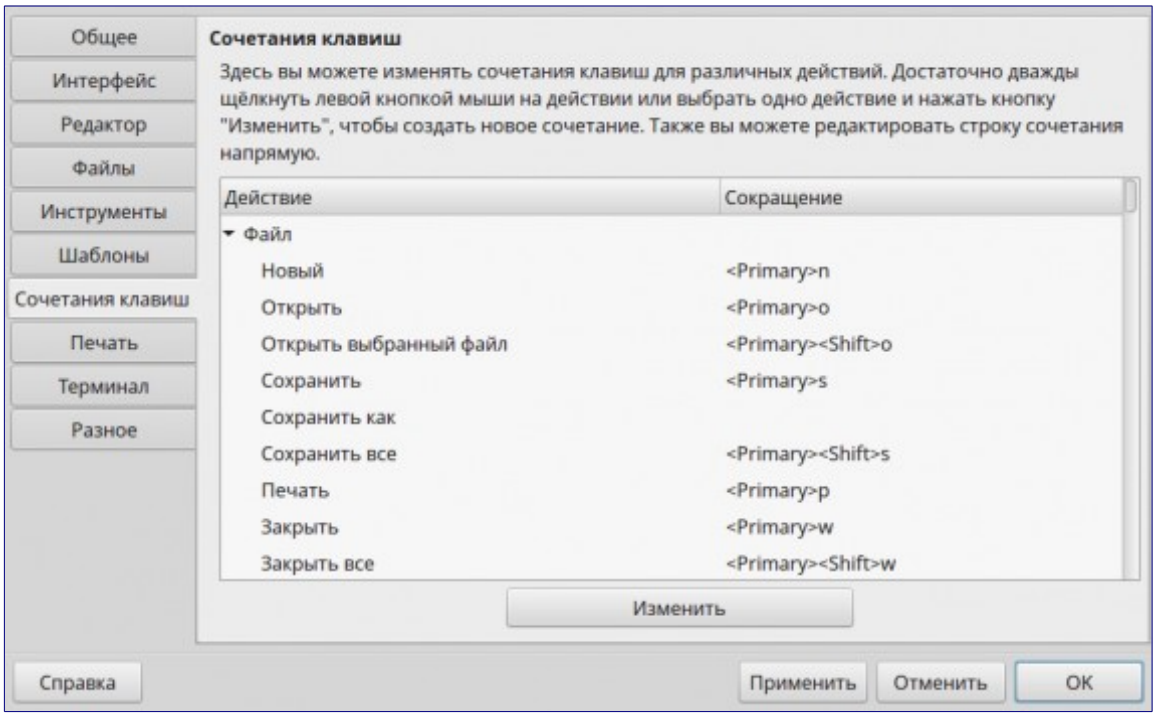
Вкладка Инструменты к панели инструментов не имеет никакого отношения: здесь определяются внешние программы, вызываемые для выполнения определённых действий. Пользователю нужно следить за тем, чтобы умолчальные значения всех полей, подходящие в большинстве случаев, всё же соответствовали реалиям его системы. То есть чтобы для действие Терминал осуществлялось в окне предпочитаемого эмулятора терминала, для поиска текстовых фрагментов применялась нужная утилита grep-семейства.



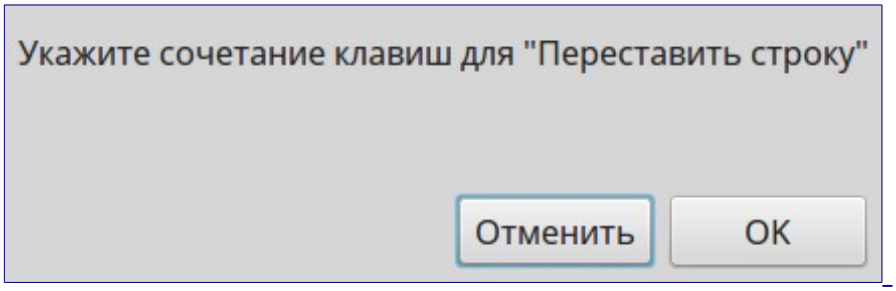
Во вкладке Шаблоны вводятся те самые личные сведения, которые потом окажутся в комментариях ко всем файлам, создаваемым посредством действий Файл -> Создать из шабла -> [тип файла]: имя и фамилия автора, адрес его электронной почты и тому подобное. В отличие от всех остальных изменений, вступающих в действие немедленно по нажатии кнопки Применить или OK в правом нижнем углу панели настроек, переопределение сведений о шаблоне обретёт силу только при следующем запуске Geany.



Во вкладке Сочетания клавиш, как легко догадаться, можно переопределить «горячие» клавиши для всех действий, предусмотренных в редакторе Geany, а также приписать их тем действиям, к которым никакие клавишные комбинации по умолчанию не определены. То есть сделать то, к чему нас призывали разработчики в меню Помощь -> Горячие клавиши.



Для переопределения существующих клавишных комбинаций или создания новых достаточно выделить в списке нуждающееся в этом действие и нажать кнопку Изменить (или просто щелкнуть на нём дважды). После этого, по появлении панельки Захватить клавишу, надо набрать желаемую комбинацию клавиш, которые тут же высветятся на панельке, и затем нажать кнопку OK.



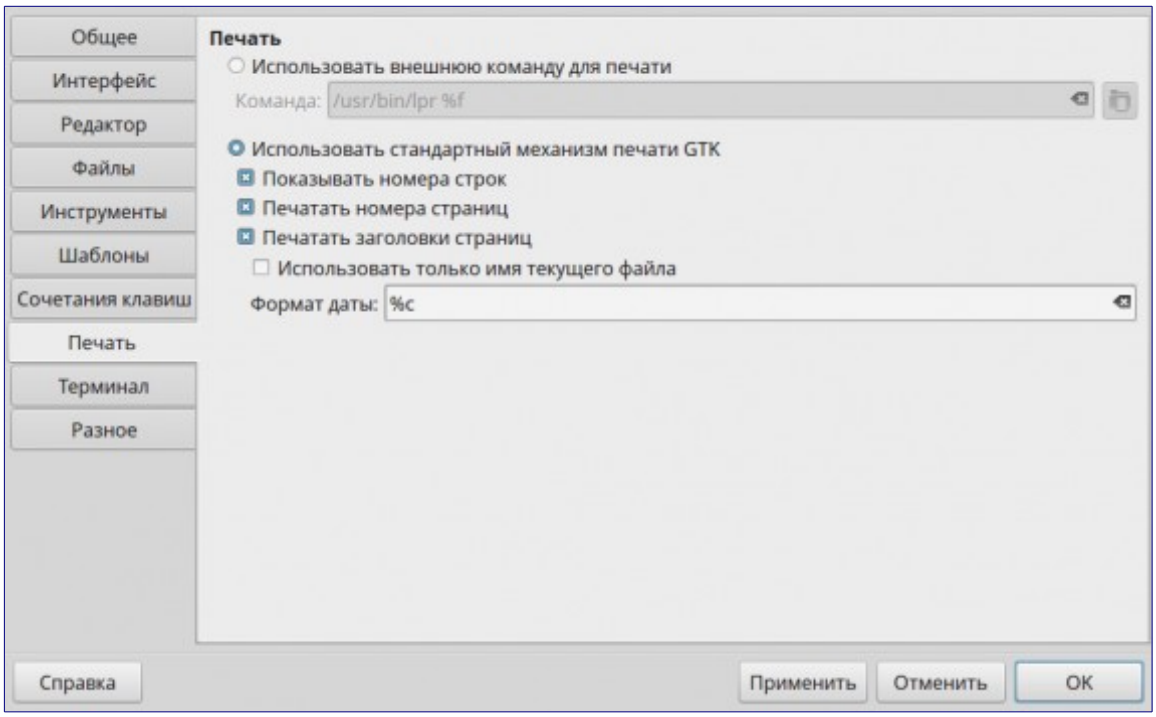
Клавиатурные комбинации можно редактировать и напрямую: для этого надо лишь, выделив строку подлежащего изменению действия, щелкнуть мышью непосредственно на обозначении горячих клавиш для него, после чего ввести желаемые значения вручную.

Я не буду останавливаться на вопросе, какие сочетания горячих клавиш следует использовать для тех или иных действий: это вопрос сугубо личный, можно даже сказать — интимный.

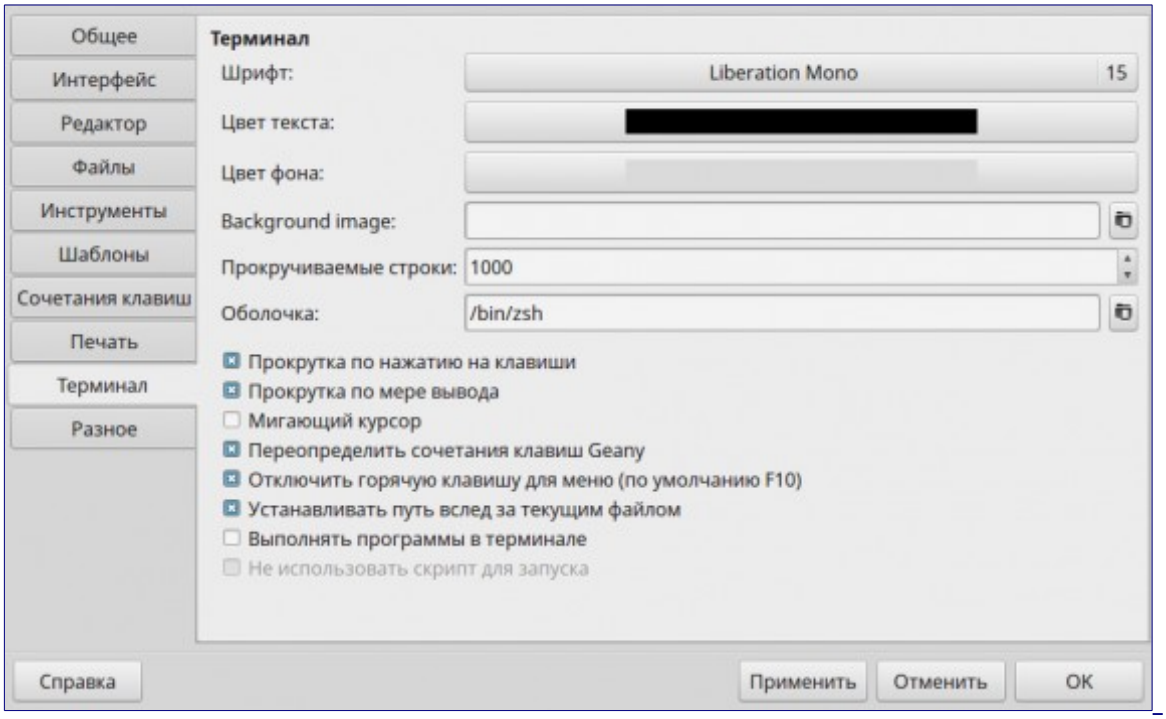
Мало что скажу также и о вкладке Печать, ибо следуем заповеди POSIX'ивистов, сформулированной Сергеем Голубевым:

Не настроил принтер — сохранил дерево.

Так что тем, кто деревьев не жалеет, в содержимом этой вкладки предоставляется разбираться самостоятельно.



А вот на содержании вкладки Терминал стоит остановиться подробнее.



Перво-наперво здесь можно определить шрифт для терминального окна, его цвет и цвет фона — это делается через панель выбора цвета, о которой мы говорили при рассмотрении главного меню. Весьма элегантно выглядит оформление в общих тонах всего редактора, что легко сделать с помощью упомянутой ранее «пипетки». Впрочем, можно задать и фоновое изображение. Ну а шрифт и его размер каждый определяет в соответствие со своими вкусами и диоптриями.

Далее определяется число строк терминальной «истории» и запускаемая в терминальном окне командная оболочка (по умолчанию это будет login shell данного пользователя). Под терминальную «историю» на нынешних машинах можно отвести сколь угодно большое число строк.

Опции Прокрутка по нажатию на клавиши и Прокрутка по мере вывода в комментариях не нуждаются. Переопределение горячих клавиш Geany может быть полезным, если они пересекаются с кейбиндингами используемой командной оболочки. Ну и отключение вызова меню через горячую клавишу F10 может пригодиться, если в терминальном окне предполагается запускать программу типа Midnight Commander.

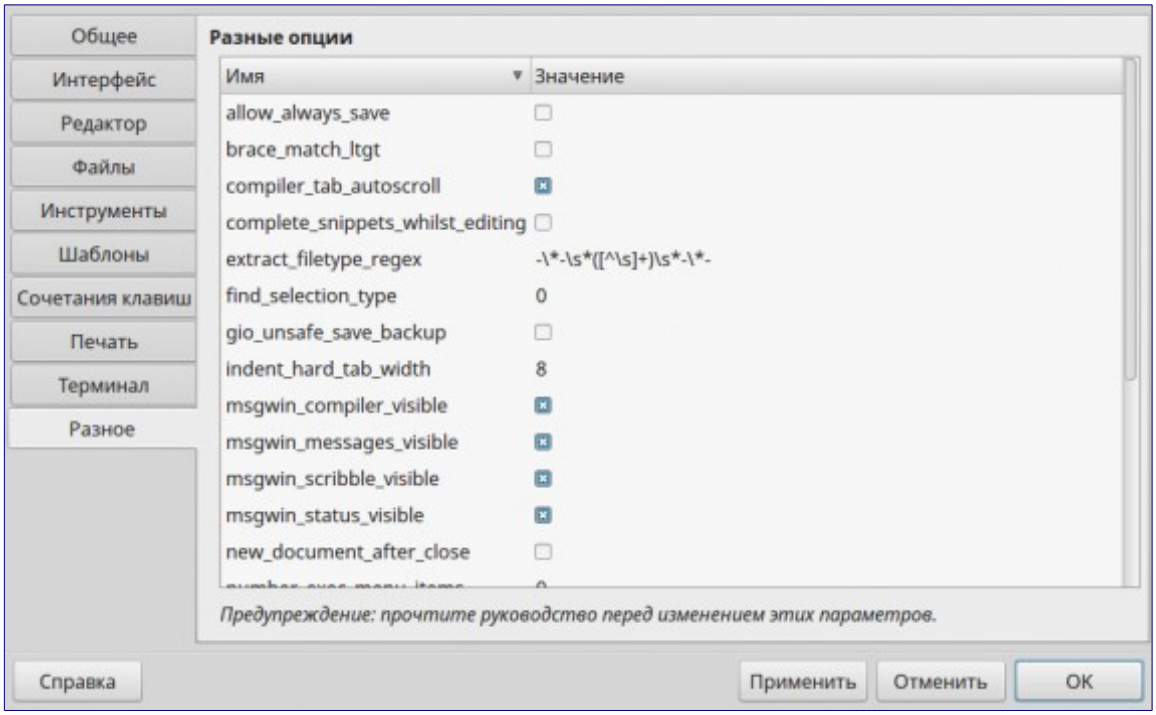
Включение опции Следовать пути текущего файла приведет к тому, что при каждом переключении между документами из разных каталогов, открытыми в поле редактирования, смена текущего каталога будет происходить и в командной строке. Полезно это, вредно ли, или безразлично, — зависит от конкретной ситуации.

Опция Выполнять программы в терминале предписывает направление исполнения отлаживаемых программ и скриптов в наш встроенный виртуальный терминал, вместо того, чтобы вызывать отдельное терминальное окно (той самой программы, которая ранее была указана в поле Терминал вкладки Инструменты). Такая возможность удобна, но именно при отладке обнаруживается её недостаток: остановить неправильно работающую программу во встроенном терминале не всегда возможно.

На этом настройку терминала можно считать законченной. Заметим только,

что точно так же её можно осуществить непосредственно из вкладки Терминал окна сообщений — через контекстное меню по щелчку правой кнопкой мыши.

Собственно, и настройки Geany вообще тоже закончены. Как и вообще разговор о его базовой функциональности — на содержании вкладки Разноеостанавливаться не буду, так как никаких параметров там не менял, вняв соответствующему предупреждению:



Настало время подвести предварительные итоги. Главный из которых таков: есть мнение, что использование этой программы целесообразно для любых текстовых работ, превышающих сложностью редактирование двух-трёх строчек в пятистрочном сценарии. Автор настоящего очерка, в частности, на протяжении ряда лет применял Geany для создания документов в форматах plain text и HTML. Удобство его для сочинительских целей определяется возможностями выполнения команд в терминальном окне параллельно с их описанием в поле редактирования, мгновенной проверки работоспособности командных конструкций, автоматизацией ввода наиболее часто используемых HTML-тегов, сочетанием средств навигации внутри находящегося в работе текста с функциями обзора файловой системы и многим, многим другим. А дополнительный функционал редактору придают многочисленные плагины, к рассмотрению которых мы и переходим.

Geany и его плагины

Надеюсь, на прошлых страницах мне удалось продемонстрировать, что Geany и своём первозданном виде предоставляет в распоряжение применителя массу возможностей для работы с текстом. Однако функционал его многократно усугубляется бесчисленными дополнения — так называемыми плагинами (plugins). Чтобы получить представление о их количестве, достаточно в CLI задать поиск по шаблону geany-plugin:

```
$ apt search geany-plugin | wc -l
```

вывод которой в Mint 17.1 Rebecca на момент сочинения этих строк будет таким:

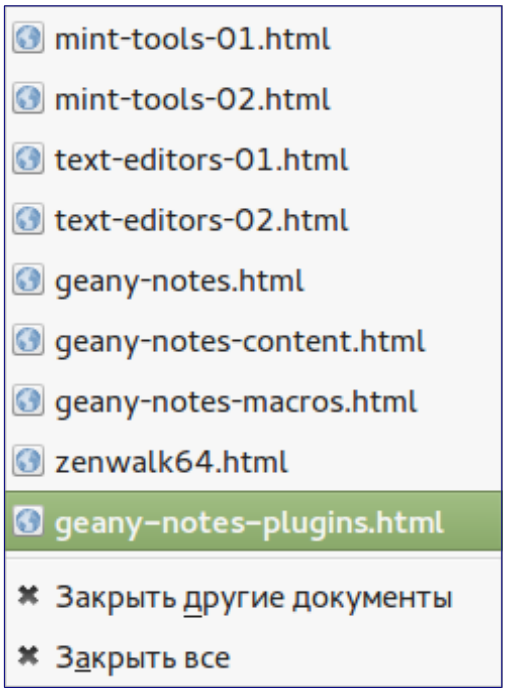
66

То есть на 20.01.2015 для Geany сочинено 66 примерно плагинов. Почему примерно — сейчас расскажу.

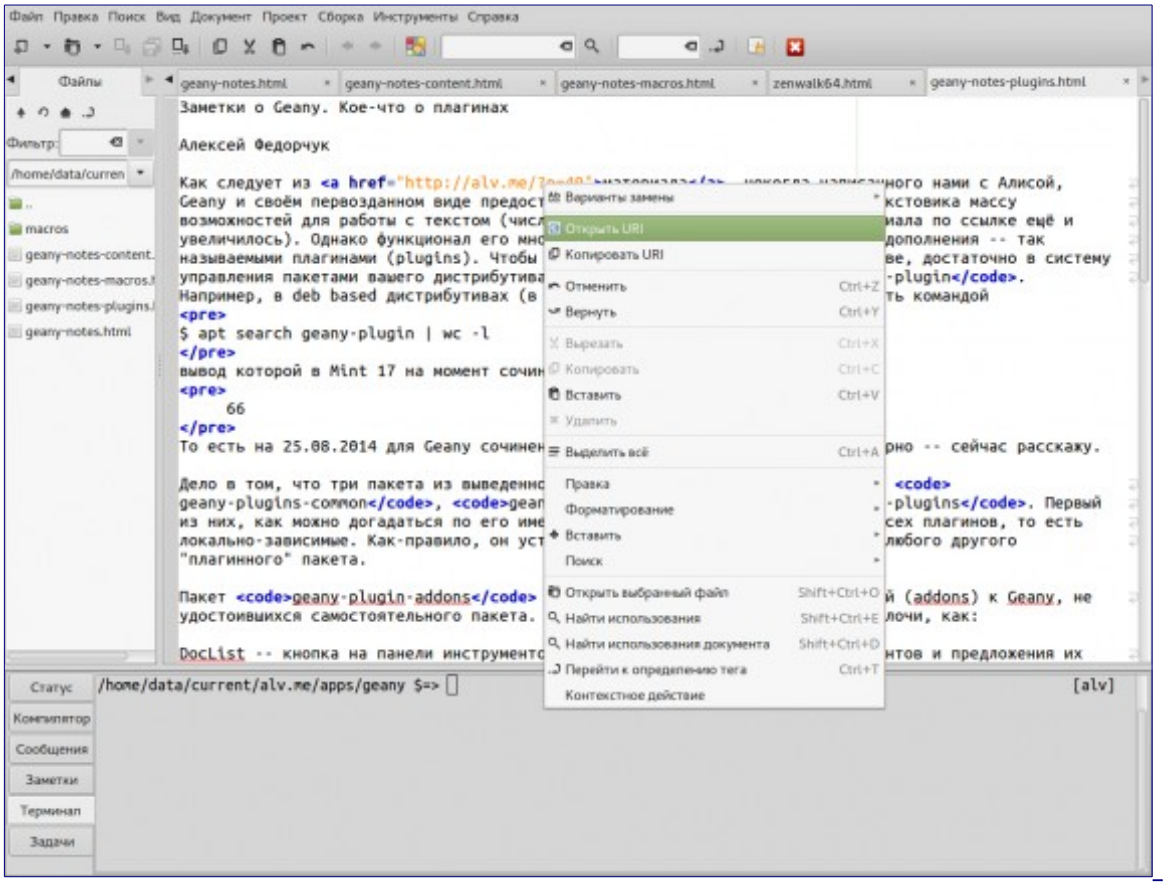
Дело в том, что три пакета из выведенного списка играют особую роль. Это — geany-plugins-common, geany-plugin-addons и geany-plugins. Первый из них, как можно догадаться по его имени, содержит компоненты, общие для всех плагинов, то есть локально-зависимые. Как-правило, он устанавливается в качестве зависимости любого другого «плагинного» пакета.

Пакет geany-plugin-addons включает в себя ряд мелких дополнений (addons) к Geany, не удостоившихся самостоятельного пакета. В его составе такие небесполезные мелочи, как:

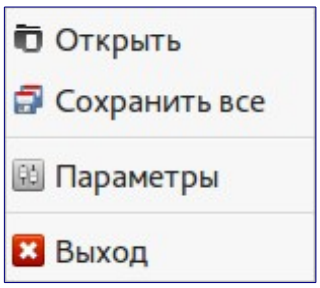
- DocList — кнопка на панели инструментов, вызывающая список открытых документов и предложения их закрытия:



- OpenURI — если элемент под курсором являет собой URI (например, http-адрес), то в контекстное меню по щелчку правой кнопкой мыши добавляются пункты Открыть URI и Редактировать URI очевидного назначения (адрес открывается в браузере, прописанном в настройках Geany):



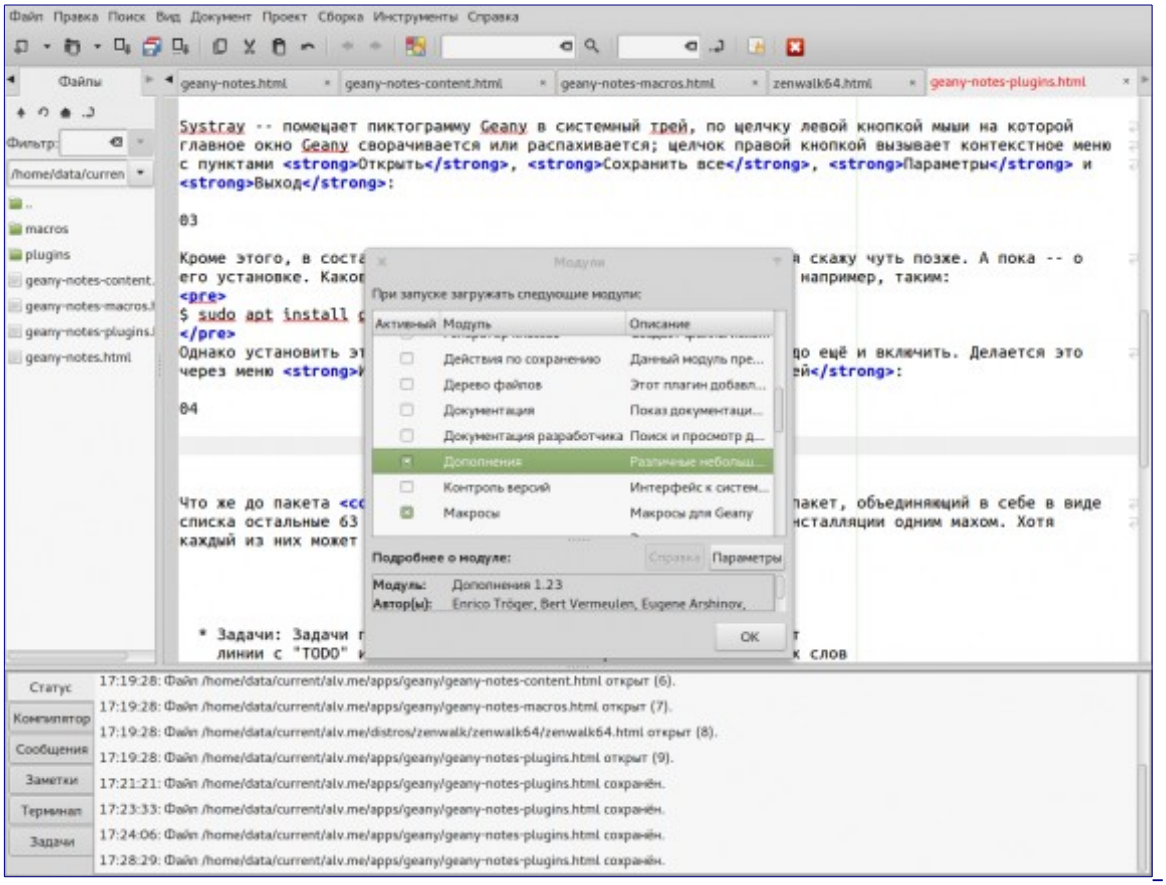
- Systray — помещает пиктограмму Geany в системный трей, по щелчку левой кнопкой мыши на которой главное окно Geany сворачивается или распахивается; щелчок правой кнопкой вызывает контекстное меню с пунктами Открыть, Сохранить все, Параметры и Выход:



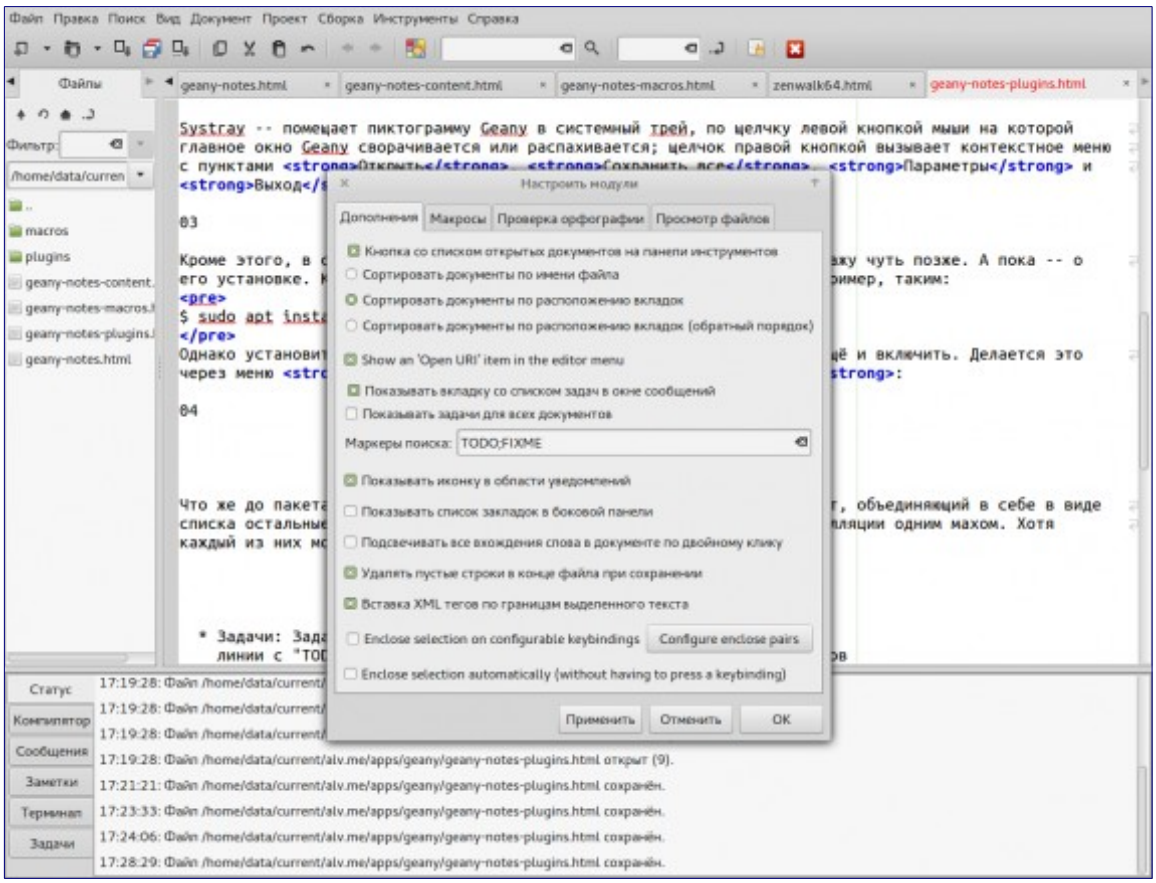
Кроме этого, в состав пакета входит ещё несколько аддонов, о чём я скажу чуть позже. А пока — о его установке. Каковая выполняется любым из стандартных способов, например, таким:

```
$ apt install geany-plugin-addons
```

Однако установить этот плагин (как и любой другой) мало — его надо ещё и включить. Делается это через меню Инструменты -> Менеджер модулей:



На скриншоте можно видеть кнопку Параметры — она может быть активной не для всех плагинов, но для geany-plugin-addons как раз активна, и вызывает панель его настроек, содержащую полный список аддонов, каковые могут быть включены или выключены очевидным образом:



Останавливаться на них я не буду, предоставив это занятие заинтересованным лицам. Отмечу только, что эта панель — общая для

настройки всех плагинов, таковую позволяющих. И может быть вызвана также через меню Правка -> Настройки модулей

Что же до пакета geany-plugins — это на самом деле метапакет, объединяющий в себе в виде списка остальные 63 «плагинные» пакеты, устанавливаемые при его инсталляции одним махом. Хотя каждый из них может быть установлен и в индивидуальном порядке. Поскольку лично мне нужны далеко не все плагины, на следующей странице в индивидуальном порядке я и рассмотрю некоторые из них. А пока напомню только, что установка полного набора плагинов не приводит к их мгновенной активизации — её нужно проделать вручную только что описанным способом. А некоторые из плагинов — ещё и настроить, о чём будет сказано в каждом конкретном случае.

Geany и его макросы

Одно из качеств, которое отличает развитый редактор от примитивного средства набора текста и исправления опечаток, — возможность наращивания его функционала. Ибо, сколь бы мощной не была исходная программа работы с текстом, предусмотреть всё, что может потребоваться впредь, мог только резиновый Полыхаев. Поэтому непременным её атрибутом должно быть наличие подключаемых пользовательских макросов. А поскольку вся эта книга ориентирована на применителей широкого профиля, желательно, чтобы процесс создания макросов не требовал чрезмерных навыков в программировании.

Geany в полной мере отвечает данным требованиям, позволяя записывать макросы простым протоколированием часто выполняемых действий, привязывать их к комбинациям горячих клавиш и, при необходимости, легко редактировать их либо собственными средствами, либо в текстовом редакторе (в том числе и в нём самом). Правда, делает он это не сам по себе, а с помощью специального плагина. Каковой и следует установить в первую очередь.

В Mint'е соответствующий плагин собран в виде отдельного пакета, который отыскивается так:

```
$ apt search geany-plugin | grep macro

p   geany-plugin-macro              - macro plugin for Geany

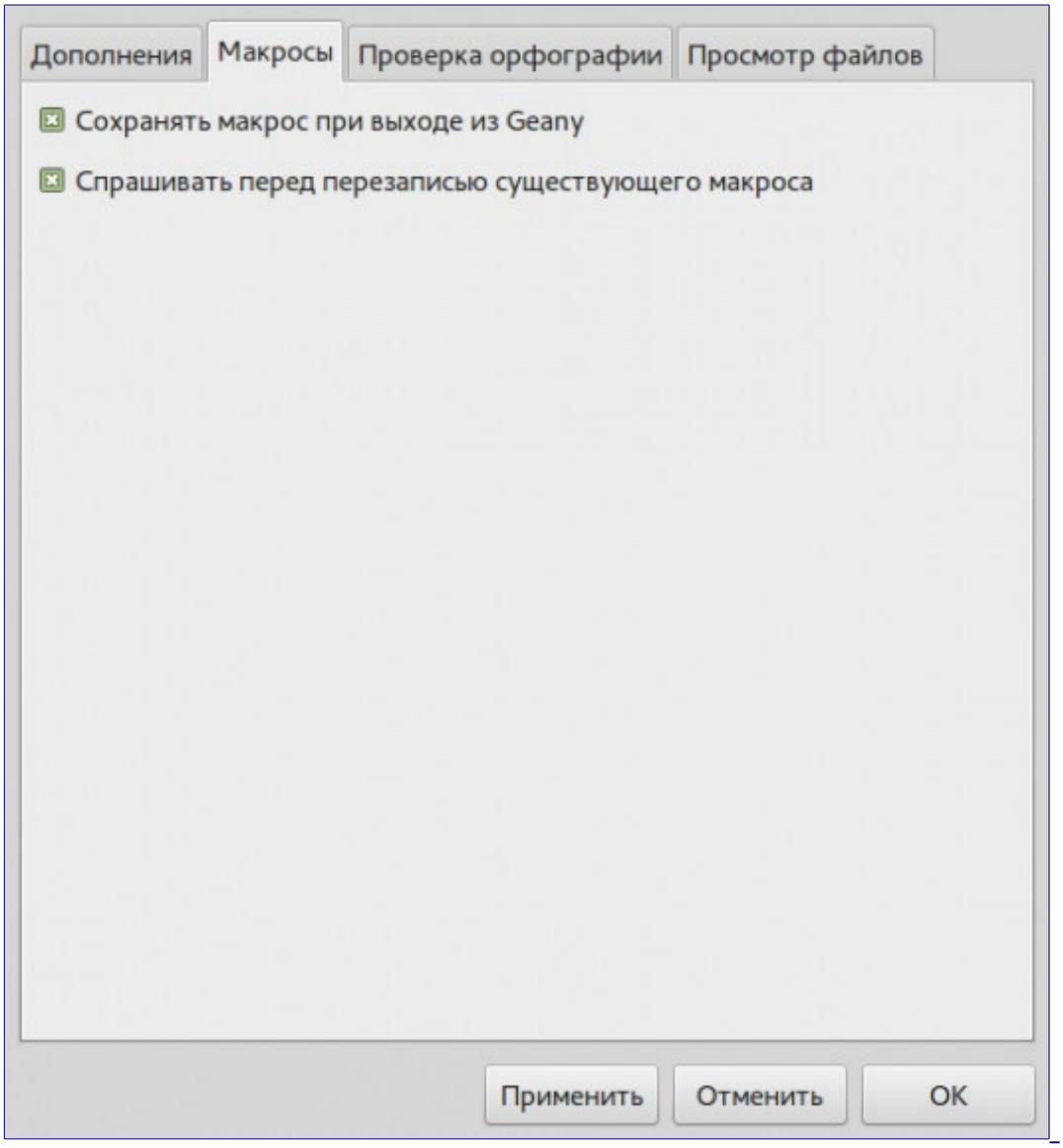
p   geany-plugin-macro:i386         - macro plugin for Gean
```

После чего его остаётся только установить:

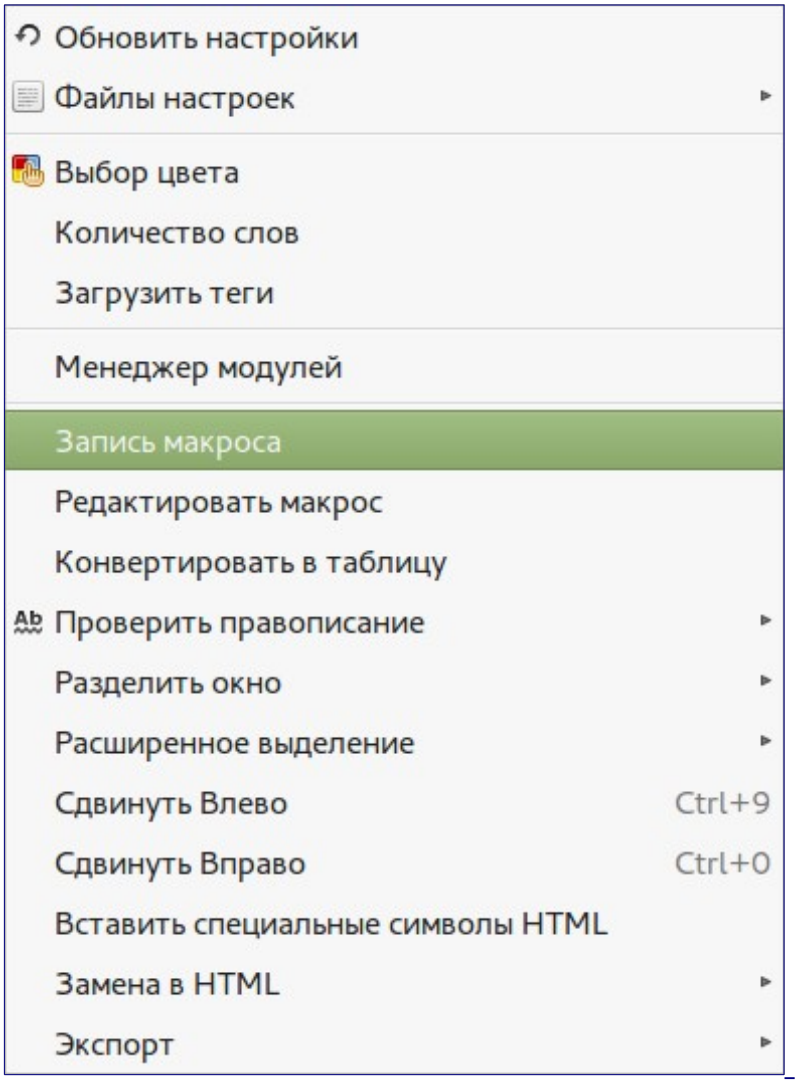
```
$ apt install geany-plugin-macro
```

Впрочем, это можно сделать и через mintinstall, о чём было сказано в соответствующем очерке.

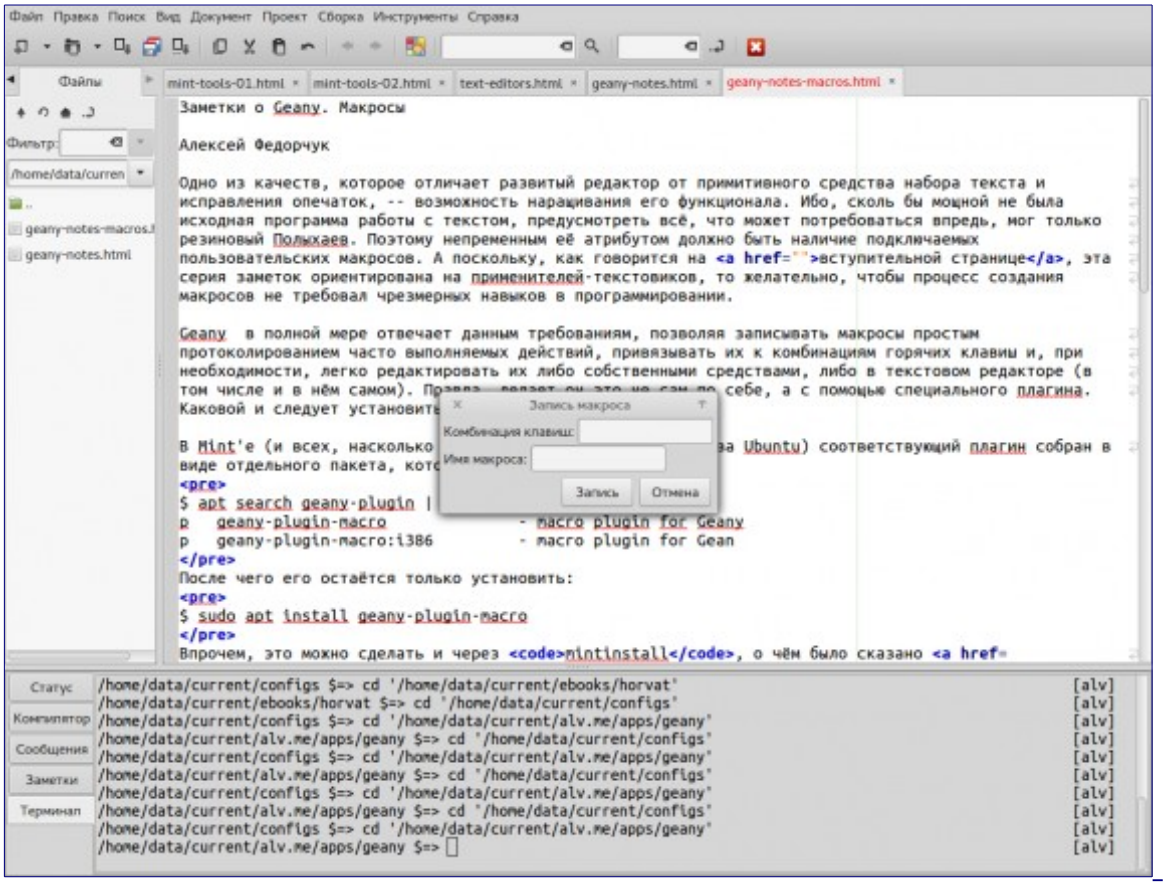
Да, разумеется, надо не забыть активировать плагин через главное меню: Инструменты -> Менеджер модулей, как это было описано в предыдущем миниочерке. А также заглянуть в его настройки через кнопку Параметры. Где, впрочем, ничего делать не надо — обе необходимые опции включены по умолчанию:



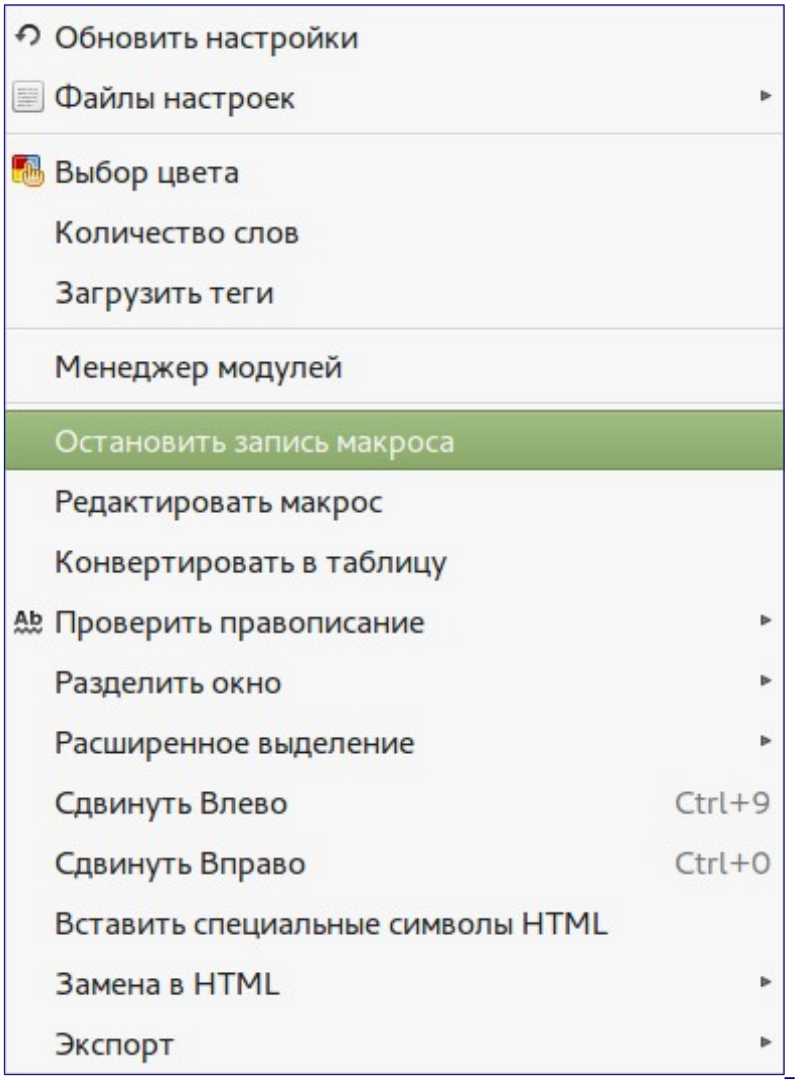
После этого в меню Инструменты появляются пункты Запись макроса и Редактировать макрос:



Как нетрудно догадаться, первый служит для протоколирования действий, которые будут претворены в макрос. Для чего требуется задать комбинацию запускающих его клавиш, имя макроса и нажать кнопку запись:



После чего выполнить действия, которые составят содержание макроса. Например, я использую их для ввода html-тегов, причём не всех, а только самых употребимых (мной), о которых легко забыть во время окончательного оформления страницы в CMS (я пользуюсь WodrPress'ом). Поэтому я просто вводу здесь текст открывающего и закрывающего тега (во избежание лишней работы автозакрытиые тегов средствами самой Geany лучше отключить). После чего снова обращаюсь к меню Инструменты, где пункт Запись макроса превращается в Остановить запись макроса



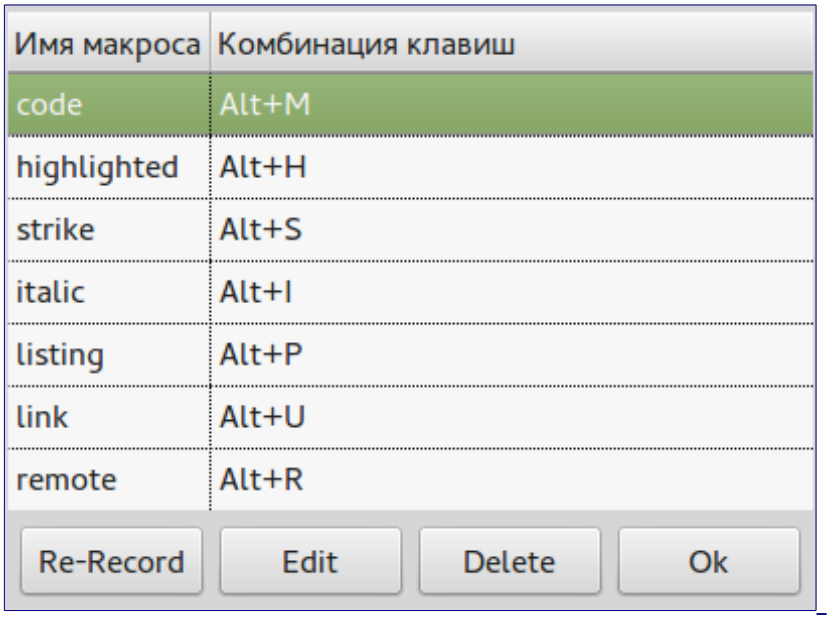
Самая сложная задача здесь — это подобрать комбинацию клавиш. Каковая должна быть

1. мнемонически значимой, и

2. не задействованной среди горячих клавиш самой Geany и рабочей среды, в которой она запущена.

Поскольку выполнение второго требования с каждым днём становится всё сложнее, то и с мнемоникой приходится извращаться. Например, для ввода тега для моноширинного шрифта мне пришлось задействовать комбинацию Alt+m — от monospace, и так далее. Впрочем, это та земля, на которой каждый умирает в одиночку — в соответствие со своими потребностями и возможностями. Кроме того, комбинации горячих клавиш, привязанные к макросам, не работают при русской раскладке клавиатуры — хотя все штатные хоткеи Geany от раскладки не зависят. Однако это горе не великое — всё равно переключаться с кириллицы на латиницу и обратно приходится очень часто.

Прямым протоколированием обычно трудно получить аккуратную макрокманду, не содержащую избыточных нажатий на клавиши. И тут стоит обратиться к пункту Редактировать макрос, которым вызывается список всех записанных макросов:



Как явствует из скриншота, любой из макросов можно перезаписать, удалить или отредактировать. Последняя операция, например, для редактирования гиперссылки выглядит так:



Порядок действий по редактированию макросов очевиден, поэтому останавливаться на этом вопросе не буду. Замечу только, что это — не единственный способ выполнения данной процедуры. Ибо описание всех макросов содержится в файле /home/alv/.config/geany/plugins/Geany_Macros/settings.conf. Это — обычный текстовый файл, и в соответствующем предыдущему скриншоту виде выглядит так:

```
[Settings]

Save_Macros=true

Question_Macro_Overwrite=true


[Macros]

A0=code

B0=109

C0=8
```

```
D0=2170,<code>,2170,</code>,2304,2304,2304,2304,2304,2304,2304

A1=highlighted

B1=104

C1=8

D1=2170,<strong>,2170,</strong>,2304,2304,2304,2304,2304,2304,2304,2304
,2304

A2=strike

B2=115

C2=8

D2=2170,<s>,2170,</s>,2304,2304,2304,2304

A3=italic

B3=105

C3=8

D3=2170,<em>,2170,</em>,2304,2304,2304,2304,2304

A4=listing

B4=112

C4=8

D4=2170,<pre>,2329,2329,2170,</pre>,2302

A5=link

B5=117

C5=8

D5=2170,<a href=\\»\\»>,2170,</a>,2304,2304,2304,2304

A6=remote

B6=114

C6=8

D6=2170,<a href=\\»\\» target=\\»_blank\\»
rel=\\»nofollow\\»>,2170,</a>,2304,2304,2304,2304
```

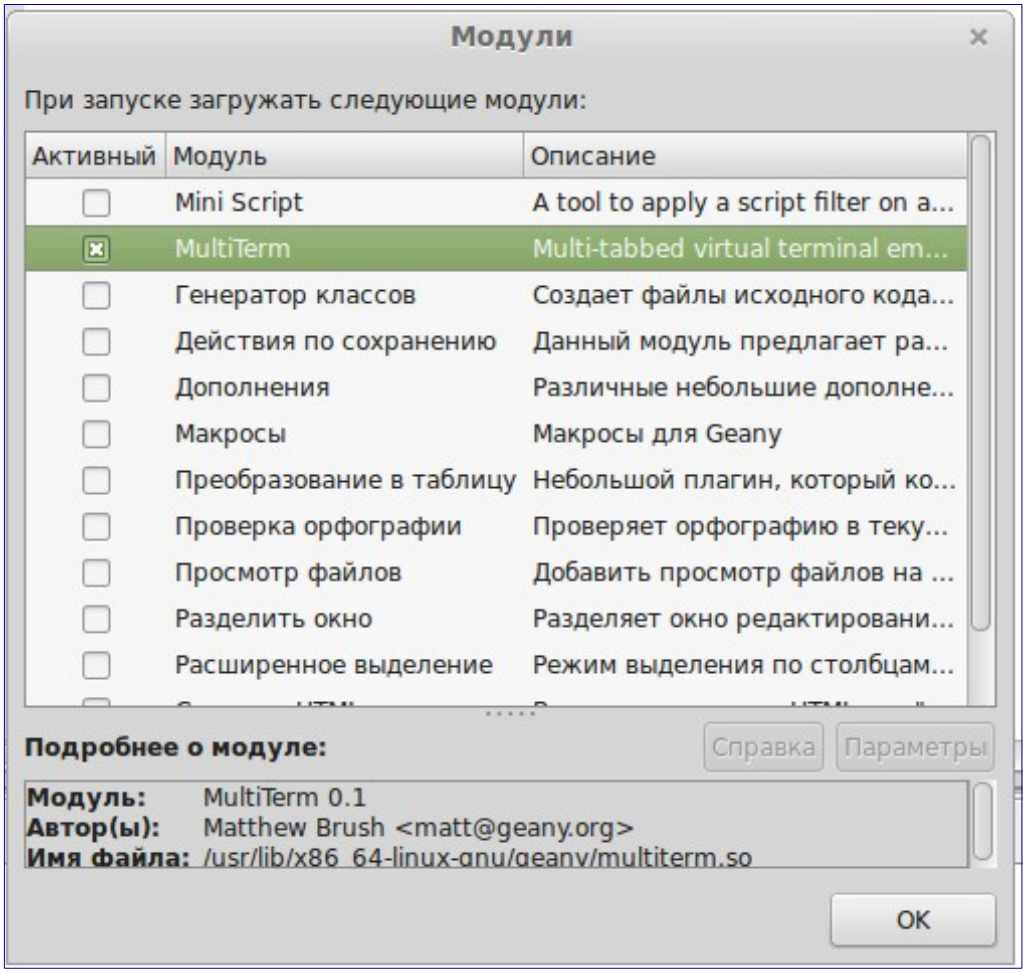
А потому может быть отредактирован непосредственно в текстовом редакторе — например, в самом Geany.

Geany и multiterm

Не так давно мы говорили о штатном встроенном терминале Geany — его хватает почти во всех случаях. Однако у него есть и более функциональный аналог — Multiterm, реализованный в виде плагина geany-plugin-multiterm. Его особенностью, как можно догадаться из названия, является поддержка вкладок (Tabs). Плагин этот входит в состав метапакета geany-plugins, но может быть установлен и отдельно, например, так:

```
$ apt install geany-plugins
```

После чего его надо активировать — никаких настроек для него на первый взгляд не предусмотрено:

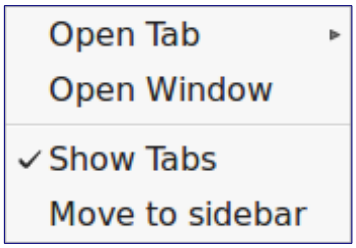


После этого в окне сообщений появляется новая вкладка, которая так и называется Multiterm. И по умолчанию имеет весьма бледный вид:



В частности, запущенный в этом окне шелл и представляется как /bin/bash. Однако он запущен в режиме эмуляции POSIX shell, то есть не умеет ни автодополнения, ни истории команд... короче, ничего, за что мы так любим современные командные оболочки. Вызванное по аналогии со штатным

терминалом контекстное меню позволяет открыть новую вкладку или переместить окно Multiterm в боковую панель (зачем это нужно — не знаю), но не содержит никаких возможностей для настройки:



Кстати, в Mint'e не работает и пункт Open Window — и сейчас станет ясно почему.

Однако доступ к настройкам возможен прямым редактированием конфига этого плагина — ~.config/geany/plugins/multiterm/multiterm.conf. Он разделяется три секции:

- General Settings — общие свойства;
- Default Shell — свойства умолчальной оболочки;
- Other Shells — свойства прочих оболочек.

В первой из них можно, в частности, переопределить значение параметра external_terminal с умолчального xterm на, например, gnome-terminal. После этого волшебным образом заработает пункт Open Window из контекстного меню — он будет открывать терминал GNOME: дело в том, что в Mint'e по умолчанию xterm не устанавливается.

В секции Default Shell следует в первую очередь заменить этот самый default'ный шелл на что-то более человеческое, отредактировав строку

```
command=sh
```

должным образом, например, у меня таким:

```
command=/bin/zsh
```

Очевидно, что, отредактировав строки

```
bg_color=#ffffff
```

```
fg_color=#000000
```

```
font=Monospace 9
```

можно изменить цвет фона, текста и шрифтоначертание с кеглем для него, соответственно. Например, у меня это сделано так:

```
bg_color=#D6D6D6
```

```
fg_color=#000000
```

```
font=Monospace 15
```

А сняв комментарий со строки

```
scrollback_lines=1024
```

можно установить желаемую величину для прокрутки истории.

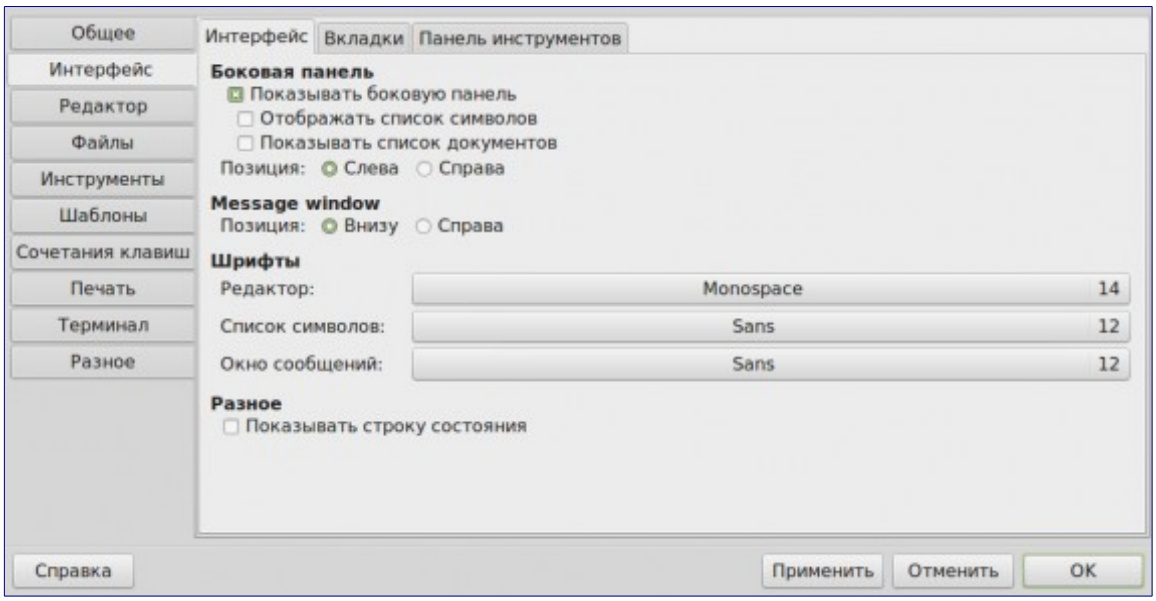
В Multiterm нет опции следования пути текущего файла — смена каталога возможна только прямой командой cd. Поэтому его можно держать включённым в паре со штатным терминалом, когда требуется одновременный доступ и к каталогу текущего документа, и к некоему фиксированному каталогу. И даже к нескольким — не будем забывать про возможность открытия табов, в каждом из которых запускается свой экземпляр шелла.

К сожалению, в Multiterm не и кое-чего другого, более важного. В частности, в нём категорически отказываются работать стандартные кейбиндинги типа Control+A, Control+E и им подобные. Поэтому в качестве замены штатному терминалу Geany он никак не годится. Но как его дополнение в некоторых случаях может быть полезен.

Geany и управление файлами

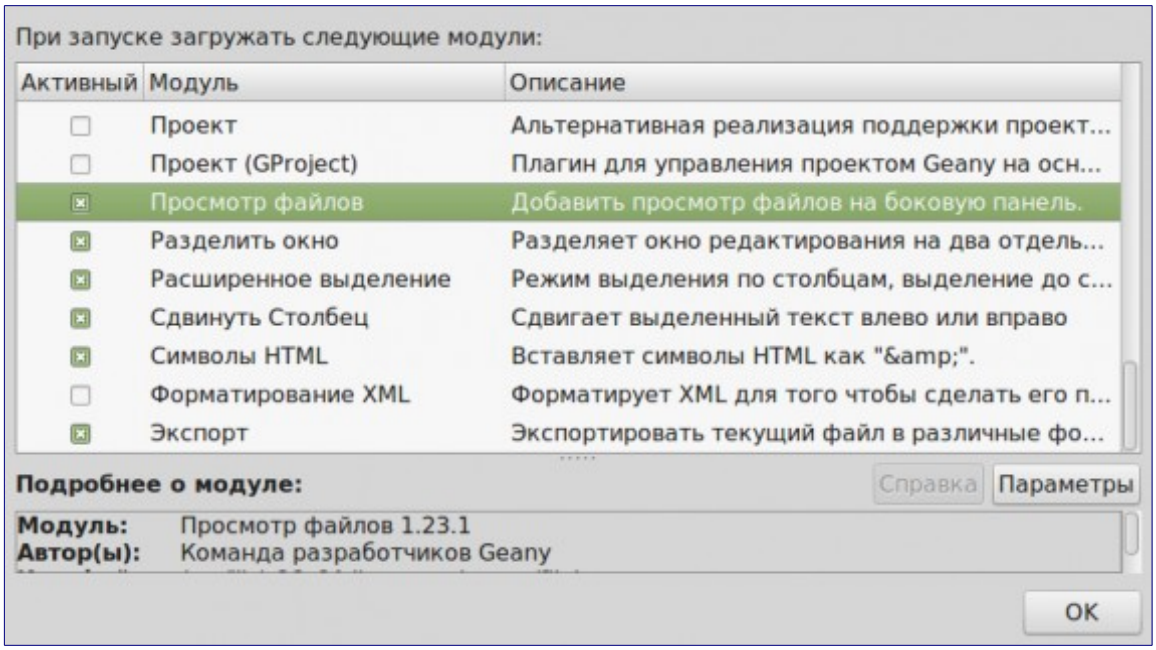
Казалось бы, управление файлами не имеет никакого отношения к сочинению и редактированию текстов, тем более нарративных. Однако практика показывает, что это не так — когда доходит до масштабных проектов, типа сочинения книги, оказывается, что средства файлового менеджмента отнюдь не лишни. Ибо они неразрывно связаны со средствами управления проектами, о которых пойдёт речь в следующем миниочерке.

Прежде чем заводить речь о средствах управления файлами в Geany, надо напомнить сказанное ранее о его боковой панели, посредством которой управление файлами осуществляется. По умолчанию, без подключени каких бы то ни было плагинов, она содержит две вкладки — Символы и Документы. Отображение любой из этих вкладок (и даже обеих сразу) можно отключить через меню — Правка -> Настройка -> Интерфейс -> Интерфейс:

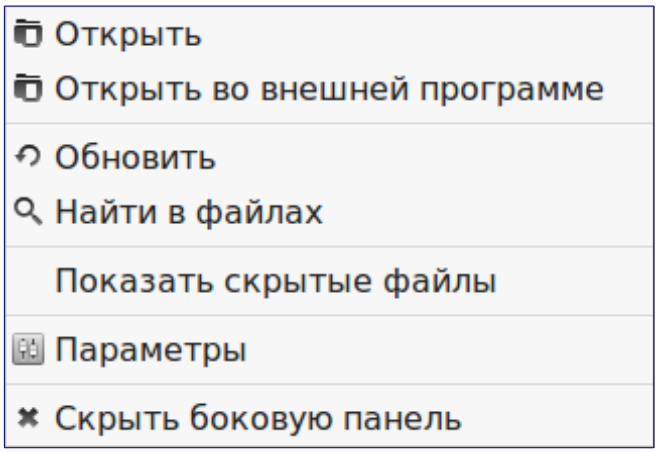


Что я и делаю, так как обычно они мне не нужны. А высвободившуюся боковую панель (вывод которой, кстати, тоже можно отключить вообще) использую как раз для управления файлами. Средств для чего в Geany как минимум два (не считая средств CLI, доступных через встроенный терминал).

Первое из них — filebrowser, который ныне входит в штатный комплект основного пакета. Однако включается он по прежнему через меню Инструменты -> Менеджер модулей ->Просмотр файлов:



Впрочем, ничего особенного, кроме просмотра файловой иерархии и простейших средств навигации, оно и не даёт:



Разве что через пункт Параметры из контекстного меню можно включить полезные пункты Следовать пути текущего файла и Использовать каталог проекта.

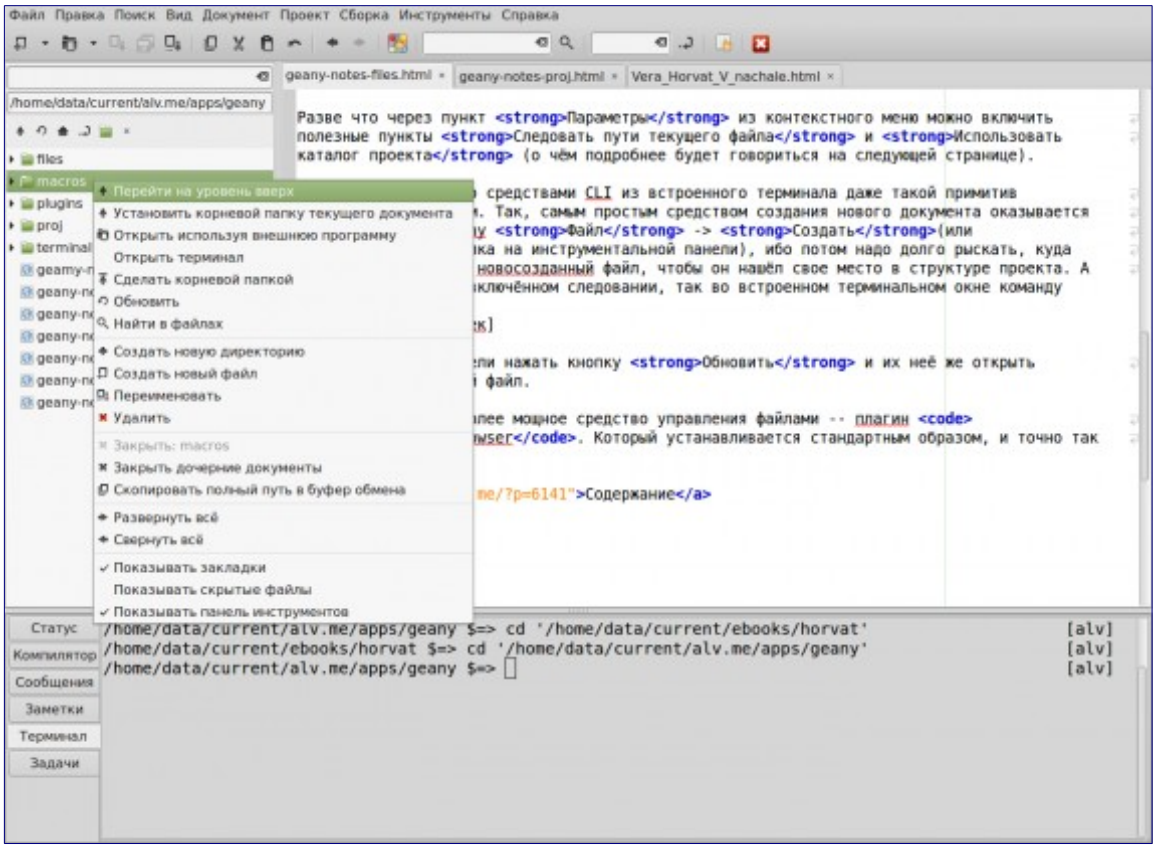
Хотя в комбинации со средствами CLI из встроенного терминала даже такой примитив оказывается полезным. Так, самым простым средством создания нового документа оказывается не главное меню Geany Файл -> Создать(или соответствующая кнопка на инструментальной панели), ибо потом надо долго рыскать, куда записать этот самый новосозданный файл, чтобы он нашёл свое место в структуре проекта. А гораздо проще, при включённом следовании, так во встроенном терминальном окне команду

```
$ touch [файл_имя_рек]
```

Затем в боковой панели нажать кнопку Обновить и их неё же открыть новосозданный пустой файл.

Но в Geany есть и более мощное средство управления файлами — плагин

geany-plugin-treebrowser. Который устанавливается стандартным образом, и точно так же включается. После чего из контекстного меню становятся доступными многие функции стандартного файлового менеджера:



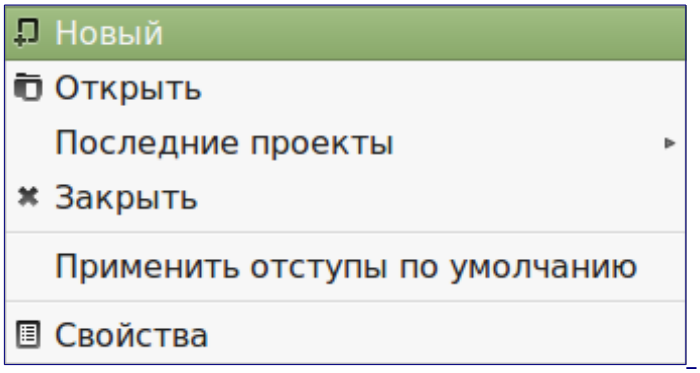
Да, не все, какие могут потребоваться. Но по крайней мере создавать новый файл в составе существующего проекта становится легко и просто. А с остальным, ребята, и сами разберётесь, если будет нужно...

Geany: управление проектами

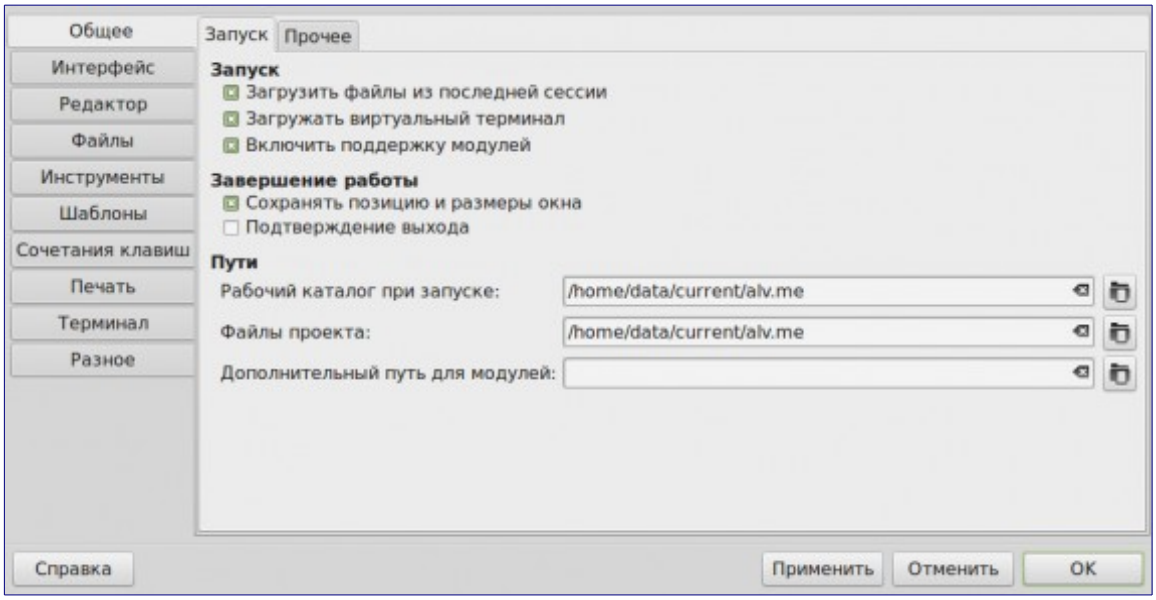
Всякому профессиональному сочинителю приходится писать не только отдельные статьи или заметки, но и их циклы или серии (а то и, страшно сказать, книги). И при этом часто работа над такими циклами/сериями разной тематики проходит в параллельном режиме, так что требуется быстрое переключение из окружения одного цикла (а это не только сами тексты, но и иллюстрирующие их картинки, а также всякого рода служебные материалы) в среду другого.

Тут-то на помощь сочинителю приходят встроенные в текстовый редактор средства управления проектами — если, конечно, они в нём есть. Благо, в Geany такие средства имеются, и даже не в единственном экземпляре: одно штатное и два внештатных, обеспечивающихся отдельными плагинами.

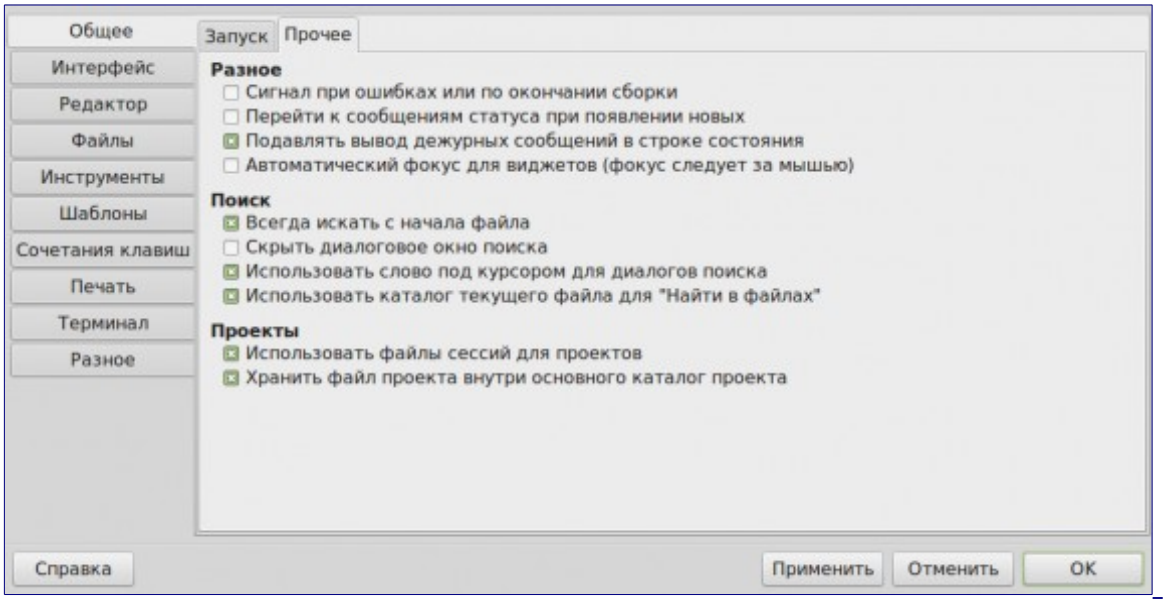
Как нетрудно догадаться, работа с проектами осуществляется через одноимённый пункт главного меню. Столь же очевидно, что начинается эта работа с создания проекта через пункт Новый:



Однако прежде не плохо выполнить некоторые глобальные настройки, общие для всех проектов. Которые выполняются через пункты Правка -> Настройки -> Общее. Сначала по вкладке Запуск указывается общий путь к файлам проекта. Поскольку проект наверняка будет не один (иначе не стоило бы и огород городить), есть смысл задать его в максимально обобщённом виде, а далее дописывать конкретные подкаталоги:

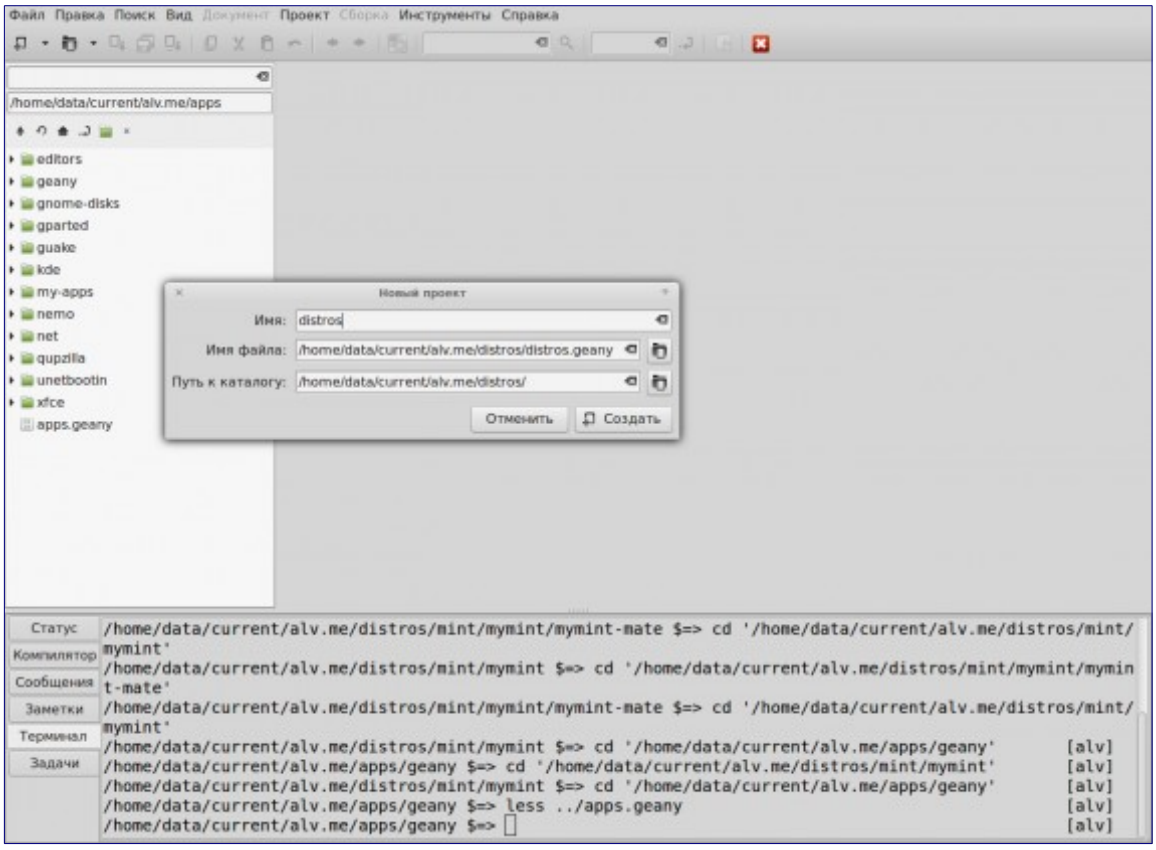


Здесь же полезно задать стартовый каталог (для меня удобно, чтобы он совпадал с путём к подкаталогам проектов, но это уж каждый для себя решит сам) и включить загрузку файлов из последней сессии. После чего перейти на вкладку Прочее и там в секции Проекты отметить пункт Использовать файлы сессий для проектов. В сочетании с загрузкой файлов из последней сессии на предыдущей вкладке это приведёт к тому, что при старте Geany будет открываться последний проект в том самом состоянии, в каком работа с ним была прервана:



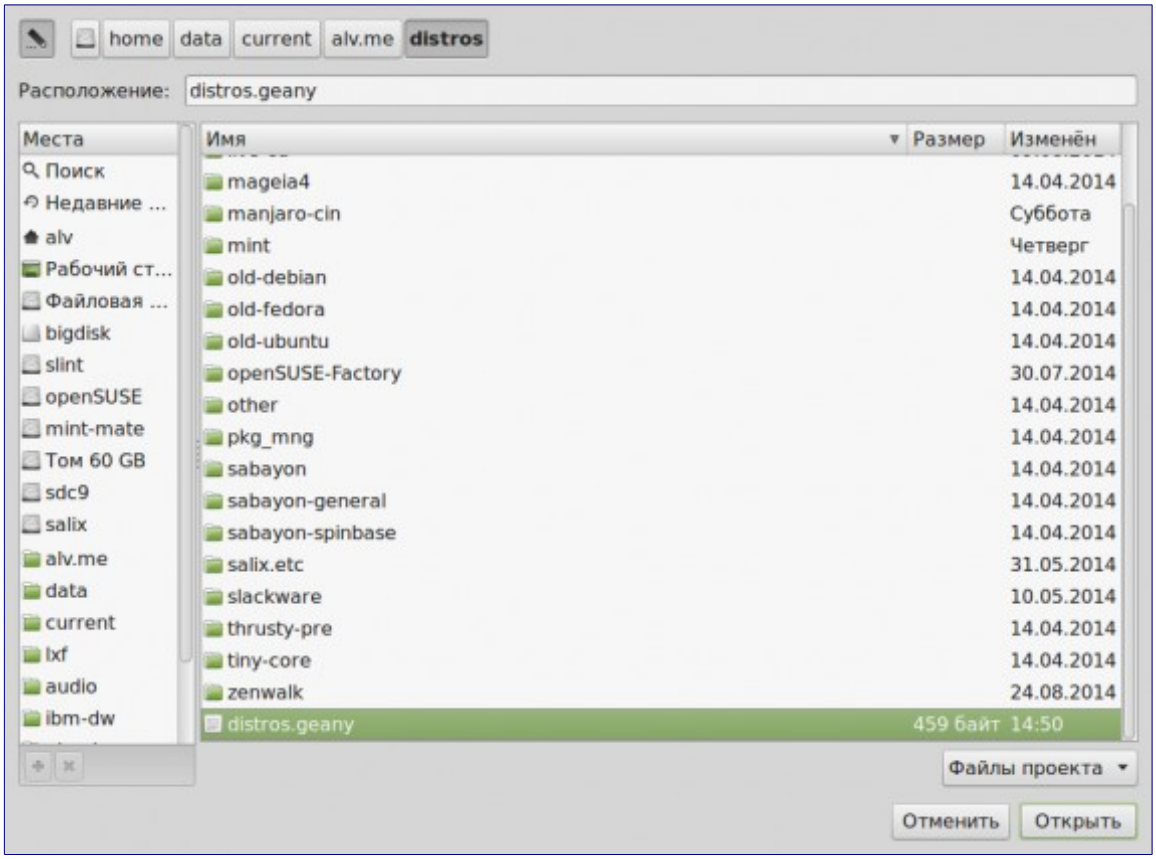
А вот пункт Хранить файл проекта внутри основного каталога проекта включается по желанию. По умолчанию Geany хранит все файлы описания проектов (вида project_name.geany)в отдельном каталоге, который так и называется — path2/projetcs (или, в русской локализации, path2/проекты). Если же включить означенный пункт, то файл описания проекта будет размещаться непосредственно в его каталоге. Мне это представляется удобным, особенно при сохранении резервных копий, однако применителю следует определить это для себя.

Теперь можно взяться за создание проекта, для чего в меню Проект выбирается пункт Новый. Если при этом был окрыт какой-либо ранее существовавший проект — последует предложение его закрыть, с чем следует согласиться: два одновременно открытых проекта Geany не поддерживает. После чего возникает вот такая панель — разумеется, пока без имени проекта:

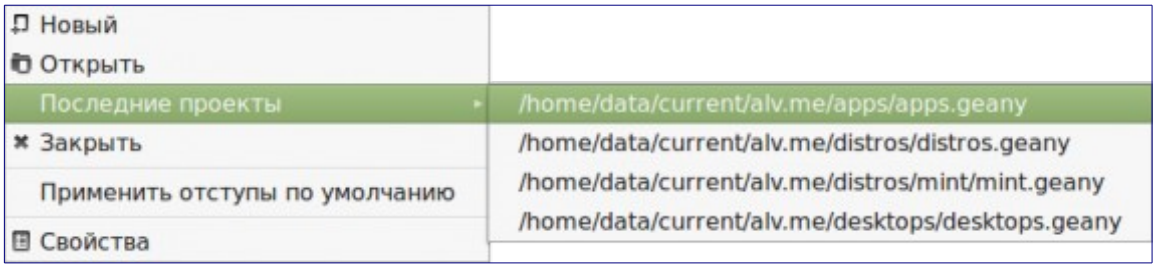


Если все настройки были сделаны, как описано выше — то достаточно задать это самое имя проекта (совпадающее с именем каталога с его файлами), и нажать кнопку Создать. И проект, вместе с файлом его описания (в данном случае /home/data/current/alv.me/distros/distros.geany), будет создан.

Для открытия существующего проекта предназначен пункт меню Проект -> Открыть, после чего в дереве файловой иерархии потребуется отыскать каталог проекта, а в нём — файло описания:

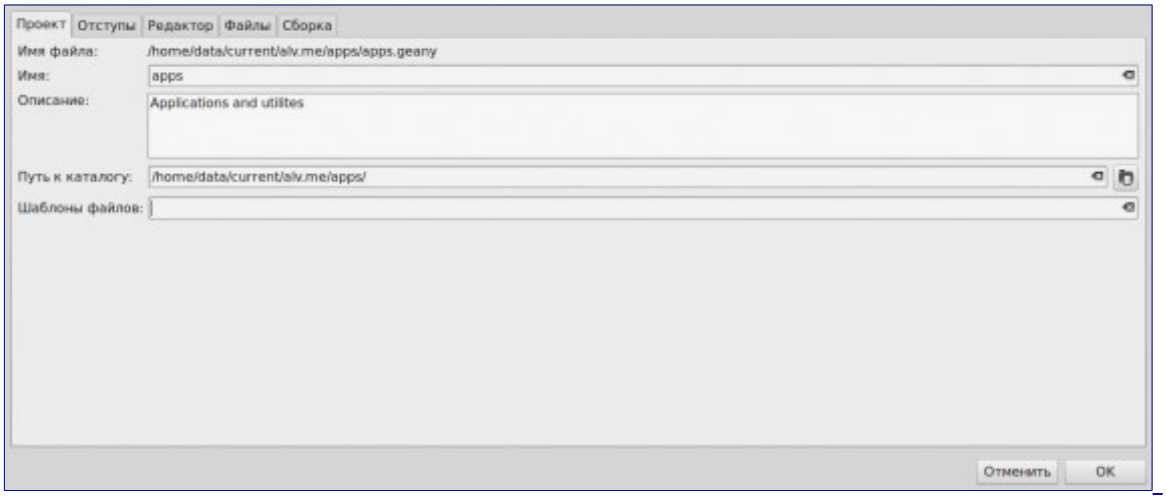


Это может быть достаточно скучно. И потому между существующими проектами проще переключаться через Проект -> Последние проекты:



Впрочем, и в том, и в другом случае, как я уже говорил, проект будет открыт точно в том состоянии, в котором с ним была прекращена работа — то есть со всеми файлами, открытыми в последней сессии.

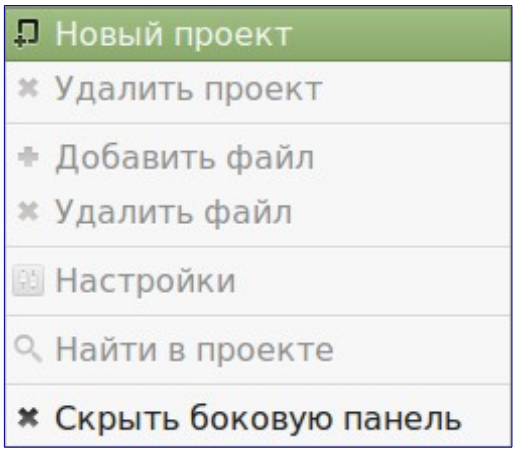
Основные данные о проекте можно посмотреть через Проект -> Свойства. Здесь, в частности, можно поменять имя проекта, его описание и рабочий каталог:



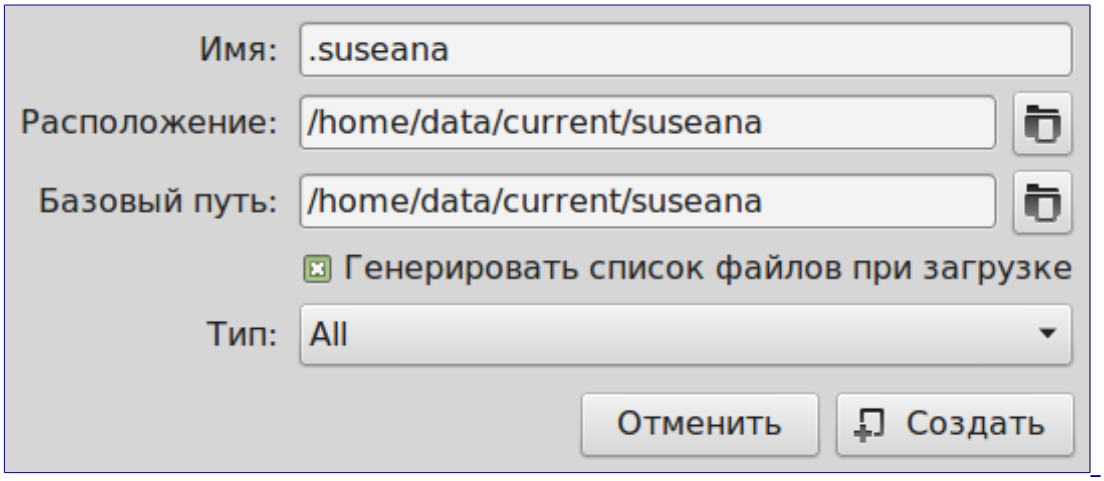
А вот изменить путь к файлу описания проекта здесь не получится. Но это можно сделать через конфигурационный файла Geany — ~/.config/geany/geany.conf. Это простой текстовый файл, в котором отражаются все настройки редактора, выполненные интерактивно. Но который можно и отредактировать непосредственно. В частности, именно здесь, в секции [project], можно изменить путь к файлу описания проекта — например, если его рабочий каталог был перемещён в другое место файловой иерархии.

Кроме штатной системы управления проектами, для этой цели в Geany существует два плагина — geany-plugin-gproject и geany-plugin-prj. Первый является дополнением штатного средства, предлагая управление проектами Geany на основе шаблонов. Включение его через Инструменты -> Менеджер модулей приводит к появлению в боковой панели вкладки Проект с несколькими пиктограммами. Честно говоря, я так и не придумал, как прикрутить шаблоны к своим проектам — и главное, зачем мне это нужно. И потому отключил этот плагин.

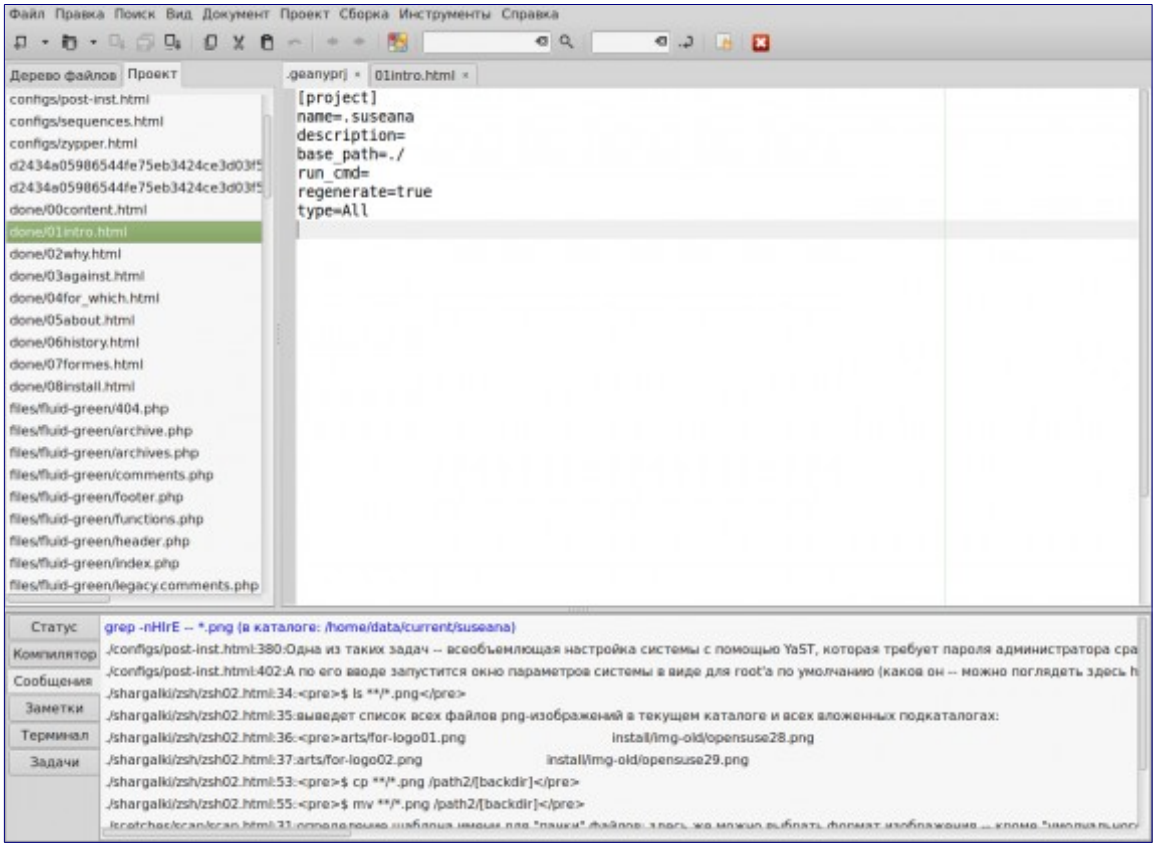
Плагин же geany-plugin-prj представляет собой полную систему управления проектами, альтернативную штатной. При его включении в боковой панели также появляется вкладка Проект — абсолютно пустая, но с контекстным меню по правой кнопке мыши:



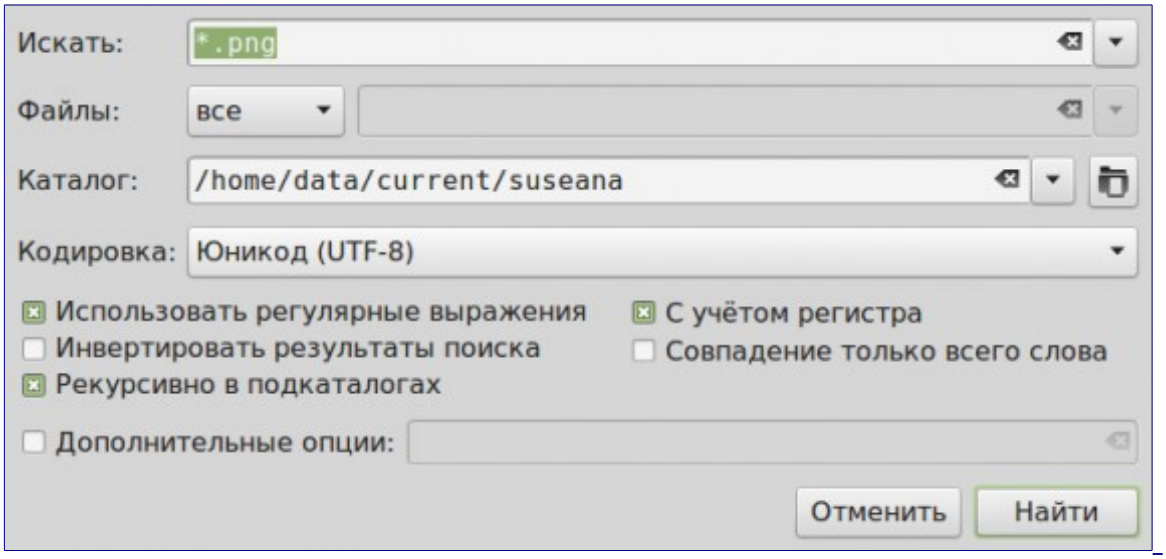
Вследствие своей альтернативности этот плагин игнорирует штатные файлы описания проектов — для него их надо создавать с нуля, как можно видеть на приведённом скриншоте. Выбор пункта Новый проект вызывает такую вот панель:



В ней указывается имя проекта, определяются расположение файла его описания (это dot-файл geanyprj и рабочий каталог. После нажатия кнопки Создать все файлы рабочего каталога включаются в проект и выводятся в соответствующей вкладке боковой панели, и генерируется файл описания:

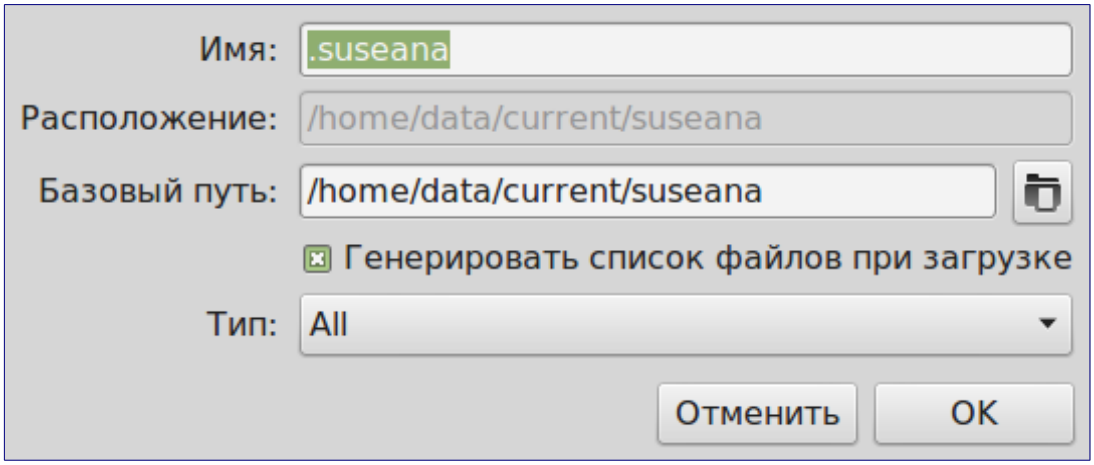


После создания проекта через контекстное меню на его вкладке в боковой панели становятся доступными, помимо прочих, пункты Найти в проекте и Настройки. Собственно, поиск в проекте — это главная особенность альтернативного управления проектами:



Результаты поиска выводятся в окне сообщений, как это видно на скриншоте, предшествующем последнему.

Ну а через пункт Настройки, как и в штатном «управителе проектами», можно поменять его имя и базовый каталог (но не положение и имя файла описания проекта):



Все функции контекстного меню дублируются также в меню главном — Инструменты -> Проект.

Повозившись с альтернативным «управителем проектов», я пришёл к выводу, что использование его имеет смысл действительно как альтернатива — совместное применение его со штатным средством этого назначения приводит к путанице. А поскольку главным для меня в этом деле — простой способ быстрого переключения между проектами (что в штатном средстве реализовано лучше), я в конце концов отключил «альтернативщика». Но возможно, что кому-то он покажется более удобным.

Текстовый редактор Komodo Edit

Текстовый редактор Geany я применял на протяжении многих лет, люблю его и более или менее знаю. А потому и уделил ему столько места. Однако последнее время основным моим инструментом для работы с текстами стал Komodo Edit. И не написать о нём здесь я не мог. Мой стаж Komodo'вания едва составляет полгода, так что особых подробностей не будет — лишь отдельные заприсовки по его применению. Но надеюсь, что кому-нибудь они

пригодятся для ориентировки в океане текстовых редакторов.

Обзор

Во избежание недоразумений (которым я в своё время поддался) надо для начала сказать, что под именем Komodo выступают две довольно разные программы:

1. Komodo IDE, полноценная интегрированная среда разработки, имеющая статус коммерческой, стоимостью под сотню баксов (для осознания того, за что они платятся, доступна трёхнедельная Trial'ная версия), и

2. Komodo Edit (далее KE), представляющий собой развитый текстовый редактор с поддержкой управления проектами, протоколированием макросов и прочими функциями, сопоставимыми с таковыми в Geany; он распространяется на условиях Mozilla Public License, то есть являет собой свободный Open Source в чистом виде.

Общее между этими программами то, что они разрабатываются одними и теми же людьми — сотрудниками фирмы Active State, Komode IDE полностью, Komode Editor — при участии сообщества.

Для установки Komodo Editor (далее KE) предлагается скачать его в виде архива бинарников с сайта фирмы (текущая на данный момент версия — 8.5.4), развернуть архив и запустить установочный скрипт. Этот рецепт подходит для произвольного дистрибутива.

Однако в дистрибутивах из кланов Ubuntu и Mint'a можно поступить проще и правильней: KE имеется на Launchpad'е в виде бинарного пакетая. Так что для его установки просто подключаем его репозиторий:

```
$ sudo add-apt-repository ppa:mystic-mirage/komodo-edit
```

Выполняем обновление локального кеша:

```
$ apt update
```

Проверяем доступность пакета и его точное имя:

```
$ apt search komodo

i   komodo-edit                  - Komodo Edit is a free, open source editor

p   komodo-edit:i386              - Komodo Edit is a free, open source editor
```

После чего только и только остаётся этот пакет установить:

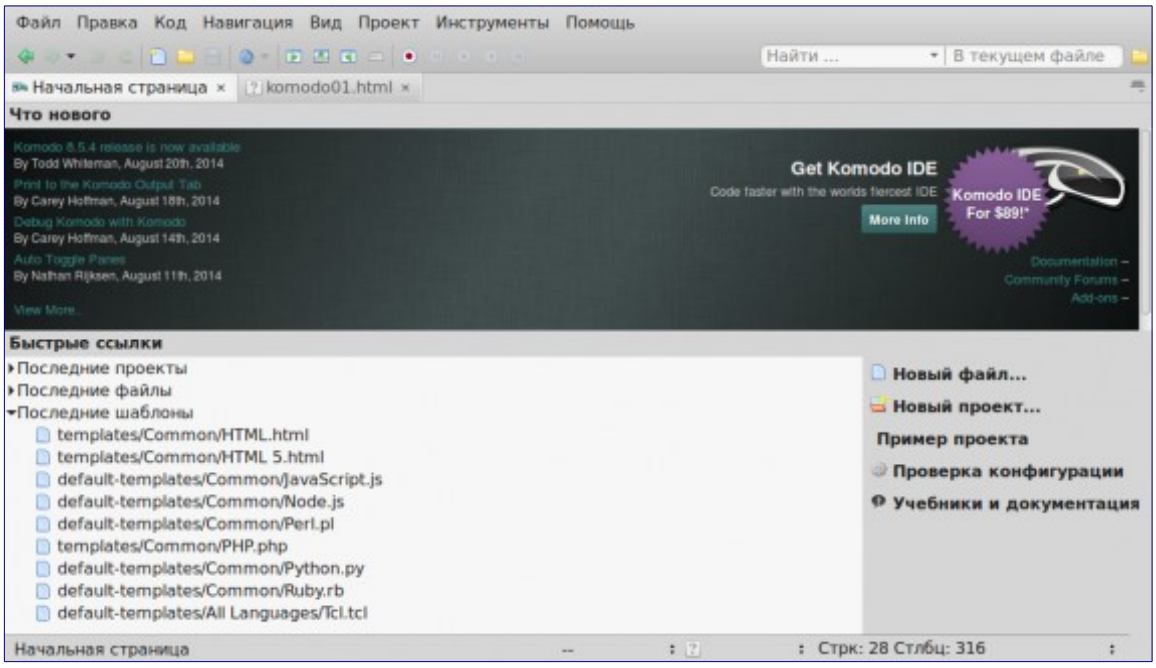
```
$=> sudo apt install komodo-edit
```

Теперь запустить редактор можно прямой командой komodo-editиз CLI или минитерминала. Ну и, конечно, из главного меню — Программирование -> KomodoEdit.

После первого запуска сначала появляется панель быстрого запуска, почти загораживающая всё остальное:



А после её закрытия (её можно удалить со старта раз и навсегда, вызывая по необходимости) KE предстаёт примерно в следующем обличье:

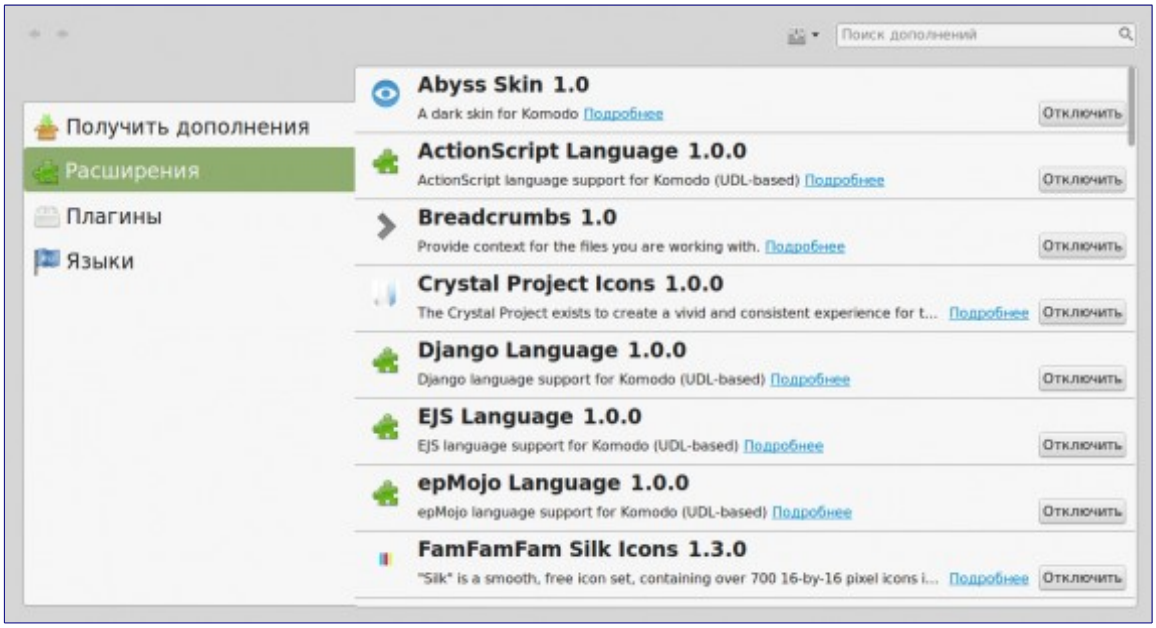


Разумеется, в таком виде KE не может внушить ничего, кроме ужаса от мысли, что в этом надо ещё и работать... Однако первоочередные шаги по приведению его в человеческий вид Bregо описаны Валерием Желябовским, и повторять его я не буду. А остановлюсь на тех моментах, которые представляют для меня самый большой интерес.

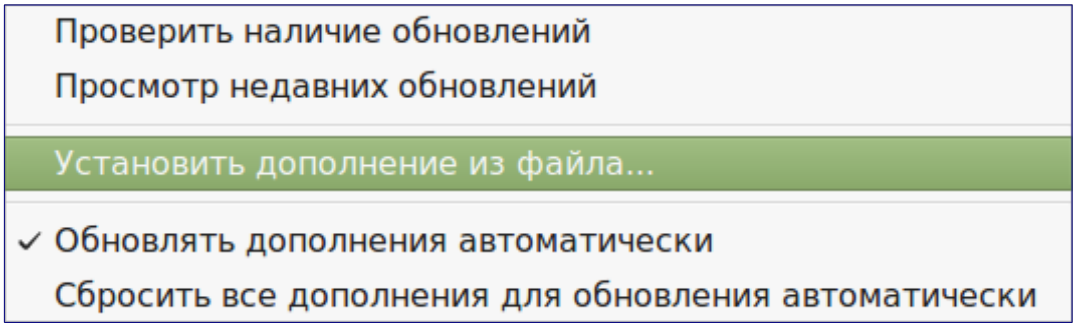
Русификация

Первое — русификация интерфейса KE. Таковая имеет место быть благодаря усилиям laborpago (за что и ему спасибо большое). И может быть выполнена двумя способами. Первый — скачать авторский пакет с его страницы. Он собран в виде расширения для KE — файла вида localru-*-ko.xpi — нужно только проследить, чтобы версия пакета соответствовала версии KE, ибо они жёстко привязаны друг к другу.

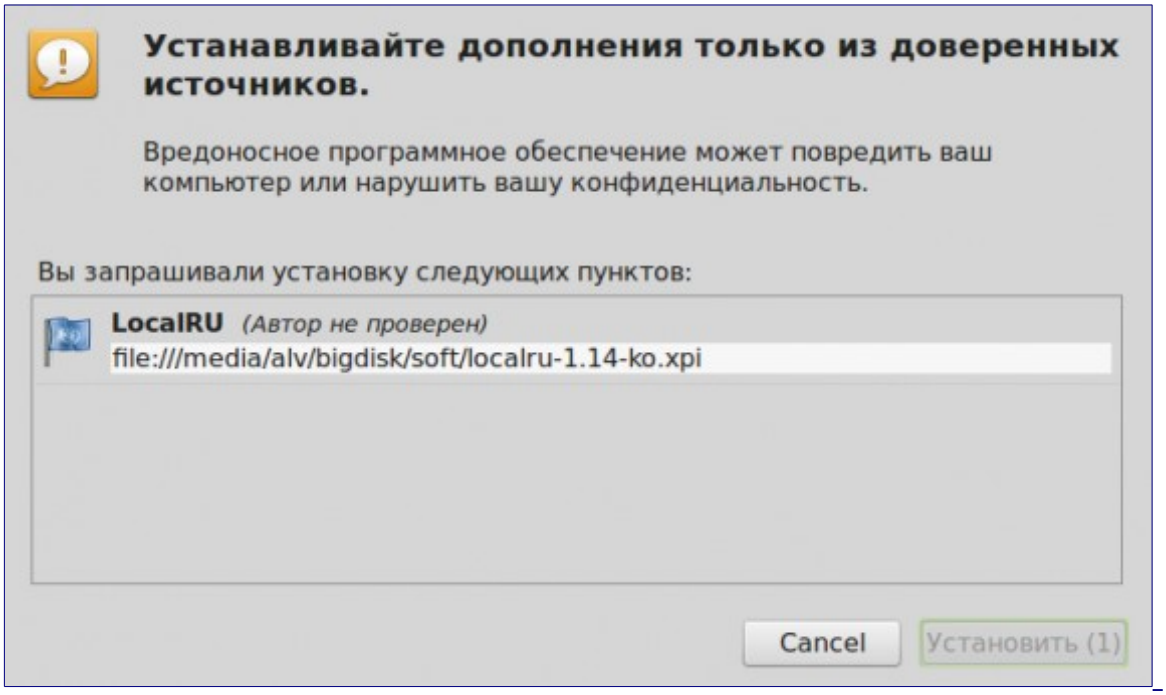
Установка русификатора в этом случае выполняется через меню KE: Tools -> Add-ons, что вызывает такую панель:



Слева от поля для поискового запроса можно видеть кнопку весьма бледного вида, которая вызывает каскадное меню — в нём нужно выбрать пункт Установить дополнение из файла:



А по выборе нужного файла в следующей панели нажимается кнопка Install Now:



После чего остаётся только перезагрузить KE. Кстати, этот миниочерк сочинялся в KE после его русификации, и потому на всех скриншотах можно видеть русские буковки в интерфейсе (не раз-русифицировать же KE обратно, как сказал Жорж Милославский).

Второй способ русификации — ещё проще: в том же репозитории ppa:mystic-mirage/komodo-edit существует и штатный пакет аналогичного назаначения:

```
$ apt search komodo-edit-ru

p   komodo-edit-ru      - Komodo Edit is a free, open source editor
```

Который, соответственно, можно установить через apt.

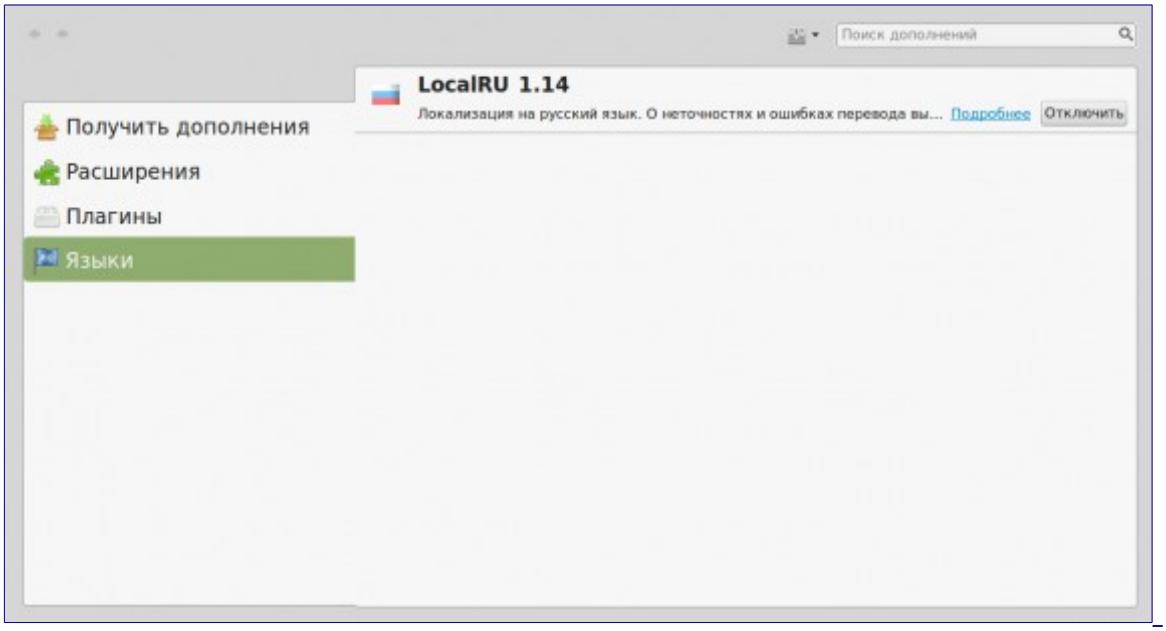
Претензий к работе русификатора, установленного как Add-ons, у меня не было, но полноты раскрытия темы я решил попробовать и этот метод. Для чего отключил расширение localru через меню: Дополнения -> Языки, перезапустил KE, чтобы убедиться в возврате англоязычного интерфейса, и установил пакет из репозитория:

```
$=> sudo apt install komodo-edit-ru
```

Однако после перезапуска KE ни малейших следов русского языка в интерфейсе не обнаружилось. Тогда я удалил localru вообще (через те же пункты меню), и выполнил реинсталляцию пакета:

```
$=> sudo apt install --reinstall komodo-edit-ru
```

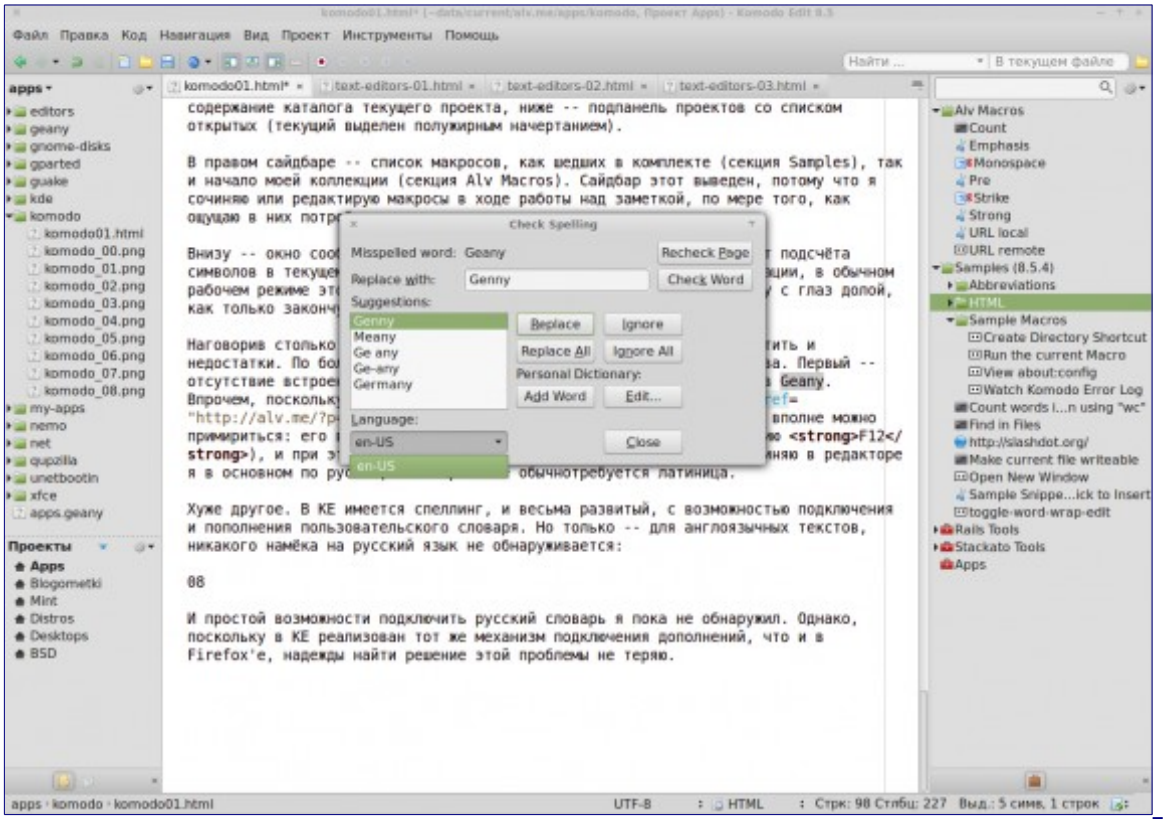
После чего интерфейс волшебным образом русифицировался, а среди дополнений в языках обнаружилось всё то же самое LocalRU 1.14:



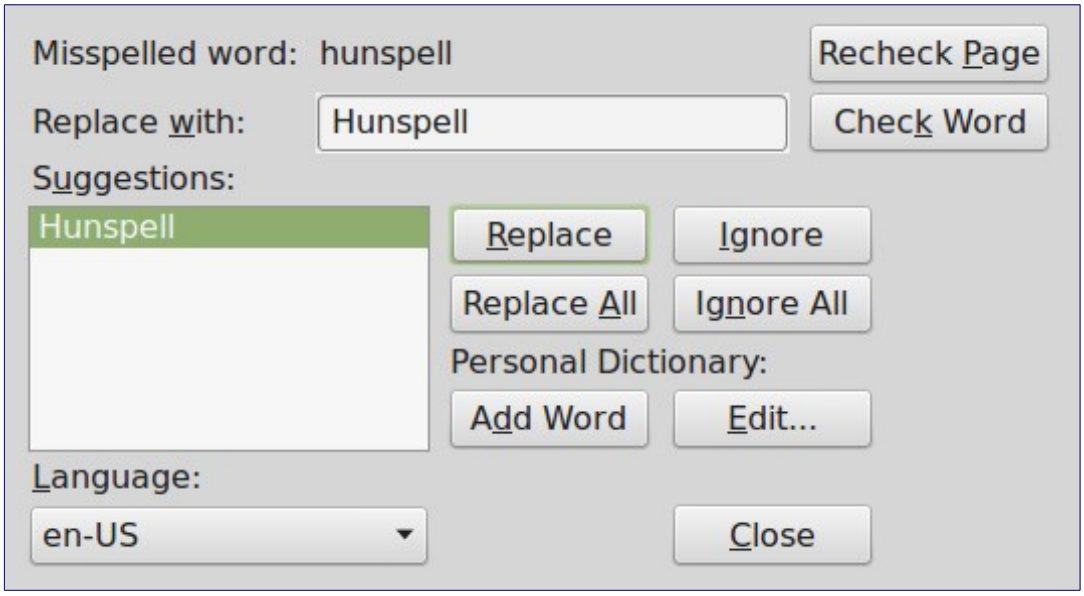
Иными словами, это просто один и тот же пакет laborpago. Только из авторского дополнения его компоненты устанавливаются в .komodoedit/8.5/XRE/extensions/, а из deb-пакета — в /usr/lib/komodo-edit/mozilla/extensions/. Что во втором случае требует, соответственно, привилегий администратора, но зато русификация распространяется на всех возможных пользователей KE.

Оба способа русификации — вполне штатные: первый — с точки зрения KE, второй — с позиций deb based дистрибутивов, оба допускают автоматическое обновление, ну и русские слова в результате получаются одними и теми же. Так что каким способом пользоваться — дело вкуса. С точки зрения сборки собственных ремиксов для меня оказался предпочтительным второй.

В KE имеется спеллинг, и весьма развитый, с возможностью подключения и пополнения пользовательского словаря. Но только — для англоязычных текстов, никакого намёка на русский язык поначалу не обнаруживается:



Однако оказалось, что проблема эта решается сама собой, причём не просто, и даже не очень просто, а ещё проще. Из просмотра сайта ActiveState Community выяснилось, что в KE используются те же самые словари, что и в Mozilla, то есть словари для hunspell'a. Однако общесистемные словари, например, вроде установленного у меня hunspell-ru-ie-yo (с поддержкой буквы Ё) он не видит — ему требуется собственные их копии, расположенные в каталоге /usr/lib/komodo-edit/mozilla/dictionaries. По умолчанию там лежат файлы только для словаря американского английского: en-US.aff и en-US.dic. Так что достаточно скопировать в него из /usr/share/hunspell файлы ru_RU.aff ru_RU.dic — и русский язык в панели спеллинга появится незамедлительно:



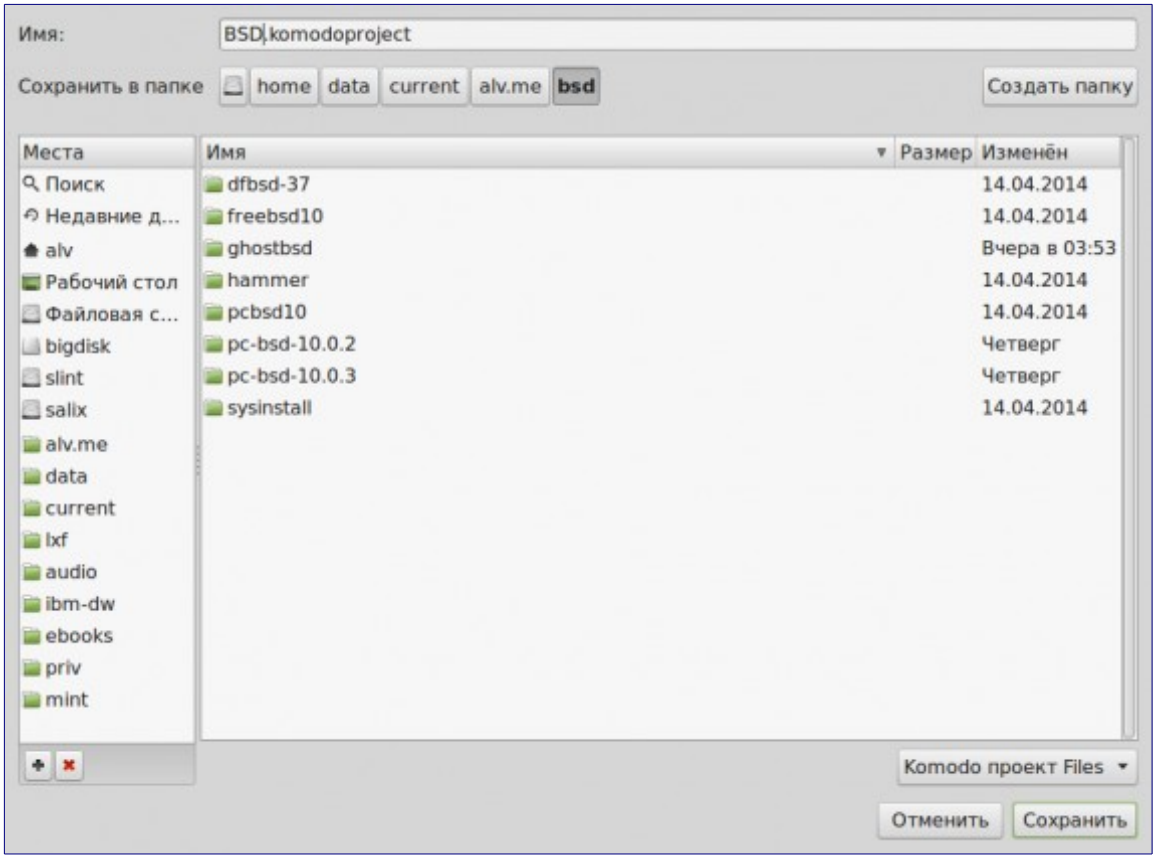


Выбранный один раз в данном документе, русский язык при проверке

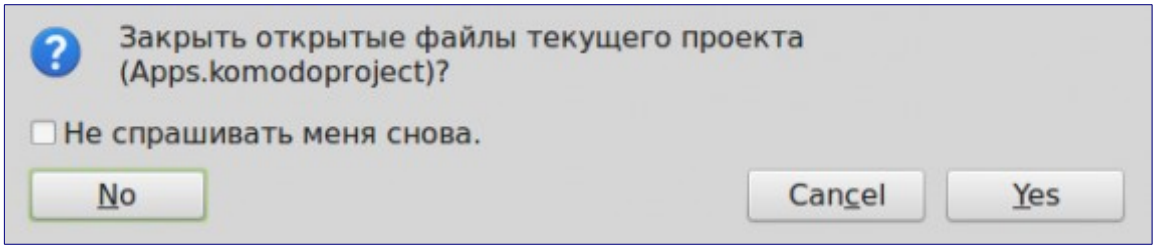
орфографии становится умолчальным не только в текущем проекте, но также и во всех открытых в данный момент проектах. Правда, подчёркивания ошибочно написанных слов в KE не предусмотрено. Что, с одной стороны, хорошо — не так рябит в глазах при обилии незнакомых словарю слов и иноязычных вкраплений. Но с другой — плохо, потому что... ну вы сами понимаете, почему.

Управление проектами

Следующий вопрос, живо меня интересующий — управление проектами. И здесь всё оказалось очень здорово. Во-первых, управление проектами в KE есть. Во-вторых, проект создаётся не просто, а очень просто (то есть проще даже, чем в Geany). Например, это можно сделать через меню Проект -> Новый проект, задав имя проекта и рабочий каталог для него:



Столь же легко создаются и все последующие проекты, сколько их потребуется. При этом задаётся вопрос, закрывать ли файлы текущего проекта? И, в отличие от Geany, вопрос этот не риторический, на него вполне можно ответить отрицательно, и новый проект будет создан при сохранении предыдущего в открытом состоянии:



В левой боковой панели легко выводится файловая иерархия внутри каталога текущего проекта (по умолчанию — до 10 уровней вложенности, при необходимости можно увеличить). Здесь же не менее легко открывается

подпанель со списком созданных проектов, через которую легко переключаться между ними. При этом, как и при создании нового проекта, файлы предыдущего проекта можно сохранить в открытом состоянии, параллельно с файлами проекта следующего. И переключаться между ними через вкладки главного окна, предназначенного для редактирования текста.

Возможность параллельной работы с файлами из разных проектов — та самая функция, которой мне так не хватало в Geany, ибо я постоянно работаю с несколькими самостоятельными, но тесно связанными проектами, между которыми необходим обмен данными. И KE мне такую возможность предоставил.

Макросы

Второй очень важный для меня момент связан с макросами. Для применителя-текстовика, работающегов разных жанрах, очень важно иметь возможность расширить базовую функциональность редактора в соответствие со своими задачами, причём сделать это простым и понятным способом. В Geany имеется удобный режим протоколирования макросов с возможностью их последующего редактирования. А если ли что-нибудь подобное в KE?

Оказалось, что в KE прибегать к сочинению собственных макросов поводов не много. Ибо он штатно снабжён их набором, вполне достаточным, например, для простой разметки html-документов, как в существующем виде, так и после минимального редактирования. Впрочем, и режим протоколирования действий для помещёния их в макросы также имеется, как и возможность последующего редактирования.

Почти нет ограничений для привязки к макросам горячих клавиш — при нахождении в окне KE им перехватывается большинство общесистмных клавиатурных комбинаций. Так что не надо, как в Geany, напрягать свою фантазию для изобретения хоткеев, с одной стороны, мнемонически прозрачных, с другой — не задействованных для нужд текущего десктопа или оконного менеджера.

Штатно в KE не обнаруживается статистики документов. Однако среди макросов нашёлся один, подсчитывающий количество слов с помощью утилиты wc. Правкой буквально одного символа можно заставить его считать число символов в документе или объём его файла в байтах.

Примеры простых макросов в KE можно умножать до бесконечности. Результаты своих упражнений я всёл в шпаргалку по этой теме. Она предназначена для внутреннего употребления, и главное в ней — это мнемоника для хоткеев, потому как они придумывались для разных редакторов в разное время, чисто ассоциативно на тот момент. То есть носят сугубо личный характер. Но возможно, что эта шпаргалка пригодится как напоминание о том, что каждый может сделать такую же для себя — со своей мнемоникой и своими ассоциациями.

Теги для выделения текста:

- Code — ввод моноширинного текста, хоткей Alt+m (от Monospace);
- Emphasis — ввод курсивного шрифтоначертания, хоткей Alt+i (от

Italic);

- Strong — ввод полужирного шрифтоначертания, хоткей Alt+b (от Bold);
- Strike — ввод зачёркнутого выделения, хоткей Alt+s (от Strike).

Теги для текстовых блоков:

- Pre — командные конструкции, скрипты etc., хоткей Alt+p (от Pre);
- Quote — цитаты, хоткей Alt+c (от Citata);
- List Or — нумерованный список, хоткей Alt+o (от Ordered);
- List Un — маркированный список, хоткей Alt+l (от List).

Теги для ссылок:

- URI Link — ссылка внутри сайта, хоткей Alt+u (от URL);
- URI Remote — ссылка вне сайта, хоткей Alt+r (от Remote);
- Name Anch — анчор внутри страницы, хоткей Alt+n (от Name);
- Name URI — ссылка на внутренний анчор, хоткей Alt+a (от Anchor).

Разные прочие теги:

- Doctype — вставка Doctype, "lang=ru-RU", "content=text/html; charset=UTF-8", хоткей Alt+d (от Doctype);
- Count — подсчёт в выделении количества строк, слов, байт и символов командой wc -lwmc, хоткей Alt+w (от Wc).

А также всё, что потребуется впредь.

В заключение хотел бы напомнить, что Alt-последовательности для ввода макросов работают только при латинской раскладке клавиатуры. Однако постоянного переключения с кириллицы на латиницу можно избежать, определив одну из «удержальных» клавиш, например, Right Control.

Настройки

В KE меются разные режимы выделения текста — всего, последовательными фрагментами, блоками, между парными скобками. И режим множественного выделения, когда выделяются не последовательные фрагменты, а куски из произвольных мест документа. Которые потом можно скопировать и вставить в другой документ — возможность, незаменимая для тех, кому часто приходится составлять компиляцию из разных материалов. И возможность эта если и не уникальна, то встречается не часто. Мне так, например, раньше не встречалась никогда и нигде.

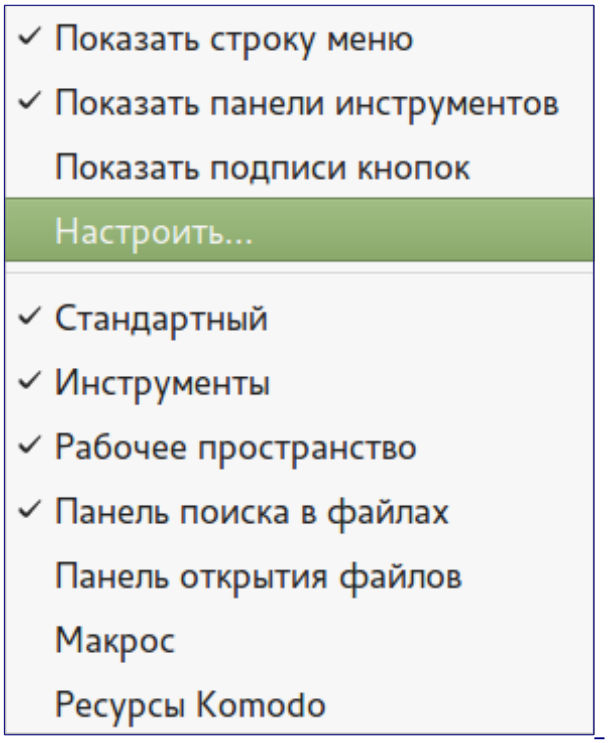
Функции поиска и замены работают как для отдельных файлов и их выделенных фрагментов, так и для каталогов, а также целых проектов. Кроме того, имеется последовательный наращиваемый поиск, как в браузерах.

Результаты поиска, в том числе и последовательного, по умолчанию подсвечиваются в течении заданного в настройках времени, которое может быть изменено в любую сторону. Не запрещается и вообще отключить режим подсветки при поиске.

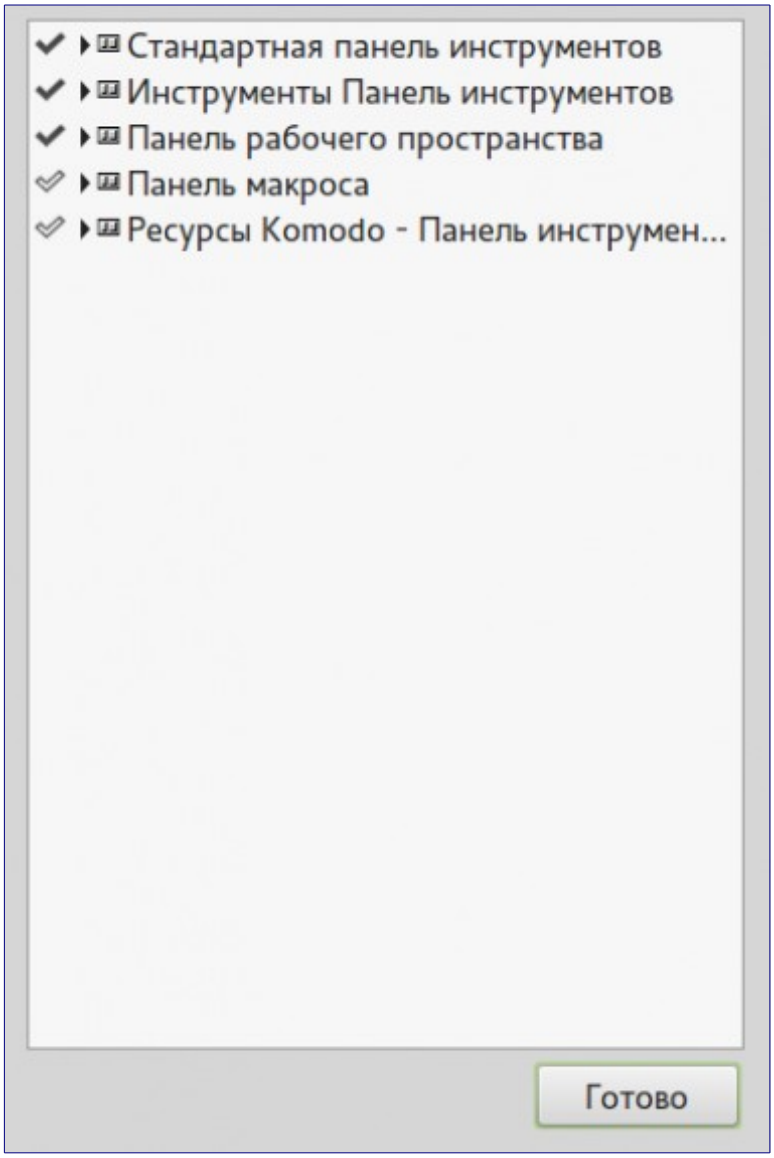
В отношении настройки клавиатурных комбинаций для штатных действий KE (не макросов) фантазия применителя ограничивается только объёмом памяти — не компьютерной, а собственной, сколько хоткеев она в состоянии запомнить: как известно, слишком хорошо — это тоже не хорошо. Но одной из возможностей этого круга я воспользовался немедленно.

Наверное, я далеко не единственный, у кого при быстром лабании по клавишам (а лабать медленно я не умею) часты перестановки соседних символов. В Kate для исправления этого (то есть для транспонирования соседних буковок) существует штатная комбинация — Control+T. Это была та фича, по которой я проливал горючие слёзы, работая в Geany — там возможности траспонирования не имеется. А вот в KE она обнаружилась — и я немедленно присвоил ей ту же последовательность, что и в Kate.

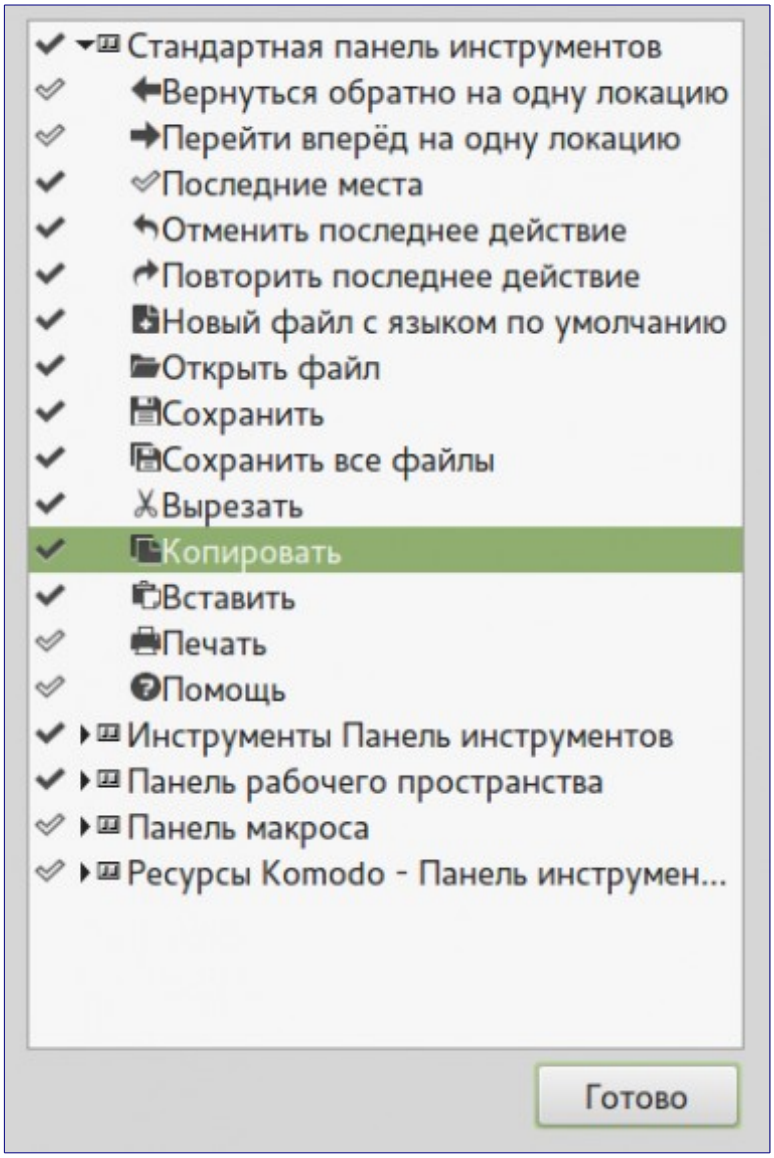
Особо следует сказать о настройке инструментальной панели. При первом запуске в KE в ней можно видеть пиктограммы для выполнения некоего набора действий, которые разработчики отнесли к числу наиболее частых. Однако мнение применителя на этот счёт может быть более иным. Учитывая это, разработчики предоставили достаточно возможностей для настраивания инструментальной панели. Чтобы ими воспользоваться, достаточно правого клика на панели и выбора пункта Настройка из появившегося контекстного меню:



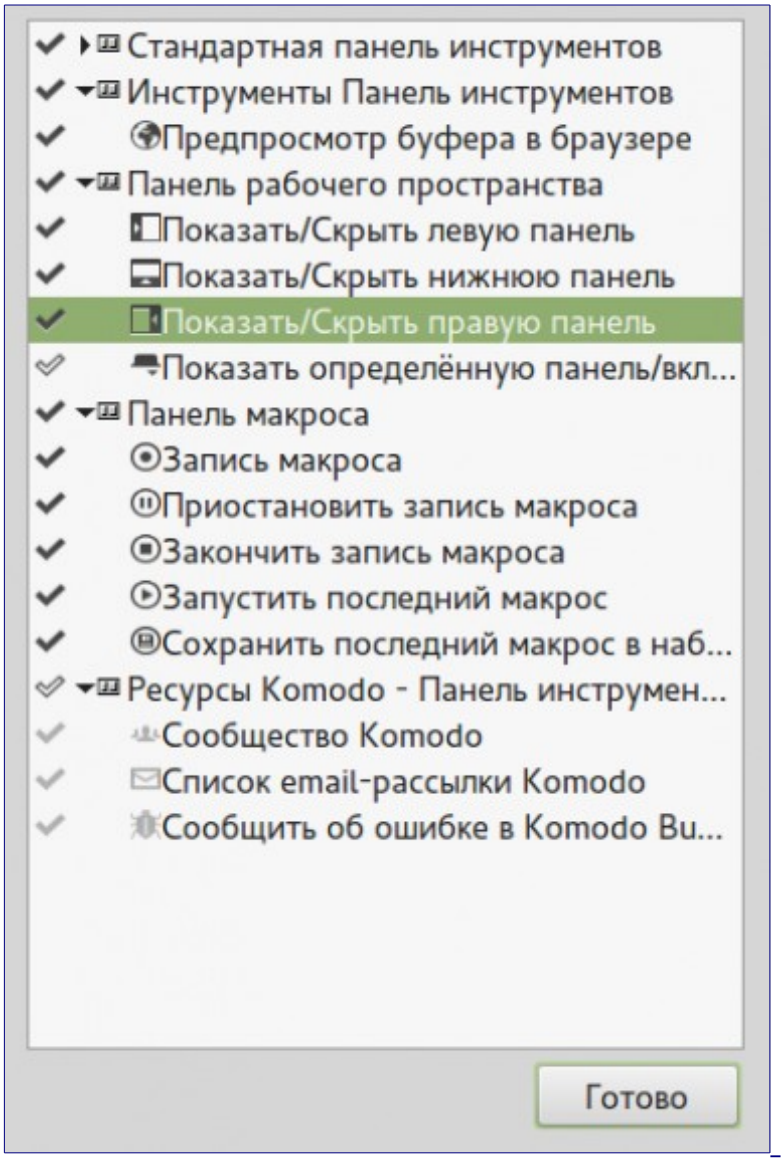
Панель настроки инструментов выглядит так:



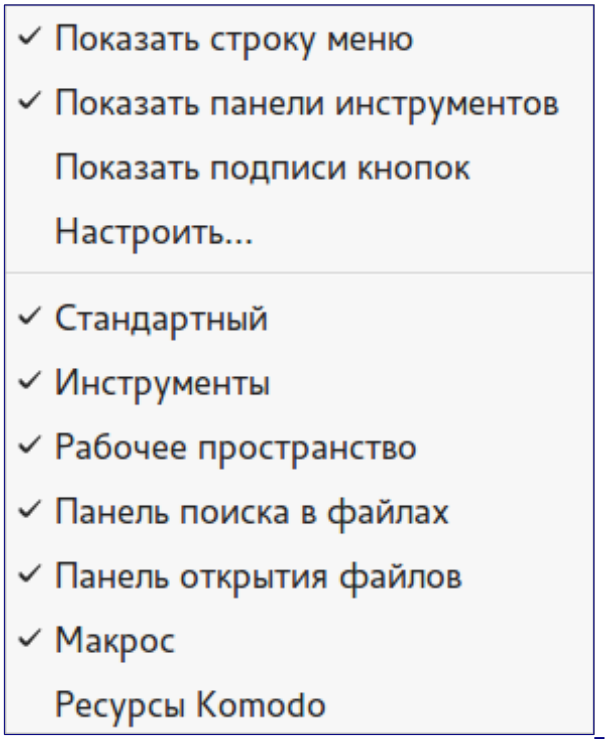
И говорить зедсь особо нечего — проще показать на скриншотах, как настроил инструменты я. Опять таки не как призыв к обезьяньичанию, просто как пример возможных вариантов. Как нетрудно догадаться, чёрные «галки» — это включённые опции, белые «вороны» — отключённые. Так что поехали по порядку — начиная со Стандартной панели инструментов:



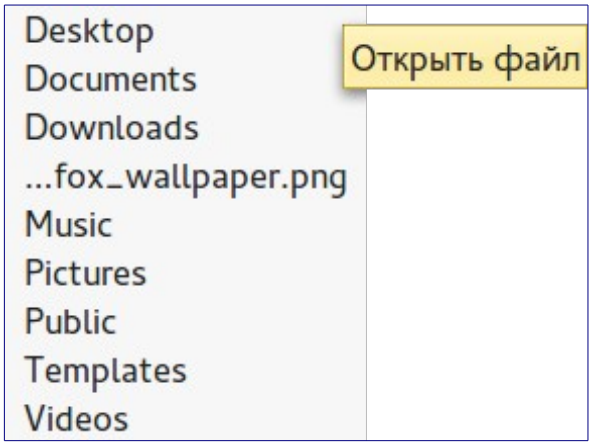
Далее я расправился с остальными панелями:



Объяснять, почему я сделал так, а не иначе, полагаю излишним — всё равно каждый применитель перекроит наборы пиктограмм на всех панелях по своему. После чего не лишне будет вернуться к контекстному меню инструментальной панели и включить панель открытия файла

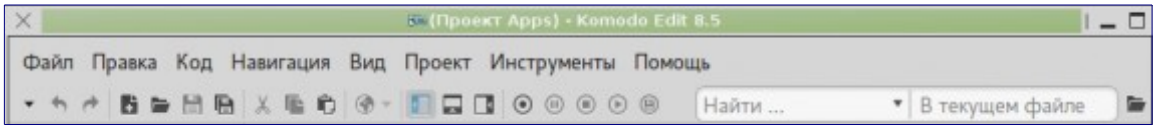


Если одноимённая пиктограмма Стандартной панели вызывается стандартное же окно с файловым древом, то здесь появляется выпадающее меню со списком подкаталогов домашнего каталога пользователя, определённого стандартом freedesktop.org: Desktop, Documents, Download и так далее:

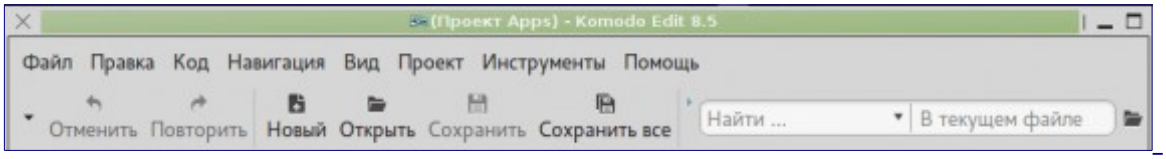


После выбора и нажатия на Enter соответствующая ветка появится в левой боковой панели главного окна KE. Впрочем, поскольку в моём домашнем каталоге никаких данных не хранится, для меня это оказалось излишеством.

После всех показанных манипуляций моя панель инструментов приобрела такой вид:

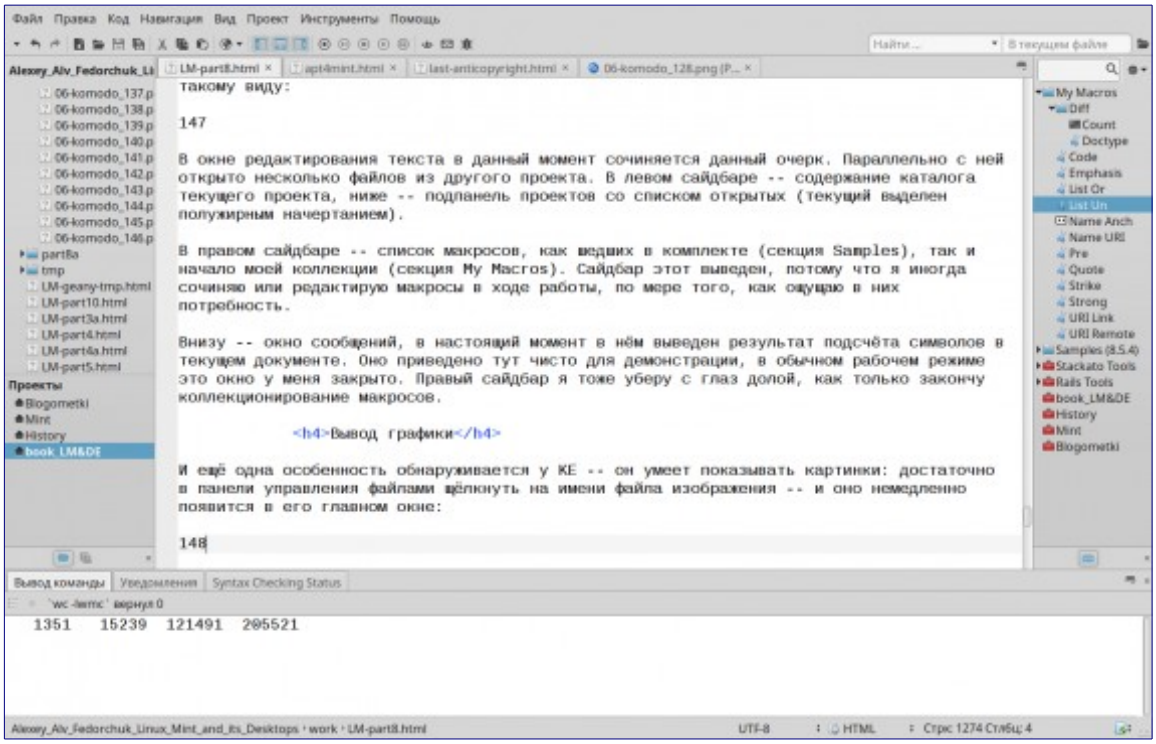


При желании можно включить и подписи к иконкам, однако я полагаю это не нужным: в результате на панели может просто не хватить места:



Тем более что всплывающих при наведении на пиктограмму подсказок даже по первости достаточно, а потом приходит автоматизм.

В общем, выполнив некоторые настройки (далеко не все возможные), я привёл KE вот к такому виду:



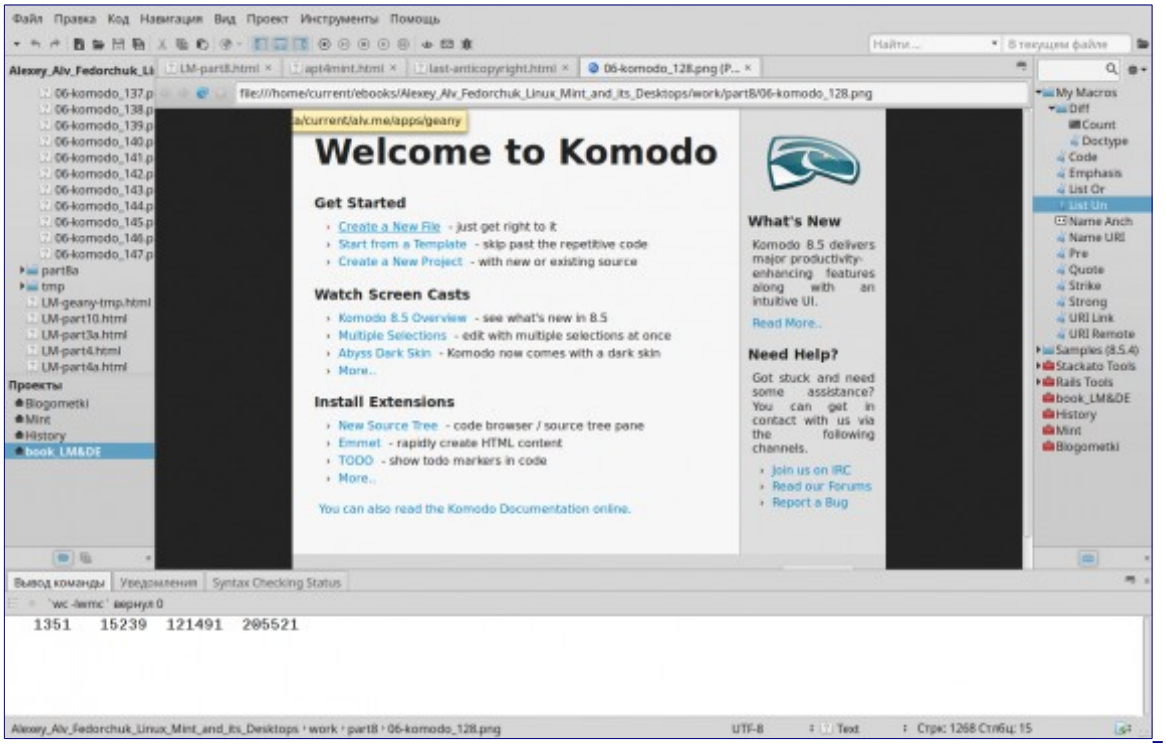
В окне редактирования текста в данный момент сочиняется данный очерк. Параллельно с ней открыто несколько файлов из другого проекта. В левом сайдбаре — содержание каталога текущего проекта, ниже — подпанель проектов со списком открытых (текущий выделен полужирным начертанием).

В правом сайдбаре — список макросов, как шедших в комплекте (секция Samples), так и начало моей коллекции (секция My Macros). Сайдбар этот выведен, потому что я иногда сочиняю или редактирую макросы в ходе работы, по мере того, как ощущаю в них потребность.

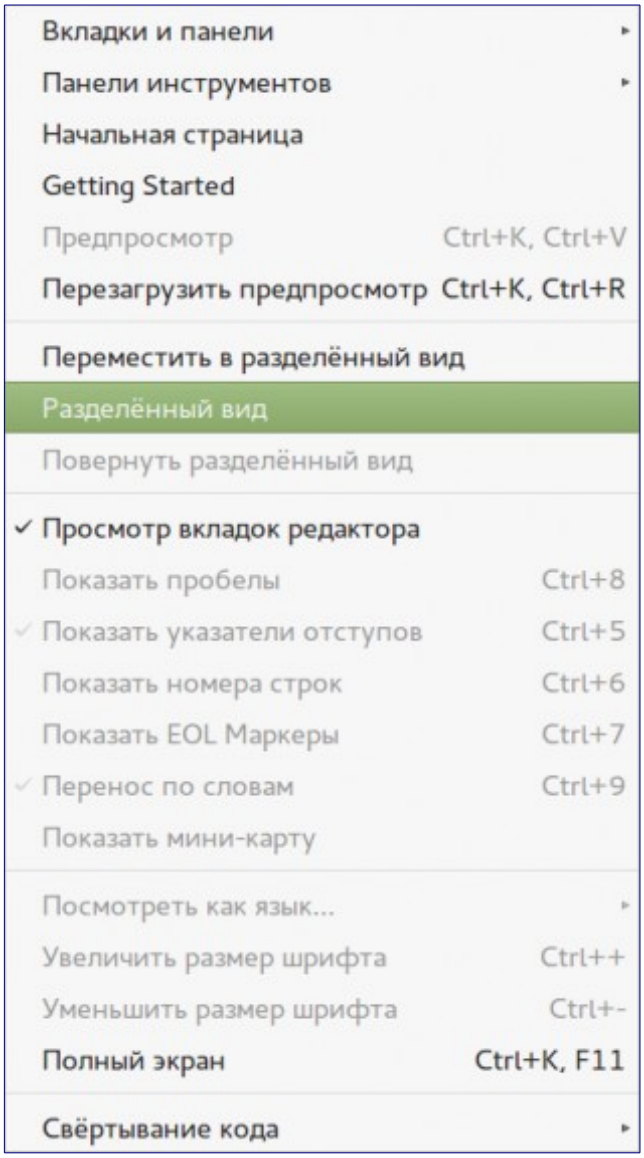
Внизу — окно сообщений, в настоящий момент в нём выведен результат подсчёта символов в текущем документе. Оно приведено тут чисто для демонстрации, в обычном рабочем режиме это окно у меня закрыто. Правый сайдбар я тоже уберу с глаз долой, как только закончу коллекционирование макросов.

Вывод графики

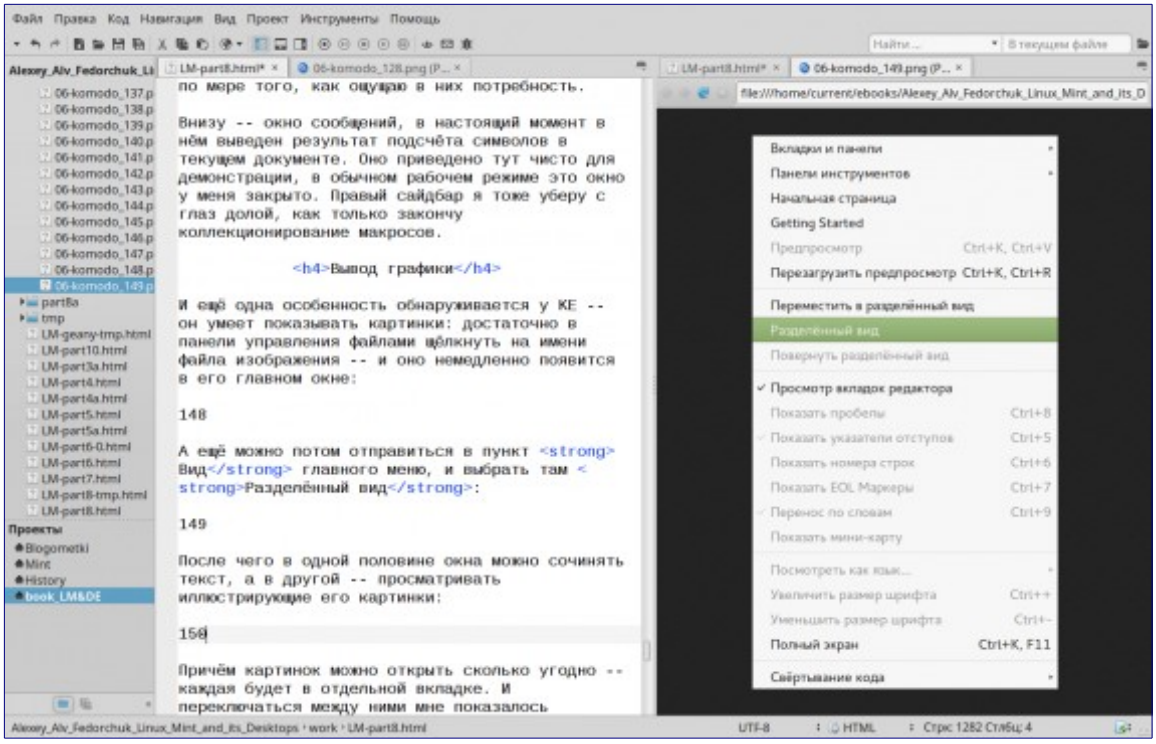
И ещё одна особенность обнаруживается у KE — он умеет показывать картинки: достаточно в панели управления файлами щёлкнуть на имени файла изображения — и оно немедленно появится в его главном окне:



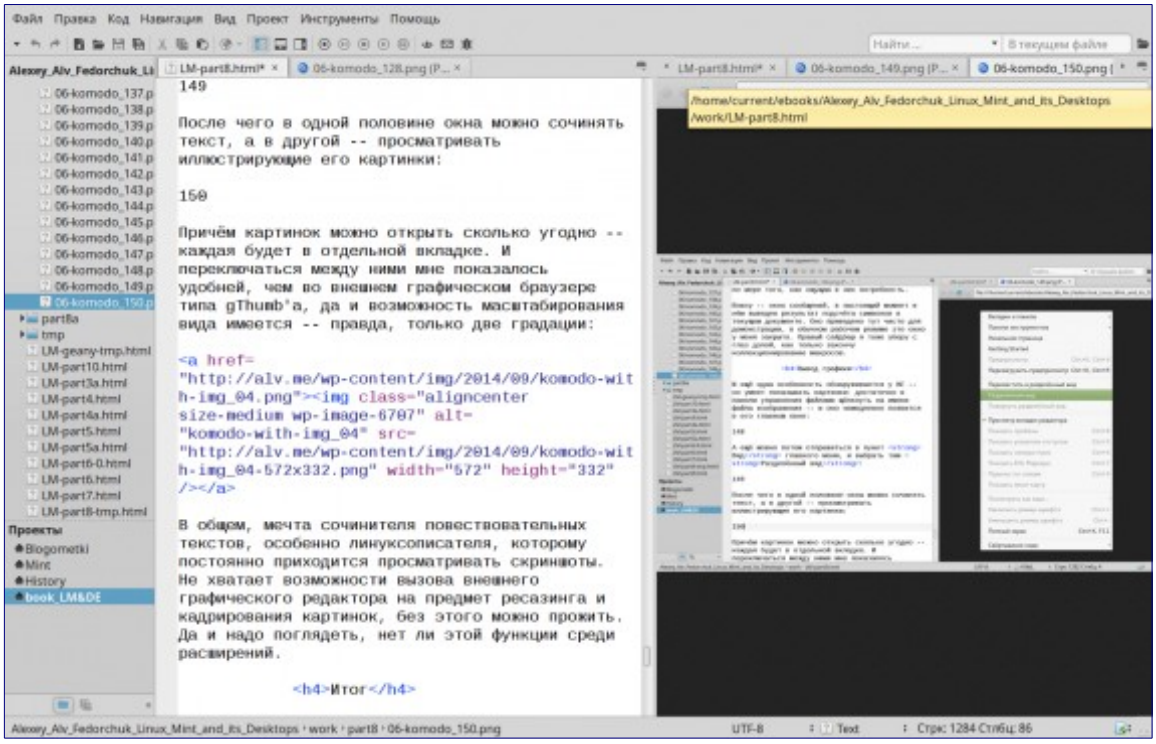
А ещё можно потом отправиться в пункт Вид главного меню, и выбрать там Разделённый вид:



После чего в одной половине окна можно сочинять текст, а в другой — просматривать иллюстрирующие его картинки:



Причём картинок можно открыть сколько угодно — каждая будет в отдельной вкладке. И переключаться между ними мне показалось удобней, чем во внешнем графическом браузере типа gThumb'a, да и возможность масштабирования вида имеется — правда, только две градации:



В общем, мечта сочинителя повествовательных текстов, особенно линуксописателя, которому постоянно приходится просматривать скриншоты. Не хватает возможности вызова внешнего графического редактора на предмет ресазинга и кадрирования картинок, без этого можно прожить. Да и надо поглядеть, нет ли этой функции среди расширений.

И последнее о Komodo Edit

Наговорив столько о достоинствах KE, было бы несправедливо не отметить и недостатки. По большому счёту таковых для меня обнаружилось полтора. Первый — отсутствие встроенного терминала, к которому я привык и в Kate, и в Geany. Впрочем, поскольку я всегда устанавливаю терминал выпадающий, оказалось, что с этим вполне можно примириться, вызывая последний горячей клавишей (по умолчанию F12). Ну а полнедостка — отсутствие подсветки слов с ошибками: как было сказано ранее, считать это однозначно недостатком не очень правильно (с моей точки зрения, разумеется).

Да, ещё, как говорят, KE показывает себя довольно тормознутым на слабых машинах. В силу того, что у меня слабых машин не водится, я этого не заметил. Но учитывать это наблюдение следует.

Заключение по редакторам

Пора подвести итог по текстовым редакторам. Если возможностей Gedit'a для работы с текстами не хватает — и Geany, и Komodo Edit будут более чем достойными ему альтернативами. Какая из них лучше? Однозначного мнения у меня не сложилось. В пользу первой альтернативы — лучшая поддержка спеллинга, более высокое быстродействие на слабых машинах и, главное, наличие встроенного терминала. За KE — более развитые средства управления файлами (в том числе и не только текстовыми) и проектами, в частности — возможность держать параллельно несколько открытых проектов. Ну и практически неограниченные возможности настройки и расширения функционала за счёт собственных макросов нельзя сбрасывать со счёта.

В результате достоинства KE лично для меня перевесили его недостатки, и в назначил его своим основным инструментом для работы с текстами. Сохранив, однако, тёплые чувства к Geany, много лет прослужившим мне верой и правдой. Почему выше этот редактор и был описан столь подробно — к аналогичному описанию KE я ещё морально не готов.

Графика

Сочиняемые тексты нередко принято иллюстрировать. А потому программы для работы с графикой почти так же необходимы, как и текстовый редактор. И программы эти разделяются на три части: средства получения изображений, средства их просмотра и инстументы для редактирования.

Для большинства из нас, не являющихся художниками и даже рисовальщиками, источниками изображений служат фотографии, сканограммы и скриншоты (если исключить потибренное из Интернета, разумеется). Полученные изображения нуждаются в просмотре, иногда — обработке, а при большом количестве — и в каталогизации.И в Cinnamon-редакции Mint есть штатные средства для всех этих операций, а в репозиториях — и альтернативы им.

Всё изобилие доступных в Mint графических программ я рассматривать не буду. Так, снимаю я мало и плохо, так что о программах, связанных с фотографией, говорить не буду. Файлов изображений у меня довольно много, однако подавляющее их большинство иллюстрируют тексты, являются частями соответствующих проектов, и потому в средствах каталогизации не нуждаюсь. Ну а про штатно устанавливаемый при стандартной инсталляции

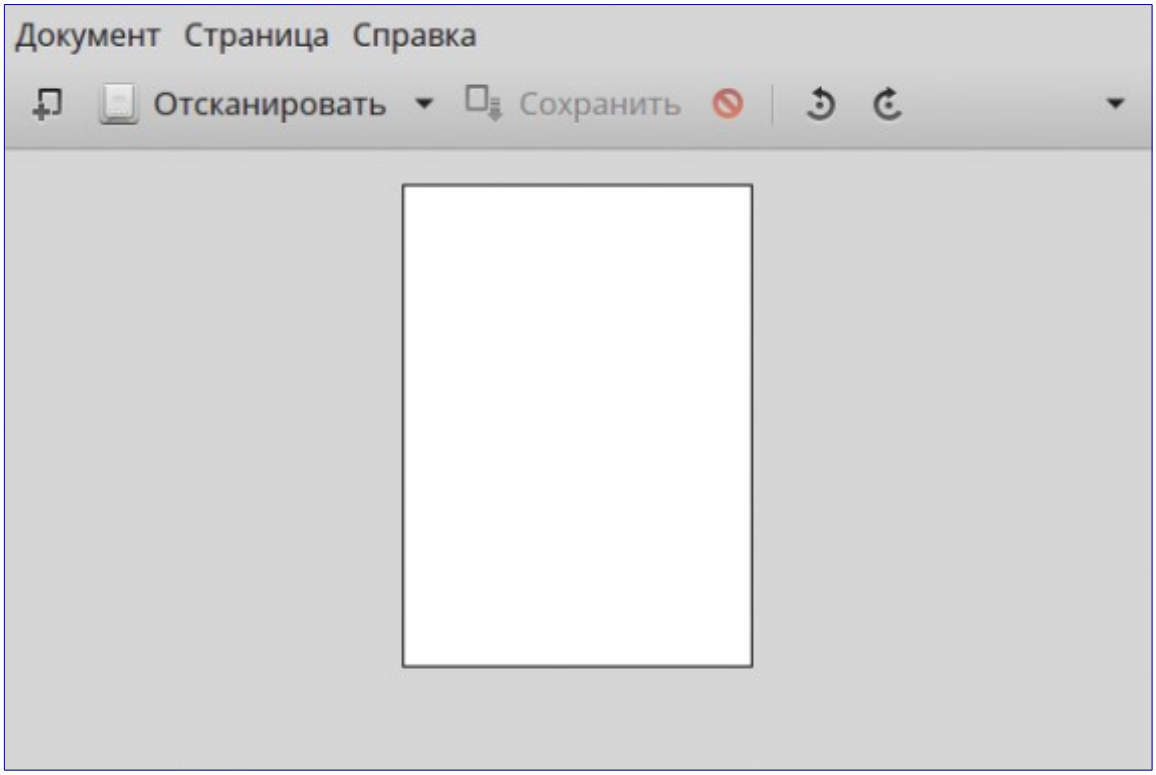
GIMP можно говорить или хорошо (то есть много), или ничего. По причине не актуальности для меня этой программы останавливаюсь на втором варианте.

Так что далее речь пойдёт о средствах сканирования, создания скриншотов, просмотра изображений и инструментах для элементарного редактирования.

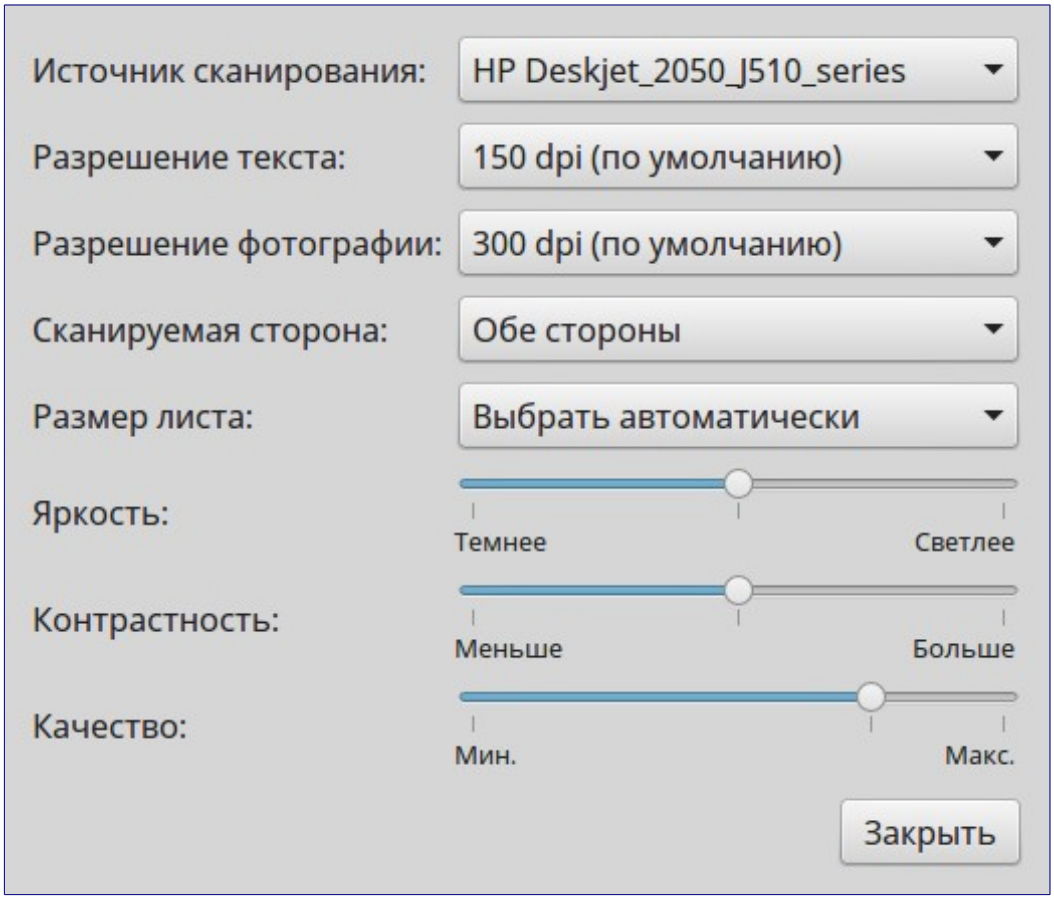
Средства сканирования: Simple Scan

Когда заходит разговор о сканировании под Linux, в первую очередь вспоминается программа sane и её графический фронт-энд xsane. Однако в Mint ни та, ни другая не устанавливаются по умолчанию, хотя и доступны в репозитории. Вместо этого в секции Графика можно обнаружить пункт Простое сканирование, под которым скрывается программа Simple Scan.

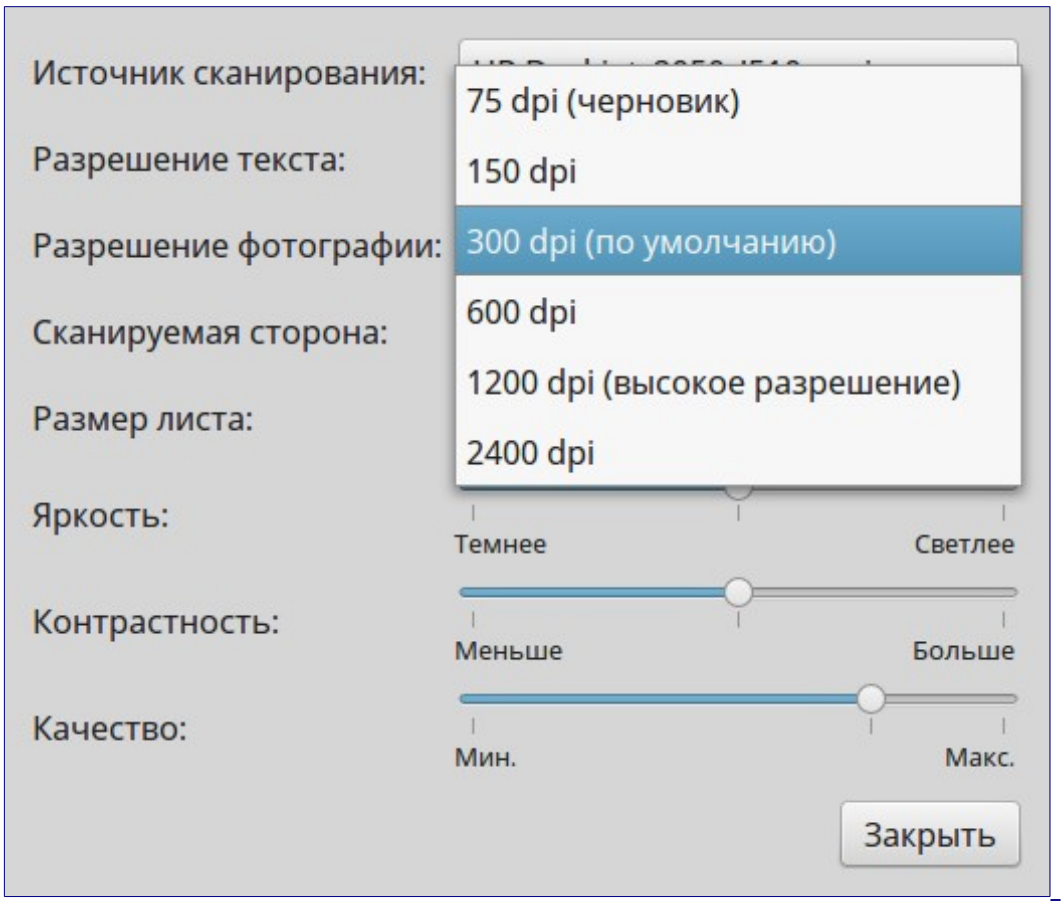
Simple Scan — самостоятельная программа, а не «морда» для sane, хотя и основана на той же библиотеке libsane, что и последняя. И она действительно оправдывает своё имя — более простой инструмент для сканирования трудно себе представить. При первом запуске она выводит такое вот окно:



А также предлагает настроить сканер. Для сканеров и МФУ производства Hewlett-Packard это не доставляет никаких проблем — при установленной системе HPLIP (HP Linux Imaging and Printing), которая в Mint имеется по умолчанию: нужно просто соглашаться со всем, что она предложит. После чего в панели настроек (вызываемой через меню Документ -> Параметры) обнаруживается соответствующее устройство. Вмоём случае это МФУ HP Deskjet 2050:



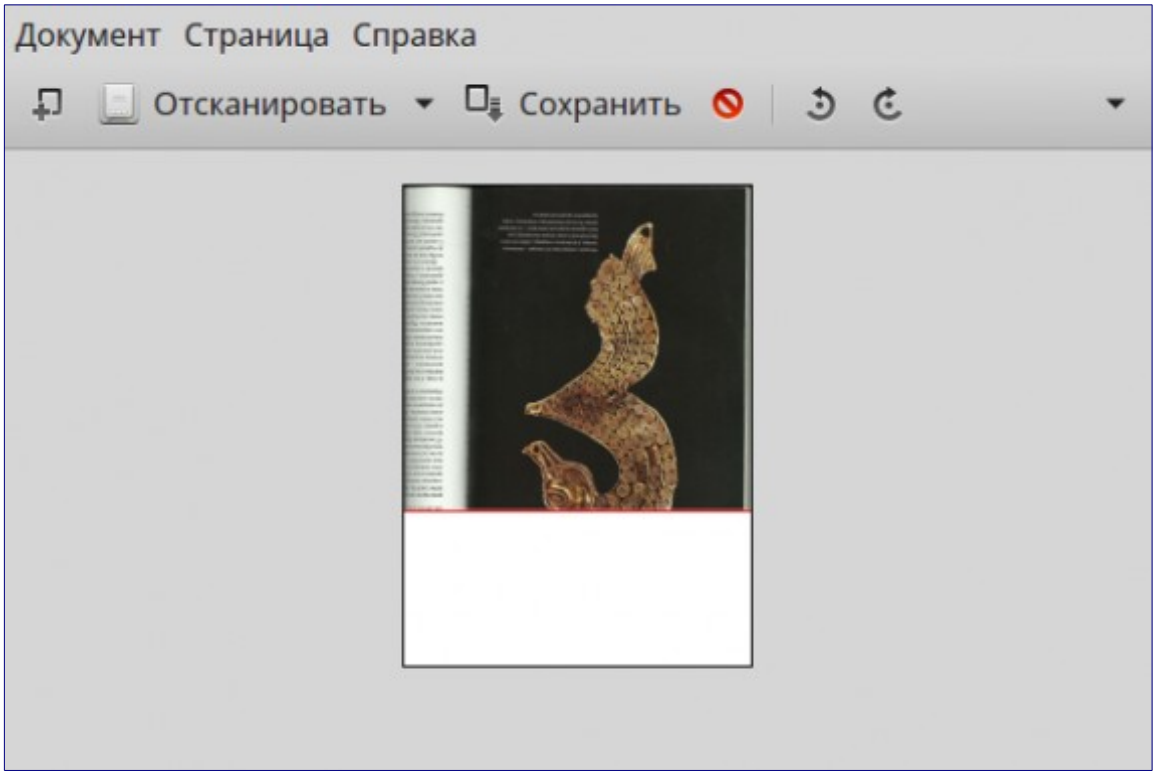
В этой же панели можно поменять и некоторые другие параметры, например, разрешение сканирования текста и фоток — у меня от 75 dpi до 1200 (физических) и 2400 (интерполированных):



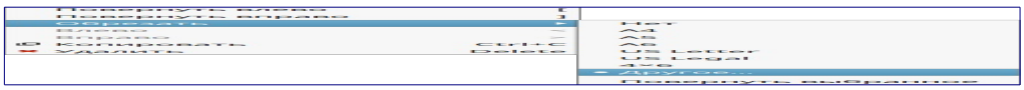
Впрочем, я оставил все параметры по умолчанию.

Правда, повторяю, всё это относится к устройствам от HP — как обстоит дело со сканерами и МФУ более иных производителей, просто не знаю.

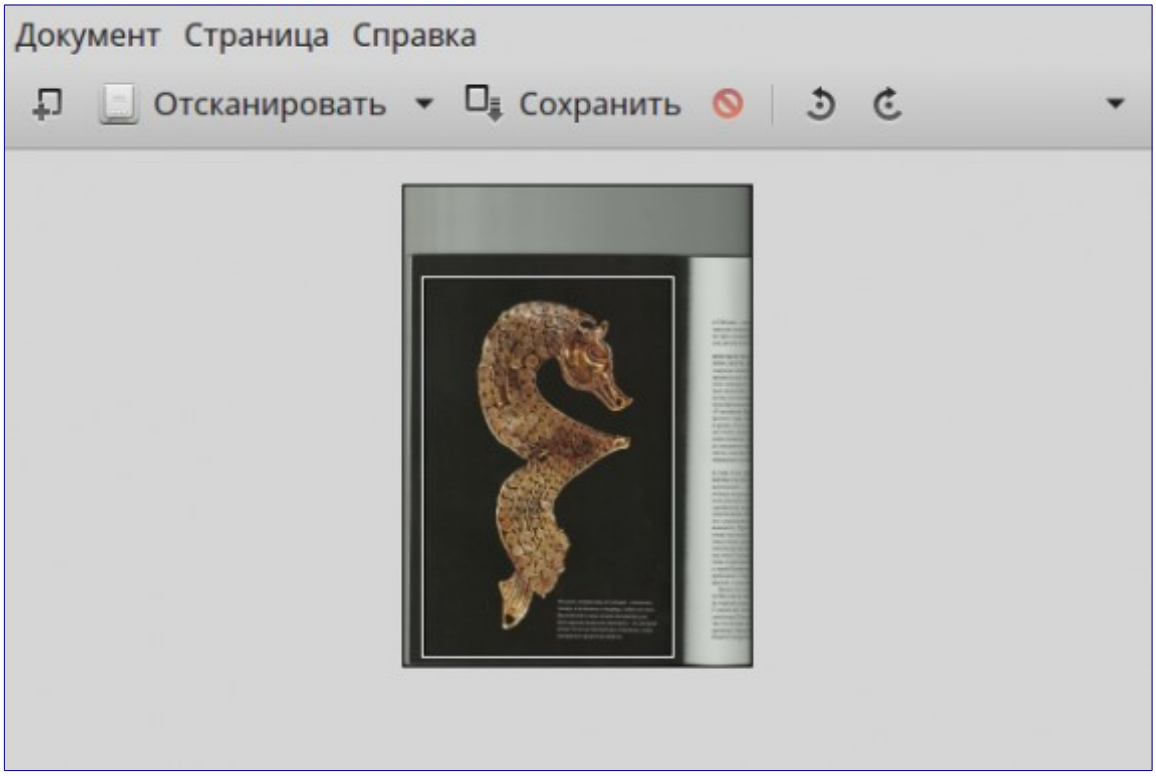
Теперь можно начинать сканирование, для чего достаточно нажать кнопку с соответствующей надписью. Процесс сканирования отображается на экране:



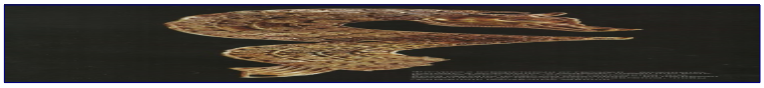
По завершении сканирования результат выводится в виде полной страницы. Разумеется, это дело надо сохранить. Но прежде изображение можно подкорректировать — в приведённом примере повернуть по часокй стрелке на 180°, а также обрезать лишнее через меню Страница -> Обрезать -> Другое:



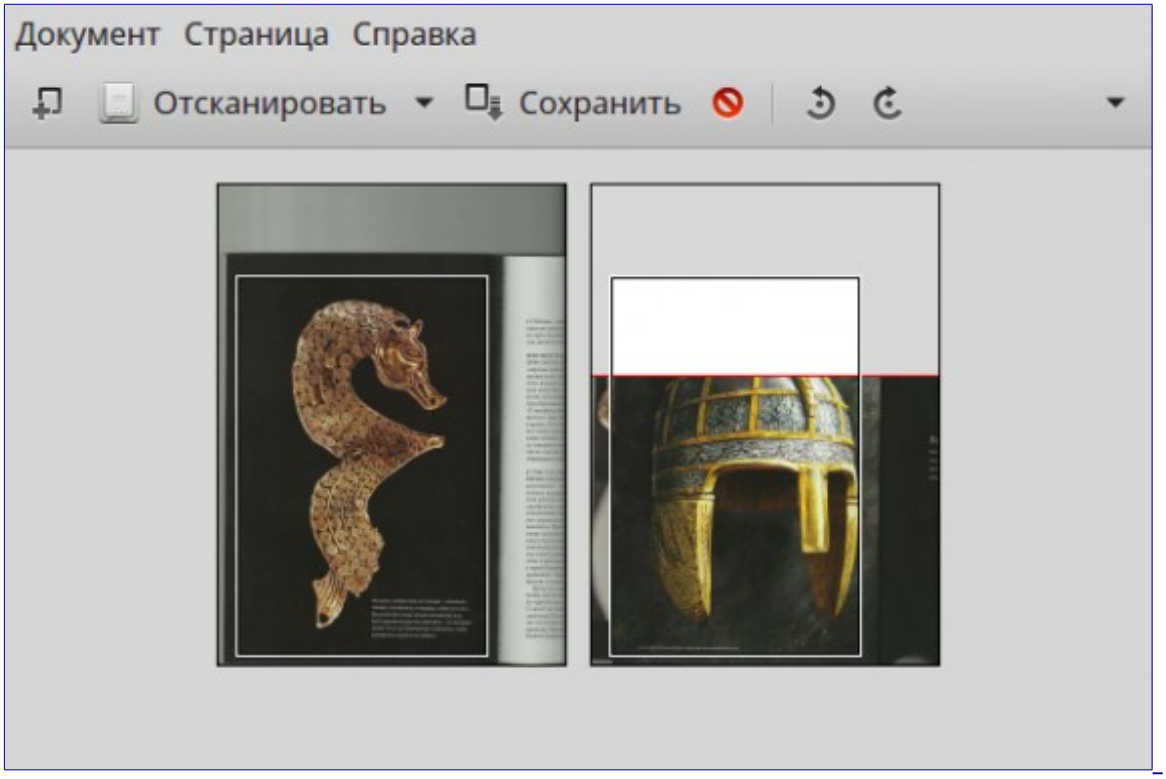
Что делается просто подгонкой рамки:



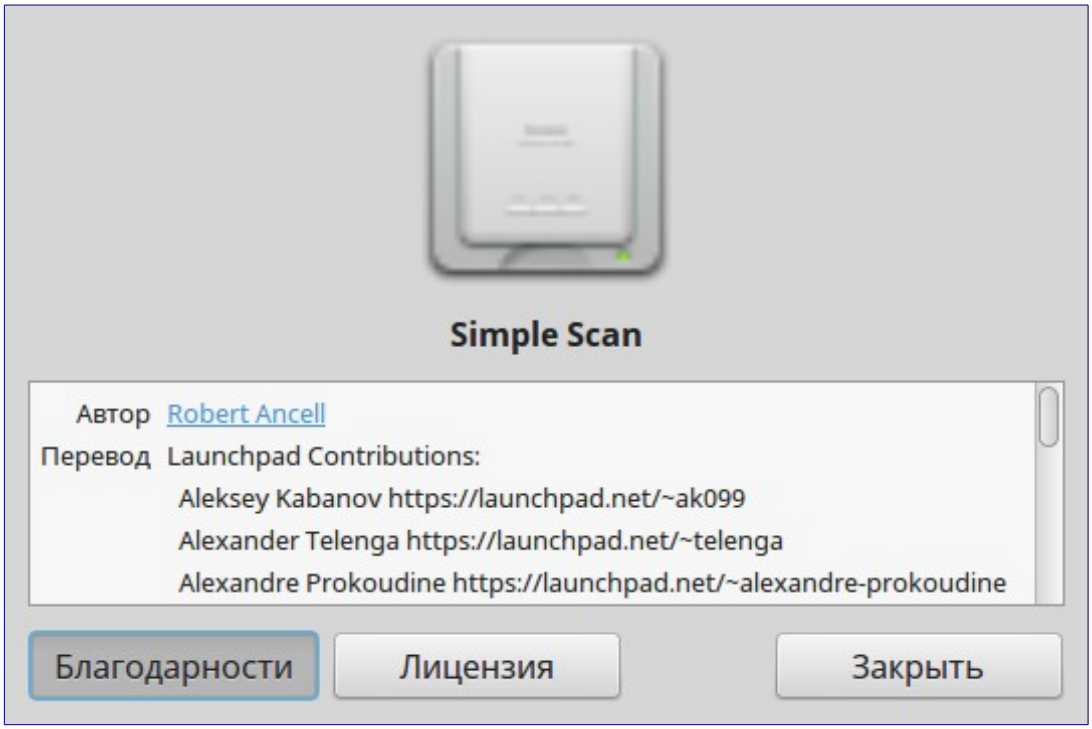
Теперь после сохранения картинка приобретёт следующий вид:



Если теперь отсканировать другие изображения — они получат то же имя, но с добавлением порядкового номера.



Вот и всё. К сказанному остаётся добавить, что программа Simple Scan была написана специально для Ubuntu Робертом Анселлом (Robert Ancell), а список переводчиков интерйса на русский язык насчитывает немало имён:



И в настоящее время программа эта включается в штатный состав большинства популярных дистрибутивов.

Создание экранных снимков

Поскольку каждому практикующему линуксописателя делать экранные снимки подчас приходится в массовых количествах (десятками, а иногда и сотнями), то к скриншоттеру предъявляются довольно жёсткие требования не только в плане функциональности, но и в отношении удобства.

Вступление

С функциональностью всё понятно: скриншоттер должен позволять делать снимки «фиксированных» элементов — всего экрана, отдельного окна, произвольной области экрана или окна. Причём как мгновенно, так и с задержкой, и время её должно поддаваться изменению. Почему? Да потому, что часто важно «снять» элементы динамические — всплывающие подсказки, выпадающие и контекстные меню, или отдельные их фрагменты. Так что надо иметь запас времени, дабы докопаться до нужного элемента многоуровневого меню, и время это в разных случаях разное. Очень существенно также иметь возможность назначить «снимаемый» элемент по умолчанию — и в большинстве случаев это бывает активное окно.

Что же касается удобства — то это в первую очередь условия сохранения получаемых файлов изображений. То есть должна быть лёгкая возможность изменения целевого каталога для файлов экранных снимков, например path2/article_name/. И, безусловно, возможность логичного автоматического именования скриншотов, типа: article_name01_001.png и так далее. Не худо иметь и возможность хотя бы простенького управления созданными файлами — как минимум, переименования и удаления.

Функции удобного просмотра изображений и их простого редактирования (кадрирование, изменение размера, конвертации в другие форматы) также не лишние, но не обязательны. Кстати, из форматов файлов, как мне кажется, актуально полтора: упомянутый png и, изредка, jpeg. Форматы типа bmp

полагаю атавизмом, а необходимость в tiff'е отпала с тех пор, как «бумажные» редакции стали спокойно принимать png.

Так вот, исходя из сформулированных требований (моих, разумеется, все от них отличные — не правильны), на протяжении многих лет лучным скриншоттером я считал штатный Ksnapshot из KDE. И, если говорить именно о программах, входящих в комплект таких десктопов, как GNOME и Xfce, то мнения своего я не изменил: ни gnome-screensot, ни xfce-screenshot до него не дотягивают по всем параметрам.

Однако в Cinnamon-редакции Mint ни малейшего Ksnapshot'а штатно, разумеется, нет, а доустанавливать его не имеет смысла — тогда уж проще переходить на KDE-редакцию. Так что и тут нужно искать альтернативу — не делать же скриншоты, с помощью GIMP'а. С другой стороны, предлагаемые от безрыбья консольные инструменты типа scrot или fbshot — это уже другая крайность.

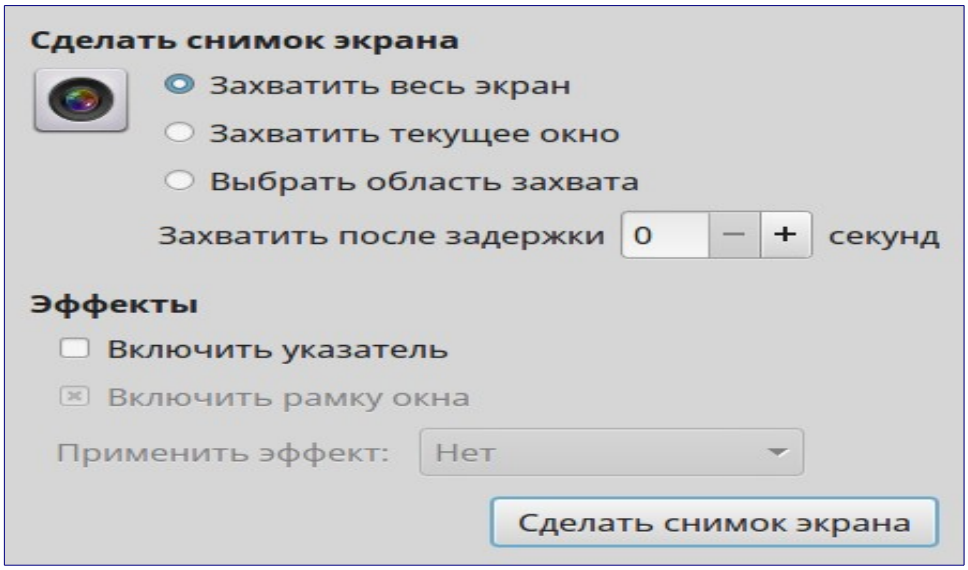
GNOME Screenshot

Однако, прежде чем заниматься поисками внештатных альтернатив, кратко рассмотрим возможности штатного GNOME Screenshot'а — ведь на первых порах приходится прибегать к его помощи. Ибо, как известно, на первоначальном бесптичье и место пониже спины — соловей.

Запустить gnome-screensot можно из секции меню Стандартные — он называется там Снимок экрана. Хотя можно обойтись и без меню: по умолчанию gnome-screensot запускается горячими клавишами — PrintScreen (снимок всего экрана), Alt+PrintScreen (снимок активного окна) или Shift+PrintScreen (снимок выделенной области).

Однако это не очень удобно: во всех трёх случаях по умолчанию скриншоты норовят записаться в каталог $HOME/Pictures, а если изменить путь к целевому каталогу вручную, то при следующем запуске скриншоттера горячими клавишами всё равно восстановится умолчальный путь.

Кроме того, при запуске через пиктограмму на панели задач, кроме снимка всего экрана, активного окна и выделенной области можно просто запустить программу в, так сказать, «общем виде»:



Обратите внимание на последний скриншот: на нём по умолчанию отмечена

опция Захватить весь экран. И если для текущего снимка изменить её на любую из двух других — при следующей запуске она вернётся в качестве отмеченной по умолчанию. Эта мелочь страшно раздражает: ведь обычно нужно снимать не экран, а одно из окон, реже — выделенную область, но сделать любую из этих опций умолчальной не получится. Второй раздражающий фактор — неудобство изменения целевого каталога. Ну а уж про именование файлов по типу Снимок экрана от 2013-07-23 22:57:04.png и говорить нечего. Оказалось, что практически в моих целях gnome-screensot можно использовать только в паре с какой-либо утилитой массового переименования. Благо в Mint я такую откопал в лице gprename, но это тема другого очерка.

В общем, поразвлекавшись с gnome-screensot некоторое время и изрядно оживив в памяти свой запас тюркской обсценной лексики, я решил поискать что-нибудь более приличное среди приложений, оставшихся за кадром штатной инсталляции Ubuntu. И, разумеется, как всякий ищущий да обрёл таковое — программу Shutter, о которой речь пойдёт в следующем миниочерке.

Shutter

Программа Shutter имеется в официальном репозитории Mint (точнее, конечно же, Ubuntu), так что доступна для установки любым стандарным методом — от

```
$ apt install shutter
```

до Synaptic'а и Менеджера программ.

Описание Shutter'а, выдаваемое командой

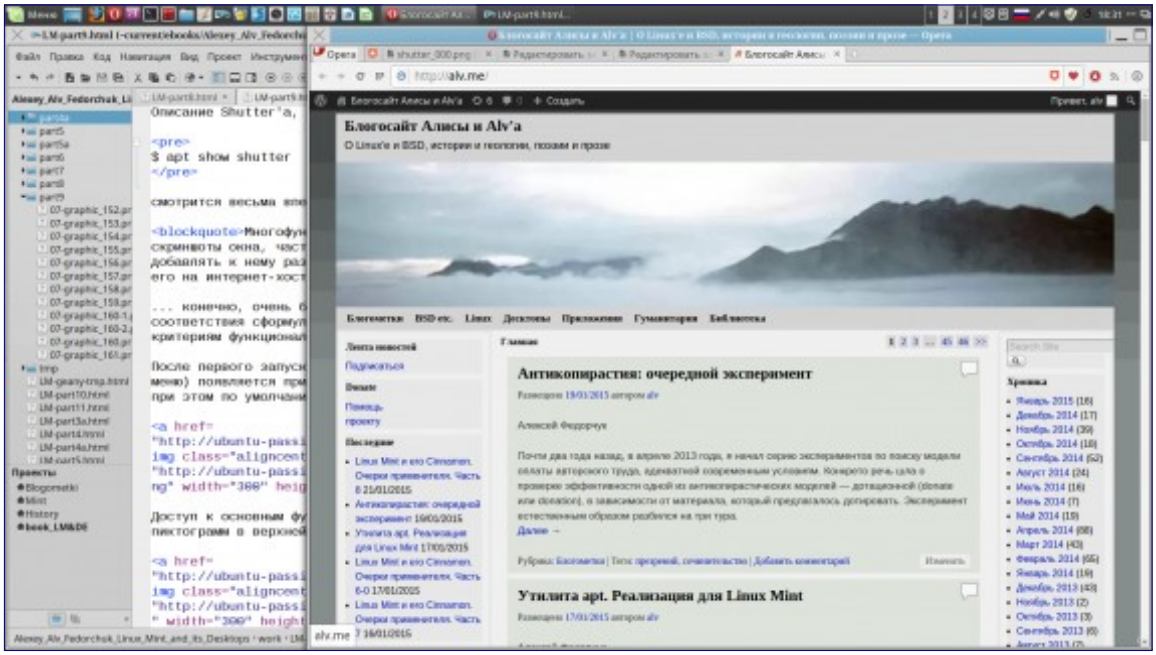
```
$ apt show shutter
```

смотрится весьма впечатляюще:

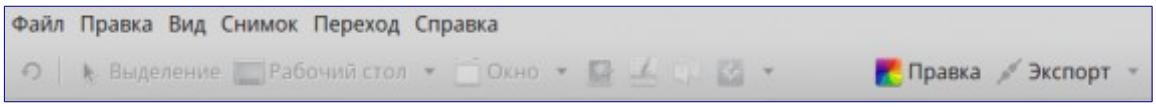
Многофункциональная программа, позволяющая делать ... скриншоты окна, части эрана, всего экрана, или даже веб-сайта, потом добавлять к нему различные эффекты, рисовать на нём, и в конце загрузить его на интернет-хостинг изображений. И всё это...

... конечно, очень благородно, но как в нём на счёт ~~баб~~ соответствия сформулированным ранее требованиям линуксописателя? То есть критериям функциональности и удобства. Начнём с функциональности.

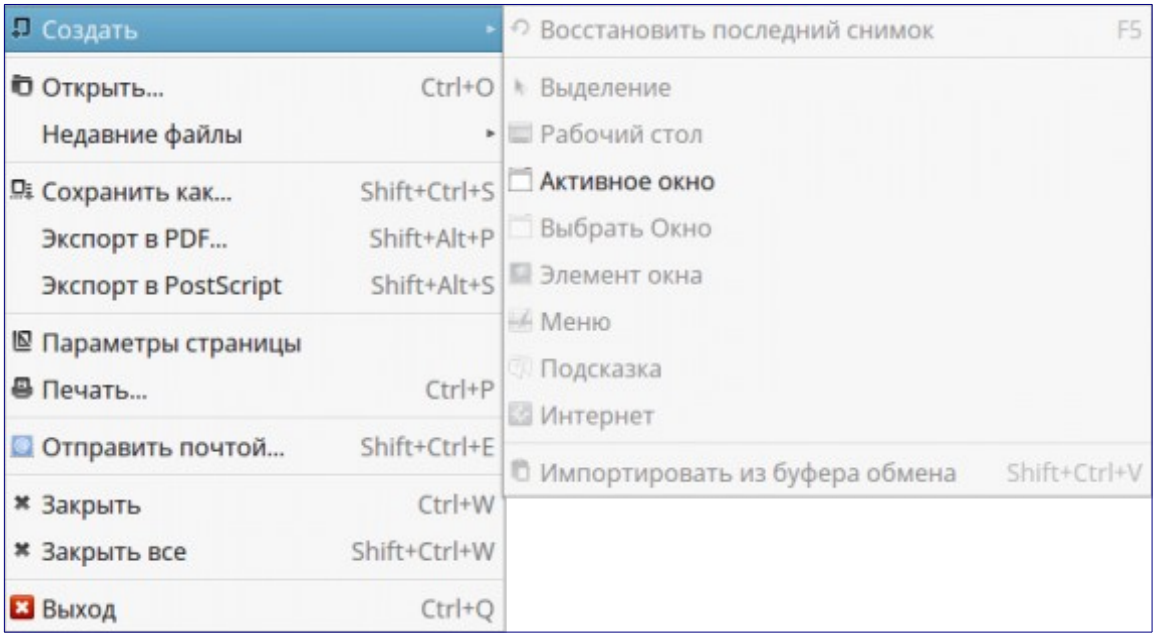
После первого запуска (из секции Стандартные главного меню) появляется примерно такое окно — снимок текущего рабочего стола при этом по умолчанию делается автоматически:



Доступ к основным функциям программы можно получить через строку пиктограмм в верхней части окна:



Или же сделать это через главное меню — через пункты Файл -> Создать -> [нужный объект]:



Пиктограммы панели Shutter'а следующие (слева направо):

- повторение последнего снимка — понятно без комментариев;

- выделение мышью прямоугольной области экрана для снимка; щёлкнув на стрелке рядом, можно выбрать инструмент выбора — простой или усовершенствованный (по умолчанию); отличие второго в том, что он позволяет масштабировать выбеленный участок перед созданием скриншота;

• рабочий стол — с её помощью можно снять не только текущий, но и любой другой из наличных виртуальных десктопов, и даже все сразу в виде одного скриншота;

• окно — снимается активное или любое из открытых, по выбору (в том числе и из свёрнутых);

• снимок отдельного элемента окна, к сожалению (у меня?) не работает, выдавая или сообщение об ошибке, или снимая выбранное окно целиком;

• зато выбор меню в приложении работает превосходно, позволяя сделать снимок выпадающего или контекстного меню любой степени вложенности;

• столь же неизменно превосходный результат даёт и захват всплывающих подсказок; при этом (как и при снимках меню) даётся время для выбора — по умолчанию 10 секунд.

Подчеркну одно из главных (с моей точки зрения) достоинств программы: с помощью её можно делать снимки любых меню и подсказок без всяких дополнительных ухищрений, типа расчёта времени на щёлканье мышью, съёмки всего экрана и последующего выпиливания из него нужных элементов.

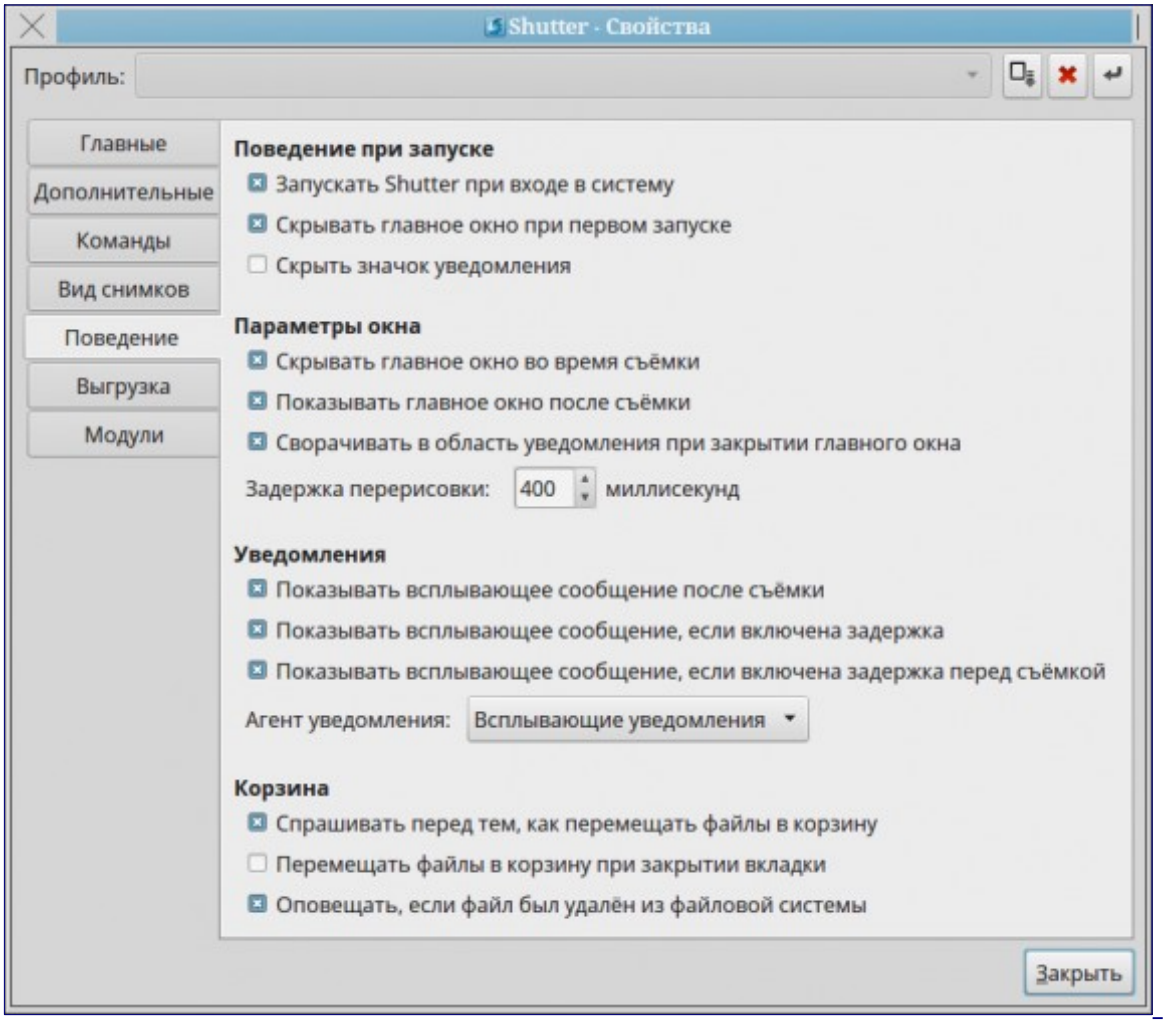
Таинственная кнопка, требующая установки gnome-web-photo, у меня не активизирована (за отсутствием последнего). А пиктограммы Правка и Экспорт предоставляют те самые супер-функции Shutter'a, о которых упоминалось в описании — встроенный редактор изображений и средства помещёния их на соответствующие серверы. Правда, доступ к встроенному редактору требует установки пакета libgoo-canvas-perl, который в стандартной инсталляции Mint отсуствует. Что, однако легко поправимо:

```
$ apt install libgoo-canvas-perl
```

Основные функции, доступные через меню — практически те же самые. Добавлю только, что у Shutter'a есть ещё и функция вьювера изображений, доступная через меню: Переход -> Вперёд/Назад/В начало/В конец. А если в меню Вид включить пункт Показать панель навигации, то перемещаться между изображениями можно с помощью стрелок, которые появятся в нижней части окна программы.

Таким образом, даже беглое рассмотрение возможностей Shutter'a свидетельствует, что его характеристика в описании ничуть не преувеличена: эта программа действительно может «заскриншотить» практически всё. Остаётся посмотреть только, насколько удобно с этим «заскриншоченым» хозяйством управляться.

Чтобы оценить меру удобства Shutter'a при его практическом использовании, надо обратиться к настройкам этой программы. Каковые, как это в обычае для Gtk-приложений, находятся в главном меню через пункты Правка -> Параметры и выглядт следующим образом:



Можно видеть, что окно настроек включает ряд вкладок, из которых я в рамках сегодняшней темы остановлюсь лишь на некоторых. Начав, естественно, с Главной.

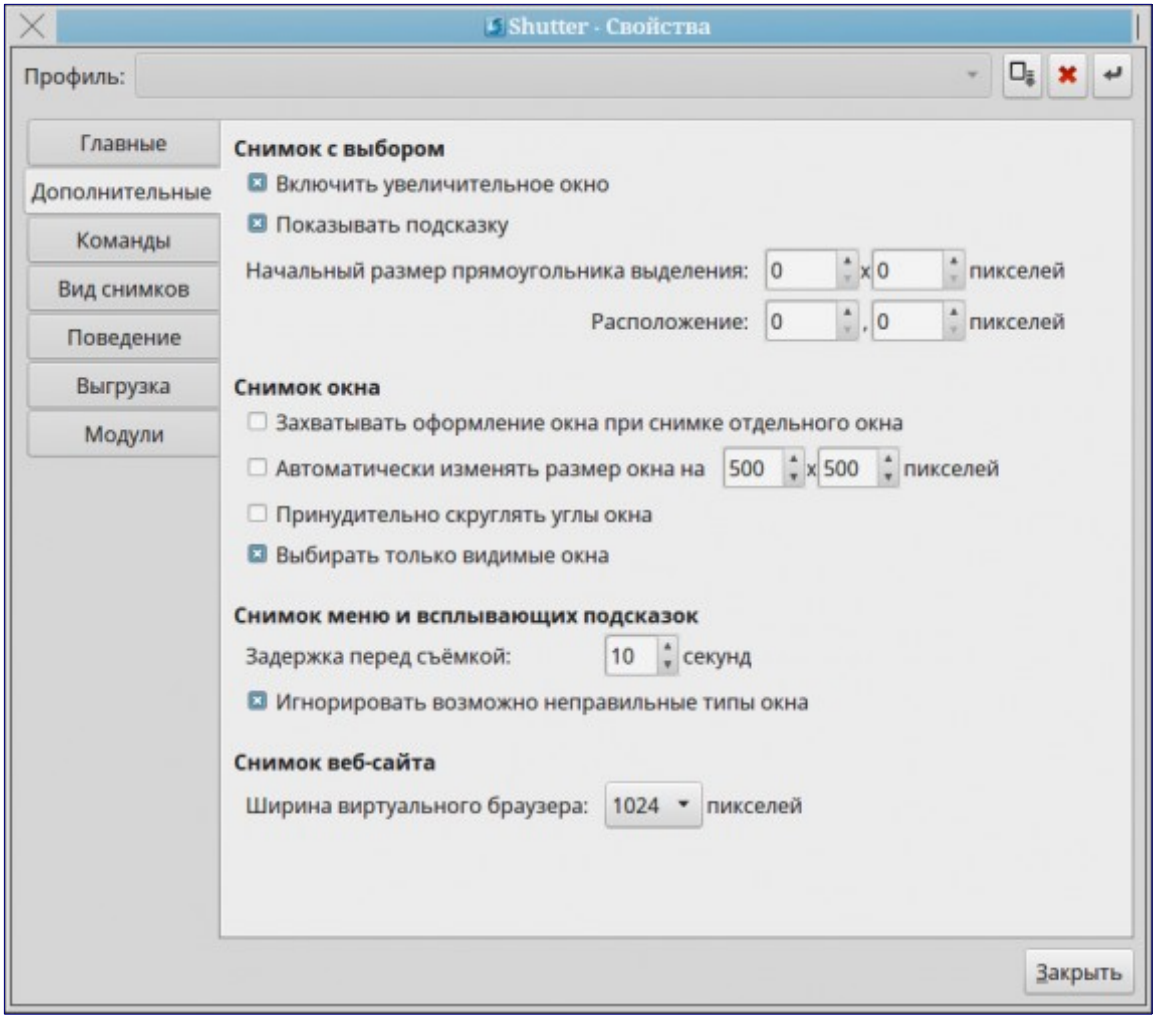
Говорить о степени сжатия или поддерживаемых форматах особо нечего — это и так все знают. Замечу только, что, кроме умолчального png, в принципе скриншоты можно снимать в иногда нужном для размещёния в web'e jpeg'e и даже в реликтовом bmp — для совместимости с допотопными редакторами.

А вот настройки условий сохранения очень важны для линуксописателя (да и вообще для писателя, иллюстрирующего свои тексты картинками). По умолчанию включено автоматическое сохранение скриншотов в каталоге $HOME/Pictures, а имена их файлов генерируются по схеме name_номер. Где в качестве переменной name фигурирует заголовок окна (если делается снимок окна), слова меню, рабочий стол и так далее (при выборе соответствующих объектов съёмки).

Всё это при практическом сочинительстве очень неудобно. Поэтому имеет смысл включить опцию Каждый раз спрашивать, куда сохранять, а целевой каталог по умолчанию и систему именования файлов установить в соответствие со своими свычаями и обычаями — у всех практикующих линуксописателей они разные, да и зависят от обстоятельств.

Автоматическое копирование снимка или имени его файла включаются по желанию и потребностям, как и захват курсора (последнее в некоторых случаях нужно, но обычно мешает). Задержка перед съёмкой относится к «скриншотированию» окон и рабочих столов, и в данном случае обычно не

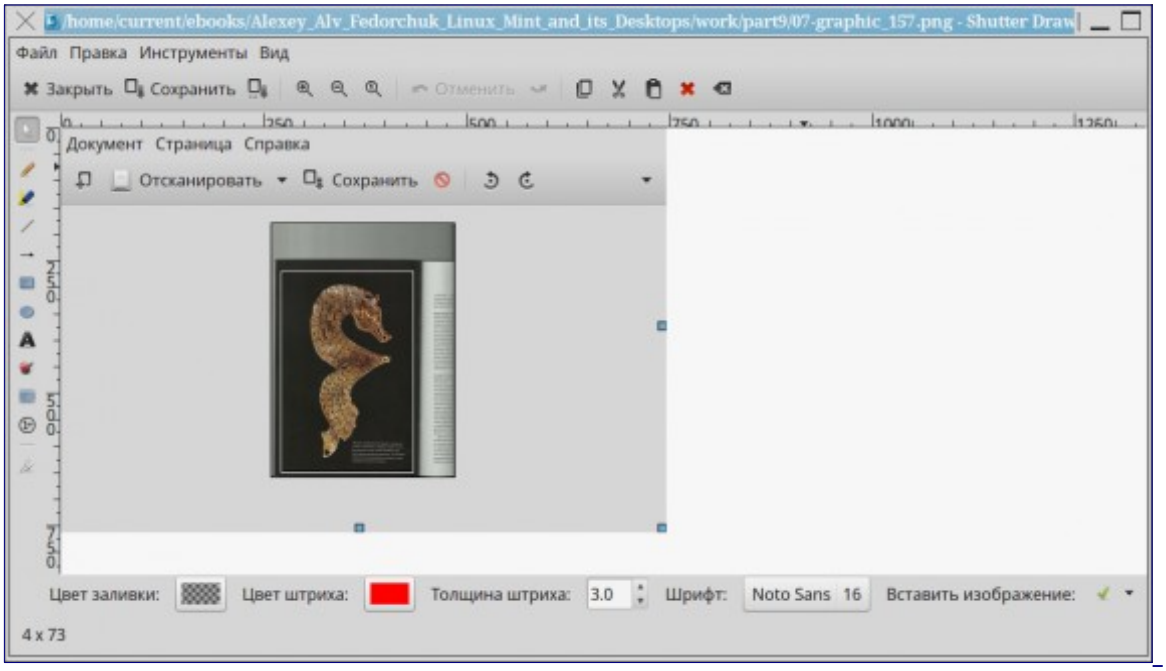
нужна. А когда нужна — об этом на следующей вкладке, каковая называется Дополнительной и выглядит так:



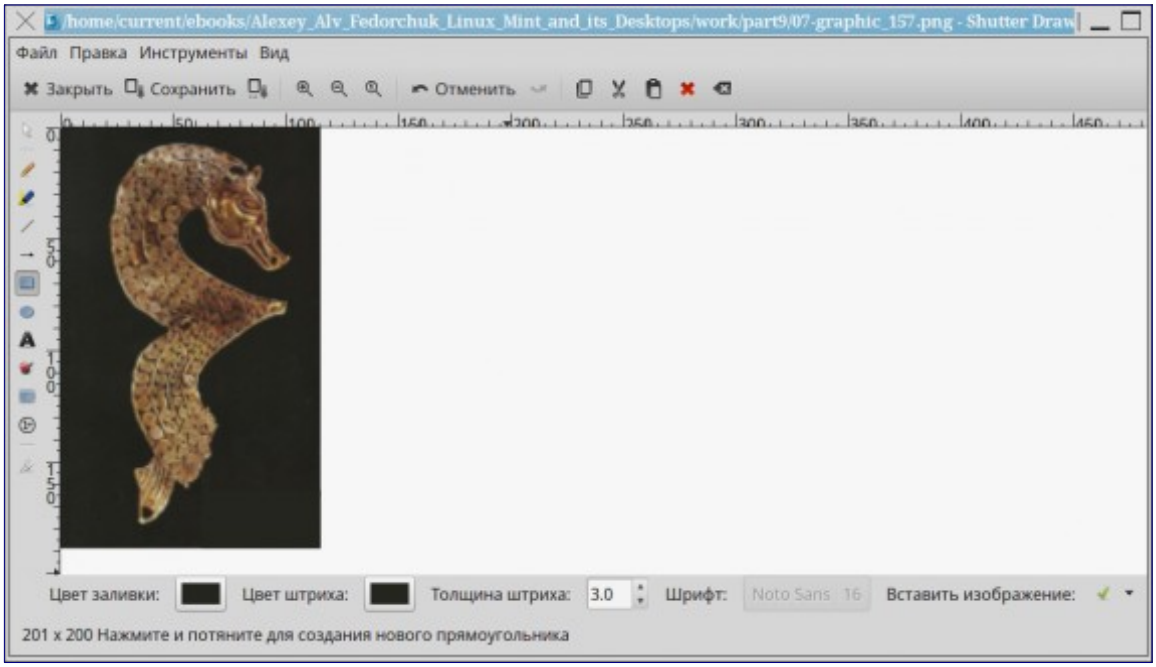
Где как раз и выставляет время задержки перед съёмкой меню и всплывающих подсказок, в зависимости от количества подготовительных действий и быстроты реакции. Прочие же опции здесь определяются вкусом и ситуацией. Что, впрочем, относится и к остальным вкладкам.

Теперь бросим взгляд на встроенный графический редактор Shutter' — для запуска его на предмет редактирования текущего файла надо нажать кнопку Правка — напомню, что она активизируется только после установки пакета libgoo-canvas-perl.

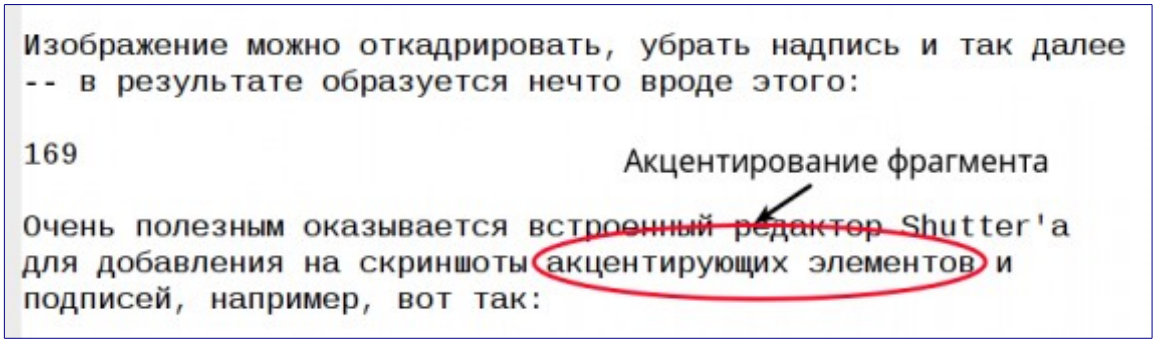
В качестве примера откроем его во встроенном редакторе один из ранее приведённых скриншотов:



Изображение можно откадрировать, убрать надпись и так далее — в результате образуется нечто вроде этого:

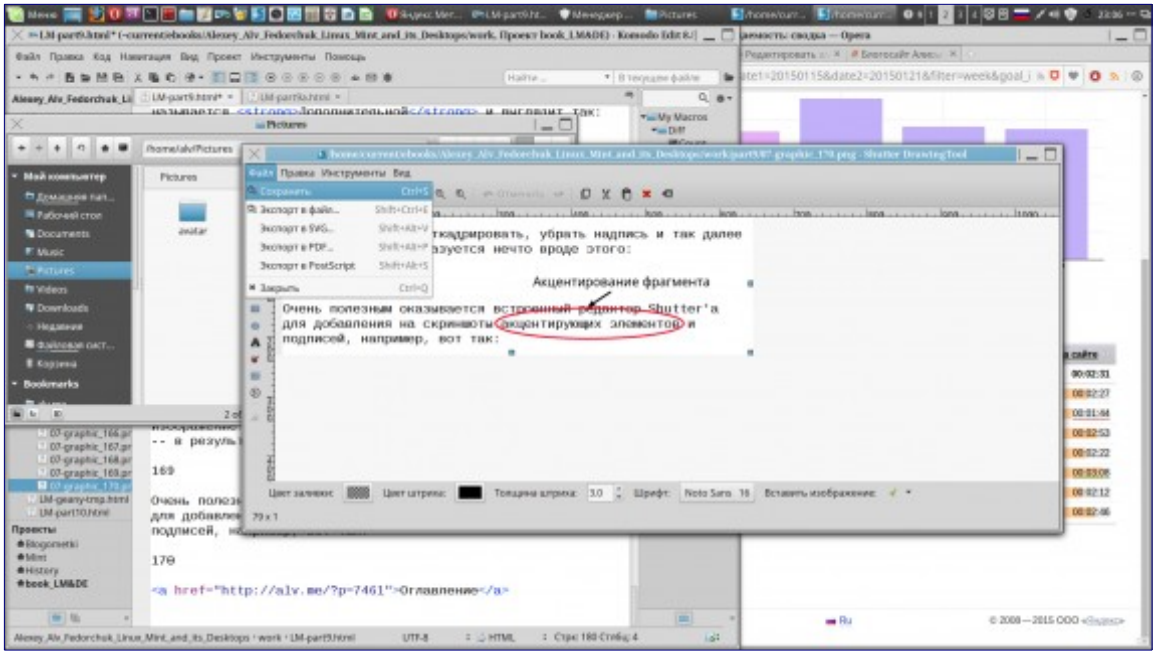


Очень полезным оказывается встроенный редактор Shutter'a для добавления на скриншоты акцентирующих элементов и подписей, например, вот так:



По завершении редактирования файл можно сохранить под тем же именем.

Прямого сохранения с переименованием (то есть Save as...) не предусмотрено. Однако это можно заменить экспортом в тот же формат под другим именем:

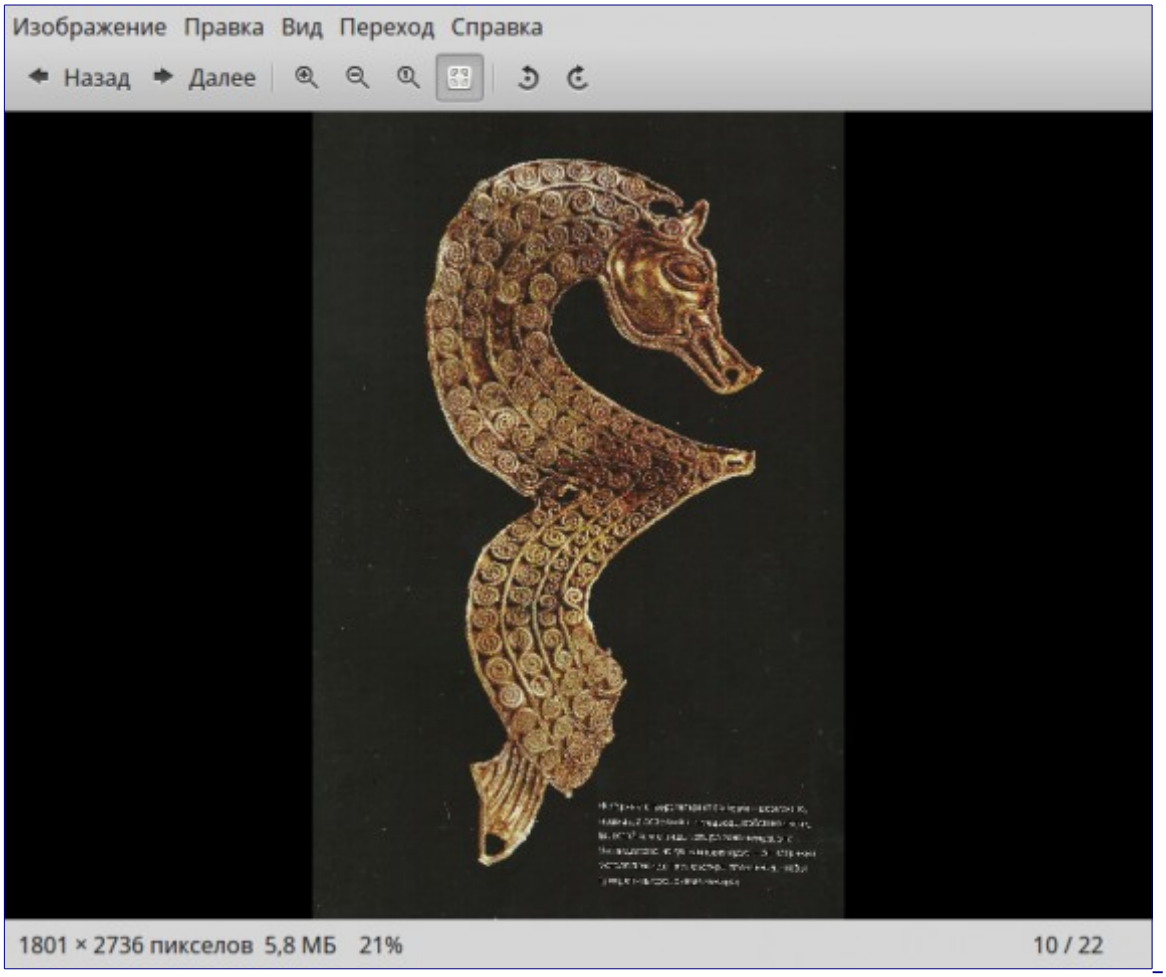


С остальными функциями встроенного редактора предоставляю разбираться читателям.

Вьюверы изображений

Изображения, полученные любым из перечисленных способов, необходимо просматривать, а иногда и подвергать некоторому редактированию. С первой задачей прекрасно справляется штатный вьюевер изображений, известный как Eye of GNOME (или, сокращённо, eog) — в секции Графика главного меню Cinnamon он скрывается под псевдонимом просмотр изображений.

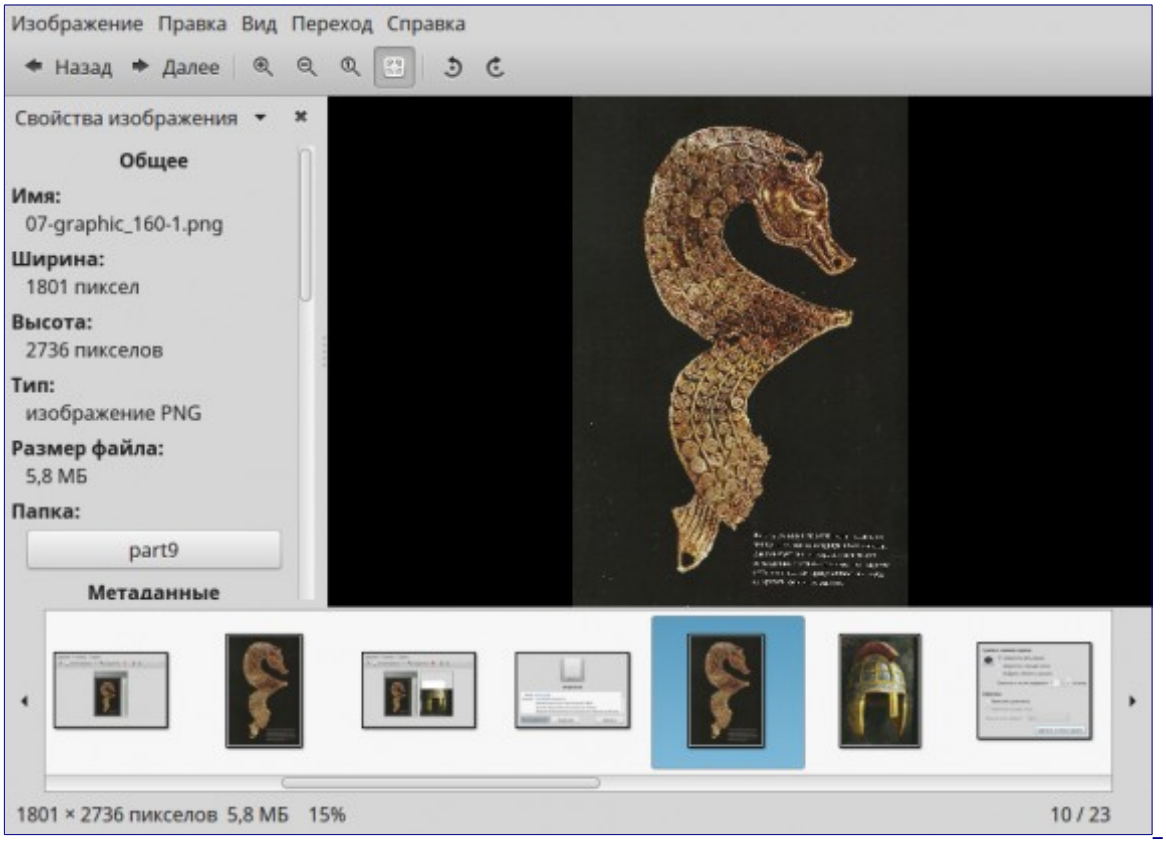
Однако обычно Eog вызывается из файлового менеджера щелчком на имени графического файла известного ему формата — а ему известны практически все растровые форматы. После чего он предстаёт примерно в таком виде:



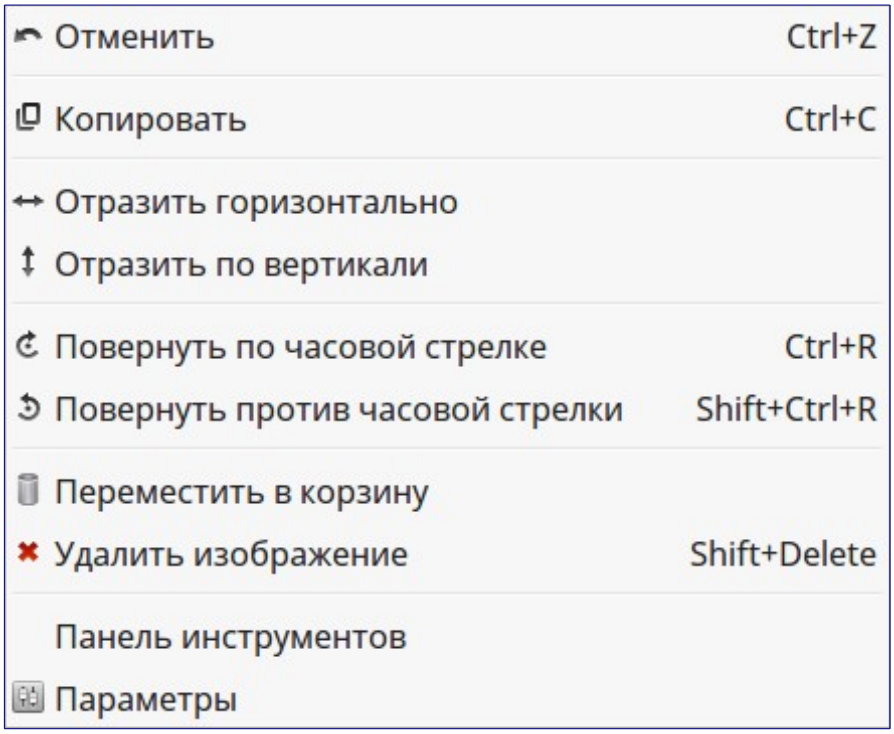
Если в текущем каталоге имеется более одного графического файла — их можно просматривать, щёлкая по пиктограммам со стрелками Далее и Назад.

Через меню Вид можно включить Галерею изображений и Боковую панель.

В первой — миниатюры картинок, находящихся в текущем каталоге, во второй — подробная информация о выведенном в данный момент файле:



Над просматриваемыми файлами можно произвести некоторые действия, типа вращения и отражения:



Более сложные манипуляции с файлами можно выполнить в программе gThumb, которая также входит в стандартную инсталляцию Cinnamon-редакции дистрибутива Mint. Это также по преимуществу вьювер изображений, наделённый, однако, функциями лёгкого растрового редактора. Он вызывается следующими способами:

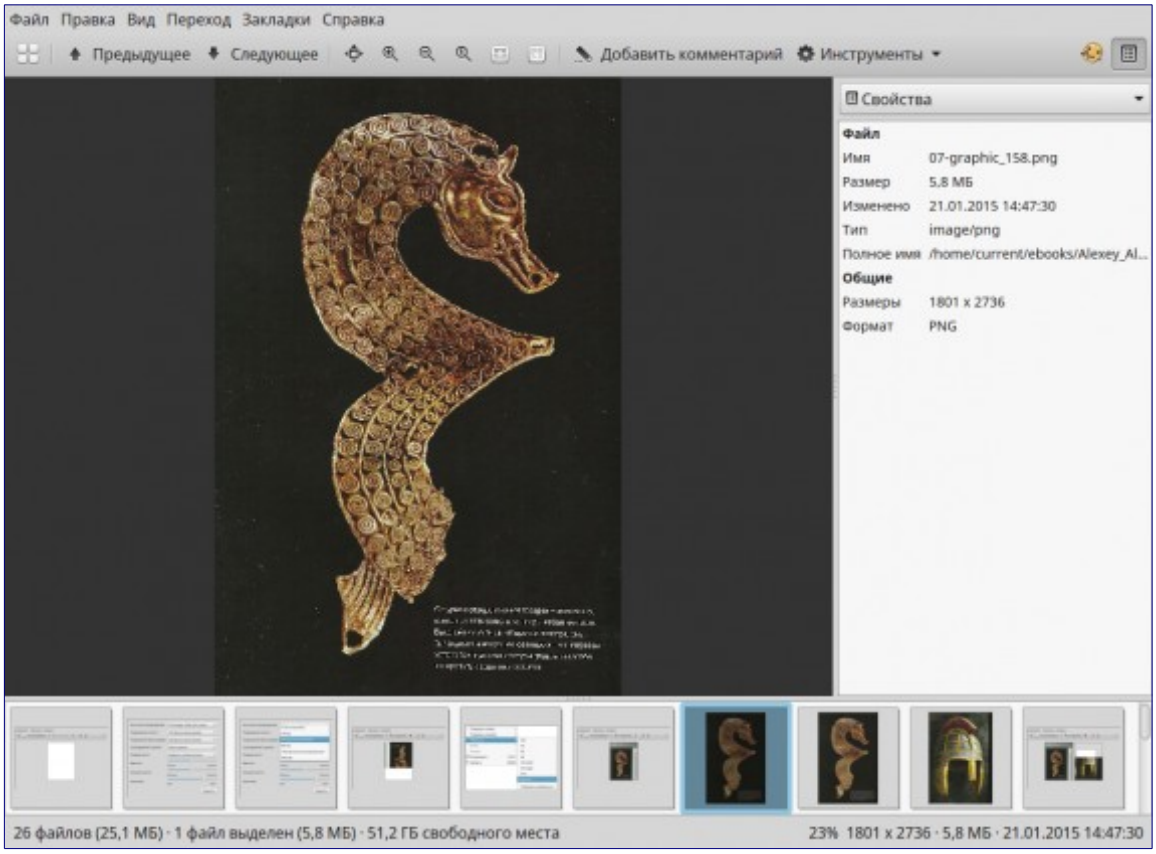
- из главного меню — где он под собственным именем обнаруживается в секции Графика;

- из контекстного меню Nemo через пункт Open with;

- из контекстного меню Shutter'a и Eog'a — через пункты Открыть в и Открыть с помощью, соответственно.

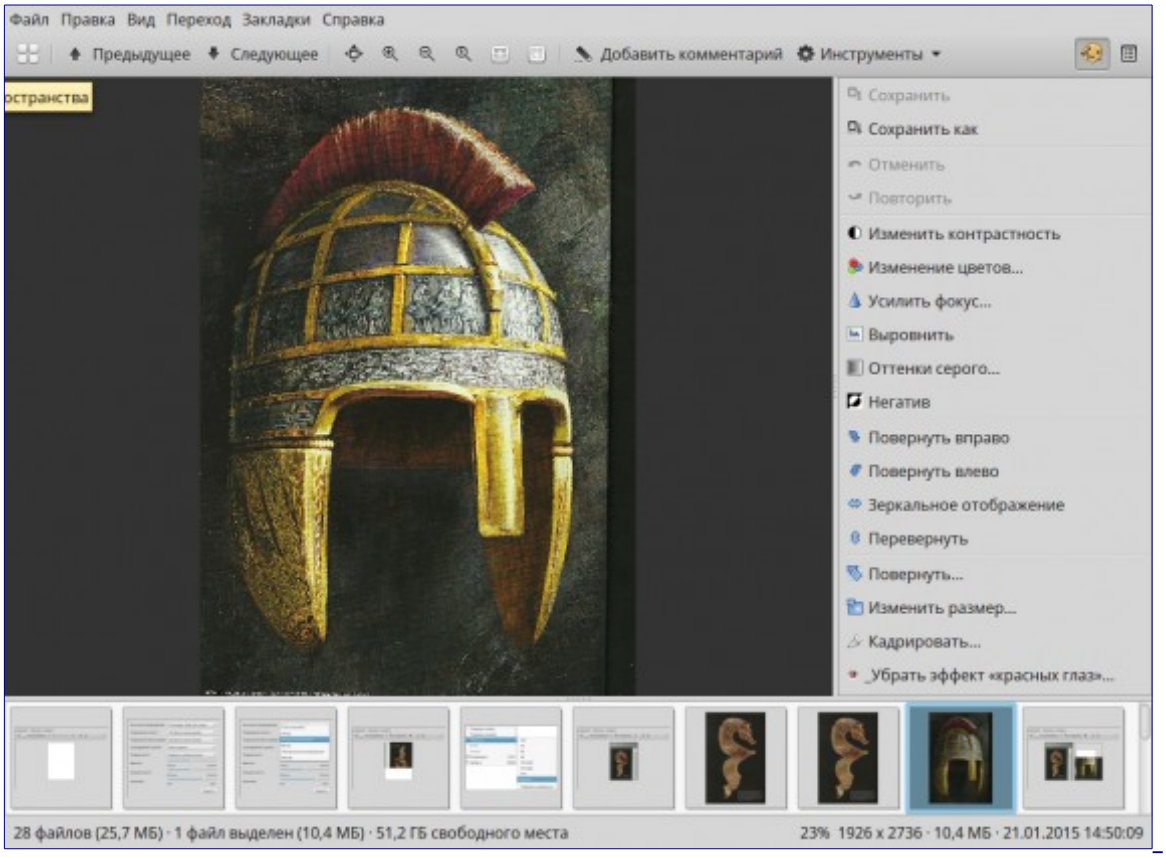
Про банальный вызов этой программы из командной строки терминала или минитерминала я уж и не говорю.

В любом случае после запуска gThumb будет выглядеть примерно так:

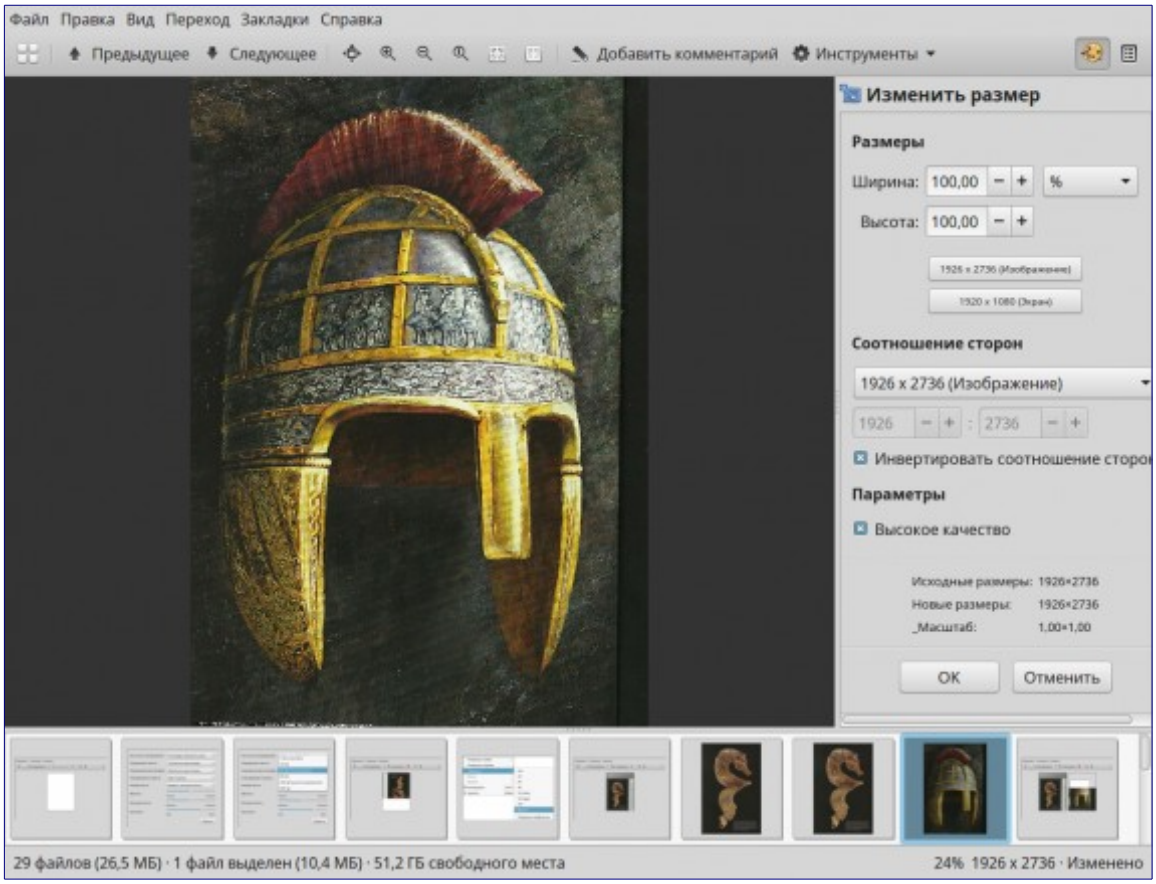


На скриншоте можно видеть боковую панель, именуемую Свойства, и Панель миниатюр — и ту, и другую можно отключить через меню Вид.

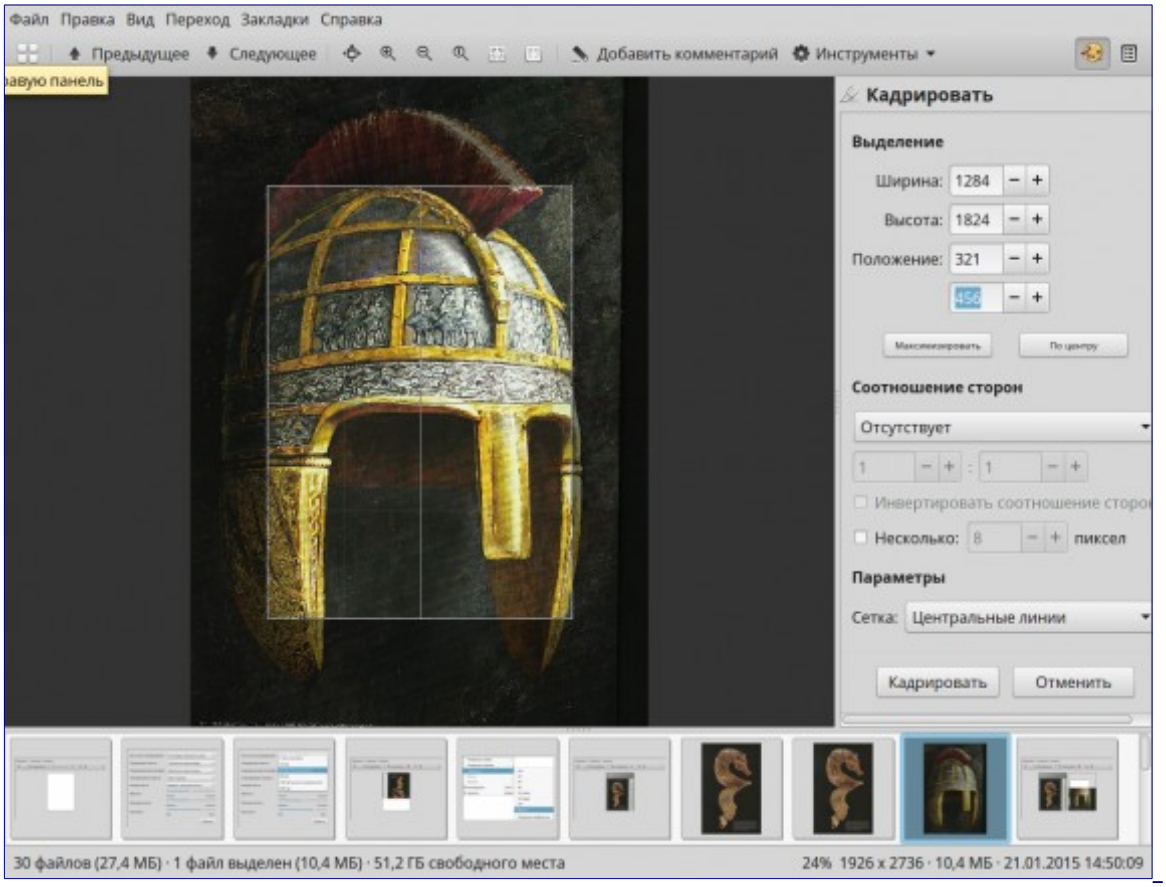
Думаю, вопроса, как с помощь gThumb просматривать изображения, ни у кого не возникнет. Однако главная сила этой программы в другом. Если щёлкнуть по изображению палитры в право верхнем углу, в боковой панели вместо информации об изображении появятся инструменты для его редактирования:



Как этими инструментами пользоваться — понятно без комментариев. Так что просто проиллюстрирую две наиболее востребованные (по крайней мере, мной) операции — Изменение размера



и Кадрирование:



К этим скриншотам можно только добавить, что, если численно задать размеры изображения и положение его левого верхнего угла, они сохранятся в течении всего сеанса (если их не поменять снова). То есть gThumb способен к обработке большого количества однотипных файлов почти в пакетном режиме.

Офисные приложения

Когда речь заходит об открытых и свободных офисных пакетах, вспоминают, как правило, LibreOffice и Apache OpenOffice.org, реже — вечный долгострой проекта KDE, KOffice, ныне перевоплотившийся в Calligra. И мало кто упомянет в этой связи компоненты так называемого GNOME Office — текстовый процессор Abiword и табличный процессор Gnumeric.

Вступление

Во всех редакциях дистрибутива Mint (как, впрочем, и в подавляющем большинстве «больших» дистрибутивов) в роли офисного пакета выступает LibreOffice в полной комплектации. Однако многим применителям, включая автора этих строк, из всего офисного богачества требуются только текстовый процессор (точнее, word processor, ибо text processor — это неинтерактивные программы типа groff) и процессор табличный (он же — электронная таблица).

Можно, конечно, поудалять лишние компоненты полного LibreOffice. А можно сразу обратиться к офису «незнаменитому» — Abiword и Gnumeric, которые подчас вполне искусственно объединяются в пакет GNOME Office. Искусственно — потому что обе эти программы вполне самостоятельны, и к среде GNOME относятся постольку, поскольку используются и в ней тоже.

Самостоятельность Abiword и Gnumeric нисколько не умаляет их достоинств. Каковыми считаются лёгкость, быстродействие, простота

освоения и применения. Но при этом забывают о функциональности — а ведь каждая из этих программ обладает своими уникальными особенностями.

Для Abiword'а это средства коллективной работы. Во-первых, он поддерживает мультиверсионные документы — в том числе и те, что были сделаны таковыми в MS Word. Во-вторых и главных, Abiword располагает инструментами удалённого редактирования, по собственному протоколу AbiCollab.net, прямому подключению TCP и, наконец, по протоколу XMPP — то есть через самый обычный Jabber-клиент.

Ну а «фирменные фичи» Gnumeric'а — это изобилие статистических и инженерных функций (более ста из которых уникальны) и широчайшие возможности для построения технических диаграмм и графиков.

И Abiword, и Gnumeric включены в официальный репозиторий Mint. Там же можно найти и плагины к ним — для первой программы это abiword-plugin-grammar и abiword-plugin-mathview, для второй gnumeric-plugins-extra. И всё это вместе установить любым из стандартных способов, проще всего — комнадой apt с указанием всех перечисленных пакетов. После чего можно приступать к их использованию.

Abiword

Рискну предположить, что очень многие применители Linux используют текстовые процессоры с одной-единственной целью — обмениваться документами с пользователями Windows, часто не подозревающими о существовании иных форматов, кроме MS Word. Ну и, возможно, для эпизодического составления документов собственных — тех, которые требуют того или иного стандартного форматирования (например, докладных записок). Такие пользователи, как правило, не нуждаются ни в красивых презентациях, ни в финансовых или инженерных функциях, ни в связи с базами данных, предлагаемых им соответствующими компонентами интегрированных офисных пакетов. Более того, не нужно им и все изобилие функций собственно текстовых процессоров из тех же пакетов. И вот им-то самое время вспомнить о существовании AbiWord.

Название этой программы происходит от испанского слова «Abierto» (открытый) и вездесущего компонента Word. Она начала разрабатываться в 1998 году вполне коммерческой компанией SourceGear как часть AbiSuite — кросс-платформенного офисного пакета, в который предполагалось включить только открытые компоненты. В дальнейшем интересы компании сместились в другие сферы, и разработка AbiSuite была заброшена. Но дело развития AbiWord взяла в свои руки группа добровольцев, которые продолжают его и поныне. На рубеже тысячелетий проект GNOME включил эту программу как текстовый процессор своего GNOME Office, но сам по себе AbiWord возник и развивается независимо от последнего, как кросс-платформенное приложение (помимо сборок для всех полнофункциональных дистрибутивов Linux и портов для BSD-систем, существует также Windows-сборка пакета, доступная в качестве самораспаковывающегося архива, не требующего никаких дополнительных компонентов).

Собственно, AbiWord объединяет с GNOME Office только одно — использование Gtk в качестве базовой библиотеки. Правда, как правило, AbiWord собирается майнтайнерами дистрибутивов с поддержкой библиотек GNOME, и дистрибутив Mint тут не составляет исключения, но в принципе это отнюдь не обязательно. Степень интеграции Aboword в виртуальный пакет

GNOME Office практически нулевая, и от его пользователя не потребуют принудительной установки остальных компонентов последнего. Хотя Gnumeric, электронная таблица из этого офисного пакета, также является штатным компонентом стандартной редакции дистрибутива Mint.

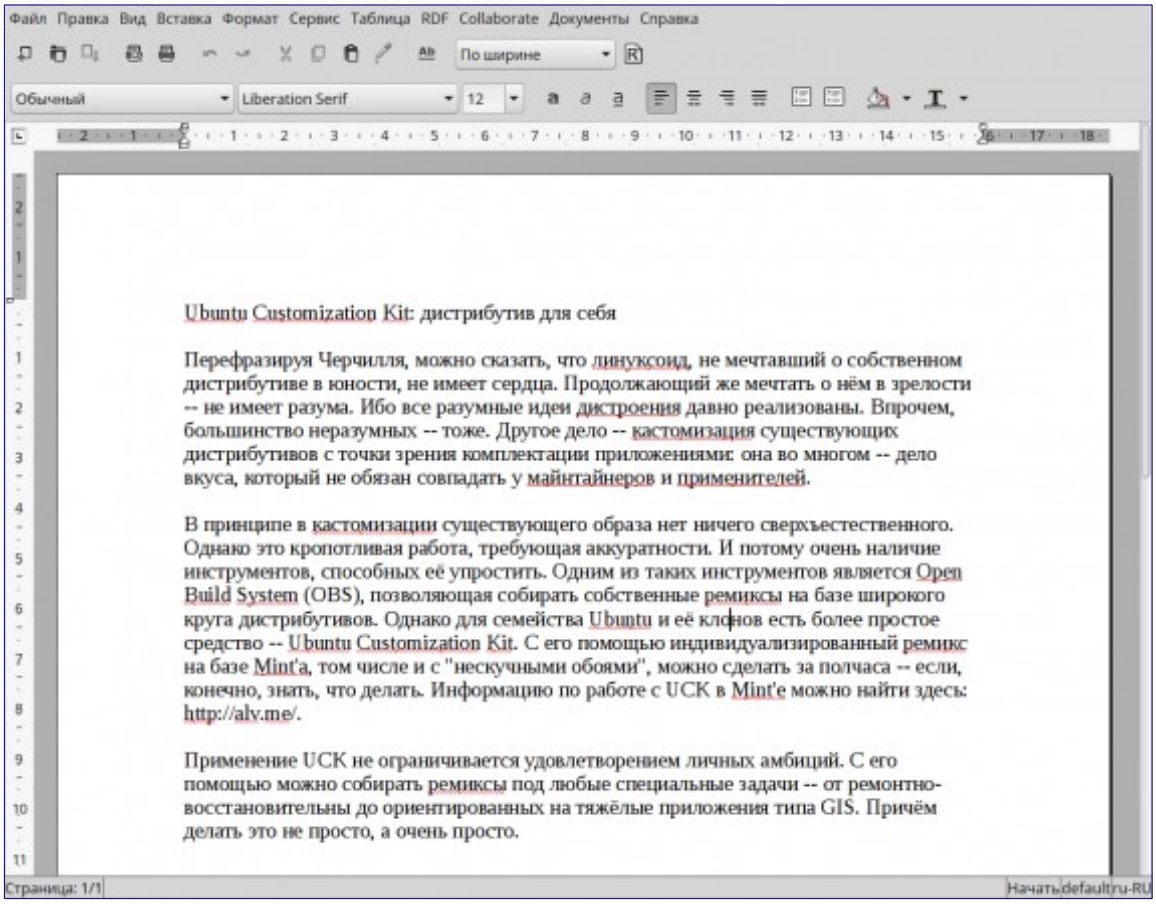
Традиционно Abiword представляется разработчиками и обозревателями как быстрый и легкий, но при этом полнофункциональный текстовый процессор WYSIWYG-типа, а некоторые добавляют к этому ещё и определение «элегантный». И почти со всеми этими эпитетами можно согласиться. Предметом спора может быть, разве что, определение полнофункциональности — тут у каждого пользователя свои представления, ничуть не более объективные, нежели понятия об элегантности.

Во-первых, AbiWord действительно быстр и лёгок — запускается в «голом» виде он практически мгновенно, запуск вместе с открытием небольших файлов в формате что в собственном формате doc и odt тоже происходит очень быстро, хотя с большими документами он работает несколько медленнее. А его исполняемый файл имеет объём шесть с половиной мегабайт, занимая в памяти на порядок меньше места, чем OpenWriter.

В-вторых, AbiWord, при всей своей легкости обладает большинством атрибутов развитого текстового процессора: поддержкой шаблонов, стилей, настраиваемых пользователем, таблиц, проверки орфографии, вывода статистики, возможностью вставки специальных символов и так далее. Не так давно он приобрел способность корректно работать с мультиверсионными документами, причём не только собственными или соответствующими стандарту ODT, но и созданными в формате MS Word. А в последней версии AbiWord'а появилась интеграция с системами онлайнового перевода, поисковой машиной Google и даже Википедией, . Дополнительные функции реализованы в виде подключаемых внешних модулей, работающих через web-интерфейс браузера, умолчального для данной системы, и потому не утяжеляют саму программу.

В-третьих, несмотря на функциональность программы, интерфейс её сохраняет простоту и не выглядит тяжеловесным и перегруженным. Количество инструментальных панелей ограничено разумным числом (их четыре, по умолчанию выводится только две — стандартная и форматирования), никаких дополнительных, принудительно всплывающих панелей не наблюдается. Наборы инструментов на панелях по умолчанию не редактируемы, но все не охваченными ими действия возможны через пункты главного меню. Контекстное меню, традиционно вызываемое щелчком правой клавиши мыши, также содержит лишь пункты для базовых действий плюс вызов внешних модулей (перевода, словарей, поиска).

Всё это вместе действительно создаёт впечатление элегантности, возникающее при первом же запуске программы. Давать подробное описание её возможностей и приёмов использования в рамках настоящего очерка я не буду. А вместо этого остановлюсь на некоторых его особенностях и ограничениях программы, после чего изложу личные впечатления от её практического использования. Думается, что этого будет достаточно пользователям для того, чтобы решить, подходит ли Abiword именно им.



Итак, одно из главных требований к текстовому процессору для Linux — это совместимость (в первую очередь — сами знаете с кем). До недавнего времени поддержка документов MS Word в AbiWord'е была реализована... скажем так, не лучшим образом: при считывании doc-файлов подчас терялось сложное форматирование, пропадало стилевое оформление, категорически не воспринималась мультиверсионность — словом, происходили разного рода неприятности. Они стали исчезать уже в прошлой версии, а ныне их можно считать окончательно изжитыми. Следует ещё раз подчеркнуть упомянутую выше корректность работы с мультиверсионными документами.

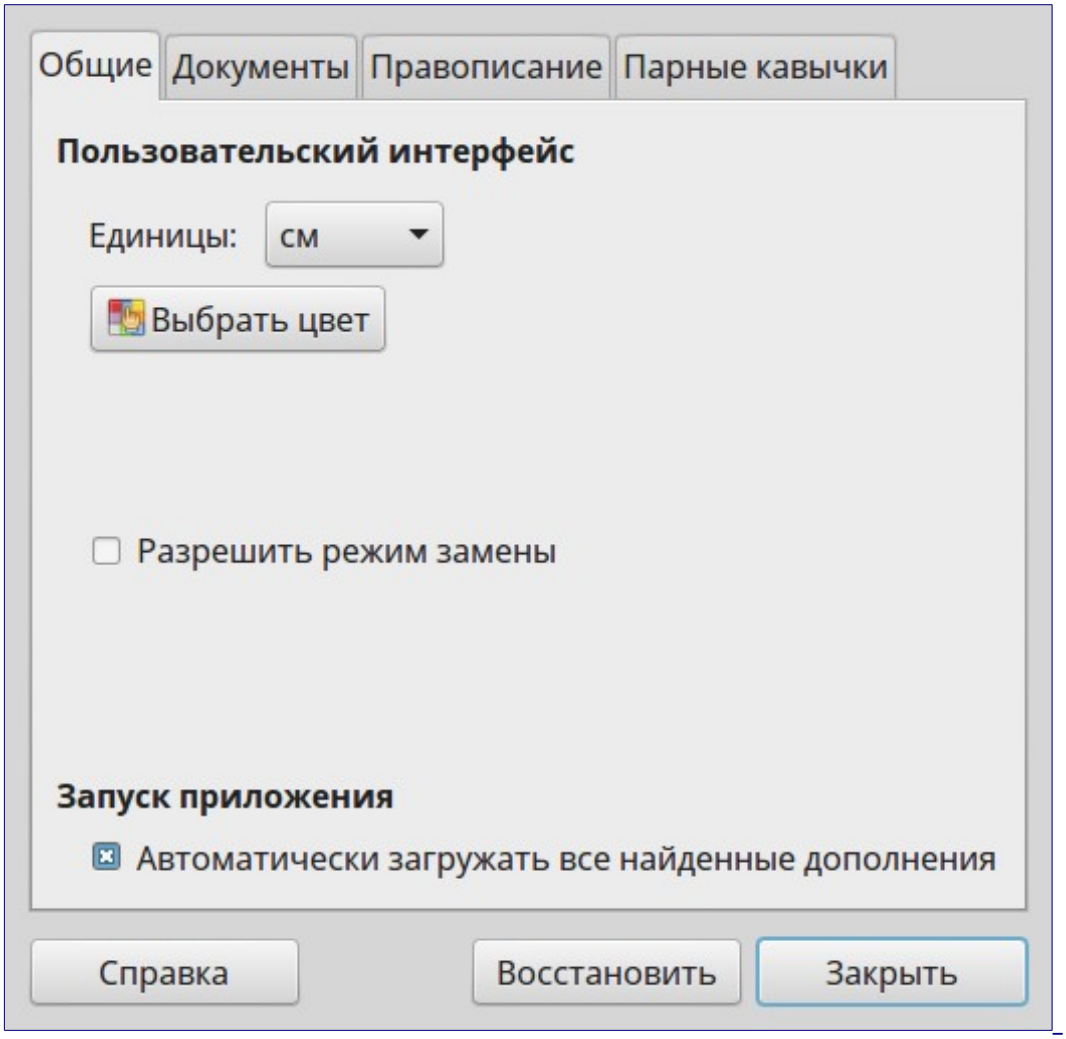
Не менее важным с точки зрения совместимости является обращение текстового процессора с документами HTML-формата. И тут Abiword'у есть чем похвастаться перед более «толстыми» собратьями: долгое время он был единственным текстовым процессором, корректно считывающим русскоязычные html-документы без определения DOCTYPE и метатега charset. То есть тех фрагментов между открывающим и закрывающим тэгами body, которые создаются при использовании различных CMS и прочих автоматизированных web-редакторов.

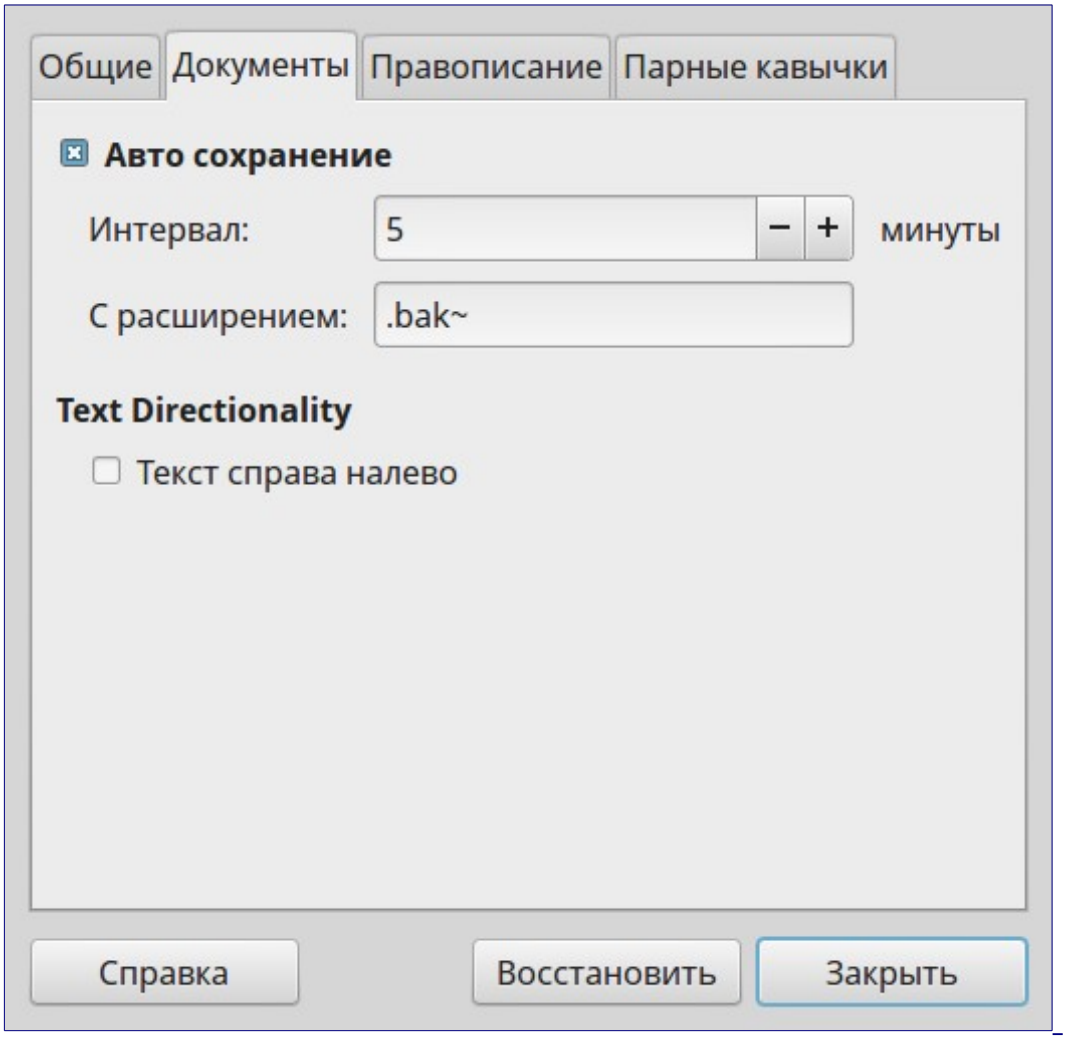
Отличается Abiword и по части экспорта документов в html-формат. Всякий, видевший результаты этого процесса для LibreOffice Writer и OpenWriter, не мог не повторить известную сентенцию: «Если это html, то дайте мне, пожалуйста, plain text». Abiword же при записи в данный формат на выходе даёт вполне чистый html-код с минимальной отсебятиной.

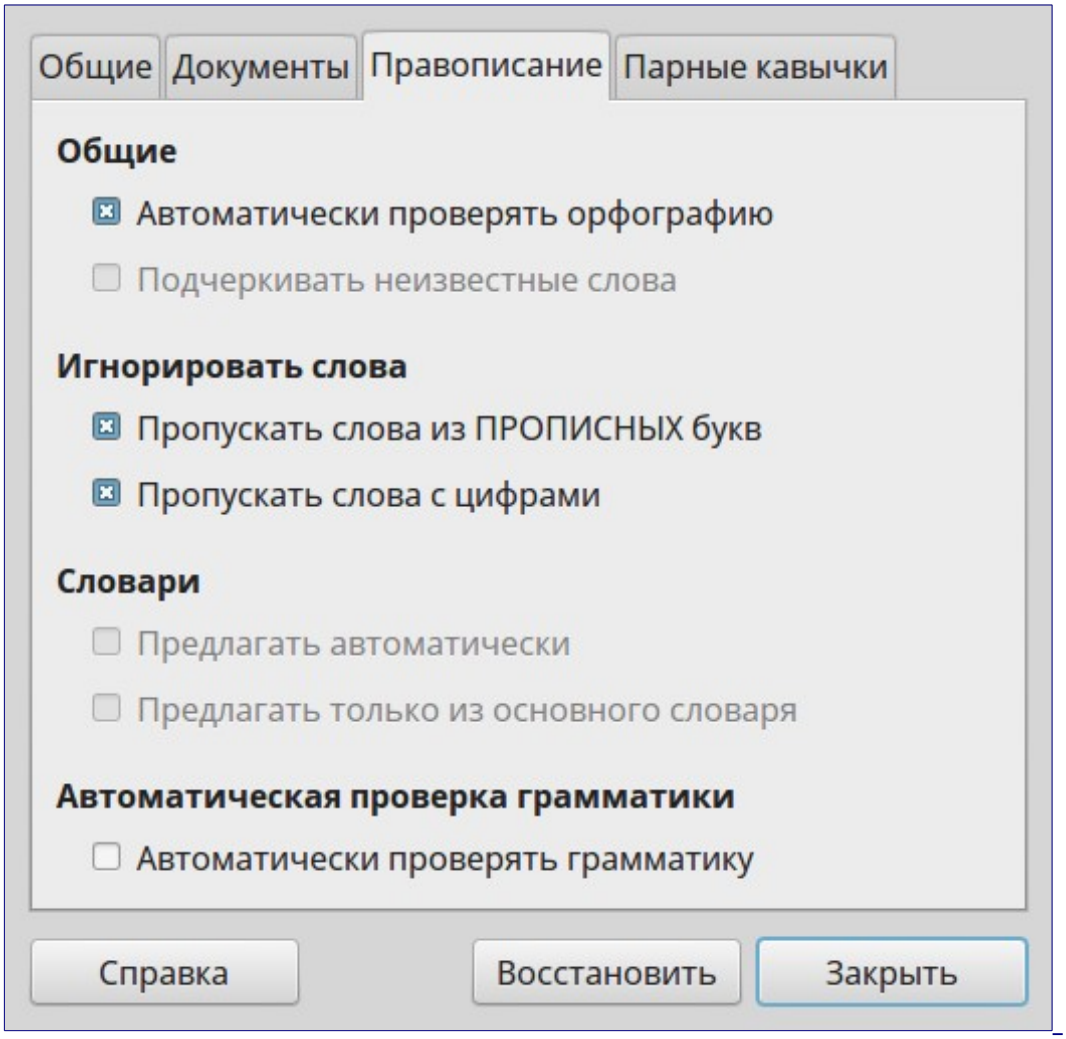
Раз уж речь зашла об экспорте, надо остановиться на таком моменте. При экспорте через пункты меню **Файл** -> **Сохранить как...** по умолчанию задаётся экспортируемый формат Abiword (*.abw, *.zabw, *.abw.gz), никем, кажется, более не понимаемый. И, чтобы указать нужный, приходится долго пролистывать длинный выпадающий список доступных форматов, в том числе весьма экзотических, что несколько раздражает.

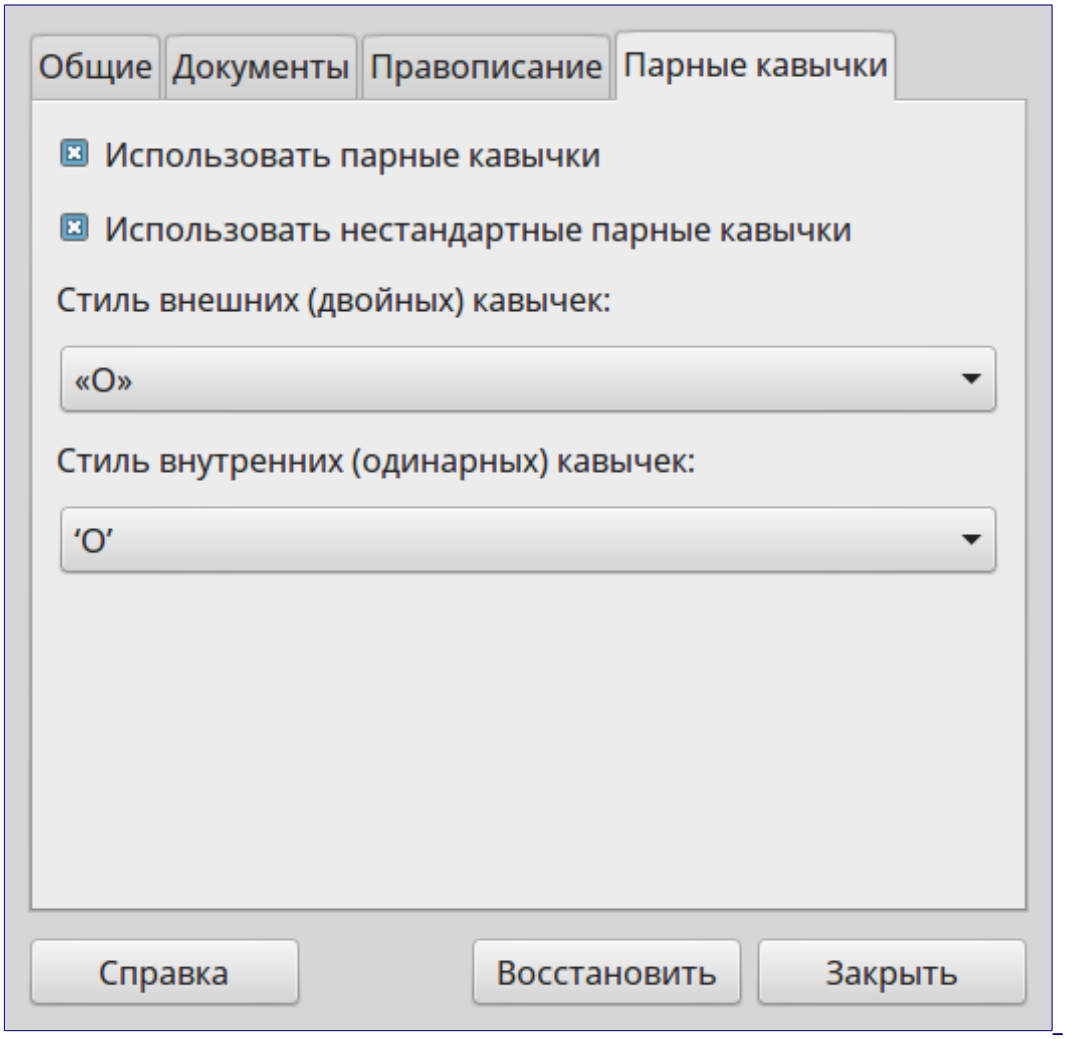
Очень важная функция любого текстового процессора — встроенная проверка орфографии, в том числе «на лету»: для многих пользователей подчеркивание неправильно написанных слов является главным стимулом к использованию текстового процессора для составления оригинальных документов. Ранее Abiword в отношении проверки русского правописания наблюдалась напряжёнка — но текущем состоянии он проверка орфографии работает безукоризненно как в автоматическом, так и ручном режимах.

Теперь об ограничениях программы. Главное из них — весьма скудные возможности её настройки. Они выполняются в пункте меню Правка -> Параметры… и сводятся к включению возможности переопределения наборов пиктограмм на инструментальных панелях (как я уже говорил, по умолчанию она отключена), выбору единиц измерения, направления текста (Abiword — программа интернациональная, и её разработчики подумали и о пишущих справа налево) и параметров проверки правописания. Умолчальные параметры нового документа, такие, как шрифт, кегль, абзацный отступ и так далее, можно задать через переопределение базового стиля Normal (через меню Сервис -> Стилист). Иных возможностей настройки вроде бы не просматривается. Впрочем, и перечисленных обычно более чем достаточно.




Общие
Документы
Правописание
Парные кавычки
Авто сохранение
Интервал:
5
минуты
С расширением:
.bak~
Text Directionality
Текст справа налево
Справка
Восстановить
Закрыть


Общие
Документы
Правописание
Парные кавычки
Общие
Автоматически проверять орфографию
Подчеркивать неизвестные слова
Игнорировать слова
Пропускать слова из ПРОПИСНЫХ букв
Пропускать слова с цифрами
Словари
Предлагать автоматически
Предлагать только из основного словаря
Автоматическая проверка грамматики
Автоматически проверять грамматику
Справка
Восстановить
Закрыть



Из недостающих функций бросается в глаза отсутствие копирования формата выделенного фрагмента — в ряде случаев весьма полезная штука. Правда, это частично компенсируется иной функцией — копированием без форматирования (в меню Правка). Не вполне удачно (точнее, вполне, на наш взгляд, неудачно) реализована вставка специальных символов. Ну и попытка вставить гиперссылку неизменно приводила у нас к выпадению программы в осадок. Впрочем, и это, скорее всего, дефект конктерной сборки.

Теперь о личных впечатлениях. Автор этих строк следит за развитием Aboword'а на протяжении уже более десяти лет. Его потребности в текстовом процессоре ограничиваются двумя задачами: чтением и редактированием doc-файлов, присланных корреспондентами, в том числе сотрудниками редакций, с которыми он работает, и конвертацией тех же документов Word'а в чистый html, не оскорбляющий эстетического чувства поклонников пуристического кода.

Обе эти задачи решаются с помощью Aboword'а, позволяя избавиться, таким образом, сразу от любого из офисных монстров со всем их сопутствующим инвентарём. Не о таком ли текстовом процессоре мечталось пятнадцать лет назад, во времена существования программы Ted, создания Витусом Вагнером проекта Пингвин при галстуке и первых сочинений Владимира Игнатова (к слову сказать, автора термина «подоконник») о легких текстовых процессорах и русификации LyX'а? И похоже, что мечта эта нашла свое воплощение.

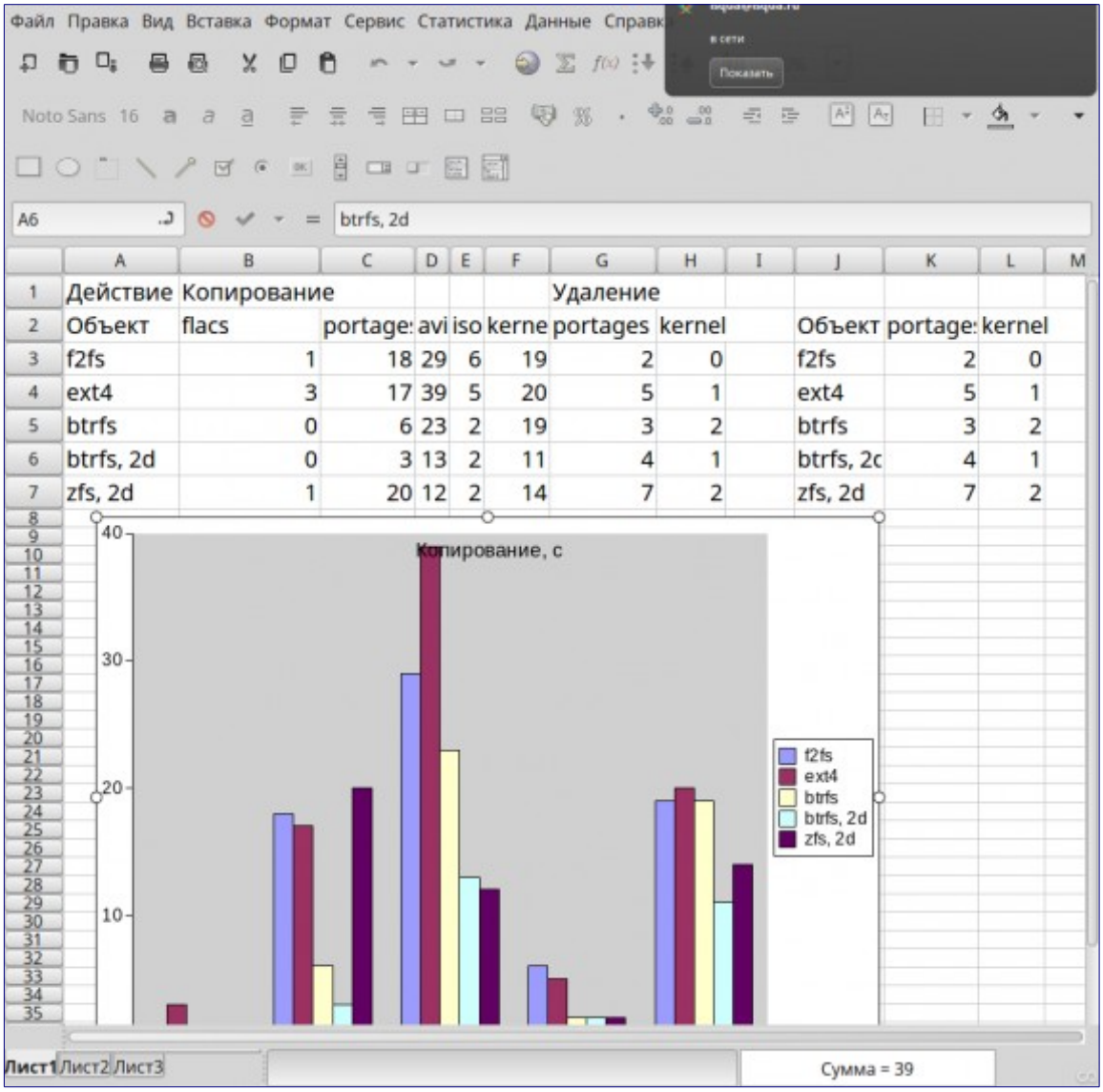
Правда, закончить этот очерк придётся всё-таки на минорной ноте: расхожее мнение о сверхъестественном быстродействии Abiword (особенно

часто повторяемое в сравнении с LibreOffice и Openoffice.org) основано на преданьях старины глубокой. Однако и в этом отношении всё не так суицидально, и по настоящему «тормознутость» Abiword'а сказывается только при работе с очень большими документами. На документах же маленьких, для которых он, собственно, и предназначен, заметить её невооруженным глазом невозможно.

Gnumeric

Электронные таблицы вообще — это весьма специфические компоненты офисных пакетов, причисляемые к ним по недоразумению: столь же обоснованно было бы отнести их к пакетам инженерным или научным. По крайней мере, в старое время лучшие представители этого семейства, такие как Lotus 123 или Quattro Pro. И Gnumeric продолжает эту традицию.

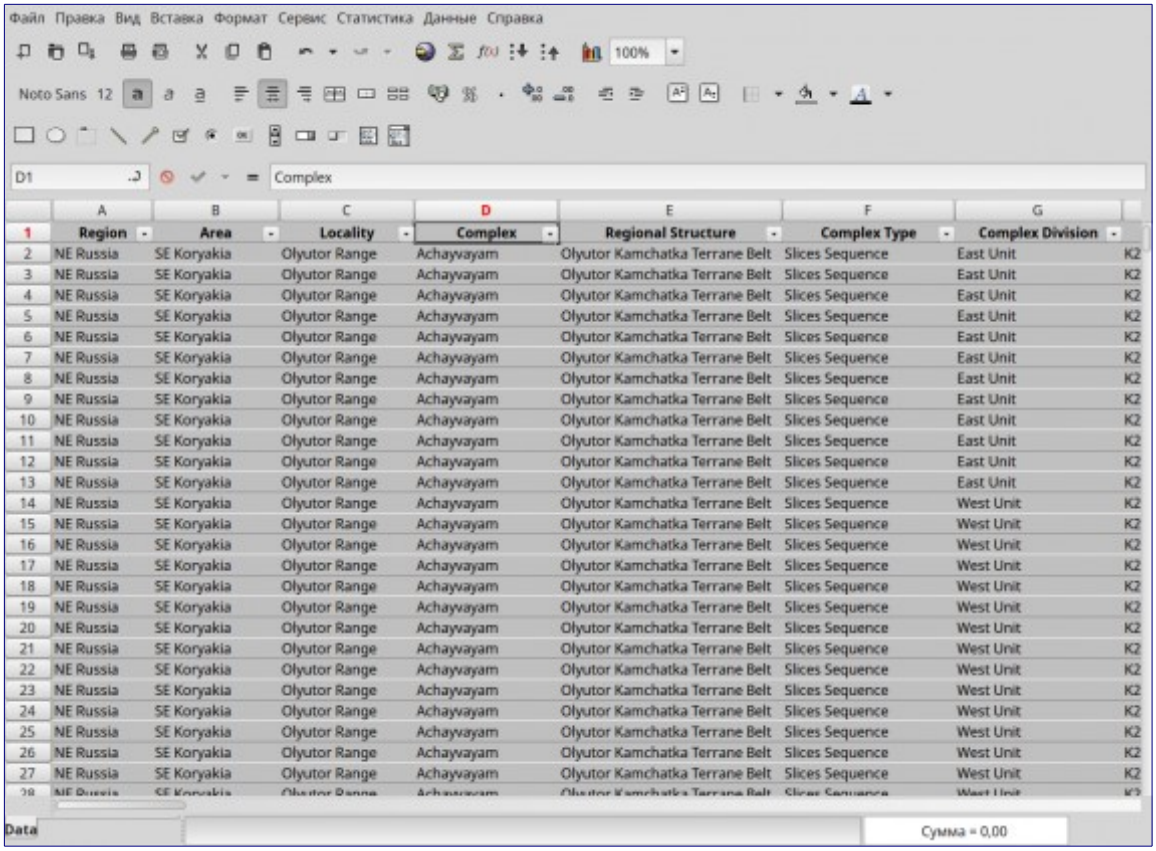
Gnumeric — это электронная таблица, входящая в состав GNOME Office, использующая библиотеки Gtk и gnomelibs. Если характеризовать Gnumeric несколькими словами, к нему можно применить те же эпитеты, что и к --; легкий и быстрый, не перегруженный излишней функциональностью, простой и элегантный:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Действие	Копирование					Удаление						
2	Объект	flacs	portage:	avi	iso	kerne	portages	kernel		Объект	portage:	kernel	
3	f2fs	1	18	29	6	19	2	0		f2fs	2	0	
4	ext4	3	17	39	5	20	5	1		ext4	5	1	
5	btrfs	0	6	23	2	19	3	2		btrfs	3	2	
6	btrfs, 2d	0	3	13	2	11	4	1		btrfs, 2c	4	1	
7	zfs, 2d	1	20	12	2	14	7	2		zfs, 2d	7	2	

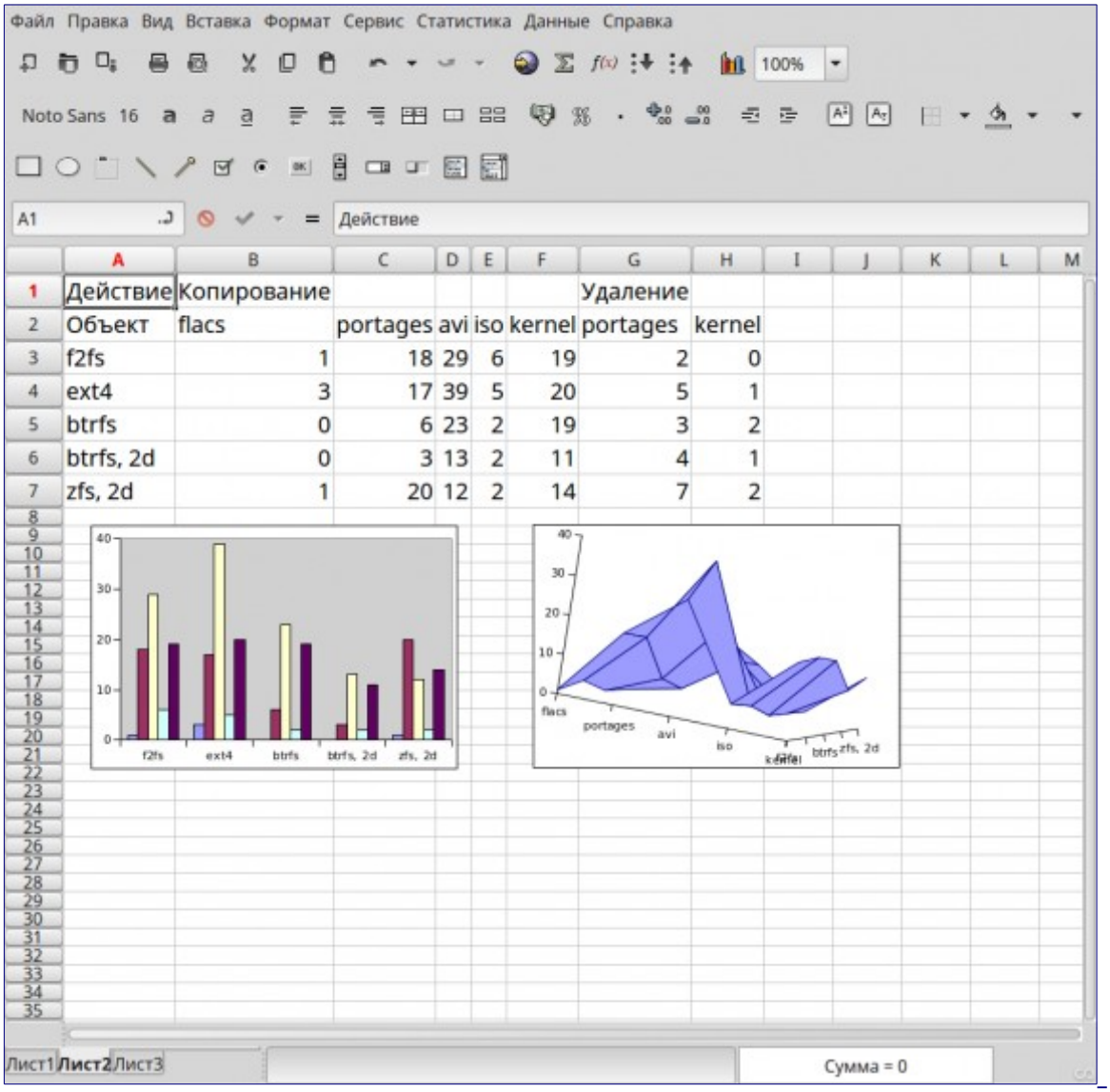
Как и AbiWord, Gnumeric использует свой формат файлов электронных таблиц, который так и называется — *.gnumeric. Однако список понимаемых им «чужих» форматов также обширен. Проверить его целиком возможности

не было, но с файлами в формате *.ods и *.xls Gnumeric справляется вполне успешно:



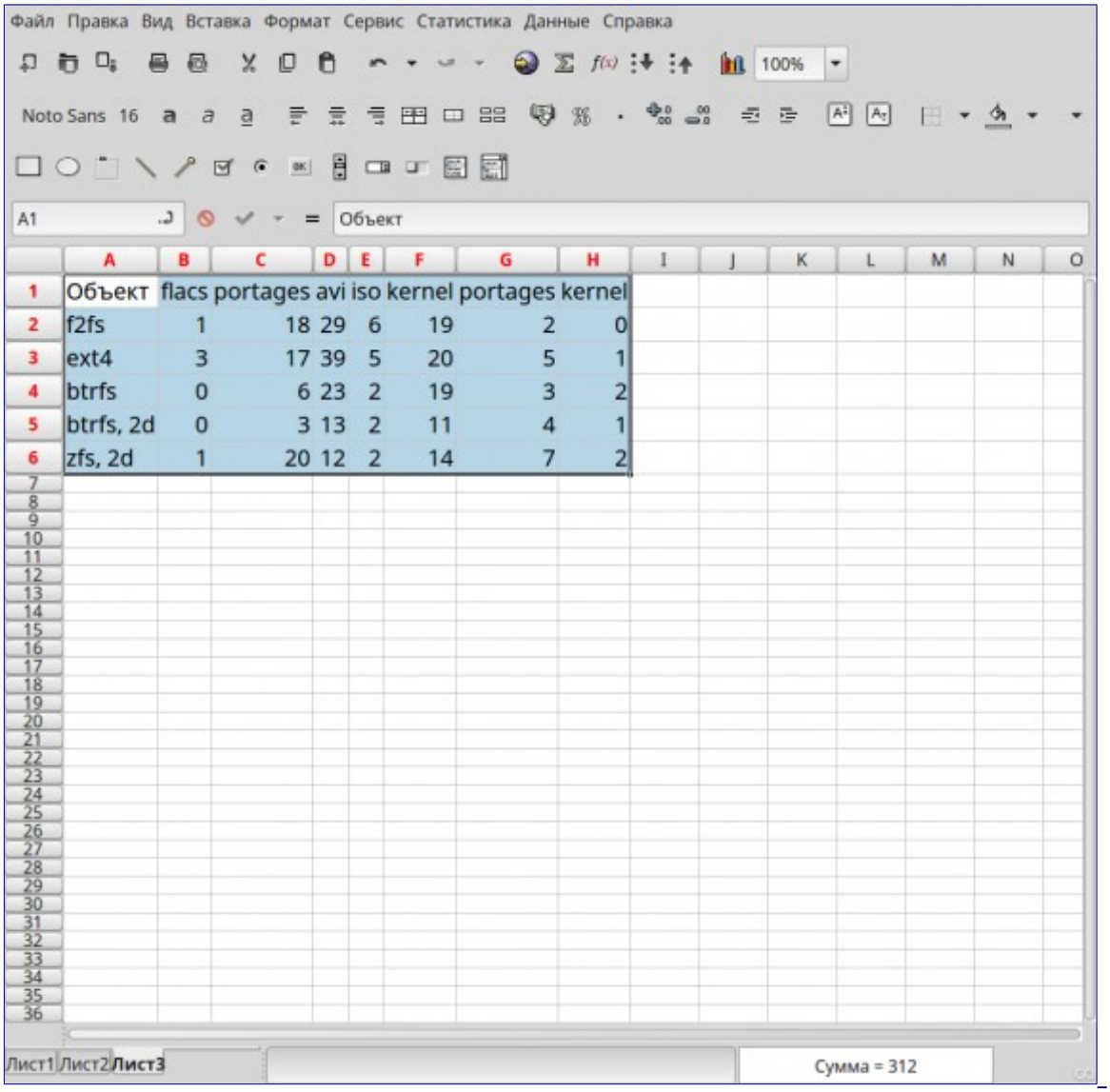
	A	B	C	D	E	F	G	
1	Region	Area	Locality	Complex	Regional Structure	Complex Type	Complex Division	
2	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
3	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
4	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
5	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
6	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
7	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
8	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
9	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
10	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
11	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
12	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
13	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	East Unit	K2
14	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
15	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
16	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
17	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
18	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
19	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
20	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
21	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
22	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
23	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
24	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
25	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
26	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2
27	NE Russia	SE Koryakia	Olyutor Range	Achayvayam	Olyutor Kamchatka Terrane Belt	Slices Sequence	West Unit	K2

Как уже говорилось во вступлении, главная сила Gnumeric'a — в изобилии его статистических и прочих технических функций, а также в построении разнообразных диаграмм, что можно просто проиллюстрировать скриншотом:



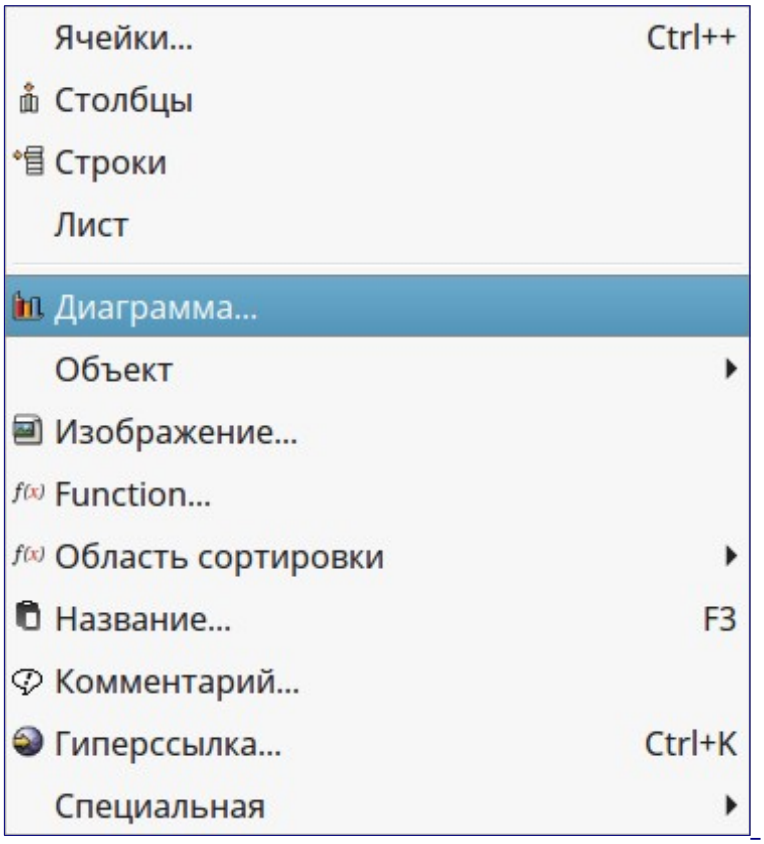
Действие	Копирование					Удаление	
Объект	flacs	portages	avi	iso	kernel	portages	kernel
f2fs	1	18	29	6	19	2	0
ext4	3	17	39	5	20	5	1
btrfs	0	6	23	2	19	3	2
btrfs, 2d	0	3	13	2	11	4	1
zfs, 2d	1	20	12	2	14	7	2

Ну а строятся диаграммы в Gnumeric'е очень просто. Для начала выделяется блок данных для будущего графика:

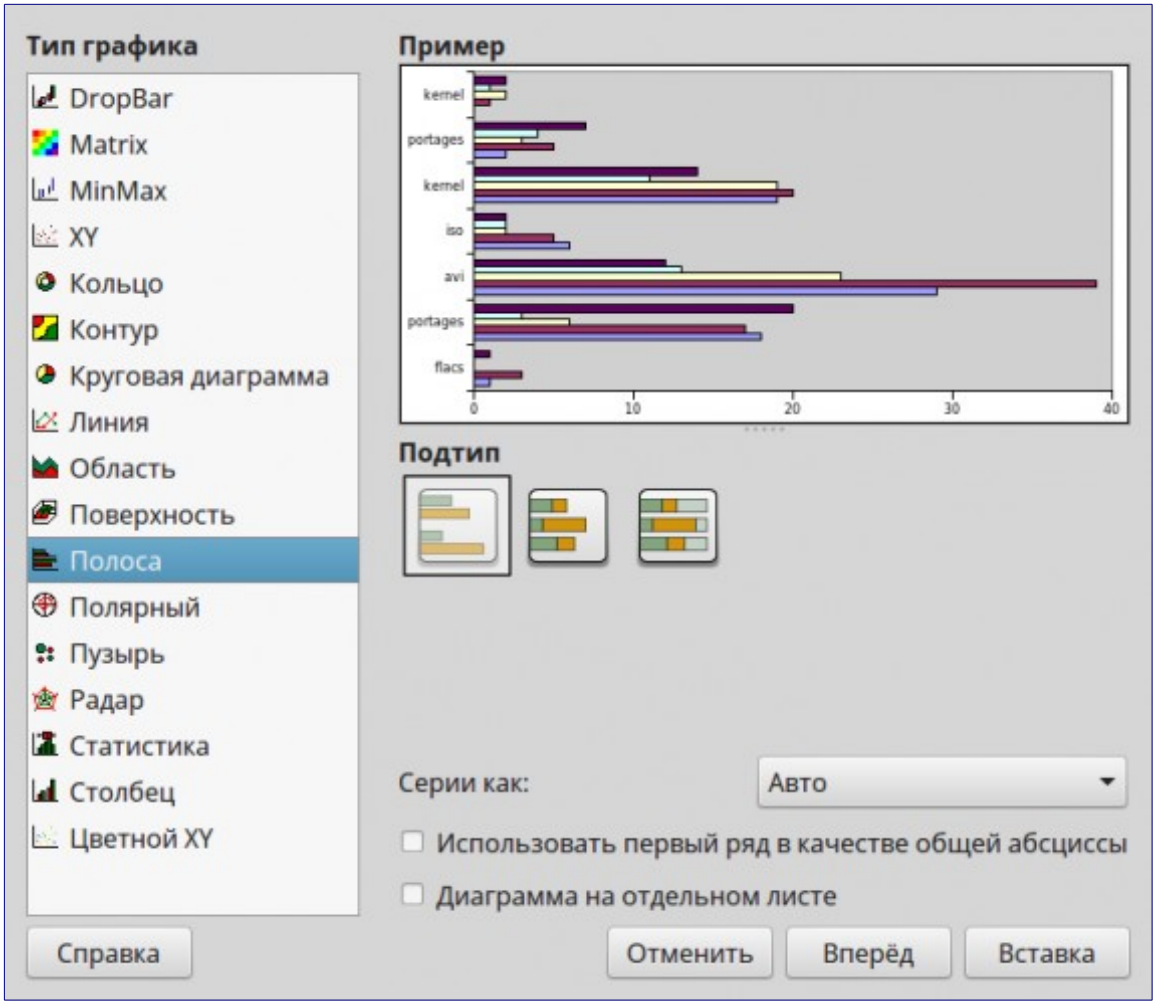


Объект	flacs	portages	avi	iso	kernel	portages	kernel
f2fs	1	18	29	6	19	2	0
ext4	3	17	39	5	20	5	1
btrfs	0	6	23	2	19	3	2
btrfs, 2d	0	3	13	2	11	4	1
zfs, 2d	1	20	12	2	14	7	2

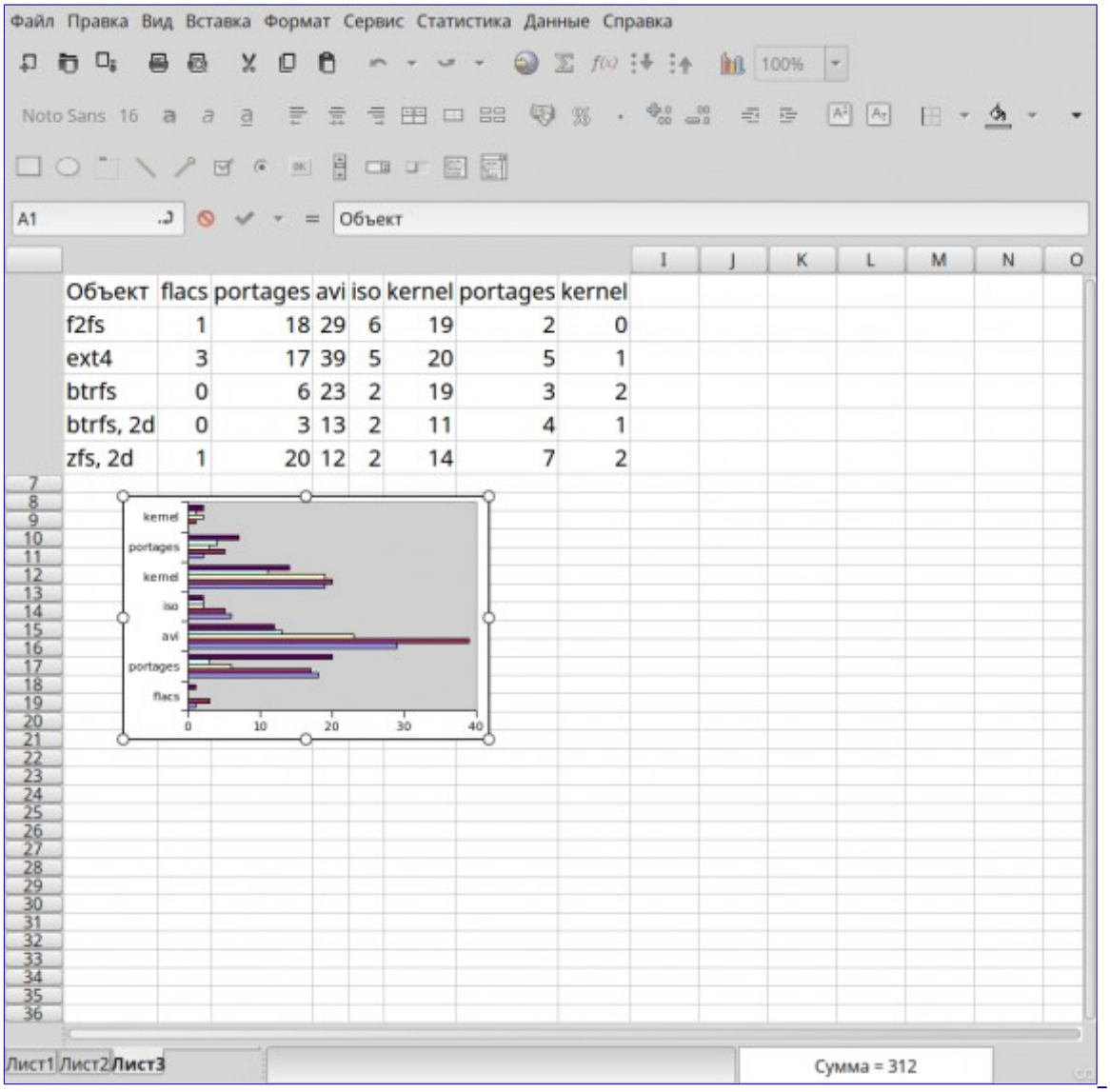
Затем — переход в меню: Вставка -> Диаграмма:



Далее — выбор типа диаграммы из предлагаемых вариантов, например, вот такой:



И — мышеклик в том месте рабочего листа, где эту диаграмму хотелось бы видеть:



Нынешняя моя сфера деятельности не даёт простора для практического применения всего функционала Gnumeric'а, поэтому я ограничился чисто умозрительными примерами. Но эта программа заставила меня вспомнить фразу, вычитанную на заре всесоюзной компьютеризации в каком-то буржуазном компьютерном журнале (цитирую по памяти):

Если задача не решается с помощью табличного процессора — скорее всего, она не решается вообще.

Конечно, это шутка — но доля шутки тут очень небольшая. Особенно применительно к Gnumeric'у.

Мультимедиа

> Но недавно случилась история —
> Я купил радиолу «Эстония»,
> И в свободный часок, на полчасика,
> Я прилёг позабавиться классикой.
> *Александр Галич,* ***Баллада о прибавочной стоимости***

Применителю любого дистрибутива Linux, дабы позабавиться классикой, нет необходимости тратиться на радиолу «Эстония» — он вполне может делать это, не отходя от своего рабочего инструмента. Не рискуя, к тому же, услышать своё фамилиё в неподходящем контексте.

Медиа-потребительство

И дистрибутив Mint тут не исключение: в стандартной установке его Cinnamon-редакции имеются:

- аудиоплейер Banshee;
- Totem — видеоплейер, который прекрасно проигрывает также и аудиофайлы;
- потоковый медиаплейер VLC.

Всё это — прекрасные средства для воспроизведения разного рода медиа-контента, сопровожающиеся всеми необходимыми библиотеками и кодеками для безукоризенной работы с медиа-форматами, которые некоторые отсалые личности полагают проприетарными, то есть частнособственническими (причём искренне считают их своей собственностью). Да вот только я всем этим богачеством не пользуюсь, и ничего не могу про него рассказать. Потому что с давних пор привык в качестве универсального аудио- и видеопроигрывателя применять mplayer — обычно в консольном исполнении, но не брезгую и графическими к нему «мордами» (последнее время в качестве таковой применяю GNOME MPlayer).

Программа mplayer — проигрыватель аудио- и видеофайлов. Её отличительная особенность в том, что как её зависимости идут кодеки для воспроизведения практически всех необходимых медиафайлов — по крайней мере, все необходимые мне. И всё это хозяйство, включая графические фронт-энды, имеется в официальном репозитории Mint. Так что если набрать в командной строке

```
$ apt install gnome-mplayer
```

то всё оно будет установлено как зависимости данного пакета. Поскольку в последнее время развитие mplayer'а несколько застопорилось, образовался его форк — mplayer2, обладающий некоторыми дополнительными функциями (хотя для нетребовательного потребителя мультимедийного контента, вроде автора этих строк, различий между ними не заметно). И именно он будет инсталлирован как бэк-энд графической «морды». Хотя в каталоге /usr/bin его исполняемый файл будет всё равно фигурировать как просто mplayer.

Про кодеки я уже сказал — их состав можно просмотреть в секции Зависимости вывода команды

```
$ apt show mplayer2
```

Сам по себе mplayer работает из командной строки — командой вида

```
$ mplayer path2/*.mp3
```

можно запустить прослушивание всей музыки из указанного каталога. Тем же образом можно и смотреть видео:



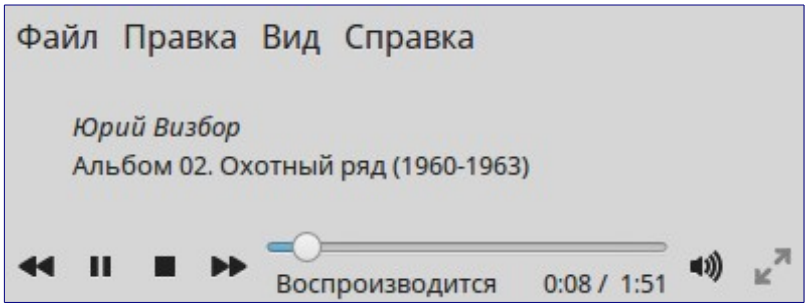
В обоих случаях никакого отвлекающего интерфейса нет. А минимально необходимые управляющие действия — пауза/продолжение и прокрутка вперёд/назад — выполняются с клавиатуры. В связи с чем даю небольшую шпаргалку по хоткеям этой программы:

- * и / — увеличение и уменьшение громкости;
- p и SpaceBar, а также правый клик мышью — пауза, повторное нажатие — продолжение;
- < и > — переход назад и вперёд по плейлисту;
- <- и -> — переход назад и вперёд на 10 секунд;
- Down и Up — переход назад и вперёд на 1 минуту;
- PgDn и PgUp — переход назад и вперёд на 10 минут;
- Enter — после паузы вперёд по плейлисту, по завершении его — окончание воспроизведения без выхода;
- U — окончание воспроизведения без выхода;
- q и ESC — остановки и выход из Mplayer'а.

Ну а использование программы GNOME MPlayer настолько тривиально, что можно ограничиться скриншотом окна программы при воспроизведении видео:



А при прогирывании аудиозаписей окно это выглядит ещё проще:



Тем не менее, не смотря на непритязательность интерфейса графической «морды» и его полное отсутствие в консольном варианте, свои потребительские функции mplayer выполняет вполне справно.

Оцифровка аудио-компактов

Некоторое время назад мне казалось, что задача оцифровки аудио-компактов утратила актуальность — всё, что могло (и должно) быть оцифровано, было оцифровано. Однако в связи с отмиранием устройств чтения оптических носителей вопрос этот опять стал злободневным, почему я и уделю посвящу ему данный мини-очерк.

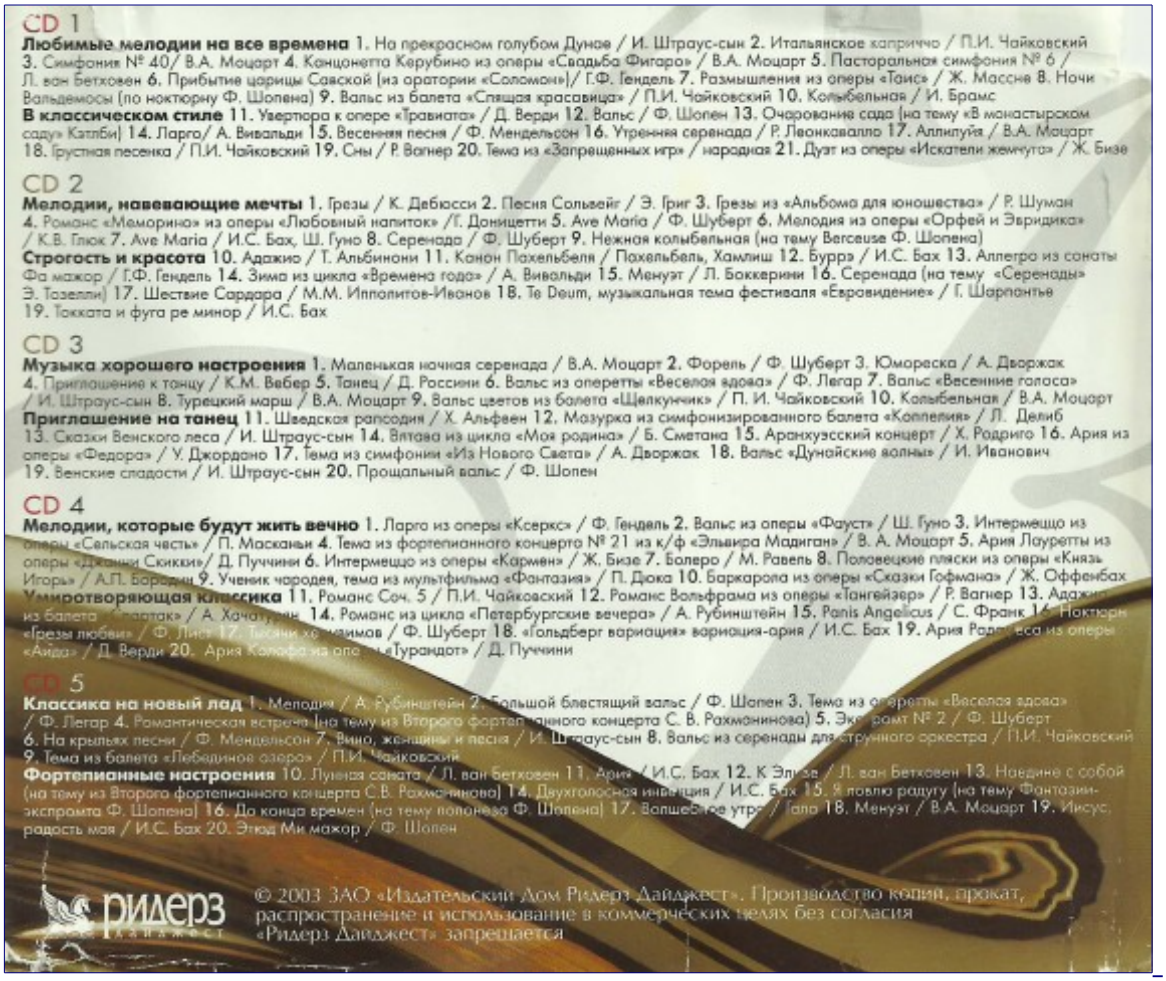
С давних пор для оцифровки всяческой музыки я использую программу Asunder — не потому, что считаю её лучше других, ибо сравнивать мне не с чем. Просто некогда она подвернулась мне под руку, показалась годной, и более добра от добра я не искал.

Asunder — программа для захвата треков с аудио-компактов, результаты которого можно сохранить в различных форматах: в виде чистых wav-файлов, в mp3, ogg и flac. В принципе, возможно и подключение дополнительных форматов, но оно потребует доустановки соответствующих кодеков.

Как работает Asunder — проще всего рассмотреть на конкретном примере. Тем более что давеча у меня был повод обратиться к ней: в закромах Родины обнаружился комплект из четырёх дисков под названием Золотая классика, который, безусловно, представлял собой непреходящую ценность. Вот лицевая сторона бокса:



А вот — оборотная, на которой приведено содержание всех дисков набора, их четыре штуки:



В стандартной инсталляции Cinnamon-редакции дистрибутива Mint программы Asunder нет, но она имеется в официальном репозитории, и потому её легко установить:

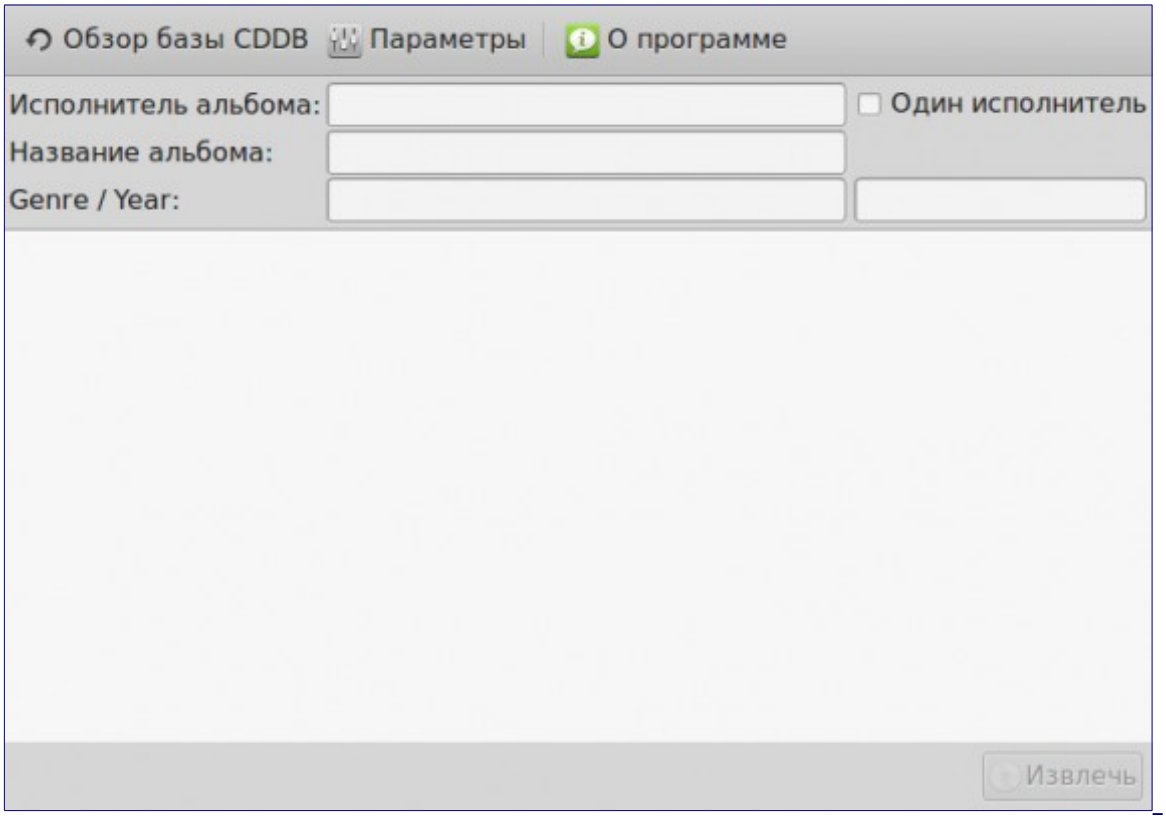
```
$ apt install asunder
```

Кроме того, неплохо также установить такие пакеты:

```
$ apt install lame flac
```

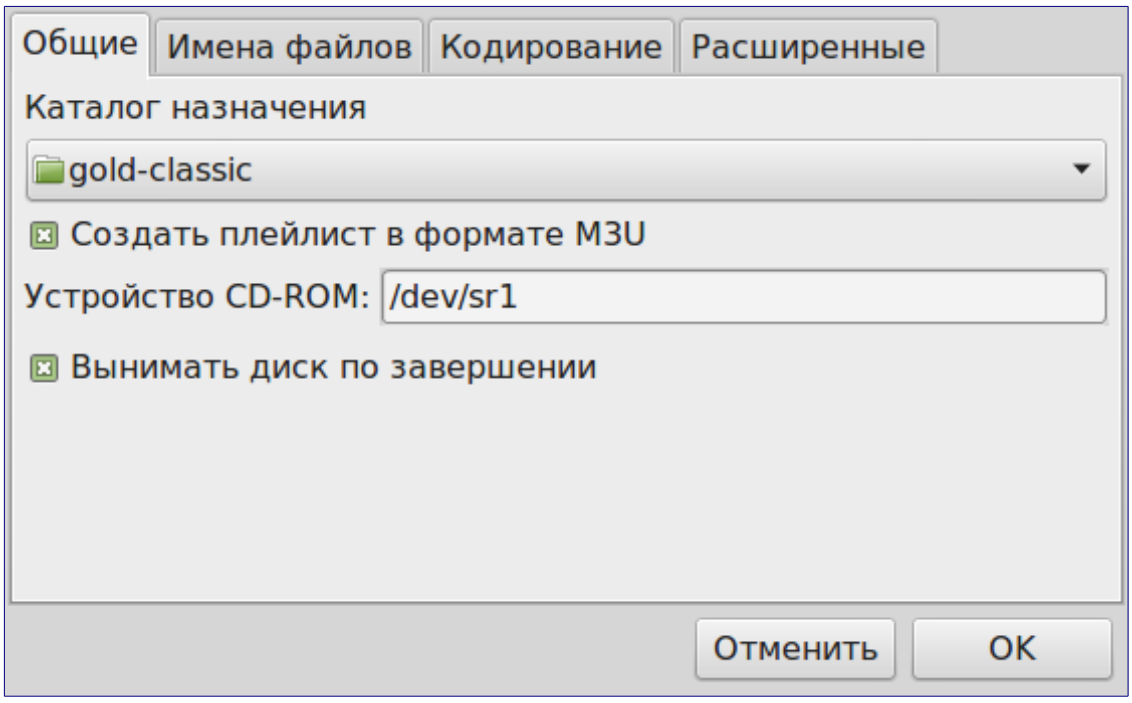
Они потребуются для сохранения звуковых дорожек в форматах mp3 и flac, соответственно.

После этого Asunder запускается из меню Аудио и видео и предстаёт в следующем виде:



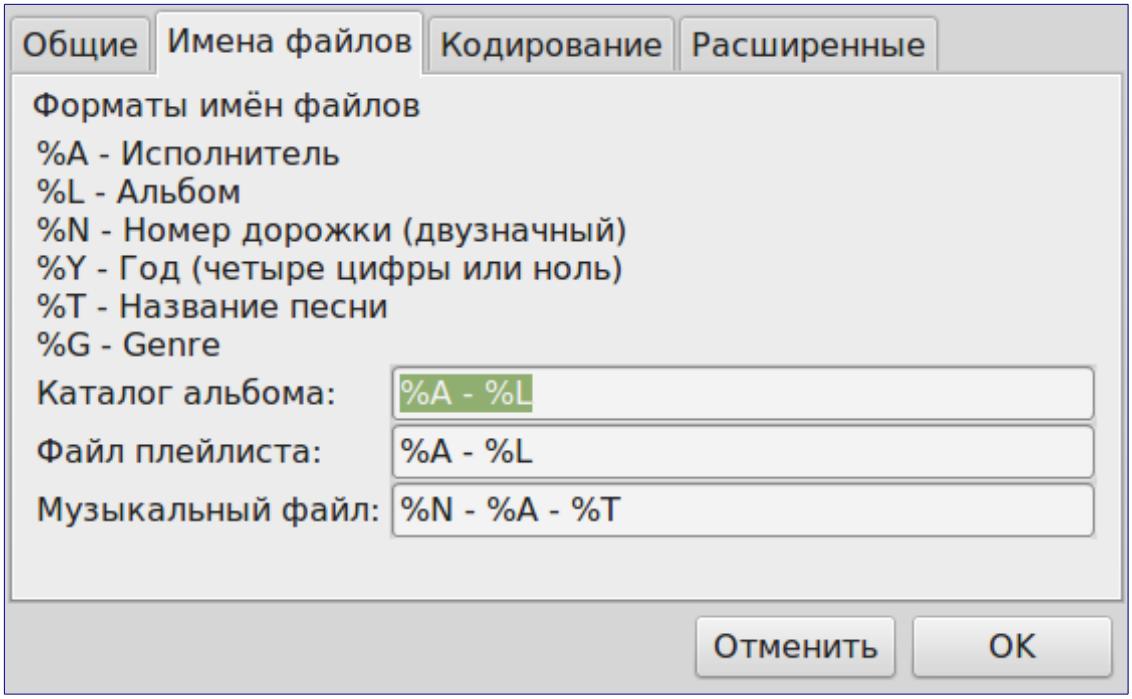
Прежде чем начинать с ним работу, желательно выполнить некоторые настройки. Как легко догадаться, они вызываются через пункт Параметры меню программу. Я приведу те настройки, которые сделал, занимаясь той самой Золотой классикой.

Для начала во вкладке Общие определяется каталог для будущих аудиофайлов и устройство чтения компакт-дисков. Последним по умолчанию является /dev/cdrom — символическая ссылка на файл реального устройства (обычно это /dev/sr0). Но у меня внутренний привод дышит на ладан, поэтому для оцифровки я подключит внешний, USB'шный, который получил имя /dev/sr0. Прочие опции указываются по желанию:

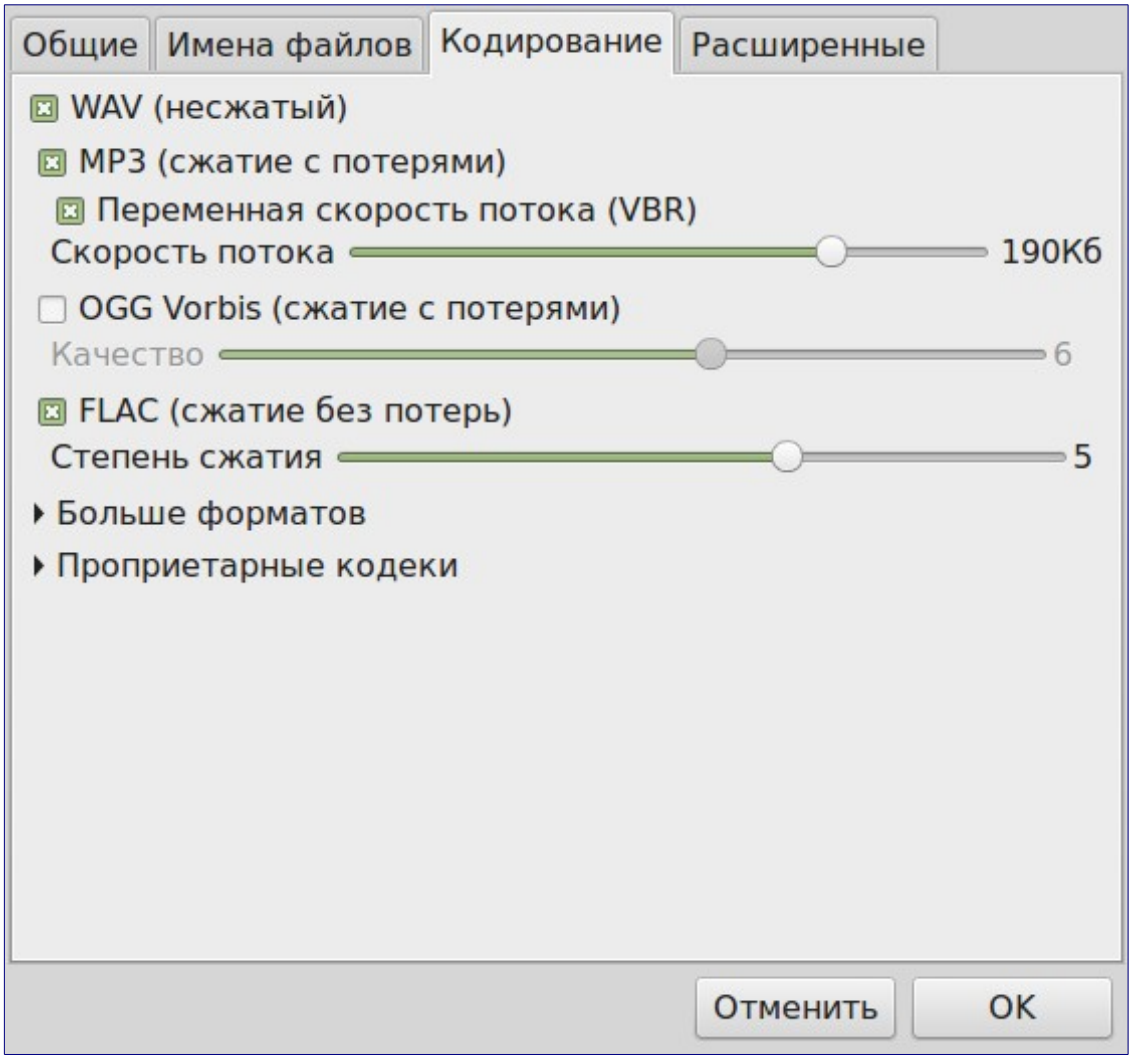


Во вкладке Имена файлов определяется система именования файлов,

полученных в результате оцифровки. Я оставил умолчания программы без изменений — они понятны без комментариев:

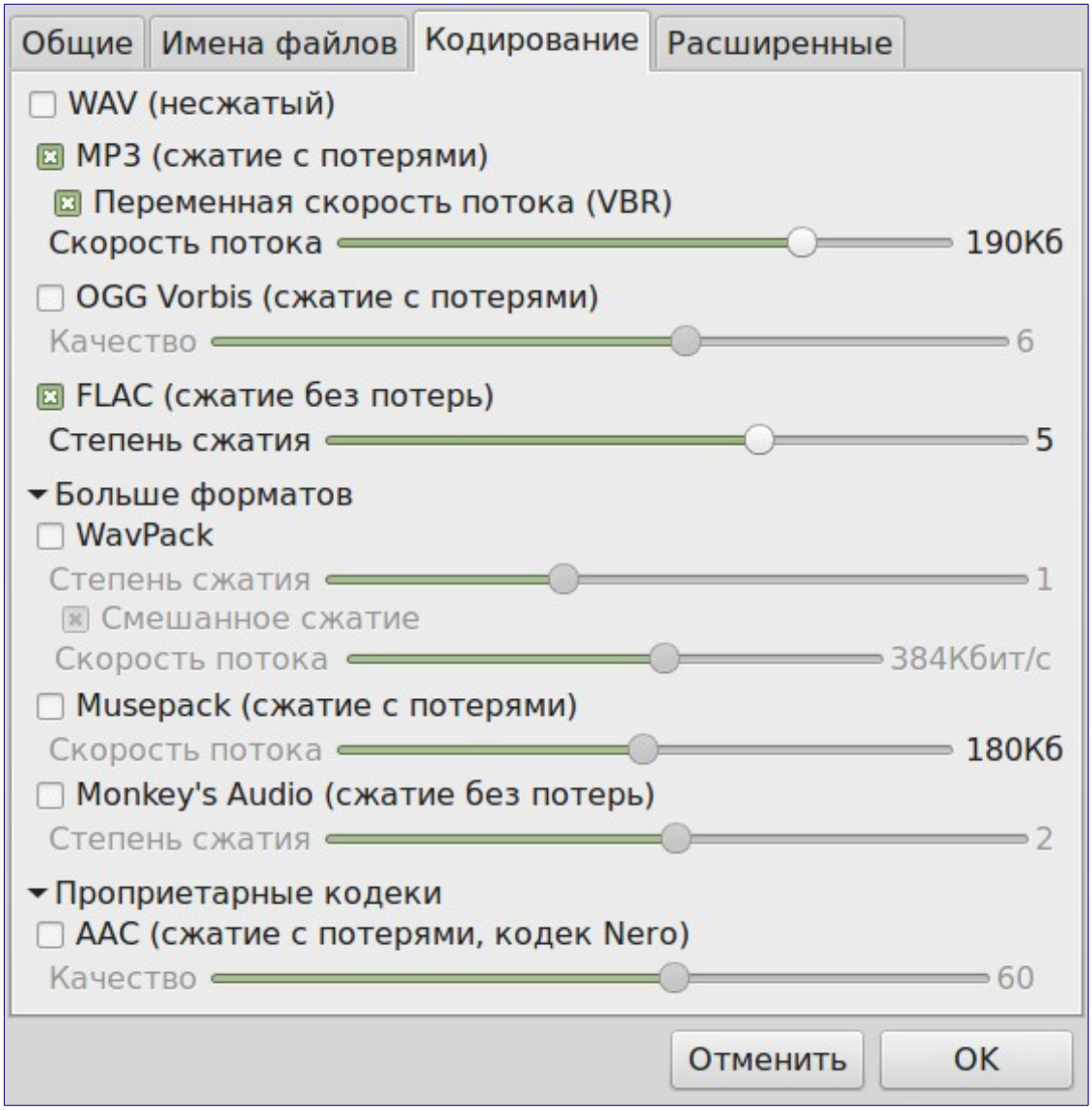


Далее, во вкладке Кодирование, указываются форматы выходных файлов — мне было достаточно сжатого с потерями MP3 и сжатого lossless, FLAC:

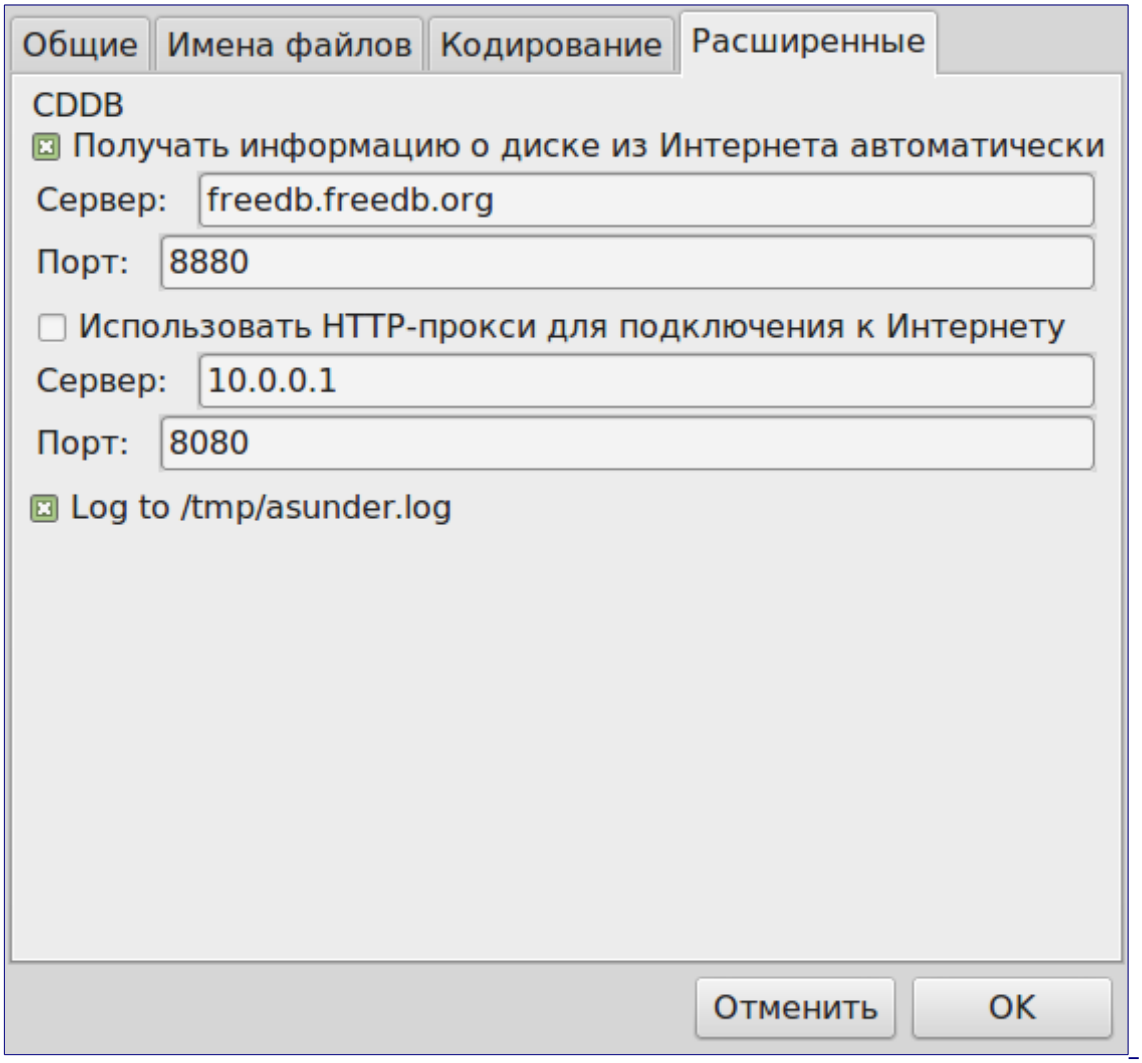


При желании можно указать ещё несколько форматов, в том числе и с

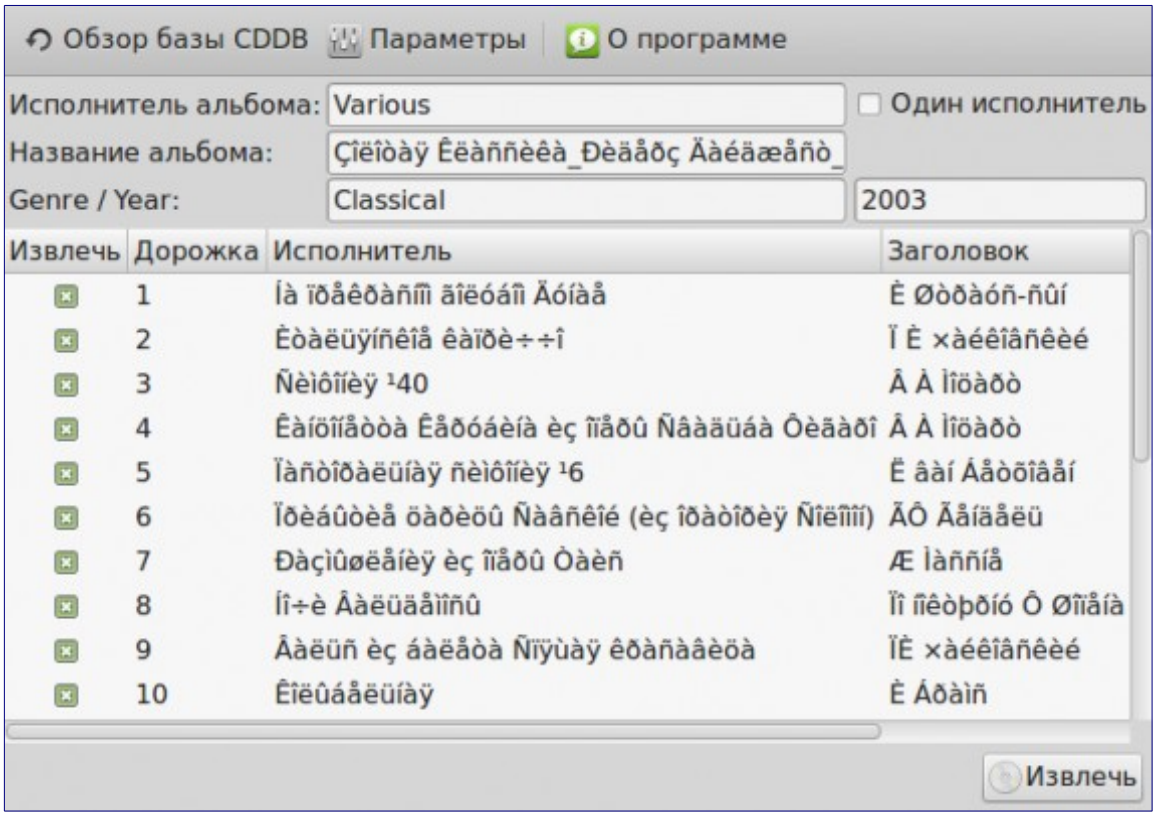
использованием проприетарного кодека Nero — правда, все они потребуют доустановки соответствующих пакетов:



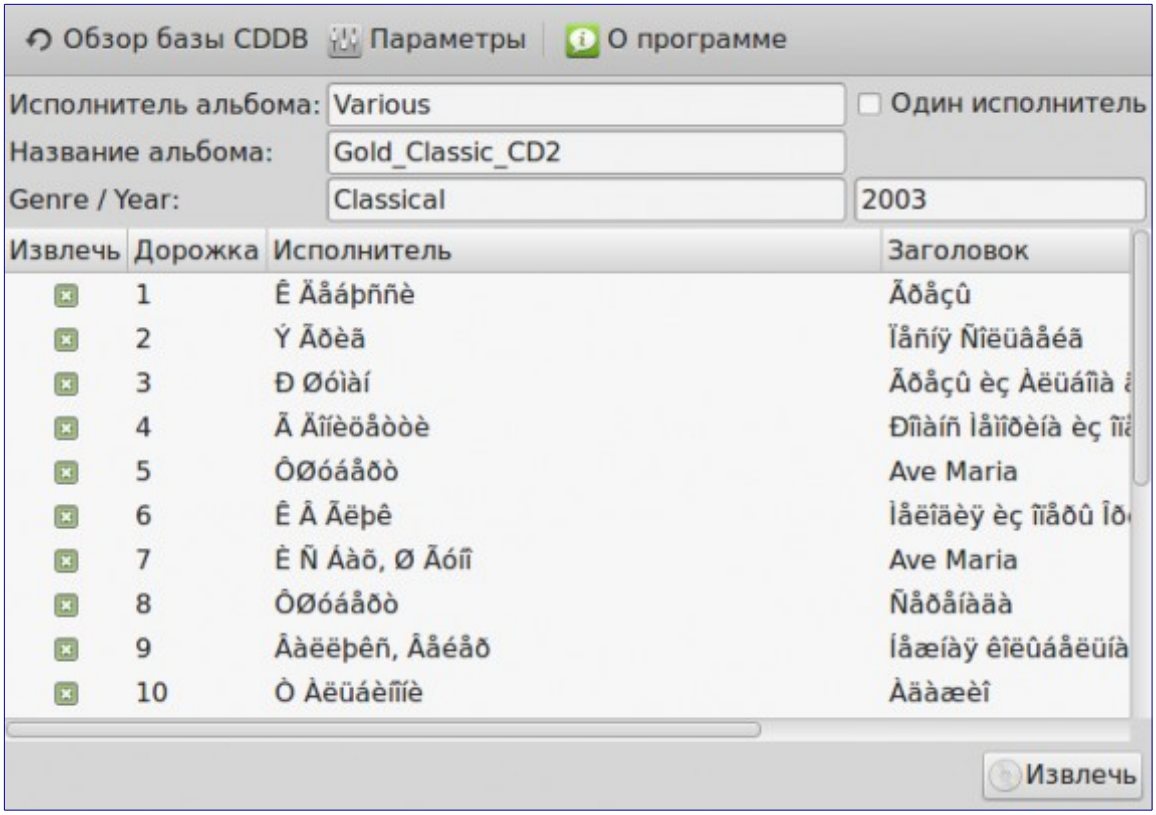
Во вкладке Расширенные я оставил умолчания без изменений, включив только на всякий случай запись log-файла:



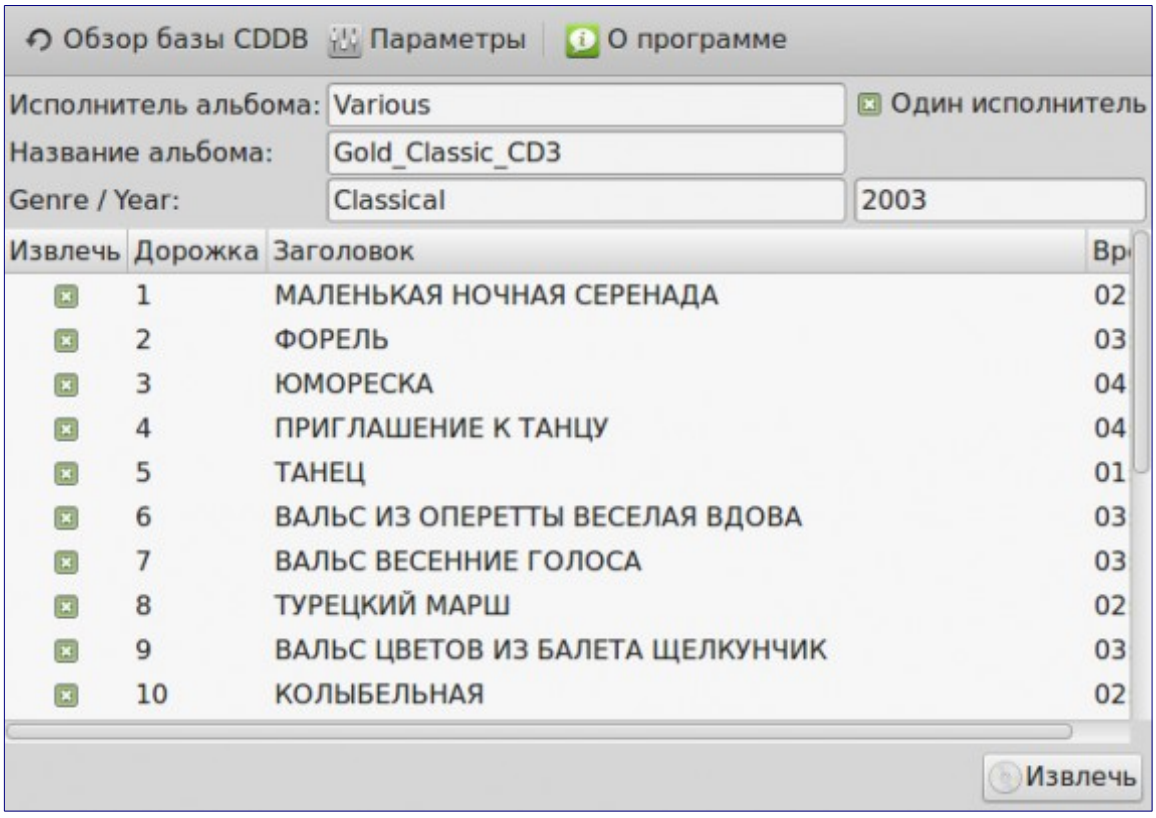
Теперь можно было вставлять первый диск набора, после чего перед глазами появляется следующее безобразие, вызванное несовпадением системной кодировки и кодировки компакта:



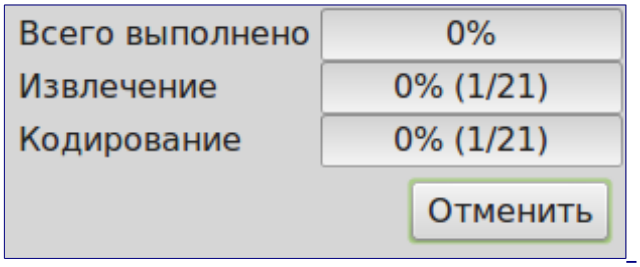
Название альбома я поправил, а переименовывать сами файлы мне было лениво:



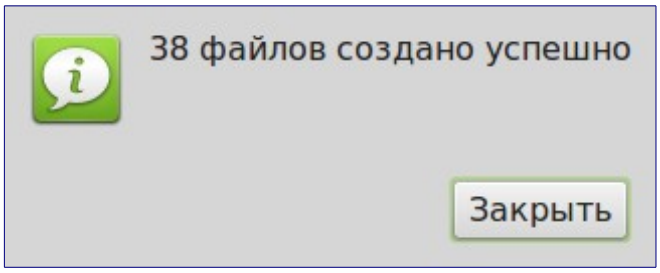
Впрочем, один из дисков набора прочитался почему-то по человечески:



Теперь оставалось только нажать кнопку Извлечь — и наблюдать за ходом процесса, который может занять немалое время, не смотря на то, что считывание трака и его кодирование осуществляются параллельно:



Однако рано или поздно процесс заканчивается, что знаменуется соответствующим сообщением:



По идее, тут лоток с диском должен бы выдвинуться, если, как у меня, отмечена соответствующая опция. Но с моим внешним приводом это не сработало, так что пришлось нажимать кнопку извлечения на самом агрегате.

Вот, собственно, и всё. Далее можно сепарировать файлы различных форматов (к сожалению, такой опции в настройках не предусмотрено), придать им осмысленные имена (автоматического перекодирования также нет), ну и, разумеется, слушать то, что получилось...

softRAID, LVM, ZFS

Следующие три очерка посвящаются рассмотрению довольно специальных вопросов — применению в Mint программного RAID, технологии LVM и системы размещёния данных ZFS. Тех, кого эти темы не интересуют, могут смело пропустить соответствующие страницы.

Общее введение

При наличии в машине двух или более накопителей возникает вопрос, как организовать работу с ними оптимальным образом. Очевидно, что разместить на одном устройстве корневую файловую систему, а остальные просто примонтировать к ней — не самое удачное решение, и к нему прибегают обычно не от хорошей жизни. И здесь для применительского десктопа напрашивается три варианта объединения накопителей, каждый из которых предоставляет ещё и некоторые дополнительные возможности:

1. программный RAID, позволяющий повысить быстродействие дисковой подсистемы (softRAID Level 0);

2. технология LVM, дающая возможность простого подключения дополнительных носителей и, при соблюдении некоторых условий, изменения размера файловых систем;

3. система размещёния данных ZFS, объединяющая в себе функции управления логическими томами и файловой системы.

Возможны и другие варианты, например, softRAID Level 1, обеспечивающий, опять же при некоторых условиях, надёжность хранения данных, или файловая система BTRFS, функцйионально сходная с ZFS. Однако первое решение для настольной машины (а в данных очерках рассматривается только этот случай, о серверах тут речи не будет) имеет не много смысла. Ибо эта самая надёжность гарантируется только тогда, когда есть возможность замены вышедшего из сторя диска аналогичным по объёму и, желательно, характеристикам — согласитесь, не частое явление для индивидуала-надомника.

Что же до BTRFS — что бы ни говорили о готовности этой системы к промышленному использованию (а настольная машина применителя для него самое что ни на есть промышленное использование), на сей счёт в народе (в том числе и у автора этих строк) существуют небезосновательные сомнения. Характерно, что в openSUSE, кажется, первой взявшей BTRFS на вооружение как умолчальной, она задействуется под корневую файловую систему, тогда как бесценные пользовательские данные по умолчанию предлагается размещать на XFS.

В силу вышесказанного в настоящей книге рассмотрены только три перечисленных варианта. Каждый из них имеет свои преимущества и недостатки. И поэтому ни один из них не может быть однозначно рекомендован во всех случаях жизни. Надеюсь, что следующие очерки дадут читателю некоторую информацию для того, чтобы выбрать вариант, подходящий именно для него. Своё же мнение по сравнению их я выскажу в общем заключении.

И ещё одна необходимая оговорка: все три варианта я рассматриваю исключительно в контексте хранения пользовательских данных, то есть, фигурально говоря, ветви /home файловой иерархии. Предполагается, что корень последней размещён на обычном дисковом разделе с традиционной файловой системой, которая, как говаривал Генри Форд Старший, может быть любой. При условии, что это будет Ext4, ибо все остальные нынче не дадут применителю ничего, кроме возможных проблем, в причины возникновения которых здесь вдаваться неуместно. А вот сказать пару слов об инструментах дисковой разметки — необходимо.

Инструменты дисковой разметки и форматирования

Как было сказано во вступительном очерке, далее речь пойдёт о прикручивании специальных систем размещёния данных к уже установленной системе. И любой из этих процессов в этом случае начинается с разметки разделов под них, а заканчивается созданием файловых систем (каковое далее для краткости буду называть форматированием, хотя в общем случае это не тождественные понятия). И потому начать разговор следует с обзора инструментария, для этих целей предназначенного.

Виды дисковой разметки

Некогда тема дисковых разделов подробно рассматривалась в любом руководстве по Linux и соплеменным системам, а также во множестве специальных документов, как в Сети, так и на бумаге. С этой процедуры начинало знакомство с Linux не одно поколение грядущих его применителей. А устрашающие к ней комментарии были непременным атрибутом «курса молодого линуксоида».

«Потом пришли другие времена» — и необходимость в столь подробных описаниях отпала. Да и число актуальных схем дисковой разметки резко поуменьшилось, сведясь к двум с половиной вариантам:

1. разметка в стиле msdos;

2. разметка в стиле gpt;

3. полварианта для любителей и ценителей — разметка в стиле bsd.

На полуварианте останавливаться не буду — те, кто держит на своей машине Linux параллельно с какой-либо BSD-системой, знают о нём не меньше меня. Тем более, что это, как и msdos, частный случай MBR-разметки, о которой сказать необходимо.

Разметка в стиле msdos возникла вместе с первыми IBM PC и их BIOS, предусматривающим Главную Загрузочную Запись (MBR — Master Boot Record). Она целиком умещается в так называемый нулевой сектор носителя, объёмом 512 байт. И в его части, отведённой под таблицу разделов, предусмотрено место для четырёх записей — то есть Primary Partitions. Большее количество разделов можно создать по «матрёшечному» принципу, путём объявления одного из первичных разделов Extended Partition.

Расширенный раздел выступает в качестве контейнера, в который последовательно, как в матрёшку, вкладываются один логический раздел и ещё один расширенный раздел. Последний, в свою очередь, выступает контейнером второго уровня, и может включать ещё один логический раздел и следующий по очереди расширенный, — и так до бесконечности. Правда, аналогия с матрёшкой - не совсем строгая, потому что для пользователя все эти вложенные разделы видятся как равноправные части «головного» Extended-раздела. Да и на счёт бесконечности — тоже несколько преувеличено: на самом деле существует практический лимит для восприятия логических разделов, определяемый числом 63.

Разметка в стиле GPT (GUID Partition Table) — это новый формат таблицы разделов на носителях информации (традиционных винчестерах, SSD-накопителях, флэшках, SD-картах). Как явствует из названия, он основан на Globally Unique Identifier (GUID) — статистически уникальных 128-битных идентификаторах всего на свете, в том числе и носителей.

Таблица разделов GUID (далее для краткости я буду называть её просто GPT) существенно больше, нежели MBR.. Она занимает первые 34 блока (с нулевого по 33-й). Из них нулевой блок занимает всё тот же MBR — точнее, его защищённая (или защищающая? — protected) копия, предназначенная для программ, не понимающих GPT. Благодаря ему, скажем, утилита fdisk опознаёт винчестер с GPT как единый раздел неизвестного типа, но на самом деле работать с ним не может.

Следующий блок — это оглавление таблицы разделов, в котором предусмотрено место для 128 записей. Это, соответственно, максимальное число разделов при разметке в GPT-стиле. Наконец, остальные 32 блока предназначены для записи данных о разделах.

Таблица разделов GUID существует в двух экземплярах: основной находится в первых 34 блоках носителя, а дублирующий (полная копия основного, за

исключением MBR) — в последних. При повреждении основной GPT (фиксируемом несовпадением контрольной суммы, хранящейся в оглавлении) она автоматически восстанавливается из таблицы дублирующей.

В Linux традиционно использовалась MBR-разметка в стиле msdos, и для последней предназначались соответствующие утилиты.

Несколько лет назад казалось, что железный конь GPT уверенно идёт на смену крестьянской лошади MBR: инсталляторы ряда популярных дистрибутивов, на которые не будем указывать пальцем, стали предлагать разметку в GPT-стиле как альтернативу, а некоторые — даже и по умолчанию. Однако довольно быстро оказалось, что это не даёт применителю ничего полезного, а вот некоторые проблемы с загрузчиком и особенно его восстановлением при сбоях создаёт. И даже самые прогрессисты от дистроения откатились обратно, сохранив поддержку GPT лишь в качестве опции.

Надо сказать, что Ubuntu и её последователи не поддавались модному влиянию, и в инсталляторах всего этого клана GPT-разметка не поддерживалась никогда, и не поддерживается по сию пору. Хотя с дисками, размеченными в GPT-стиле внешними программами все Ubuntu'иды работают вполне справно. Однако далее я буду говорить только про MBR-разметку — интересующимся прогрессом ради прогресса могу предложить почитать вот это.

CLI: инструменты разметки

В Linux для разметки диска в MBR-стиле из командной строки можно использовать:

- низкоуровневую утилиту командной строки sfdisk — инструмент очень гибкий, но сложный в обращении и требующий очень большой аккуратности — все изменения дисковой разметки совершаются там в реальном времени;

- интерактивную диалоговую программу fdisk — почти столь же гибкую, как и sfdisk, но более простую и, главное, более безопасную в обращении — изменения дисковой разметки происходят тут только после соответствующего подтверждения пользователем правильности своих действий;

- интерактивную меню-ориентированную программу cfdisk, которая считается ещё более простой в использовании, чем fdisk (для которого она служит фронт-эндом) и столь же безопасна с точки зрения сохранности данных;

Кроме этого, существует универсальная утилита parted, которая позволяет создавать не только дисковые разделы, но и файловые системы на них. В числе её функций также модифицирование существующих разделов — изменение размера, копирование и перемещёние. Однако платой за её универсализм является сложность использования без постоянной практики. А поскольку разметка диска — не то занятие, которому тпичный применитель Linux'а предаётся по три раза на дню, говорить о ней я не буду.

Утилита fdisk

Начнем с fdisk: именно им больше всего пугали в старые времена начинающих пользователей Linux, предлагая дружественные альтернативы типа Disk Druid. Однако при ближайшем рассмотрении выясняется, что ничего устрашающего в ней нет.

Происхождение fdisk теряется во мраке веков, уходя во времена первых UNIX для PC-архитектуры — насколько я понимаю, раньше необходимости в ней не было, а главными инструментами дисковой разметки были утилиты типа disklabel или bsdlabel. Мне не удалось также выяснить, когда эта утилита появилась в Linux. Могу только предполагать, что на самых ранних стадиях создания утилит обрамления для его ядра — т.н. linux-utils.

Для начала следует запомнить, что запуск команды fdisk в любом качестве, даже просто для получения информации о диске, возможно только с правами суперпользователя, каковые и надо обеспечить себе обычным для Mint образом, то есть через sudo.

Если команду fdisk дать без опций и аргументов, она выведет краткую справку об её использовании:

```
$ sudo fdisk



Usage: fdisk (-l) (-b SSZ) (-u) device

E.g.: fdisk /dev/hda  (for the first IDE disk)

  or: fdisk /dev/sdc  (for the third SCSI disk)

  or: fdisk /dev/eda  (for the first PS/2 ESDI drive)

  or: fdisk /dev/rd/c0d0  or: fdisk /dev/ida/c0d0  (for RAID devices)
```

В качестве аргумента команды фигурирует имя файла устройства — физического диска целиком. Поскольку в современных версиях ядра Linux все диски, вне зависимости от их интерфейсов (PATA, SATA, SCSI, SAS, USB) определяются единой подсистемой ATA-SCSI, на самом деле имена эти будут иметь вид /dev/sda, /dev/sdb и так далее.

Смысл опций команды fdisk следующий:

- l не предписывает выполнения каких-либо действий, а лишь выводит информацию о диске и его разделах, если таковые имеются;
- b задаёт размер блока — единицы измерения дискового пространства; по умолчанию, без указание этой опции, он равен физическому блоку (512 байт), прочие возможные значения кратны его размеру — 1024, 2048 или 4096 байт;
- u запускает fdisk, являясь опцией по умолчанию.

Перво-наперво посмотрим на информационную функцию fdisk, для чего запустим её следующим образом:

```
$ sudo  fdisk -l /dev/sdd
```

Ответом будет вывод примерно такого вида:

```
Диск /dev/sdd: 500.1 Гб, 500107862016 байт

255 головок, 63 секторов/треков, 60801 цилиндров, всего 976773168 секторов

Units = секторы of 1 * 512 = 512 bytes

Размер сектора (логического/физического): 512 байт / 512 байт

I/O size (minimum/optimal): 512 bytes / 512 bytes

Идентификатор диска: 0x000b24d0


Устр-во Загр     Начало      Конец      Блоки   Id  Система

/dev/sdd1          2048   629147647   314572800    5  Расширенный

/dev/sdd2     629149696   746337195    58593750   83  Linux

/dev/sdd3     746337196   863525803    58594304   83  Linux

/dev/sdd4     863525804   976773167    56623682   83  Linux

/dev/sdd5          2111    62504095    31250992+  82  Linux своп / Solaris
```

Если опустить аргумент команды, то аналогичные сведения будут выведены для всех физических дисков данной машины: сначала — общая информация о диске, включающая его размер, число головок, секторов и цилиндров, а затем для каждого существующего на диске раздела указываются его первый и последний цилиндры (символом + маркируются разделы, не занимающие последний цилиндр полностью), размер в блоках (физических или заданных опцией b), идентификатор типа файловой системы и его название.

Для каких-либо манипуляций с дисковыми разделами команду fdisk следует запустить в интерактивном режиме:

```
# fdisk /dev/sdd
```

Что можно сделать без всяких опций, но вот указание аргумента тут будет обязательным.

После этого мы получаем в свое распоряжение некий интерфейс, требующий ввода определенной команды, исполнение которой сводится к ответу на несколько вопросов. С полным списком доступных команд можно ознакомиться благодаря прекрасной системе помощи, вызываемой командой m.

Так, команда p выведет текущий список дисковых разделов с указанием их типа и размера. Далее, разделы можно создавать (командой n) или удалять

(командой d), однако до команды записи изменений (w) никаких необратимых действий, могущих разрушить ранее существовавшую разметку (и, соответственно, файловые системы и данные, к ней привязанные), не последует: неудачно созданные разделы можно удалить и на их месте создать новые. И в любой момент командой q можно без всяких последствий выйти из программы.

При создании раздела средствами fdisk сначала определяется, будет он первичным (primary) или расширенным (extended). Рассмотрим сначала первый случай. При нем далее просто указывается номер раздела (от 1 до 4). В этих пределах номер может быть любым — можно сначала создать раздел 2, а потом 1, или даже весь диск отвести под раздел 4. Номер раздела останется на века: именно он будет идентифицировать файл устройства, соответствующий созданному разделу (например, /dev/sda2, или /dev/sdb1).

Далее задается начальный цилиндр создаваемого раздела (по умолчанию - первый свободный, для пустого диска — просто первый). Однако никто не мешает указать любой другой цилиндр в качестве стартового (на неразбитом пространстве, разумеется). А потом — конечный цилиндр (по умолчанию — последний физический на неразбитом дисковом пространстве), или просто размер раздела в каких-либо общепринятых единицах измерения информации, например, +300M; и +, и M (или что-либо аналогичное) — обязательны, иначе объём диска окажется равных ровно трёхстам цилиндрам.

В современных версиях fdisk возможно указание объёма в двух системах единиц (не считая цилиндров). Во-первых, он может задаваться в том, что мы испокон веков привыкли называть мегабайтами и гигабайтами, то есть степенях двойки. Однако ныне авторитетные товарищи утверждают, что такие единицы измерения должны величаться мебибайтами, гибибайтами и так далее. Их следует указывать так: +1000MB, +10GB и так далее.

А можно определять объём раздела и в «настоящих», с точки зрения пуристов от метрологии, мегабайтах и гигабайтах, представляющих собой собой степени десятки — тех, в которых производители жёстких дисков издревле указывают размеры своей продукции. И тогда он определяется таким образом: +1000M, +10G etc.

При задании размера в единицах, отличных от цилиндров, он всегда будет округляться (по обычным правилам округления) до ближайшего числа, кратного целому количеству последних. Так что не следует удивляться, если вместо искомого раздела в 20 Мбайт возникнет 16-мегабайтный, а вместо 22-мегабайтного — раздел в 24 Мбайт.

При создании расширенного раздела сначала все происходит точно также — задание номера (очевидно, что в том же диапазоне 1--4), указание начального цилиндра и конечного (или — объёма в мегабайтах). Однако это ещё полдела, нужно поделить расширенный раздел на разделы логические. И потому при следующей команде на создание раздела нам будет предложен уже выбор между первичным (если число последних ещё не исчерпано) и логическим (ведь второй extended-раздел средствами fdisk создать нельзя):

```
Command (m for help): n
```

Command action

l logical (5 or over)

p primary partition (1-4)

Дальше же логический раздел создается аналогично первичному.

Для каждого вновь создаваемого средствами fdisk раздела (первичного или логического) по умолчанию устанавливается идентификатор типа файловой системы Linux native (83 в шестнадцатеричном исчислении). Расширенный же раздел также автоматически получает правильный идентификатор своего типа — 5. Однако типы эти не есть нечто неизменное. Более того, по крайней мере в одном случае, при создании раздела подкачки, изменение типа раздела — необходимость. Это потребуется также и для использования таких технологий, как Software RAID или LVM, о которых будет говориться позднее.

Делается это командой t, после чего запрашивается номер раздела, тип которого должен быть изменен, а затем — идентификатор желаемого типа. Полный список поддерживаемых типов файловых систем (и их идентификаторов) можно вывести командой l. Напомню, что идентификатор типа файловой системы раздела — отнюдь не файловая система, которая на нем размещается. И на разделе Linux native, как это подчеркивает название, можно создать любую файловую систему из числа тех, которые поддерживаются Linux в качестве родных (ext2/ext3, ext4, XFS, ReiserFS, JFS, btrfs, NILFS2).

Теоретически fdisk позволяет присвоить созданному разделу идентификатор типа почти любой из мыслимых файловых систем — от FAT12 до Free-, Open- и NetBSD. Однако сами по себе файловые системы средствами fdisk не создаются, и потому для разделов любого типа в дальнейшем потребуется их форматирование специальными командами типа mkfs, о которых будет говориться в соответствующей рубрике.

Сказанного, надеюсь, достаточно, чтобы осознать великое достоинство fdisk — исключительную гибкость: можно определить раздел строго определенного размера и точно позиционировать его на диске. Или зарезервировать в любом месте накопителя неразбитое пространство, с двух сторон окруженное созданными разделами.

Сложность же применения fdisk — кажущаяся: благодаря системе подсказки дисковые разделы создаются легко и непринуждённо. Тем не менее, если таки она удручает своей недостаточной наглядностью, можно воспользоваться утилитой cfdisk с меню-ориентированным интерфейсом.

Утилита cfdisk

Как уже говорилось, утилита fdisk часто оказывает устрашающее действие на начинающих пользователей. И потому, идя навстречу их невысказанным пожеланиям, Кевин Мартин (Kevin E. Martin) написал к ней консольный фронт-энд с меню-ориентированным интерфейсом, получивший имя cfdisk.

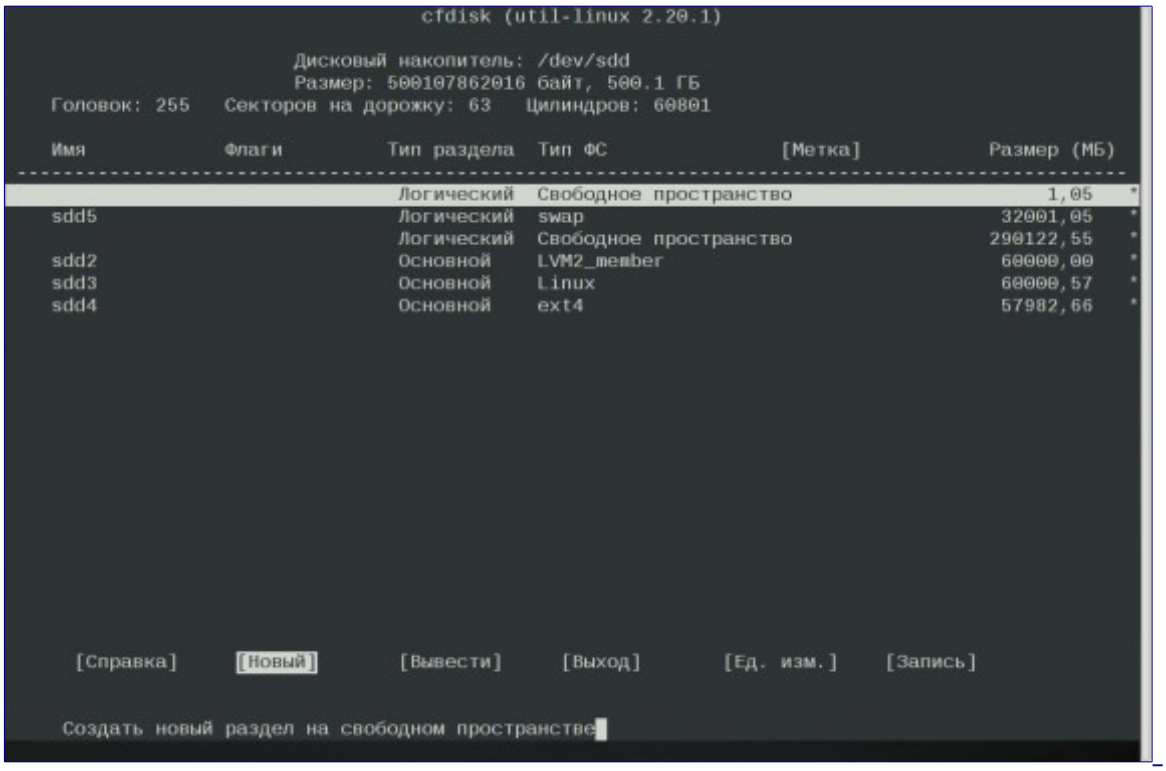
Утилита cfdisk описывается в литературе гораздо реже, хотя традиционно она считается более удобной, чем fdisk — впрочем, это субъективно и зависит от привычки.

Запустить cfdisk можно одноименной командой, с указанием имени дискового устройства в качестве аргумента:

```
# cfdisk /dev/sdb
```

Если аргумент в командной строке опущен — по умолчанию команда будет исполнена для первого физического диска машины.

Разумеется, для использования утилиты требуются права администратора. Если попытаться запустить её от лица обычного пользователя — программа стартует с сообщением. А после запуска программы (в консоли или окне терминала) мы видим следующую картину:



На ней выводится информация о диске, первом физическом или том, что был указан в качестве аргумента (имя файла устройства, размер, число головок, секторов, цилиндров), таблица существующих разделов (если, кончено, они действительно существуют) и меню из следующих пунктов: Bootable, Delete, Help, Maximize, Print, Quit, Type, Units, Write. Это — для диска с существующими разделами. Если же диск не разбит (или в таблице разделов курсор зафиксирован на неразбитом пространстве), меню ограничивается пунктами Help, New, Print, Quit, Units, Write.

Смысл пунктов, думаю, понятен из их названий, как и возможности программы вообще. Замечу лишь, что здесь, как и в fdisk, до выбора пункта Write (в котором будет запрошено подтверждение действия) никаких необратимых изменений не происходит: через Quit всегда можно покинуть программу без боязни за существующие разделы и данные на них.

И ещё: по умолчанию размеры разделов в таблице указаны в тех мегабайтах, к которым мы привыкли — 220 байт, которые, как нынче

считается, положено называть мебибайтами. Однако через пункт Units (сиречь единицы измерения) можно переключиться на показ его в секторах или цилиндрах. Для создания раздела выбирается пункт New, выводящий подменю: Primary, Logical, Cancel.

После выбора типа раздела просто задается желаемый его размер (в мегабайтах) и запрашивается, приписать ли раздел к началу диска или его концу. А потом остается только сохранить разбиение в таблице разделов выбором пункта Write (повторяю, с запросом подтверждения, и не просто как у, а вводом полного слова yes — дабы дать дополнительные мгновения на раздумье).

Таким образом, все происходит почти также, как в fdisk. Это и не удивительно: cfdisk по сути лишь интерфейсная для fdisk оболочка. Хотя cfdisk несколько менее гибок: например, раздел в середине неразбитого дискового пространства создать нельзя.

Утилиты форматирования

Утилит для форматирования в Linux существует столько же, сколько и поддерживаемых её ядром файловых систем: чтобы убедиться в этом, достаточно набрать в командной строке mkfs и нажать Enter, что даст примерно такую картину:

```
mkfs        mkfs.ext2    mkfs.ext4dev  mkfs.minix   mkfs.reiserfs

mkfs.bfs    mkfs.ext3    mkfs.fat      mkfs.msdos   mkfs.vfat

mkfs.cramfs mkfs.ext4    mkfs.jfs      mkfs.ntfs    mkfs.xfs
```

И это — оглашение не всего списка: его легко поплнить установкой пакетов поддержки таких файловых систем, как nilfs, f2fs и других, о которых я и не слыхал.

Зато из данного списка становится очевидным, как выполнять форматирование: достаточно дать команду на создание желаемой файловой системы и имя файла устройства (то есть дискового раздела) в качестве её аргумента, например:

```
$ sudo mkfs.ext4 /dev/sdg1
```

Как легко догадаться, конечно же, команда даётся от имени root'а. А результатом её работы будет создание на указанном разделе файловой системы ext4fs.

Можно ограничиться и универсальной командой mkfs — но в этом случае, кроме аргумента, потребуется ещё и опция -t с явным указанием файловой системы. Прочие опции всего семейства команд mkfs специфичны для отдельных файловых систем, и говорить о них тут не будем. Как воздержимся и от обсуждения вопроса, какая из файловых систем — самая рассмая и гарантирующая своему применителю счастие немыслимое.

Про графические морды и особенно про GNOME Disks

Как уже говорилось, средств для управления носителями и разделами для них в Linux'ах немало. И если большинство применителей «со стажем»

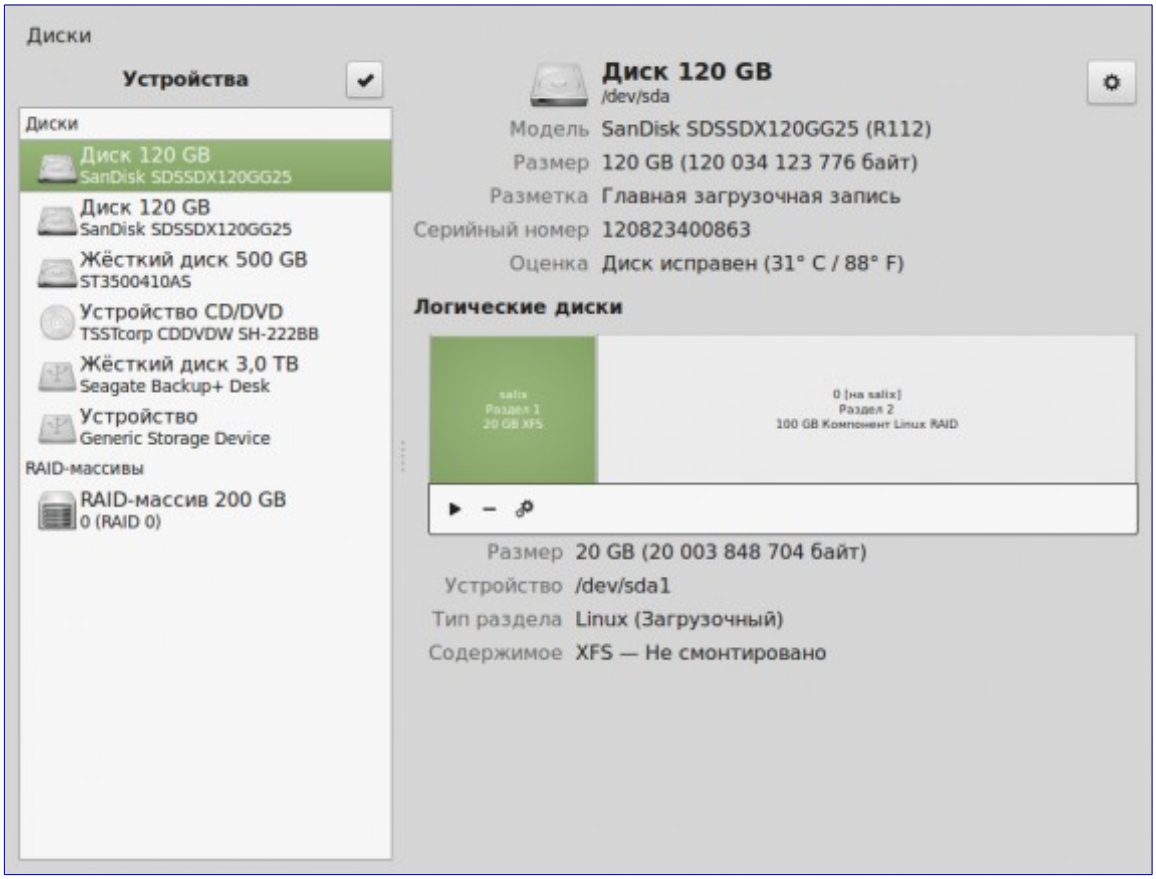
используют в этих целях набор специализированных утилит командной строки, то применители начинающие отдают предпочтение графическим фронт-эндам, интегрирующим функционал всех перечисленных инструментов. Впрочем, последними не брезгуют и те из опытных применителей, кто не одержим фанатизмом и уже переболел детской болезнью крутизны. Среди таких фронт-эндов наибольшую известность и распространение получила программа GParted. Она же применятся обычно во всякого рода Live-дистрибутивах ремонтно-спасательной ориентации, вроде Parted Magic и GParted Live.

Автор этих строк тоже не пренебрегал программой GParted, когда ему требовалось разметить какой-либо носитель на скорую руку (или когда просто лениво было обращаться к инструментам CLI). Однажды, уже в бытность применителем Mint, такая потребность возникла у него в очередной раз. И потому он собрался обратиться к ~~знакомым пистолетам~~ знакомому и привычному фронт-энду. Однако следствие показало, что в Mint 17 таковой в штатном комплекте отсутствует. А вместо него предлагается некий gnome-disks, фигурирующий в разделе Стандартные главного меню Cinnamon'a под простым именем — Диски. Хотя доустановить GParted из репозитория труда бы не составило, я решил опробовать этот инструмент. Ибо до сих пор только слыхал о нём — и не лучшие отзывы. В частности, что в последних его версиях отказались от поддержки softRAID, мотивируя обычным для GNOMEделателей аргументом: Народу это не нужно.

Забегая вперёд, скажу, что опасения по части softRAID оказались не напрасными: в том варианте, в котором gnome-disks присутствует в Mint'e, он позволяет создавать разделы с данным идентификатором — и ничего более. Для дальнейших действий приходится всё равно обращаться к утилите mdadm, но это будет следующим номером нашей программы.

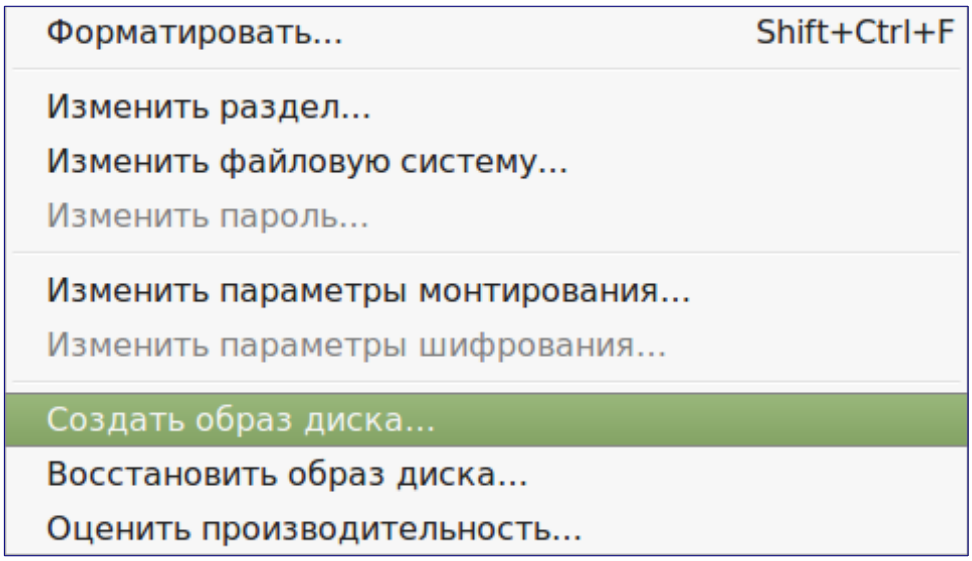
Однако начну по порядку. Запустить gnome-disks можно или через CLI, одноимённой командой, или из меню. Обращаю внимание — в отличие от GParted, пароль пользователя в момент запуска не запрашивается: его спросят только перед выполнением действий, которые на самом деле требуют привилегий администратора.

Будучи запущенным на моём десктопе, gnome-disks показывает следующую картину:

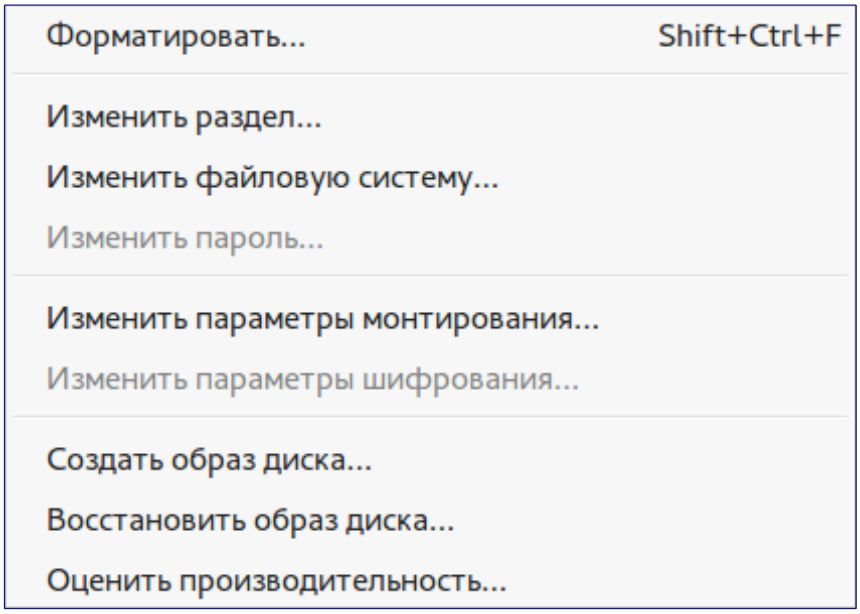


Что соответствует реалиям моей дисковой подсистемы, в детали описания которой я сейчас вдаваться не буду — задача, как уже было сказано, передо мной стояла очень частная. Отмечу только, что, вопреки опасениям, мой программный RAID был распознан, и для него были доступны некоторые манипуляции, типа добавления или удаления разделов. Чего я, по понятным причинам, не делал. Но пару слов о возможностях программы вообще скажу.

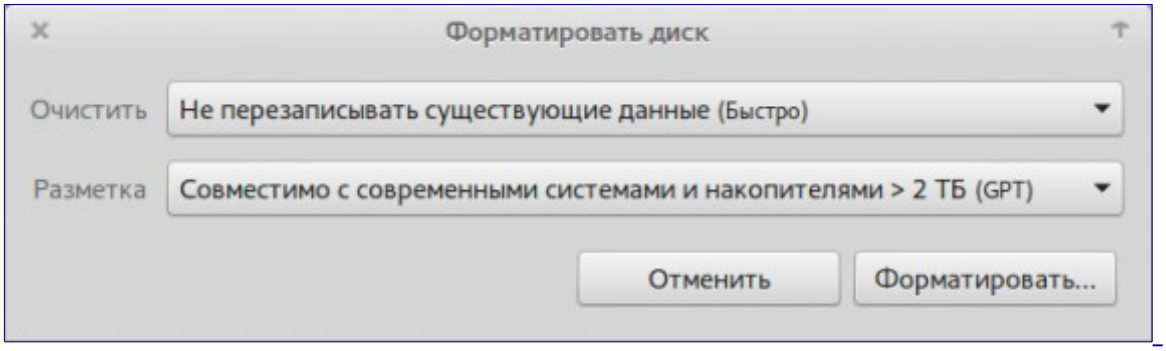
Шестерёнка в правом верхнем углу — вызов главного меню gnome-disks, которое выглядит так:



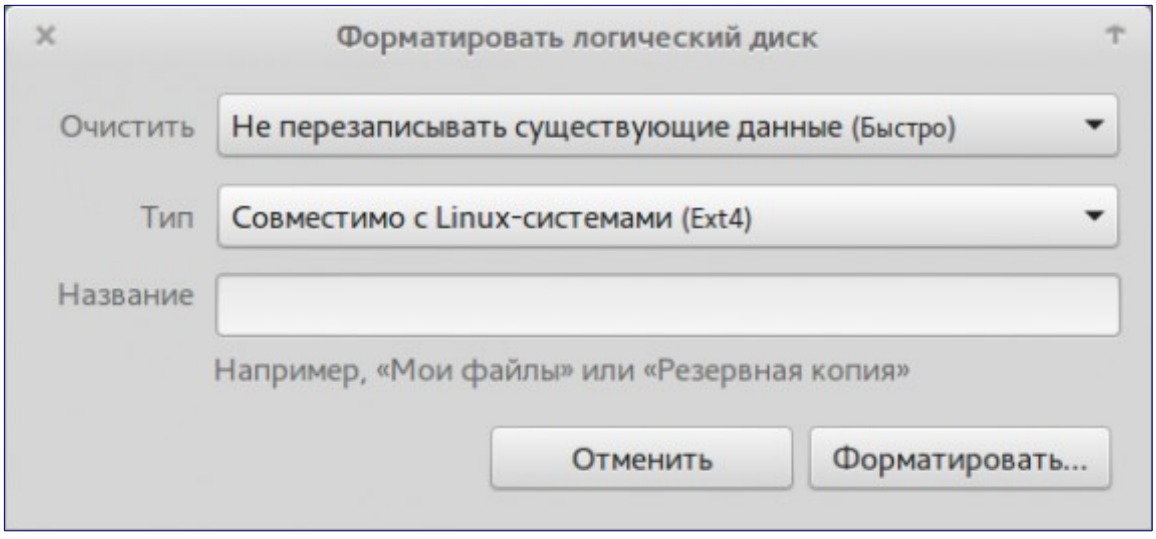
Пара сцепленных шестерёнок ниже графика распределения диска по разделам вызывает очень похожее, хотя и не идентичное, меню:



Смысл всех пунктов обоих меню интуитивно понятен. Следует только учесть, что пункты «верхнего» меню соответствуют манипуляциям над дисками в целом, «нижнего» — над его разделами. Этим и объясняются некоторые различия в этих меню: очевидно, что SMART не имеет смысла для раздела, а изменение параметров монтирования — для диска в целом. Кроме того, пункты, именованные в обоих меню одинаково, выполняют разные действия. Так, Форматирование из «верхнего» меню — ни что иное, как создание таблицы разметки диска (варианты — в стиле MSDOS или GPT):

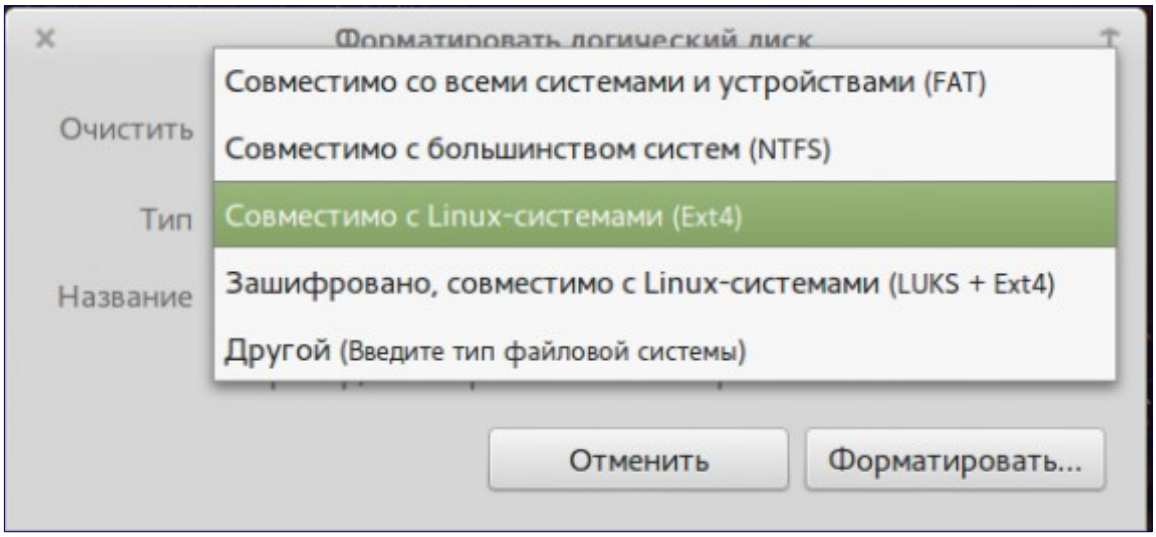


А одноимённый пункт из меню «нижнего» — это действительно создание файловой системы на разделе:



Причём через меню предусмотрен выбор только между FAT, NTFS и Ext4 (в

том числе с шифрованием):

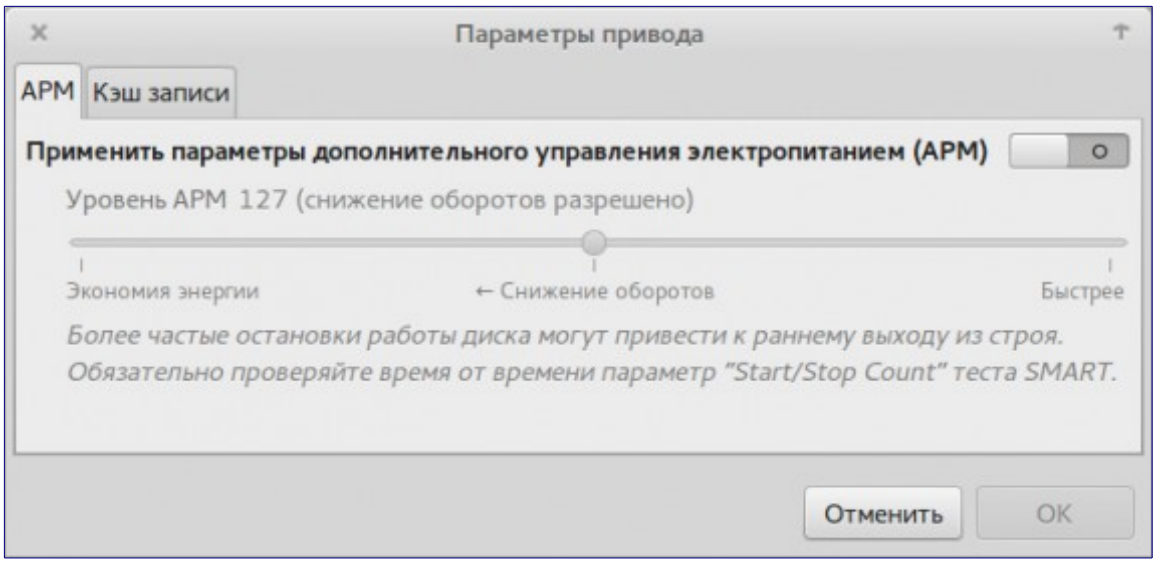


Названия прочих файловых систем следует, выбрав строку **Другой**, ввести вручную (например, btrfs). Список поддерживаемых файловых систем зависит от установленного в системе инструментария для работы с ними. Например, чтобы отформатировать раздел в JFS, необходимо иметь в установленном виде пакет jfsutils.

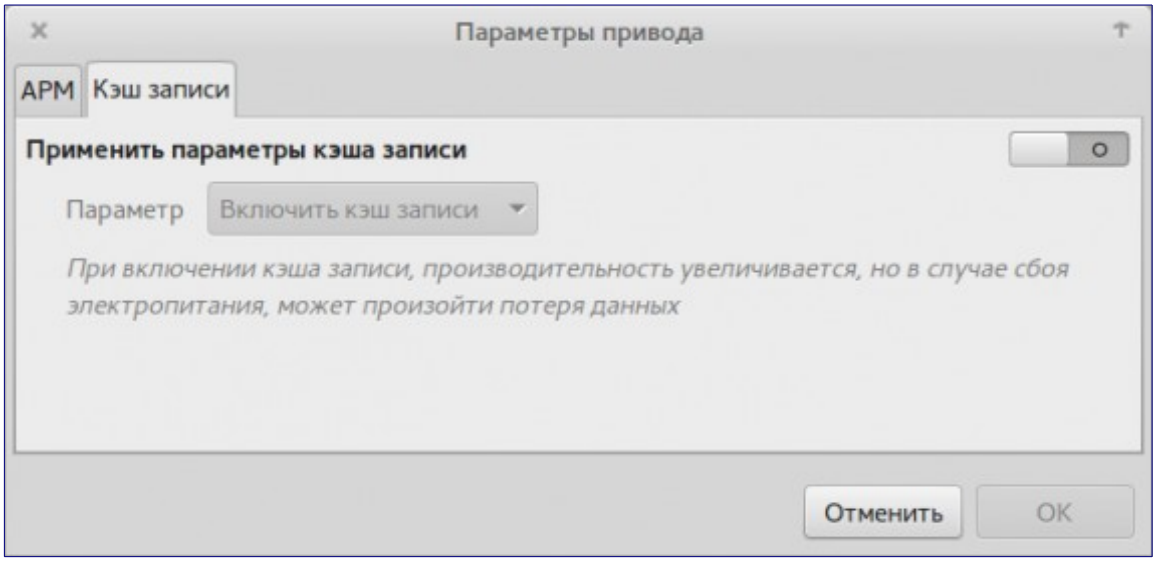
Кстати, поле с именем Название — ни что иное, как метка диска.

Думаю, понятно, что оба «форматирования» возможны только на отмонтированных носителях и требуют прав администратора. Причём пароль на доступ к оным запрашивается в самый последний момент, после всех подтверждений серьёзности своих намерений.

Перед этим неплохо бы рассмотреть параметры приводов. В зависимости от типа накопителей, они оказались разными. Так, для SSD информационная панель включала две вкладки — управления питанием

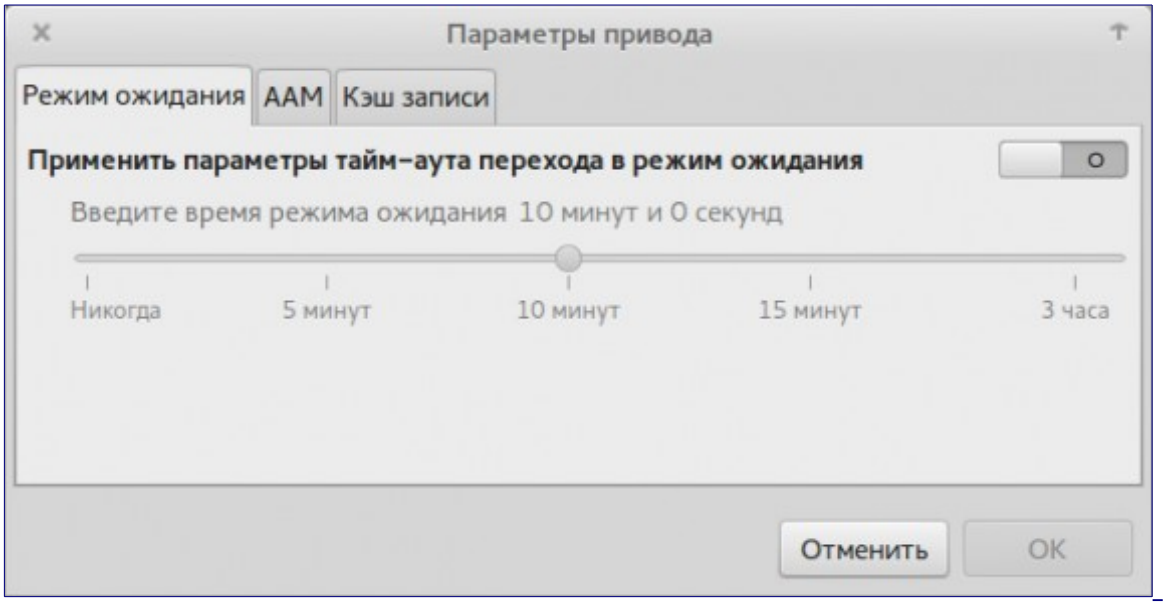


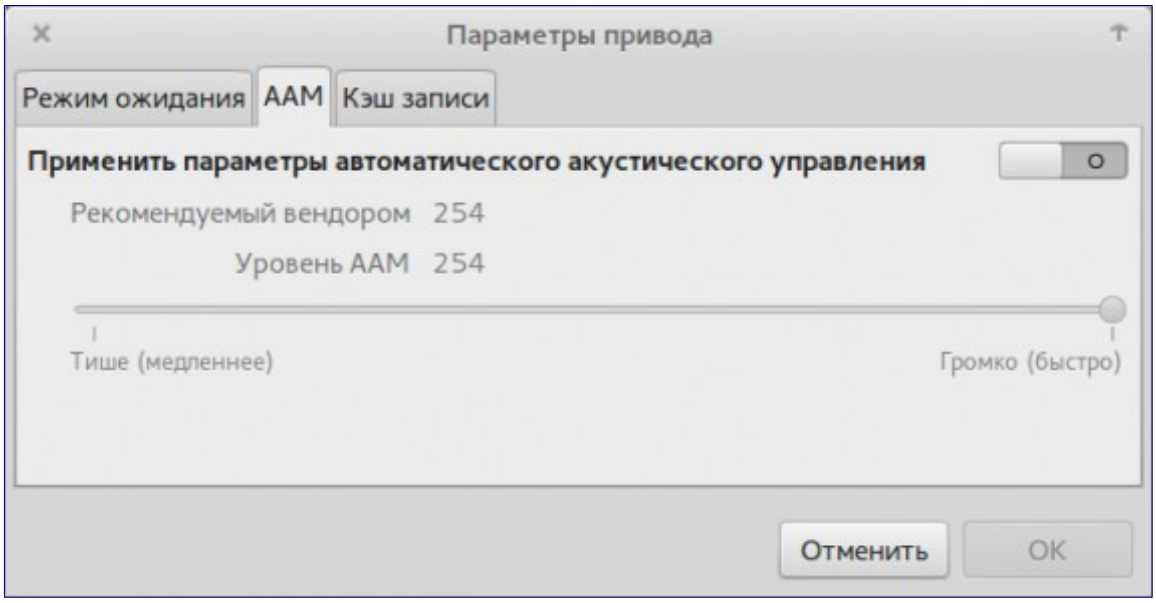
и включения кеша записи:

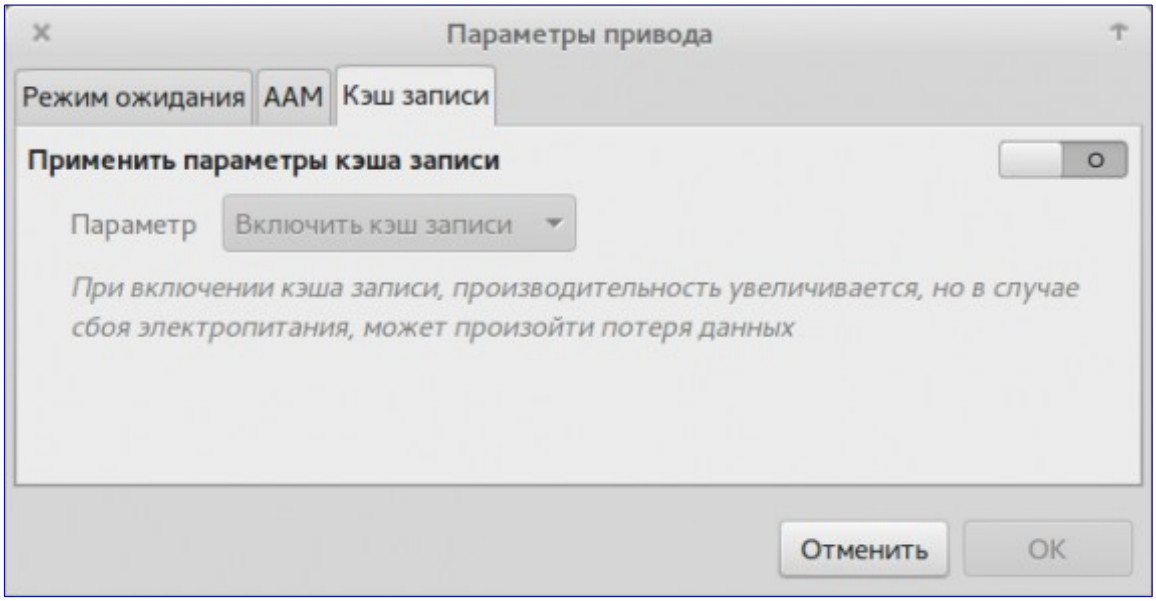


Обе функции по умолчанию отключены. Оно и понятно — для десктопа управление питанием не требуется. А с включением кеширования я решил пока не экспериментировать — ведь речь шла не о тестировании нового накопителя, а о практической работе на реальной машине.

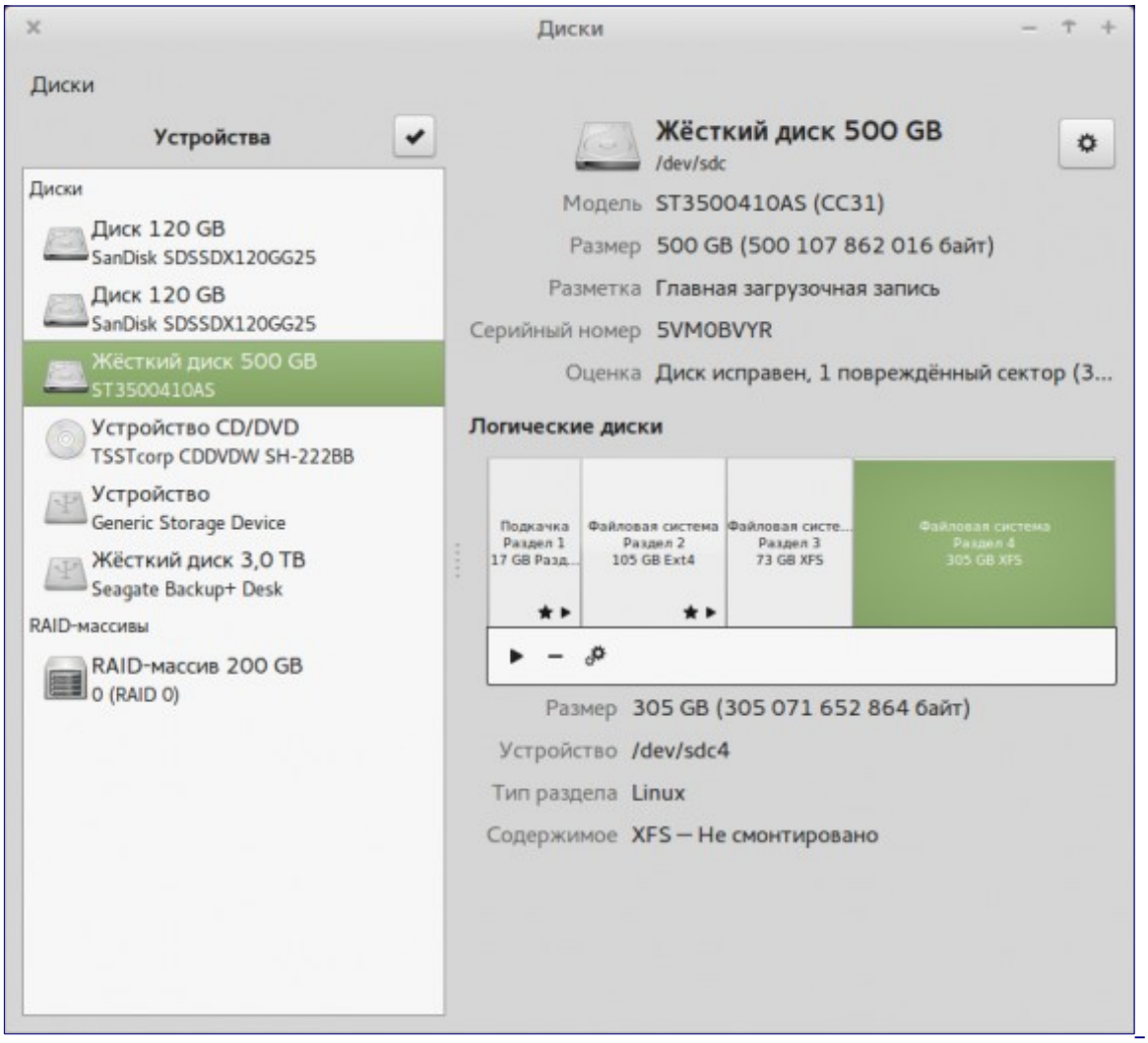
Для традиционного винчестера на 500 ГБ параметры выглядели так:



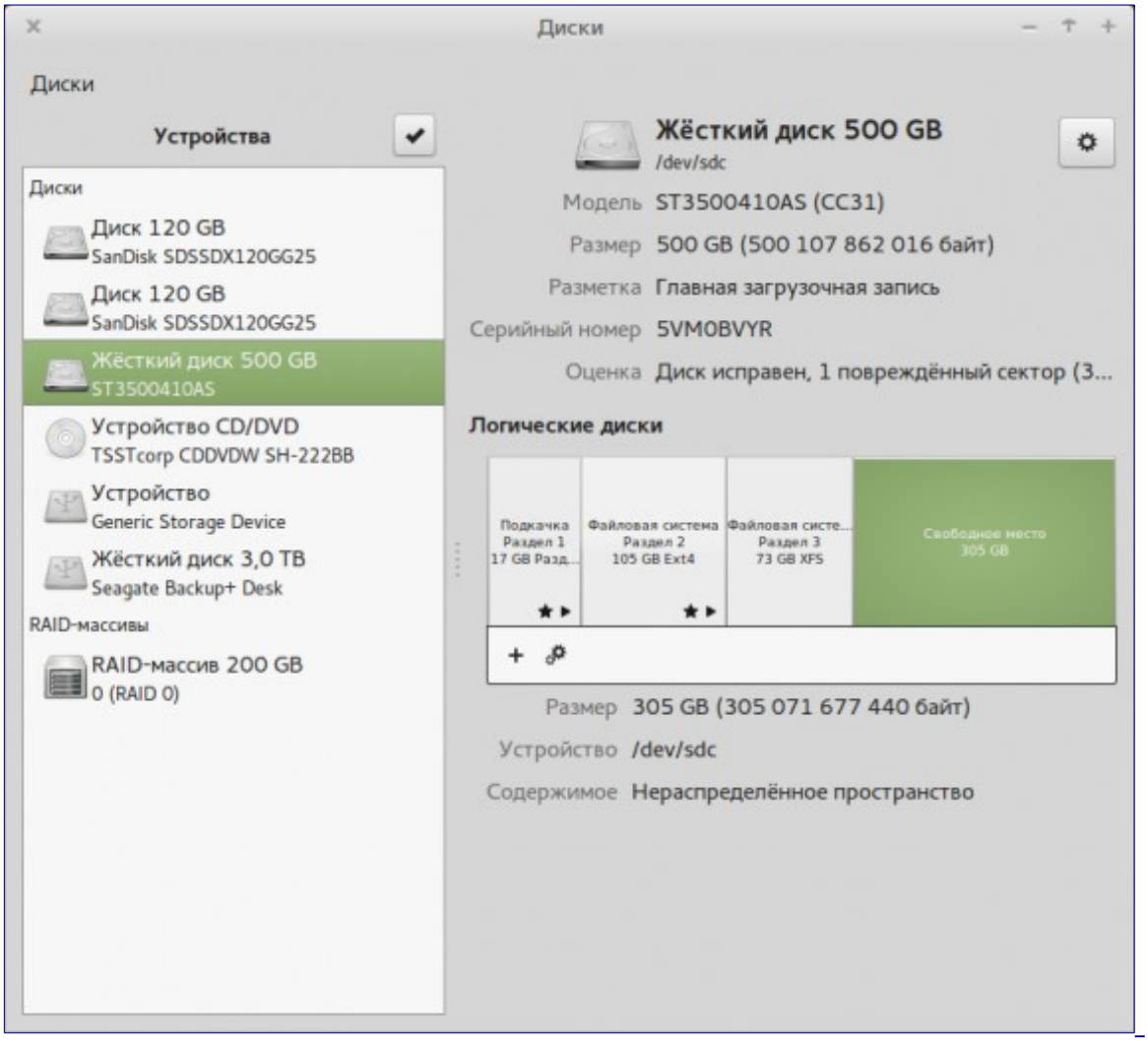




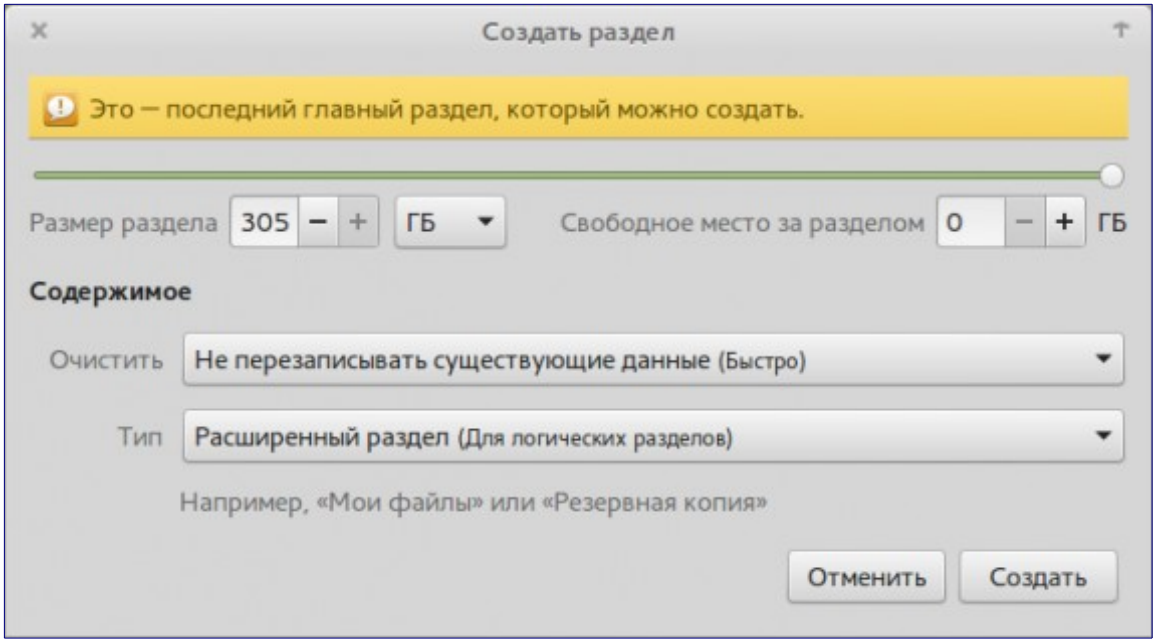
Вволю поигравшись с функциями gnome-disks и сделав зарубки на счёт того, какие из них мне могут понадобиться в дальнейшем, я приступил к выполнению поставленной боевой задачи — переразметке экспериментального диска /dev/sdc. Каковой выглядел следующим образом:



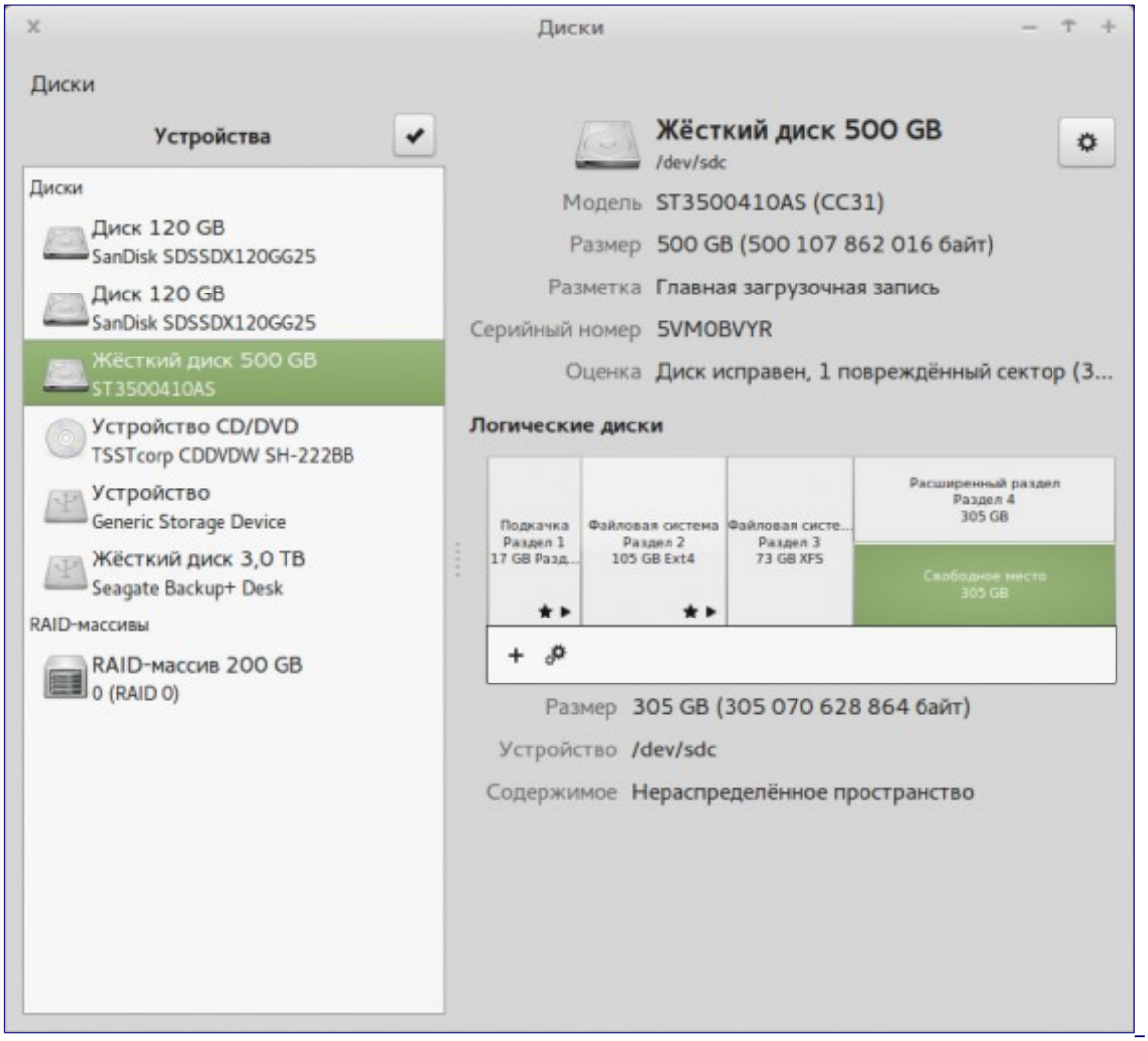
Так что первой задачей было удаление некоего «отсебятного» раздела. Для чего, как подсказала мне солдатская смекалка, требовалось нажать кнопку с минусом рядом с кнопкой вызова «нижнего» меню. Что, как и ожидалось, привело к следующему результату:



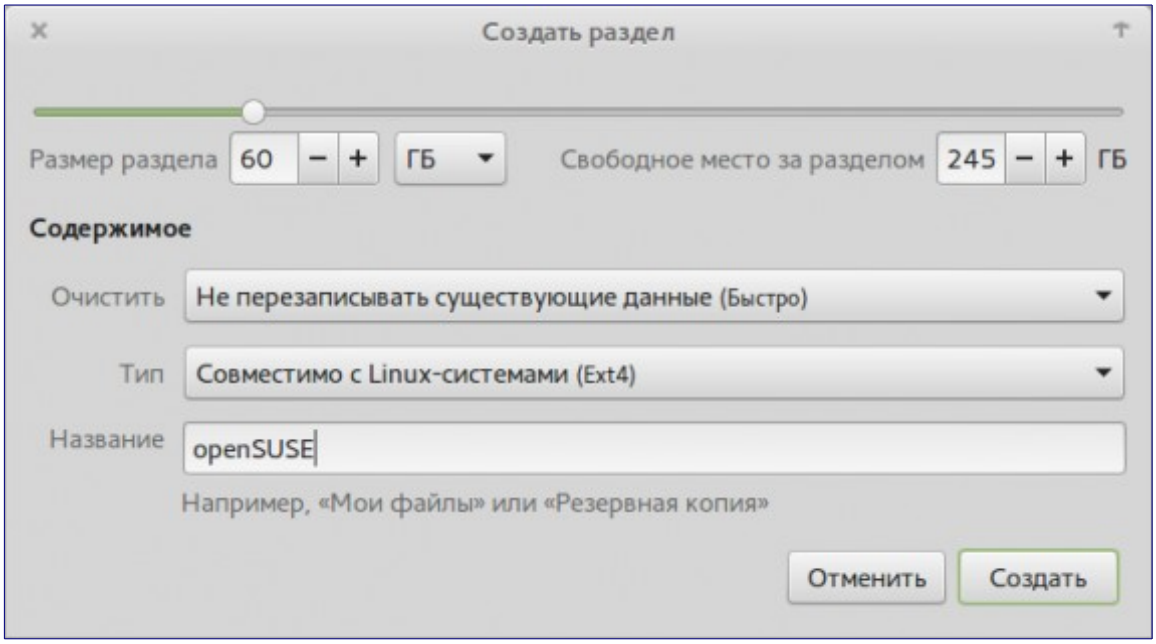
Та же солдатская смекалка говорила, что нажатие кнопки с плюсиком приведёт к созданию на пространстве, ставшем неразмеченным, нового раздела, который оставалось только определить как расширенный:



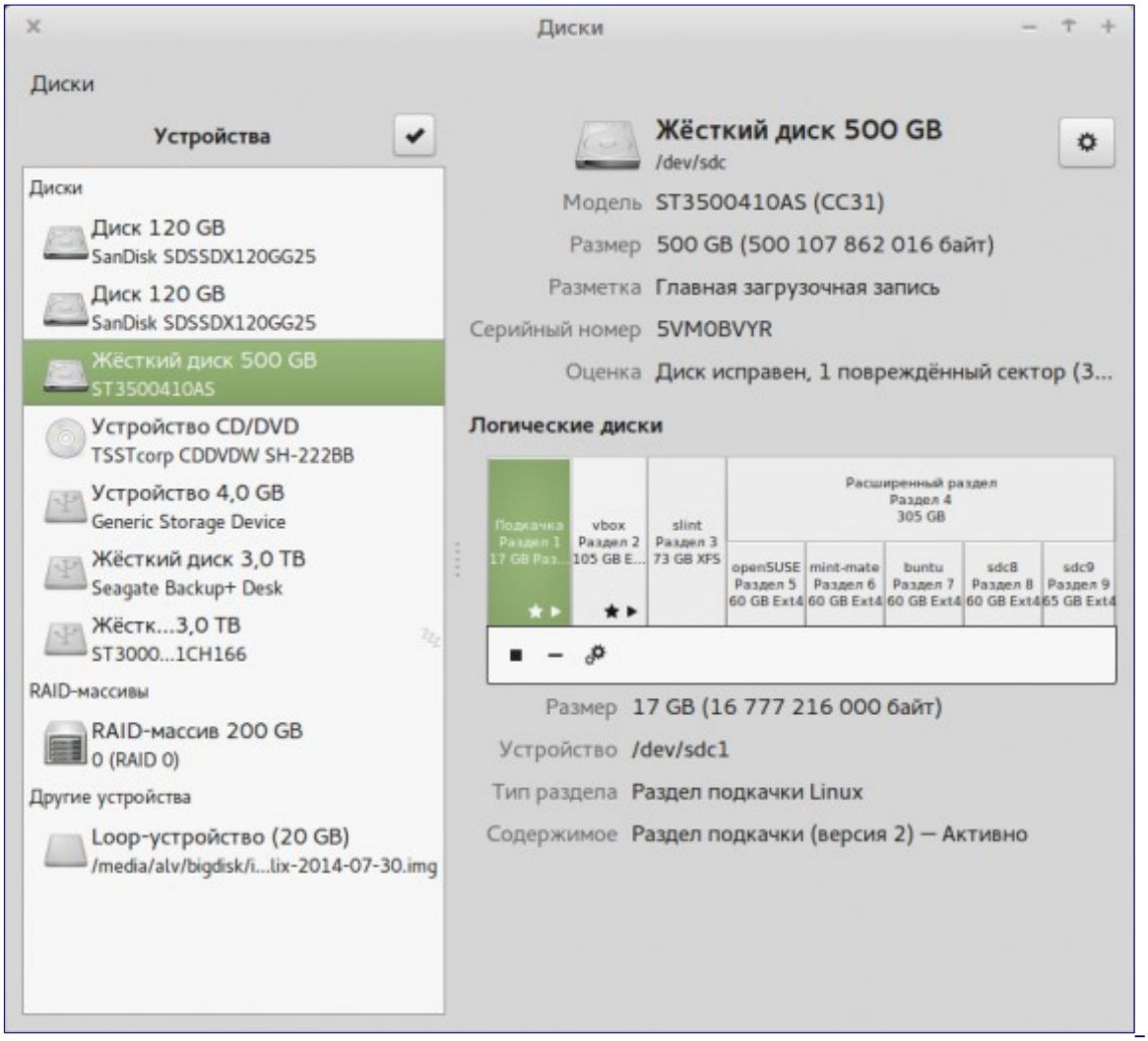
В результате чего разметка устройства /dev/sdc приобрела такой вид:



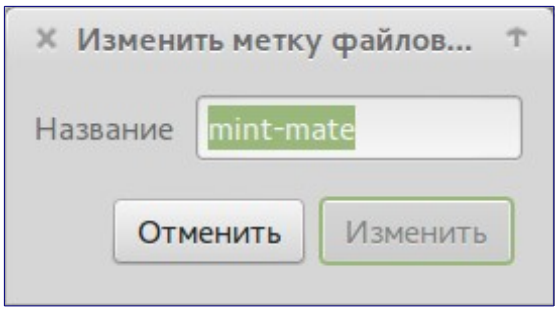
Наличные примерно 300 ГБ расширенного раздела я планировал разметить на 5 примерно равных по объёму разделов логических. Что, с помощью той же самой кнопки Плюс, и проделал для первого, предназначенного в будущем для openSUSE Factory, задав ему для определённости соответствующую метку:



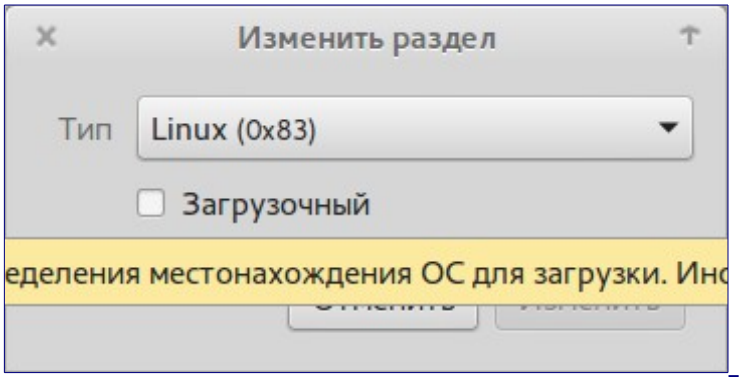
Далее оставалось проделать ту же процедуру для остальных четырёх логических томов, после чего получилась следующая картина:



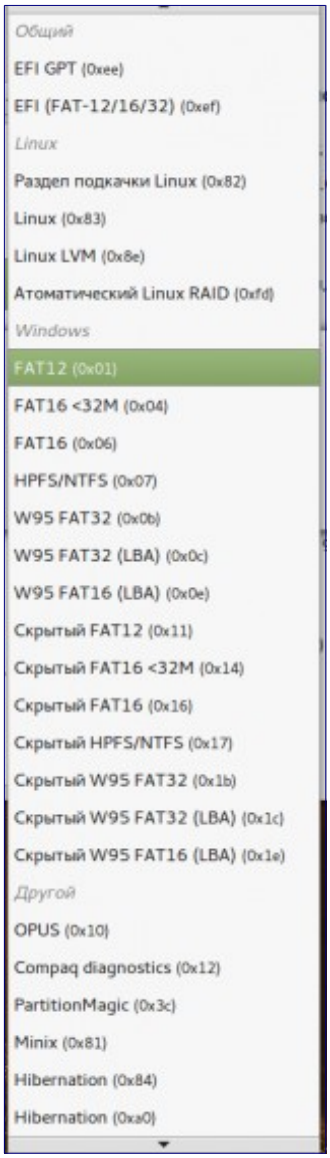
Выбором файловых систем я не заморачивался — он будет сделан при инсталляции целевого дистрибутива. А метки для разделов я присвоил или в соответствие с названиями предполагаемых на них систем, либо просто по именам устройств. Для уже созданных (но не смонтированных) разделов (точнее — файловых систем, ибо label — это её аттрибут) метку можно задать через пункт Изменить файловую систему «нижнего» меню:



К слову сказать, через предыдущий пункт того же меню, Изменить раздел, можно задать индентификатор его типа:



Каковых имеется более чем вдоволь:



Впрочем, из этого изобилия практически значимы лишь менее полудюжины. Да и делать это для разделов, несущих файловые системы, тем более с данными, не очень рекомендуется. Конечно, изменение идентификатора типа к разрушению файловой системы и потере данных не приведёт, проверено на опыте. Но как будут восприниматься идентификаторы, отличные от Linux'ового 0x83 всякими программами работы с дисками — ведомо только Ахурамазде. А самое главное — это нафиг не нужное занятие.

Подведу итоги. Сравнивать GParted напрямую с gnome-disks не возьмусь. Но одно преимущество последней утилиты лежит на поверхности: если GParted запрашивает пароль на доступ к административным правам сразу при

запуске, то gnome-disks — только непосредственно перед выполнением операций, для которых они на самом деле требуются. Так, просто поглядеть на схему разметки, параметры носителя и его состояние можно в режиме обычного пользователя. Конечно, это не гарантирует от ошибок, которые в деле разметки дисков могут быть критическими, но несколько снижает их вероятность.

А в целом функционал gnome-disks показался мне вполне достаточным для выполнения всех повседневных практических задач в области работы с дисками, разделами и файловыми системами. И в большинстве обыденных случаев он вполне может заменить комплекс утилит CLI, включая и команду dd для создания образов дисков (кстати, GParted, кажется, делать этого не умеет). Если, конечно, не требуются какие-то специфические параметры разметки или монтирования — тут специализированные утилиты командной строки вне конкуренции.

Mint и softRAID

Теоретически softRAID Level 0 — самый простой способ объединения двух носителей информации, конкретно дисковых разделов. Однако как раз в Mint оказывается, что задействовать его «искаропки», то есть на стадии инсталляции, не получится. Причина проста до банальности: на установочном носителе любой редакции этого дистрибутива отсутствует пакет mdadm, отвечающий ныне за управление программным RAID. Что, однако, не препятствует подключению его в любой момент времени после установки.

Зачем? Вопрос, нужен ли RAID народу, и если нужен — то какой, да и какие ветви файловой иерархии на нём размещать, я здесь обсуждать не буду, ибо неоднократно высказывался по этому поводу ранее. А потому буду исходить из следующих постулатов:

- народу (в лице одного из его лучших представителей) RAID нужен позарез;
- он необходим ему в форме softRAID Level 0;
- размещаться на нём должна ветка /home файлового древа.

Для чего, как нетрудно догадаться, требуется установка пакета mdadm, в ходе которого автоматически выполняется сканирование на предмет наличия softRAID'a. И после рестарта машины нужные модули (raid# и всё, что с ними связано) загружаются автоматически, появляется устройство /dev/md0.

Теперь остаётся определить устройство /dev/md0 на его законное место — я уже несколько лет держу свои рабочие данные на отдельном носителе (разделе, на пуле ZFS или, как в примере, на программном RAID'e), который монтируется в каталог /home/data. Каковой был немедленно создан, и командой chown ему были присвоены атрибуты принадлежности alv:alv (точнее, 1000:1000 — UID и GID моего главного рабочего пользователя, вне зависимости от того, как его зовут и какова его основная группа). Затем командой

```
$ sudo blkid
```

для устройства /dev/md0 был определён его UUID, под которым он был

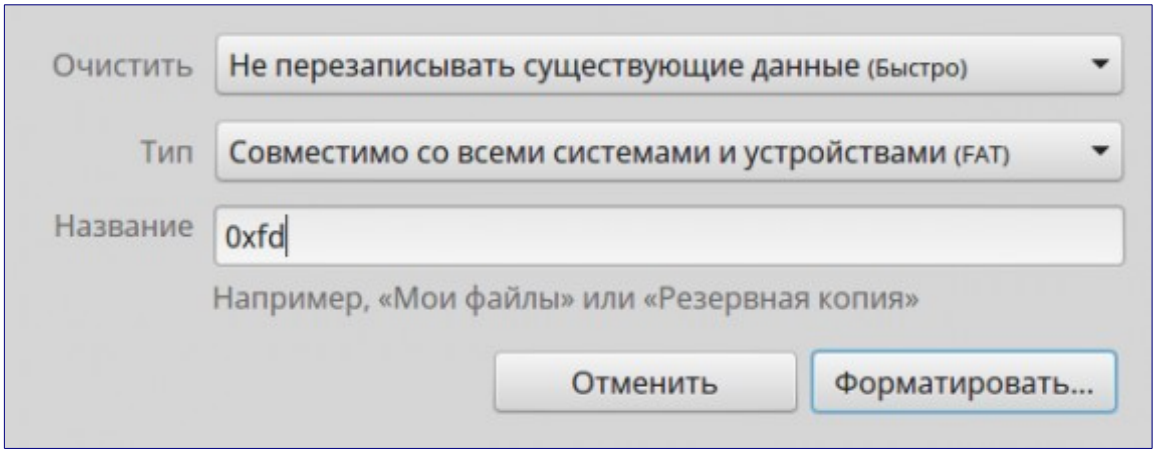
вписан в /etc/fstab строкой вида

```
UUID=очень-длинное-бла-бла-бла /home/data ext4 defaults,noatime 0 0
```

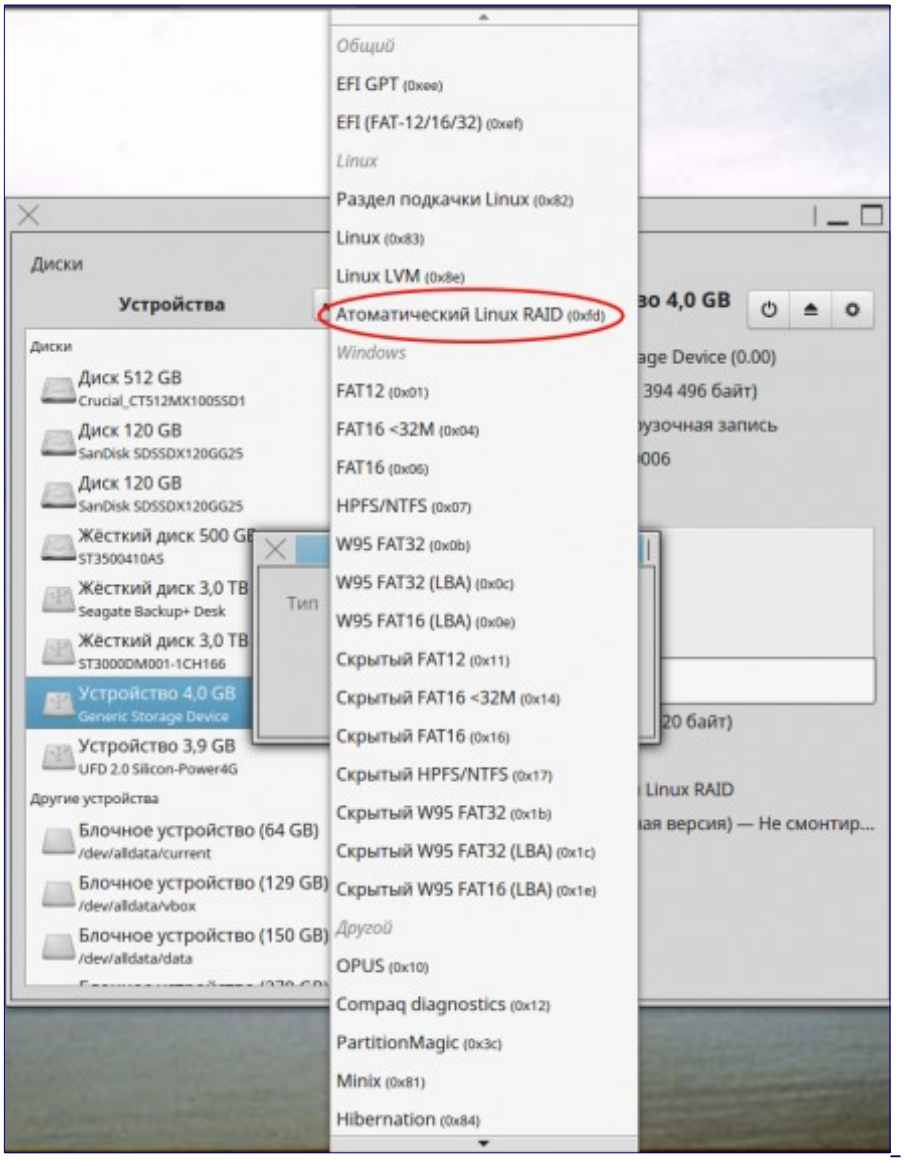
Разумеется, можно было обойтись и без UUID, занеся RAID под его так называемым именем верхнего уровня, то есть /dev/md0. Но уж раз в Ubuntu и её потомках принято именование устройств по UUID'у — будем придерживаться фирменного стиля.

Сказанное выше относилось к подключению уже существующего softRAID Level 0. Однако создание последнего «с нуля» ничуть не сложнее. Для начала с помощью одной из утилит, fdisk или cfdisk на каждом из носителей, предназначенных для включения в массив, создаются по разделу. Для обоих следовало устанавливается идентификатор типа файловой системы fd — Linux raid autodetect.

Эти действия можно проделать с помощью графической утилиты gnome-disks, задав при создании разделов идентификатор их типа вручную, как 0xfd:



Или, если не обременять память этим сложным шестнадцатеричным числом, выбрать его из списка, выводимого через пункт Изменить раздел, вызываемый «нижней шестерёнкой»:



Дальнейшая работа выполняется с помощью утилиты mdadm, которая в моём случае была запущена в такой форме:

```
$ sudo mdadm --create /dev/md0 --auto=yes --level=0 --raid-devices=2 /dev/sd(a,b)2
```

Здесь --create (или -C) — субкоманда создания массива, в качестве аргумента которой указывается имя его файла устройства (к этому вопросу я ещё вернусь), --level — определение его уровня (а я уже говорил, что именно нужно народу), --raid-devices — число входящих в массив устройств с указанием их имён (/dev/sda2 и /dev/sdb2). Опция же --auto=yes, как было установлено эмпирическим путём, препятсвует переопределению имени файла RAID-устройства.

ТПосле создания RAID'а результат своих действий можно проверить таким образом:

```
$ sudo mdadm --detail /dev/md0
```

Что должно дать примерно такой вывод:

```
/dev/md0:

        Version : 1.2
```

```
  Creation Time : Tue Apr 15 00:06:59 2014
     Raid Level : raid0
     Array Size : 195371008 (186.32 GiB 200.06 GB)
   Raid Devices : 2
  Total Devices : 2
    Persistence : Superblock is persistent

    Update Time : Tue Apr 15 00:06:59 2014
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

     Chunk Size : 512K

           Name : salix:0
           UUID : a32dc435:25c68870:18fa63ca:010d8910
         Events : 0

    Number   Major   Minor   RaidDevice State
       0       8        2        0      active sync   /dev/sda2
       1       8       18        1      active sync   /dev/sdb2
```

Вместо субкоманды --detail можно использовать её сокращённую форму -D. И заметка на будущее: все утилиты субкоманды mdadm, даже не выполняющие никаких действий, а только выводящие информацию, требуют прав суперпользователя. Исключение — запрос помощи

```
$ mdadm --help
```

и детализирующие его просьбы типа

```
$ mdadm --create --help
```

К слову, в выводе последнего запроса (или из man mdadm) можно узнать о дополнительных опциях, с помощью которых определяются, например, такие параметры, как величина блока «распараллеливания» (Chunk Size), которая теоретически должна влиять на быстродействие (чем больше, тем лучше), Однако сведений, насколько это чувствительно в десктопной обстановке, я не нашёл, и потому положился на умолчание mdadm; как можно видеть из вывода субкоманды -D, оно составляет 512 Кбайт (в старых версиях — 64 Кбайт).

Завершив создание RAID'а, на нём следует создать файловую систему, например, так:

```
$ sudo mkfs.ext4 /dev/md0
```

И обеспечить её монтирование при старте машины, внеся в файл /etc/fstab строку такого вида:

```
UUID=очень-длинное-бла-бла-бла /home/data ext4 defaults,noatime 0 0
```

Где очень-длинное-бла-бла-бла, как и раньше, определяется командой

```
$ sudo blkid
```

которая выведет значения UUID для всех наличных накопителей.

Mint и LVM

Тема этого очерка образовалась в значительной мере как результат случайности. Которая началась с того, что я стал счастливым обладателем SSD-накопителя производства Crucial MX100 объёмом 512 ГБ, и в результате дисковая подсистема, упакованная внутри моего десктопа, стала выглядеть так:

- вышеупомянутый полутерабайтный Crucial — на первом SATA-разъёме;
- SanDisk Extreme SSD, 120 Гбайт — на втором;
- он же, то есть точно такой же — на третьем;
- традиционный винчестер Seagate ST3500410AS о 500-х гигабайтах — на четвёртом.

Первый SSD в ходе установки Mint 17.1 Rebecca был разбит на три раздела:

- /dev/sda1 объёмом 20 ГБ с файловой системой ext4 под корень файловой иерархии;
- /dev/sda2 объёмом 10 ГБ, также с ext4 — под каталог будущего пользователя, то есть меня — /home/alv (в домашнем каталоге я храню только dot-файлы и некоторые служебные данные, на что указанного объёма хватало с лихвой);
- /dev/sda3 на всё оставшееся пространство — без файловой системы и,

сооветственно, без точки монтирования.

На обоих SanDisk'ах было создано по разделу, занимающему их целиком (/dev/sdb1 и /dev/sdc1, соответственно), также без файловой системы. Вместе с /dev/sda3 они предназначались для объединения в хренилище моих рабочих данных — организация оного и является предметом данных очерков.

Винчестер у меня служит для экспериментальных целей, и потому разметка его постоянно меняется, да и к нашей теме не относится. За исключением того, что первые 32 ГБ диска выделены в раздел/dev/sdd1, служащий swap'ом для всех моих систем.

Очевидно, что организация хранилища из трёх устройств требовала их объединения тем или иным способом. И поначалу напрашивался выбор ZFS: эту систему хранения данных я люблю, более-менее знаю, и включал её поддержку в сборки своих вариантов Mint 17. Однако тут модули ZFS у меня неожиданно с первого раза не собрались. Правда, проблема решилась (благодаря помощи Станислава Шрамко aka stanis, о чём пойдёт речь в следующем очерке), однако, как говорится, осадок остался. Ибо случай этот послужил напоминанием не только о бренности бытия, но и птичьих правах, на которых существует ZFS on Linux.

В своё время, более десяти лет назад, я очень увлекался технологией LVM, тогда ещё 1-й версии. Потом я это дело забросил по двум причинам. Во-первых, во времена винчестеров с PATA-интерфейсом было очень сложно сконфигурировать многодисковую систему LVM без деградации производительности. Во-вторых, инструментарий CLI для создания такой системы и последующего управления ею показался мне при использовании в «домашних» условиях неоправданно сложным — особенно в сравнении с появившейся вскоре ZFS.

Ныне, в эпозу безраздельного господства SATA-накопителей, первое препятствие к применению LVM отпало полностью. Что касается второго, то оно во многом сглаживается наличием графических оболочек, изолирующих применителя от низкоуровневых команд. Одну из которых system-config-lvm, я и использовал нынче.

С тех пор, как я имел дело с LVM, появилась LVM2, предоставляющая ряд дополнительных возможностей, таких, как расширение логического тома на вновь подключённые физические тома. Однако суть технологии, терминология, утилиты CLI для работы с LVM почти не изменились. Да и в сети можно найти много не менее подробных, но более свежих материалов по теме. Так что на этих вопросах останавливаться не буду, ограничившись маленьким терминологическим конспектом.

Сама по себе система LVM — уровень абстракции между физическими носителями (дисками, их разделами и массивами) и обычными файловыми системами. Она позволяет объединять в логические тома разделы с разных дисков, изменять их размер в сторону увеличения (за счёт присоединения новых накопителей) и, с некоторыми оговорками, уменьшения. В основе её лежит понятие физического тома (PV — Physical volume). Это — обычный дисковый раздел, для которого устанавливается идентификатор типа (ID) 8e. Вопреки написанному в некоторых сетевых материалах, целый диск как raw-устройство типа /dev/sd? в качестве физического тома выступать не может. Другое дело, что созданный на нём раздел с ID 8e может занимать и весь диск и даже RAID, программный или аппаратный.

Физический том делится на «куски», именуемые физическими экстентами (Physical Extent, PE — по традиции оставлю этот термин без перевода, так как предлагаемый русскоязычной Википедией диапазон может ввести в заблуждение). Это — нечто вроде аналогов физических блоков (секторов) обычного винчестера, их размер по умолчанию 4 МБ, но может быть изменён в обе стороны.

Физические тома объединяются в группу томов (VG — Volume group), которая выступает как единое целое и может рассматриваться как логическиий аналог raw-накопителя. И, подобно последнему, делится на разделы — поскольку над физикой мы уже поднялись, они назваются логическими томами (LV — Logical volume). А уж те — опять-таки на «куски», которые, как нетрудно догадаться, носят имя логических экстентов (LE — Logical extent). По размеру логический экстент равен экстенту физическому, которому он ставится в соответствие — однозначно, но одним из двух методов:

1. линейным (linear mapping), когда логические экстенты последовательно привязываются к физическим сначала первого физического тома, потом второго, и так далее, подобно линейному режиму RAID;

2. «черезполосным» (striped mapping), при котором «куски» логических экстентов (так называемые блоки чередования, по умолчанию равные 4 КБ) распределяются по физическим экстентам чередующихся физических томов.

Последний метод, сходен с расщеплением данных в RAID Level 0 и, подобно последнему, теоретически способствует быстродействию дисковых операций. При этом очевидно (и важно в контексте дальнейшего развития сюжета), что размер логического тома с чередованием не может быть больше наименьшего из входящих в него томов физических.

В ходе создания собственного хренилища LVM, оказалось, что существует ещё и метод зеркалирования. Надо полагать, что это некое подобие RAID Level 1. Но, поскольку для меня это не актуально, выяснением деталей не занимался.

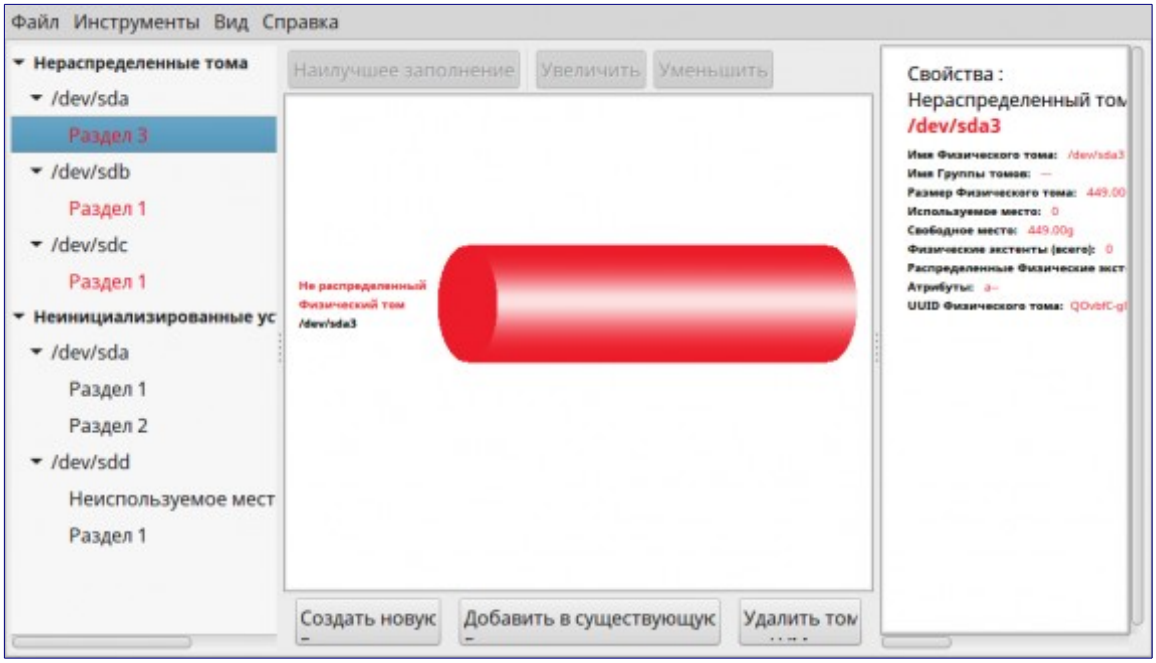
Вне зависимости от метода соответствия логических и физических экстентов, логические тома предназначены для того, чтобы нести на себе файловые системы, подобно обычным дисковым разделам, причём точно те же самые — любые нативные для Linux'a: ext2/ext3/ext4, XFS, ReiserFS, JFS.

На этом терминологическое введение можно считать законченным — перехожу к описанию действий по организации хранилища данных.

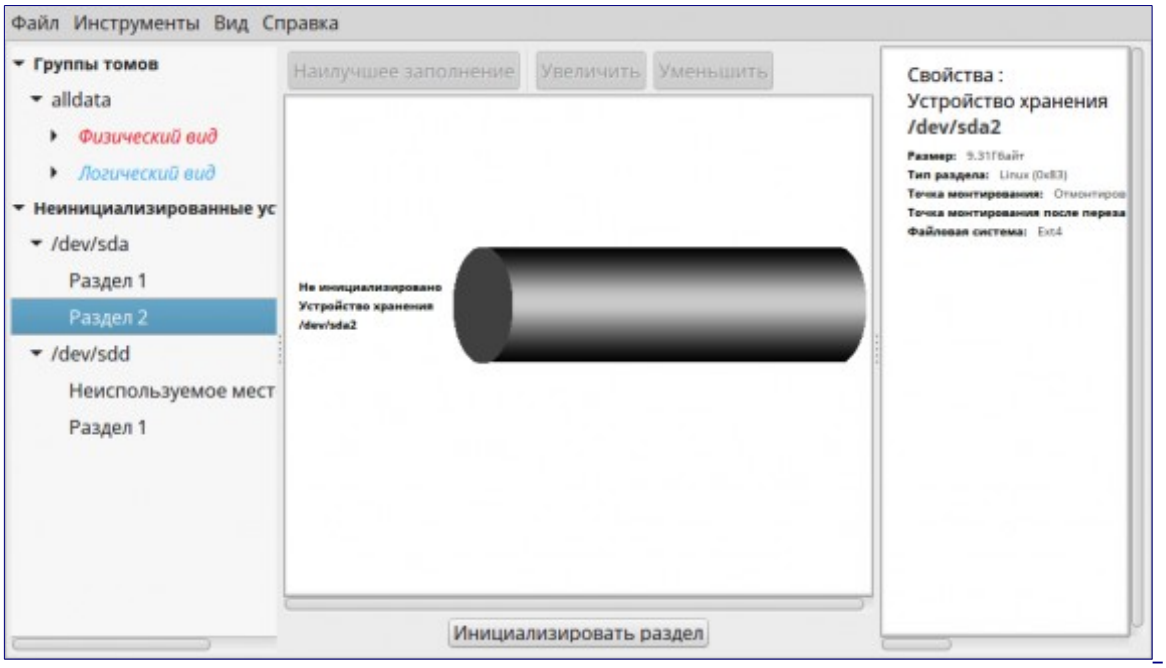
Пакет LVM2, предоставляющий утилиты CLI для работы с логическими томами, устанавливается в Mint по умолчанию. Однако, как говорилось выше, LVM — не тюрьма народов, и не заставляет применителя её зубрить полсотни команд этого пакета (я почти не преувеличиваю — их там 48). Потому что в репозиториях доступна графическая оболочка, интегрирующая все их функции — system-config-lvm. В скобках замечу, что это не просто самая распространённая «морда» к утилитам lvm2, но и единственная активно развиваемая в настоящее время. Так что перво-наперво её следует установить, что я и проделал:

```
$ apt install system-config-lvm
```

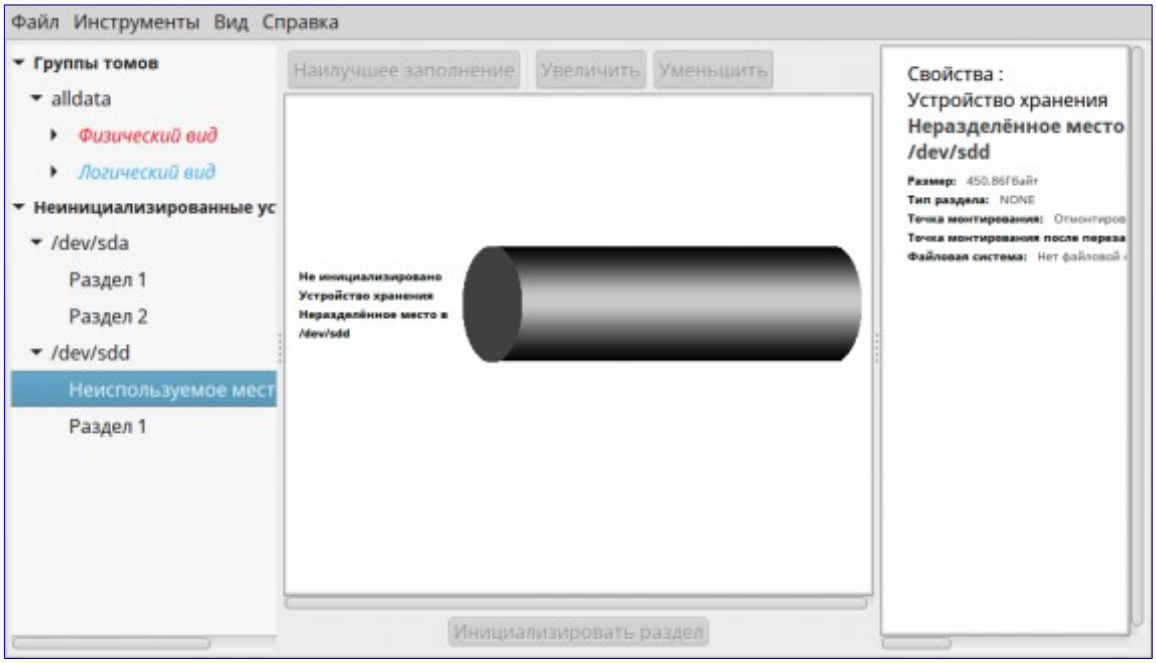
После установки программа под именем Управление логическими томами обнаружилась в секции Администрирование главного меню Cinnamon, откуда и была запущена после ввода пароля, демонстрируя в моём случае такую картину:



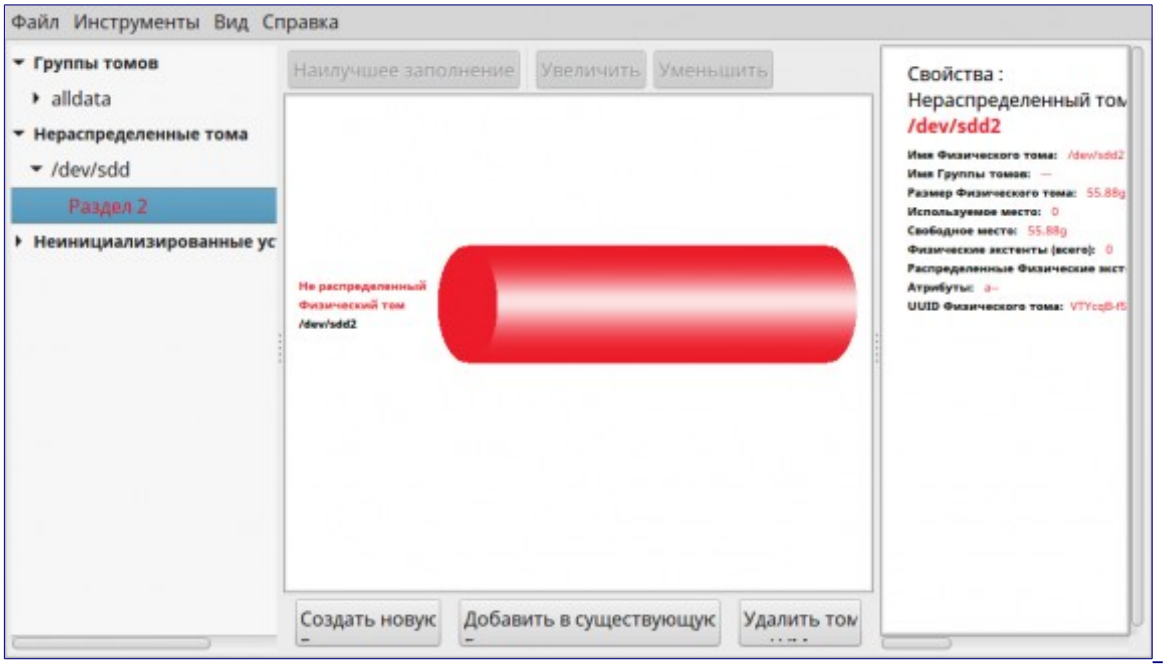
Правда, здесь надо учесть, что перед её запуском я через fdisk установил ID разделов, предназначенных на растерзание LVM, равным 8e. Что в принципе не обязательно — это можно сделать и в графической оболочке, нажав кнопку Инициализировать раздел:



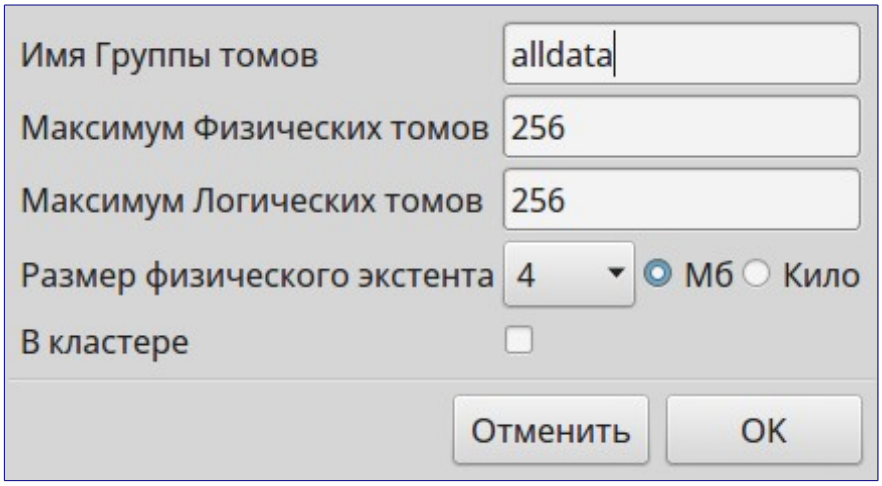
Однако сам раздел должен быть создан заблаговременно — зафиксировав курсор на неразмеченном пространстве, можно видеть, что кнопка инциализации просто не активна:



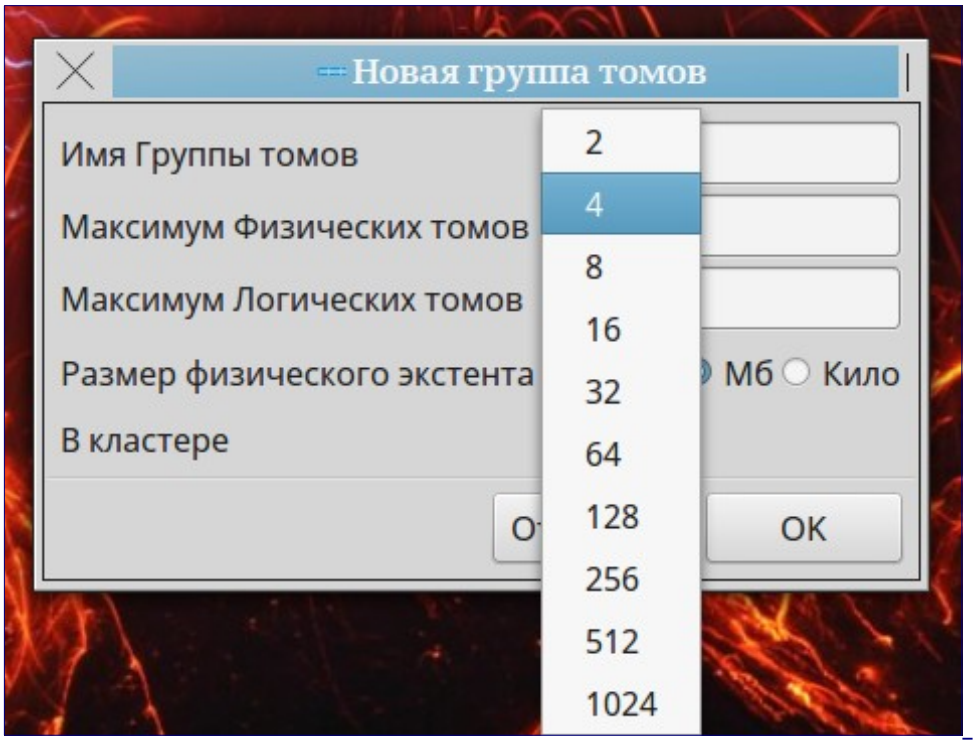
Вне зависимости от того, каким способом были инциализированы разделы, первым шагом после этого будет создание группы томов посредством фиксации одного из инициализированных разделов (я сделал это для самого «жирного») и нажатия кнопки Создать новую:



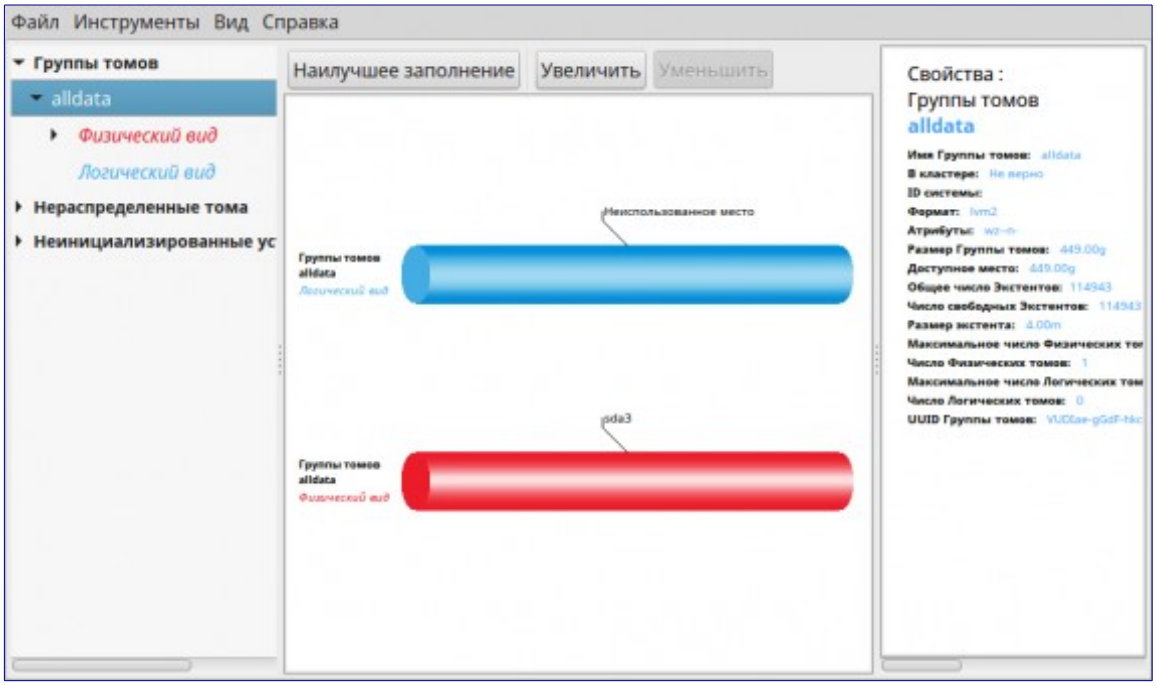
В вызываемой её панели достаточно задать какое-либо имя для группы томов, желательно осмысленное, например:



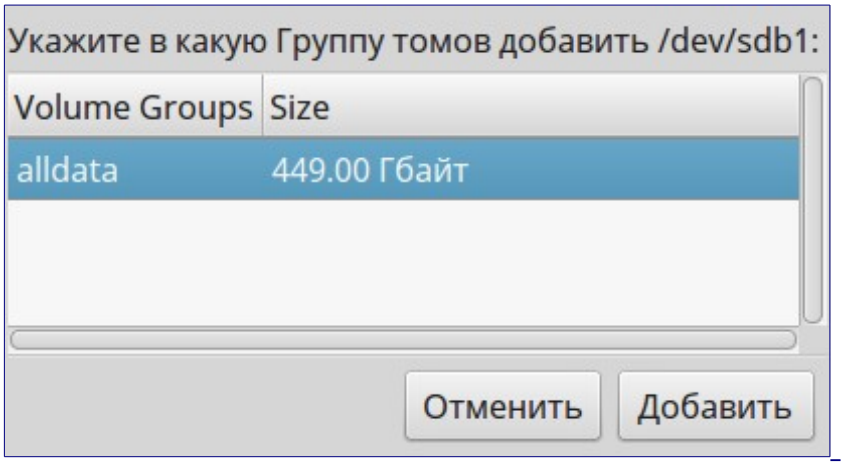
Впрочем, можно и изменить размер физического экстента в диапазоне от 2 до 1024 МБ — если, конечно, точно знаете, что это нужно (я — так не уверен, поэтому все остальные параметры оставил умолчальными):



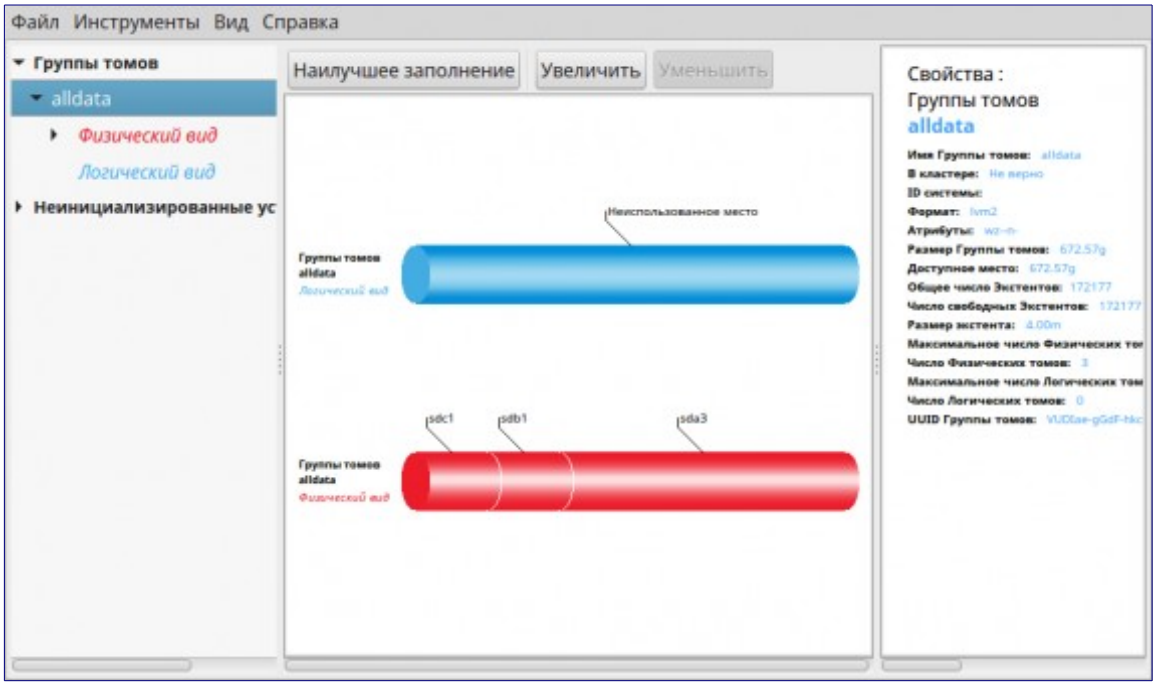
После этого я полюбовался на физический и логический вид группы томов (состоящей пока из одного физического тома, раздела /dev/sda3):



После чего занялся добавлением в группу физических томов — то есть в моём случае разделов /dev/sdb1 и /dev/sdc1. Для чего просто указывается группа, в которую надо добавить соотвтетсвующее устройство — поскольку группа у меня одна, то и ломать тут голову не над чем:

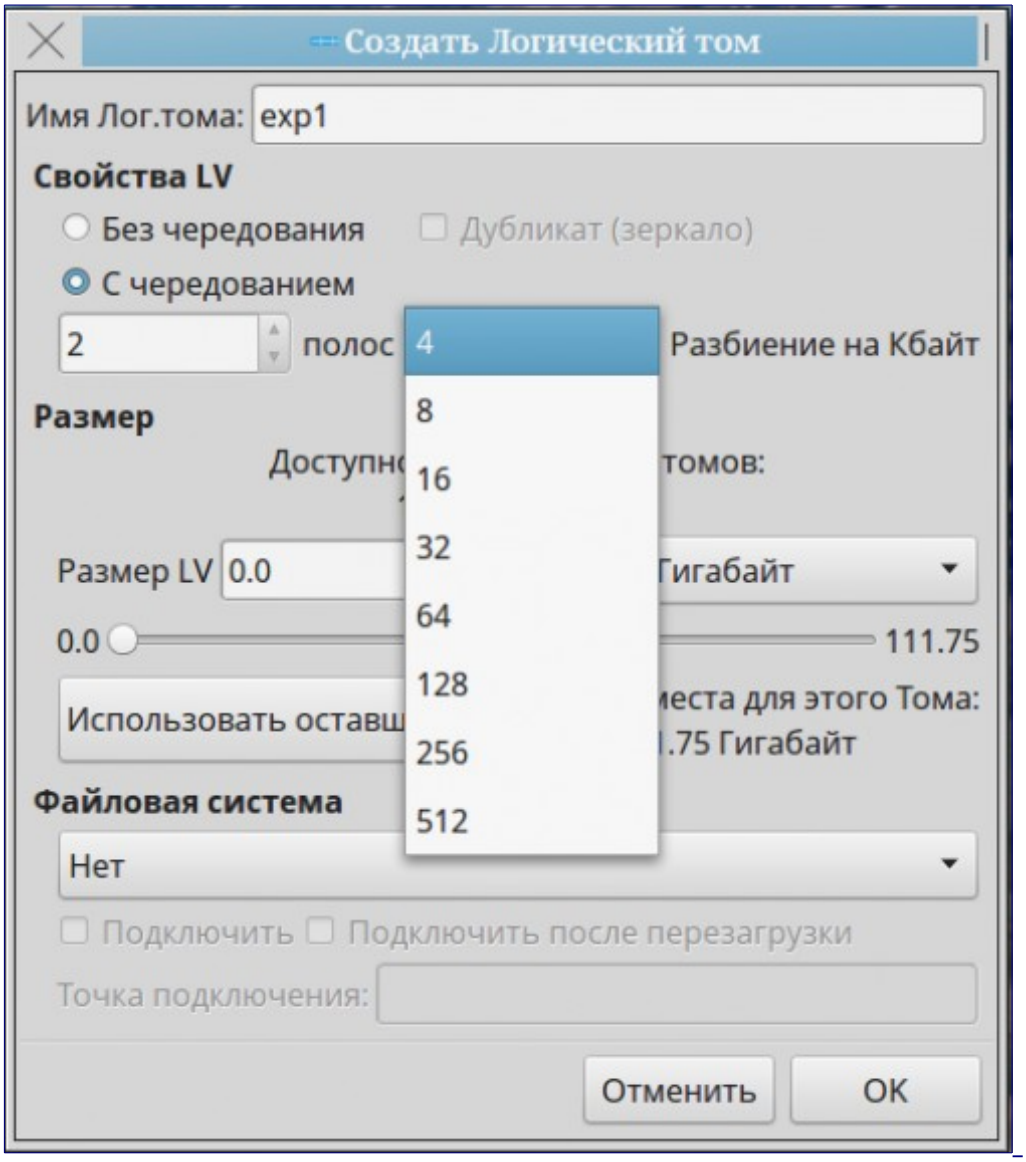


По завершении компоновки группы томов она стала выглядеть таким образом:

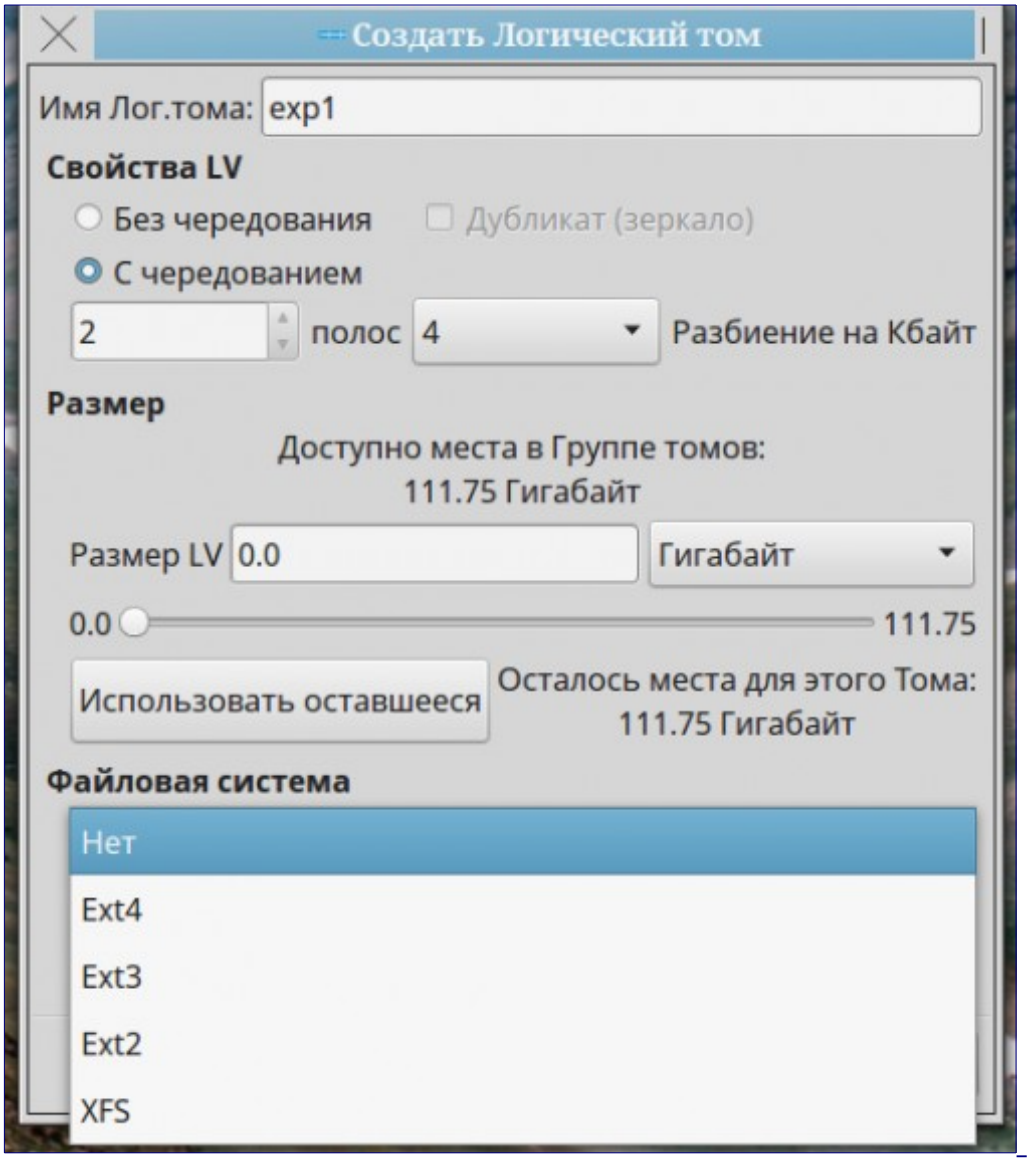


Теперь настал черёд поделить эту группу на логические тома. Оно сводится к

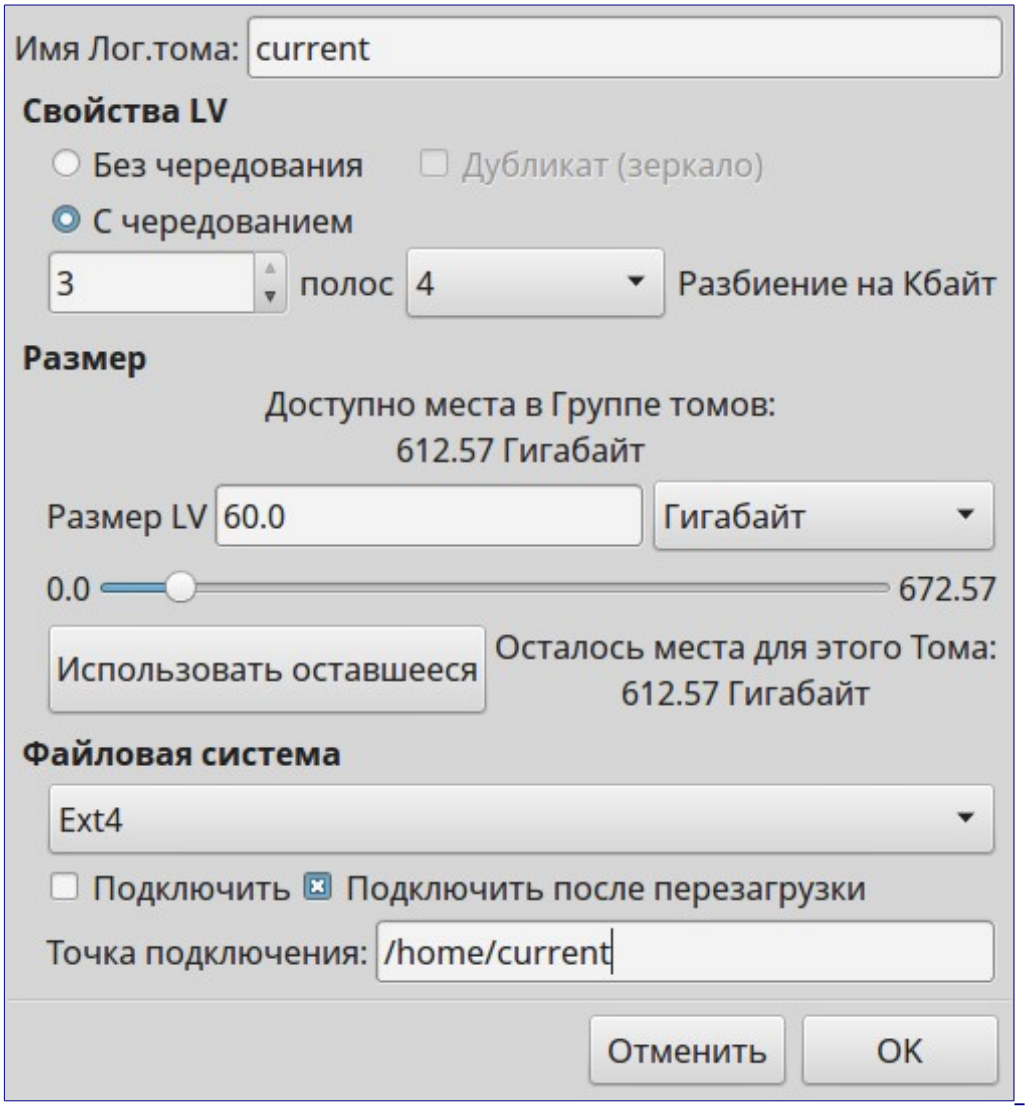
- определению имени тома (опять таки желательно осмысленного — на следующем скриншоте рассмотрен пример тома для текущих проектов);
- заданию метода соответствия логических экстентов физическим — в примере «чересполосного»;
- числа «полос» — поскольку в группе томов участвует три устройства, значение его очевидно;
- размера блока чередования — я сохранил умолчальные 4 килобайта, но его можно увеличить до 512 КБ (опять же не знаю, нужно ли это):



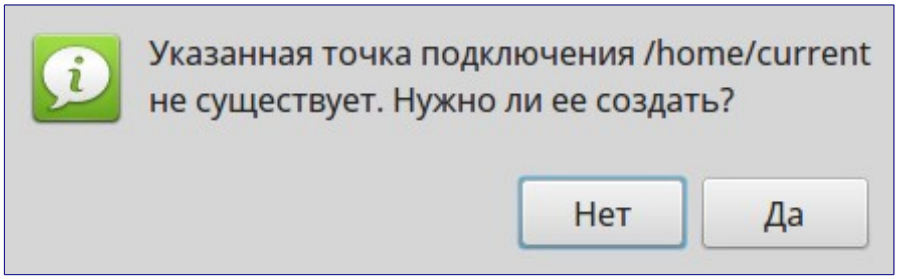
Затем я задал размер тома и файловую систему для него — в моей системе доступны ext всех видов и XFS:



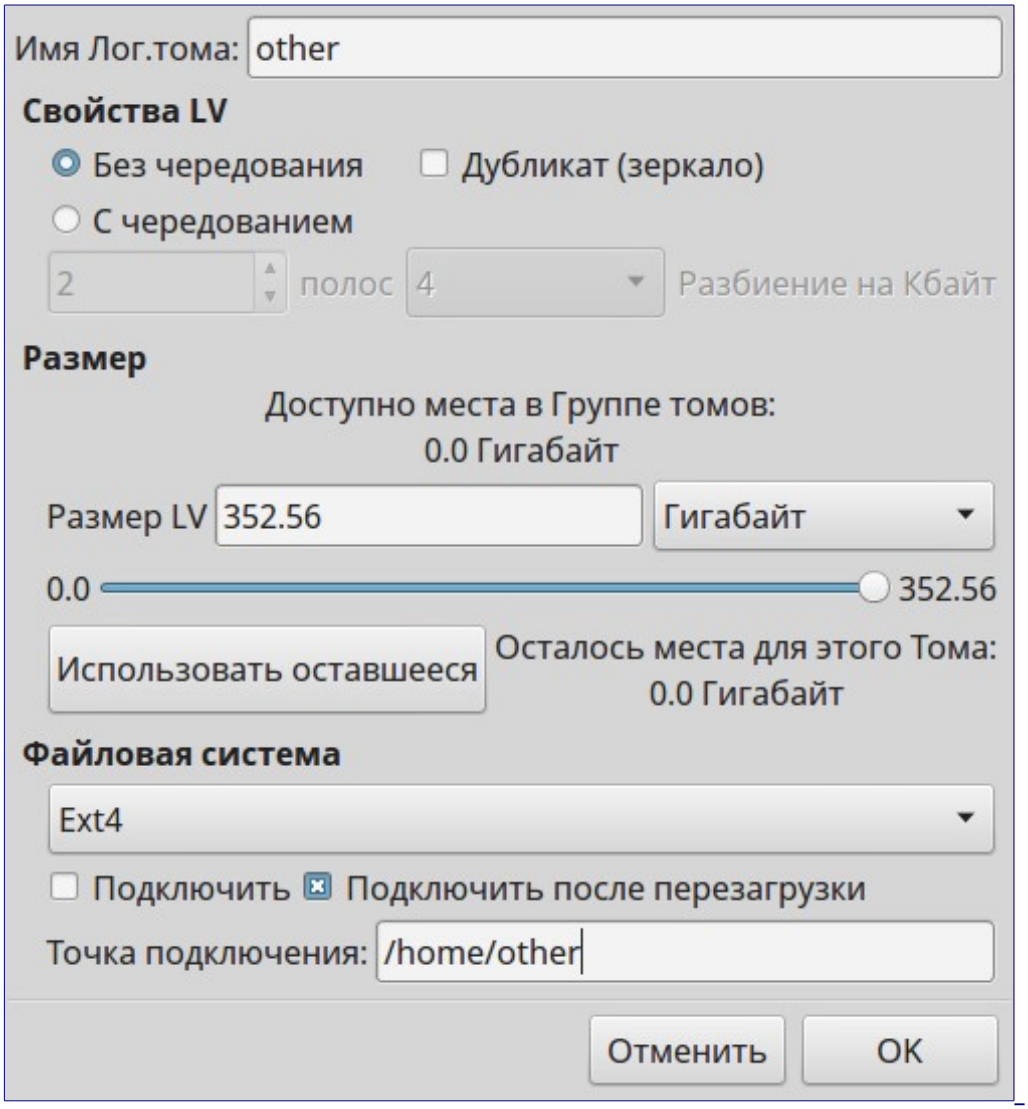
После чего осталось указать условия монтирования тома точку для оного:



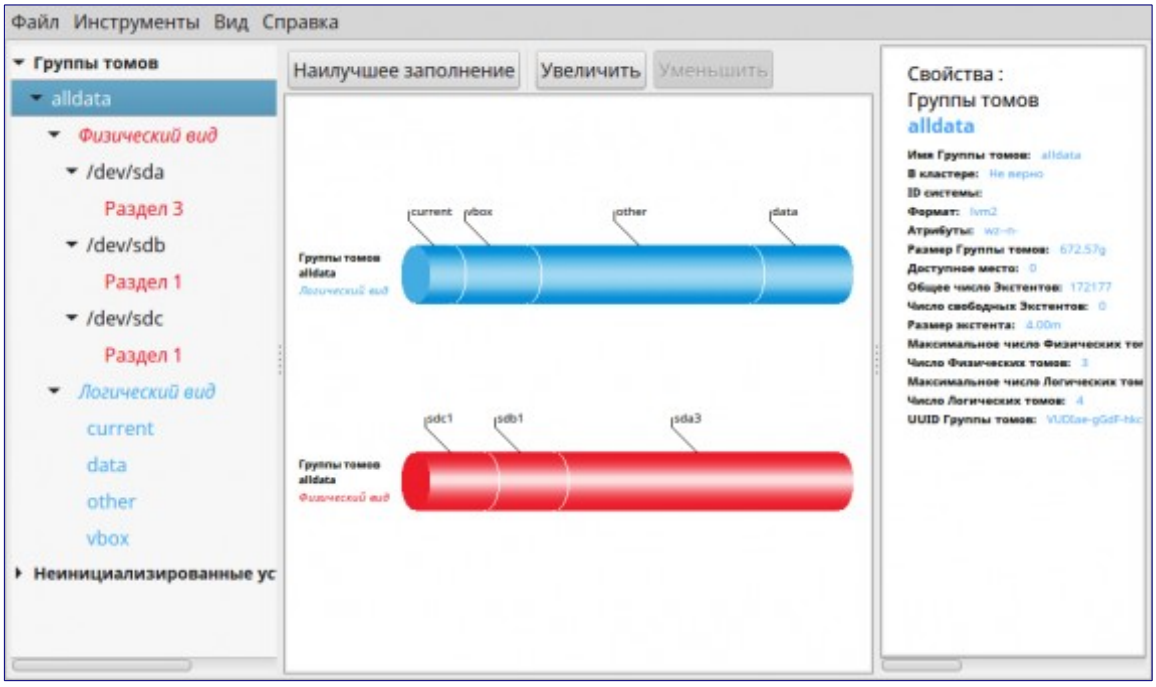
Нажав OK, я получил сообщение, что и такой точки не существует и предложение её создать, от которого, разумеется, невозможно отказаться:



Тем же манеров был создан том для размещёния виртуальных машин — как мне объяснили резонные люди, для них быстродействие дисковых операций также важно. На этом лимит томов с чередованием у меня был исчерпан, и оставшееся пространство я разделил на два «линейных» тома — под старые, но периодически нужные проекты, и под всякую всячину типа мультимедии:



После чего физическая и логическая картины моей группы томов стали выглядеть так:



Оставалось перезагрузиться (для страховки, можно было заказать и немедленное монтирование томов) и начать заполнять тома данными из бэкапов, разумеется, сделанных до начала не только развлечений с LVM, но и

до установки Rebecca.

Mint и ZFS

Соедующая серия очерков посвящена ZFS — универсальной системе размещёния данных, интегрирующей в себе собственно файловую систему и технологию управления дисковыми массивами и логическими томами. Если softRAID и LVM посвящено множество сетевых материалов, то тема ZFS в Linux'е (так называемая ZFS on Linux) на русском язые освящена существенно слабее. Поэтому я и решил остановиться на ней подробно.

Общее введение

Одна из главнейших задач при работе на компьютере — манипулирование данными: создание, модификация, копирование, перемещёние и так далее. И тут первое — это организация их размещёния. Это понятие включает в себя широкий круг частных вопросов — схемы дисковой разметки, управления дисковыми массивами и логическими томами, файловые системы и их монтирование в файловую иерархию. Они тесно связаны между собой, но традиционно решаются каждая с помощью собственного инструментария.

Однако в последние годы в Linux’е получили распространение интегрированные системы размещёния данных, объединяющие в себе и файловые системы, и задачи управления массивами и томами, и даже, частично, задачи разметки дисков. Такие системы существовали очень давно — со времен доисторического UNIX’а, но были они проприетарными. ZFS же, разработанная фирмой Sun для своей ОС Solaris, ныне распространяется свободно, под лицензией CDDL. Благодаря чему была портирована на FreeBSD, а в последние годы нативно поддерживается и в Linux’е.

Именно ZFS on Linux и будет героиней нашего романа, и не только в силу своих несравненных достоинств. А во-вторых, развитие проекта ZFS on Linux блестяще демонстрирует торжество инженерного разума над юридической заумью. И потому являет собой просто замечательный литературный сюжет, мимо которого не в силах пройти ни один сочинитель в жанре технологической новеллы. И начать этот сюжет надо издалека.

Дисковая разметка

Говорят, что во времена далекие, теперь почти былинные, файловых систем не было: информация на носители записывалась побитно, без всякой организации в именованные её наборы. Впрочем, такой способ записи данных применялся и много позднее — например, при резервном копировании на стриммерные ленты. Можно обходиться без файловых систем и при записи на стандартные блочные устройства — винчестеры, SSD, компакт-диски.

Однако в большинстве случаев данные на носителях блочного типа организуются в виде файлов, а файлы объединяются в файловые системы — плоские, как в древнем DOS’е, древовидные, как во всех UNIX-подобных операционках, или, так сказать, «многодревные», как в Windows. Каковые могут быть созданы непосредственно на носителе как raw-устройстве, но обычно накладываются на дисковые разделы.

Исторически сложилось так, что одному разделу соответствовала одна файловая система. Соответственно, и выходить за границы несущего их устройства файловые системы не могли. И если требовалось работать более

чем с одной файловой системой на одном физическом накопителе (а в UNIX-подобных ОС это почти всегда так), то был необходим тщательный расчет дискового пространства для каждой из них: ошибки в расчётах влекли весьма неприятные последствия, вплоть до необходимости переразбиения диска и переустановки ОС вообще.

Правда, дисковые разделы могут не только разделяться, но и объединяться в программные массивы или в группы томов, о которых говорилось в предыдущих очерках.

Файловые системы

Как известно ещё с советских атеистических времен, Господь Бог, создавая человека, хотел сделать его умным, честным и партийным. Но оказалось, что даже он, при всём своём всемогуществе, не смог ему дать больше двух качеств вместе.

Аналогично и с файловыми системами: разработчики хотели бы видеть их быстрыми, надежными и простыми в обращении. Давайте посмотрим, удалось ли им превзойти Господа.

В UNIX-подобных системах требование быстродействия удовлетворяется, во-первых, оптимизированным расположением каталогов, метаданных и данных файлов на физических носителях. Но во-вторых и главных — кэшированием записи.

Думаю, каждого, кто начинал знакомство с Linux’ом во времена безраздельного господства файловой системы ext2fs, поражала быстрота выполнения всех файловых операций, обусловленная их асинхронностью — то есть кэшированием данных и метаданных. Оборотная сторона медали — отказ системы по любой причине влёк за собой тяжкие последствия, вплоть до полного ее разрушения. Но и даже когда до полной катастрофы дело не доходило, отказы (например, по питанию) вызывали за собой долгую и нудную процедуру проверки целостности файловой системы.

Были разработаны различные механизмы решения этой проблемы. Однако основным в Linux стало так называемое журналирование, когда сведения о файловых операциях записываются в специальный файл журнала до того, как эти операции будут фактически выполнены. Это дает возможность после любого сбоя «откатить» файловую систему до последнего непротиворечивого состояния. Оборотной стороной чего, как обычно, является снижение быстродействия — различное для отдельных файловых систем и видов файловых операций.

Правда, с точки зрения простоты использования ни в одну из файловых систем Linux’а бросить камень рука не подымется: создание и монтирование их никаких трудностей не сулит. Так что требование «партийности» выполняется, пожалуй, при всех соотношениях «ума» и «честности». Но эта ситуация сохраняется, пока мы не начинаем комбинировать «ум, честность и партийность» файловых систем с аналогичными качествами систем управления RAID’ами или с LVM. В результате чего получаем:

- либо быстрое и относительно простое решение на основе RAID Level 0, не блещущее надёжностью;

• либо надёжное решение без ощутимой потери быстродействия на основе одного из RAID с избыточностью, не являющееся, однако, эталоном простоты;

• либо, наконец, относительно надёжное и потенциально быстрое решение при использовании технологии LVM — однако и оно не блещёт простотой.

Ибо все эти решения — многоуровневые. И очевидно, что удлинение «цепочки» уровней в любом случае приводит к снижению надежности: чем больше в ней звеньев, тем вероятней отказ всей цепи.

И тут-то и возникает вопрос: а нельзя ли уменьшить количество уровней, сделать систему более «плоской»? И системы размещёния данных, в том числе и ZFS — попытка ответа на него.

Из истории систем размещёния

Не в интересах правды, а истины ради нужно заметить, что ZFS была отнюдь не первой комплексной системой размещёния данных — хотя её исторические предшественницы также именовались просто файловыми системами.

Первой из таких предшественниц была, видимо, файловая система Veritas (или VxFS), разработанная фирмой Veritas Software и представленная миру в 1991 году. Она же претендует на звание первой в истории мироздания журналируемой файловой системы. Хотя, насколько мне известно, JFS — эпоним всех журналируемых ФС — в своей реализации для AIX появилась в 1990 году, так что вопрос приоритета остаётся не вполне ясным.

VxFS была основной файловой системой в HP UX, работает также во всех ныне живущих проприетарных UNIX’ах и теоретически может использоваться в Linux’е. Однако о практических примерах последнего я не слышал: VxFS является системой проприетарной и весьма дорогой.

VxFS тесно интегрирована с менеджером логических томов — VxVM. Благодаря чему в ней возможно изменение (в любую сторону) размера файловой системы «на лету», включение различных режимов использования томов — стриппинг данных, их зеркалирование, а также комбинации того и другого, создание избыточных массивов по типу RAID Level 5, изменение внутренней организации данных без остановки работы. Всё это позволяет VxFS (в сочетании с VxVM) претендовать на звание комплексной системы размещёния данных.

Впрочем, не меньше к тому оснований было и у AdvFS — файловой системы, разработанной к 1993 году фирмой DEC для своего проприетарного варианта UNIX, именовавшегося сначала OSF/1, затем Digital UNIX, и завершившего свою жизнь под именем Tru64 UNIX. Судьба её была печальной. Снискав заслуженное признание на своей родной платформе DEC Alpha под управлением указанной ОС, она после покупки DEC фирмой Compaq оказалась в загоне. А после того, как Compaq, в свою очередь, был поглощён фирмой Hewlett Packard, использовавшей для своего UNIX’а на платформах HP PA и Itanium только что упомянутую VxFS, AdvFS оказалась совсем не при делах.

В результате HP сделала щедрый дар сообществу свободного софта вообще

и Linux-сообществу в особенности: в середине 2008 года исходники файловой системы AdvFS были открыты под лицензией GPv2 — ради максимальной совместимости с ядром Linux. С предложением использовать их в качестве богатой технологической базы для этой ОС. Правда, оговорка, что сама HP не заинтересована в дальнейшем развитии AdvFS заставляла вспомнить народную присказку: «Возьми, боже, что мне не гоже».

Да и предложение несколько запоздало: как мы скоро увидим, к тому времени интенсивно развивались и ZFS, и btrfs.

Однако, помимо исходников, HP предоставила также доступ ко всей документации — благодаря чему об AdvFS при желании можно узнать больше, чем о любой другой проприетарной файловой системе для UNIX-подобных операционок. Это избавляет меня от необходимости описания особенностей AdvFS. Замечу только, что среди них мы увидим все черты развитой комплексной системы размещёния данных. Те самые, с которыми ознакомимся, когда дело дойдёт наконец до рассмотрения устройства ZFS. Но для начала перейдём к обзору уже её истории.

Начало истории ZFS

Разработчики ZFS поставили себе честолюбивую цель: создать систему хранения данных, которая отвечала бы всем трем критериям идеала. Разработка её проводилась в компании Sun Microsystems, командой под руководством Джеффа Бонвика и Мэттью Аренса (Matthew Ahrens). Первоначально название ZFS рассматривалось как аббревиатура от Zettabyte File System, но быстро стало просто условным именованием. Его можно интерпретировать как последнюю точку в развитии файловых систем вообще. И в последующем мы увидим: это недалеко от истины.

Результаты работы над ZFS были представлены миру в августе 2004 года. А в 2006 году она была включена в штатный состав OS Solaris 10 (релиз 6/06). То есть, подобно своим предшественницам, она также была проприетарным продуктом. И пользователям свободных UNIX-подобных систем поначалу от ее существования было ни холодно, ни жарко. Однако период камерного существования ZFS продолжался недолго — уже в ноябре 2005 года, то есть до включения в Solaris, ее поддержка была интегрирована в открытый её вариант, OpenSolaris. Ибо она основывалась на том же ядре SunOS 5, что и коммерческий прототип.

Исходники ZFS распространяются, как и собственно OpenSolaris, под лицензией CDDL (Common Development and Distribution License). Эта лицензия, базирующаяся на Mozilla Public License (MPL), не влияет на общую лицензию проекта, в состав который включены CDDL-компоненты. И потому оказывается совместимой с большинством свободных лицензий. За исключением... какой? Правильно, GPL во всех её проявлениях.

Разумеется, ZFS была задействована в клонах openSolaris, таких, как BeleniX, SchilliX и, в первую голову, в Nexenta OS. Правда, последняя развивалась в направлении коммерческой системы хранения данных, а о числе пользователей остальных можно было только гадать.

Некоторое время ZFS была доступна пользователям Macintosh’а — в Mac OS X Leopard от осени 2007 года. Правда, ходившие перед её выходом слухи, что она будет там файловой системой по умолчанию, оказались несколько преувеличенными: поддержка ZFS оказалась опциональной и лишь в режиме

«только для чтения». А в последующих версиях семейства кошачьих вообще исчезла и, видимо, уже не возродится.

Так что для широких народных масс ZFS по прежнему оставалась недоступной. Пока... пока ее не портировали под FreeBSD в 2007 году, и официально не включили её поддержку в 7-ю версию этой ОС, вышедшую в начале 2008 года. В чём, как и в дальнейшем её развитии, основная заслуга принадлежит Павлу-Якубу Давидеку (Pawel Jakub Dawidek) и Ивану Ворасу (Ivan Voras). Правда, до недавнего времени ZFS нельзя было задействовать при установке FreeBSD средствами её штатного инсталлятора и конфигуратора sysinstall. Однако это без труда можно было осуществить в дальнейшем руками. В том числе и разместить на ZFS корень файловой иерархии.

С самого начала поддержки ZFS во FreeBSD появилась и возможность задействовать её, что называется, «искаропки», в десктоп-ориентированном клоне последней — PC-BSD. А с переходом FreeBSD, начиная с версии 9.0, на новую программу установки — BSDInstall, эта функция распространилась и на материнскую систему.

Успех ZFS во FreeBSD, где она стала если не главной файловой системой, то добилась равноправия с UFS2, послужил примером для других BSD-систем. Так, ныне ZFS поддерживается в NetBSD — эта работа была начата Оливером Голдом (Oliver Gould) летом 2007 года в рамках акции Google Summer of Code. А в 2009 году Адам Хамсик (Adam Hamsik) интегрировал её код в ядро NetBSD. Правда, использование ZFS в этой операционке рекомендуется только в экспериментальных целях.

Наконец, одно время в списках рассылки DragonFlyBSD активно обсуждался вопрос о портировании ZFS и на эту операционку. Потом, правда, разговоры эти стихли — вероятно, в связи с активной разработкой файловой системы Hammer, обладающей во многом аналогичными возможностями. Однако, учитывая лёгкость адаптации к DragonFlyBSD любых сторонних файловых систем, можно не сомневаться, что поддержка ZFS на уровне обмена данными будет включена в неё тогда и если (или если и тогда), когда (и если) это кому-то понадобится.

Таким образом, пользователям большинства BSD-систем ZFS или уже доступна как нативная, или может стать доступной в ближайшее время.

Из истории юриспруденции

А что же Linux, спросите вы меня? Как обстоит дело с поддержкой ZFS в самой массовой из свободных UNIX-подобных операционных систем нашего времени? А вот с Linux’ом все оказывается гораздо сложнее. Ибо не зря поминали мы выше лицензию CDDL. Которая сама по себе очень даже свободная, и не накладывает почти никаких ограничений на распространение защищаемых ею программ.

В частности, не запрещает CDDL и коммерческого распространения производных продуктов в виде бинарников, без открытия исходных текстов. Как известно, не накладывает такого ограничения и лицензия BSD, почему включение кода поддержки ZFS в любые BSD-системы и проходит юридически безболезненно, как мы только что видели на примере FreeBSD.

А вот с лицензией GPL обеих актуальных версий (v2 и v3) CDDL входит в

диалектическое противоречие. Ибо любые продукты, производные от программ под GPL, вне зависимости от формы распространения, должны сопровождаться исходными текстами. Что делает юридически невозможным включение кода поддержки ZFS непосредственно в ядро Linux, распространяемое, как известно, на условиях GPLv2.

Кроме того, невозможность включения в ядро Linux кода поддержки ZFS объясняется тем, что GPL требует распространения всех основанных на ней продуктов под GPL же, тогда как CDDL — сохранения её для «своих» компонентов.

Правда, часть кода ZFS была открыта под GPL с тем, чтобы соответствующий патч можно было включить в загрузчик GRUB. Это обеспечило возможность загрузки Open Solaris непосредственно с ZFS-раздела. Однако оказалось недостаточным для полноценной реализации этой системы, которую можно было бы распространять под данной лицензией.

Впрочем, не будучи юристом, ломать голову над лицензионными вопросами не буду, и моим читателям не советую, ибо понять это всё равно невозможно. А достаточно лишь запомнить, что всеми резонными и юридически подкованными людьми признано, что поддержки ZFS в ядре Linux быть не может.

Таким образом, сложилась абсурдная, с точки зрения здравого смысла, ситуация: два программных продукта под свободными лицензиями (обсуждать вопрос, какая из них «свободней другой», мы сейчас не будем), созданные друг для друга, как Huggies и... э-ээ... место пониже спины (дальнейшие события показали, что технических сложностей при портировании ZFS на Linux практически нет), невозможно было использовать в составе одного проекта. По крайней мере, для законопослушных граждан, чтущих... нет, не уголовный кодекс, а принципы свободного программного обеспечения.

И, разумеется, здравомыслящие люди попытались эту ситуацию разрешить. И первая такая попытка была предпринята ещё в 2006 году в рамках Google Summer of Code. Основывалась она на поддержке ZFS через FUSE (Filesystem in Userspace). Поскольку модуль FUSE работает как пользовательское приложение, необходимости во включение кода ZFS в ядро Linux нет, что снимает все юридические вопросы. Однако встают вопросы другие — производительности и устойчивости.

Проект ZFS-FUSE развивается по сей день, хотя и не очень быстрыми темпами. Правда, находясь в стадии хронической бета-версии, он до сих пор рассматривается как сугубо экспериментальный. Да и в любом случае в таком виде ZFS выполнять свои функции — быть надёжным хранилищем данных большого объёма — скорее всего, не сможет.

Так что ZFS-FUSE нельзя считать кардинальным решением вопроса с этой системой размещёния данных в Linux.

Появление героини

И тем не менее, решение этой проблемы нашлось — и решение столь же изящное, сколь и очевидное. Его предложил весной 2010 года Брайан Белендорф, некогда один из основных разработчиков web-сервера Apache. Он создал модуль поддержки ZFS, который собирается и может

распространяться отдельно от ядра, сохраняя прародительскую лицензию CDDL. А поскольку последняя, как уже говорилось, является лицензией «пофайловой», этим самым обходится антагонистическое противоречие — запрет на распространение продуктов, в которых смешан код, лицензируемый под CDDL и GPL.

На базе разработки Брайана возникло сразу два проекта. Первый осуществлялся индийской компанией KQ Infotech, которой уже в сентябре 2010 года удалось выпустить работоспособный, пригодный для тестирования Linux-ядра с реализацией файловой системы ZFS. А в январе следующего, 2011, года появилась финальная его версия, доступная тогда в исходниках и в виде двоичных пакетов для Fedora 14, RHEL6, Ubuntu 10.04 и 10.10.

Однако весной того же года KQ Infotech была куплена фирмой STEC, занимающейся производством SSD-накопителей, каковых, впрочем, в наших палестинах мало кто видел. И работы по дальнейшему развитию нативной поддержки ZFS были свёрнуты. Хотя исходники модуля и сопутствующих компонентов до сих пор доступны, последнее их обновление происходило более года назад. А информации о дальнейшей судьбе проекта с тех пор не появлялось.

Второй проект реализовывался самим Брайном вместе с сотрудниками Ливерморской национальной лаборатории, каковая, будучи в подчинении Министерства энергетики США, занимается не только вопросами ядерного оружия (эвфемизмы вроде Минсредмаша в ходу не только в бывшем Советском Союзе), но и разработкой суперкомьютеров. В результате скоро возник проект ZFS on Linux — ZFS on Linux, в рамках которого модуль поддержки ZFS и сопутствующие утилиты поддержки, портированные из Solaris — так называемый SPL (Solaris Porting Layer), были доведены до ума, и к началу 2011 года стали пригодны для использования в экспериментальном режиме. А к настоящему времени, несмотря на формальное сохранение статуса release candidate, порт ZFS on Linux можно считать готовым к практическому применению.

Правда, майнтайнеры основных дистрибутивов не торопились включать поддержку ZFS в свои системы даже в качестве дополнительных неофициальных пакетов. Подозреваю, что не столько из косности и лени, сколько из-за очередной сложности: видимо, по всё тем же лицензионным ограничениям модули zfs и spl приходится привязывать к фиксированной версии (и даже конкретной сборке) ядра Linux. Что, при регулярных, даже корректирующих, обновлениях последнего требует и их пересборки.

Тем не менее, разработчики проекта воплотили результаты своей работы в виде PPA-репозитория для Ubuntu. Каковым без проблем могут пользоваться и применители Mint.

Обзор возможностей

Прежде чем погружаться в вопросы, связанные с ZFS, читатель, вероятно, хотел бы убедиться в том, что это стоит делать. То есть — ознакомиться с возможностями, которые она ему предоставляет.

Для начала — немного цифр. В отличие от всех предшествовавших файловых систем и систем размещёния данных, ZFS является 128-битной. То есть теоретическое ограничение на её объём и объёмы её составляющих превышают не только реальные, но и воображаемые потребности любого

пользователя. По выражению создателя ZFS, Джеффа Бонвика, для её заполнения данными и их хранения потребовалось бы вскипятить океан.

Так, объём пула хранения данных (zpool — максимальная единица в системе ZFS) может достигать величины 3×1023 петабайт (а один петабайт, напомню, это 1015 или 250 байт, в зависимости от системы измерения). Каждый пул может включать в себя до 264 устройств (например, дисков), а всего пулов в одной системе может быть тоже не больше 264.

Пул может быть разделён на 264 наборов данных (dataset — в этом качестве выступают, например, отдельные файловые системы), по 264 каждая. Правда, ни одна из таких файловых систем не может содержать больше 248 файлов. Зато размер любого файла ограничивается опять же значением в 264 байт.

Количество таких ограничений можно умножить. Как уже было сказано, они лежат вне пределов человеческого воображения. И привожу я их только для того, чтобы вселить в пользователя уверенность: ни он сам, ни его внуки и правнуки в реальности не столкнутся с ограничениями на размер файловой системы или отдельного файла, как это бывало при использовании FAT или ext2fs.

Так что перейду к особенностям ZFS, наиболее интересным, по моему мнению, десктопному пользователю. Здесь в первую очередь надо отметить гибкое управление устройствами. В пул хранения данных можно объединить произвольное (в обозначенных выше пределах) число дисков и их разделов. Устройства внутри пула могут работать в режиме расщепления данных, зеркалирования или избыточности с подсчётом контрольных сумм, подобно RAID’ам уровней 0, 1 и 5, соответственно. В пул можно включать накопители, специально предназначенные для кэширования дисковых операций, что актуально при совместном использовании SSD и традиционных винчестеров.

Пул хранения становится доступным для работы сразу после его создания, без рестарта машины. В процессе работы дополнительные диски или разделы, в том числе и устройства кэширования, могут как присоединяться к пулу, так и изыматься из его состава в «горячем» режиме.

Пул хранения может быть разделён на произвольное количество иерархически организованных файловых систем. По умолчанию размер их не определяется, и растёт по мере заполнения данными. Это избавляет пользователя от необходимости расчёта места, потребного под системные журналы, домашние каталоги пользователей и другие трудно прогнозируемые вещи. С другой стороны, не запрещёно при необходимости и квотирование объёма отдельных файловых систем — например, домашних каталогов отдельных излишне жадных пользователей.

Файловые системы ZFS также доступны для размещёния на них данных сразу после создания, никаких специальных действий по обеспечению их монтирования не требуется. Создание файловых систем внутри пула — процесс предельно простой: разработчики стремились сделать его не сложнее создания каталогов, и это им вполне удалось. Но при этом составляющие пула остаются именно самостоятельными файловыми системами, которые могут монтироваться со своими специфическими опциями, в зависимости от назначения.

Среди других возможностей ZFS, интересных настольному пользователю,

можно упомянуть:

- создание снапшотов файловой системы, позволяющих восстановить её состояние в случае ошибки;
- клонирование файловых систем;
- компрессия данных файловой системы и дедупликация (замена повторяющихся данных ссылками на «первоисточник»);
- создание нескольких копий блоков с критически важными данными и, напротив, возможность отключения проверки контрольных сумм для повышения скорости доступа к ним.

В общем, даже если не говорить об быстродействии ZFS (а оно весьма высоко, особенно в многодисковых конфигурациях), перечислять её достоинства можно очень долго. Так долго, что поневоле успеваешь задаться вопросом: а есть ли у неё недостатки?

Разумеется, есть. Хотя большая их часть — скорее особенности: например, ограничения при добавлении или удалении накопителей в пуле, или отсутствие поддежки TRIM.

По большому счёту для пользователя Linux’а у ZFS обнаруживается два кардинальных недостатка: некоторая усложнённость её использования, обусловленная юридическими факторами, и высокие требования к аппаратуре.

Первый недостаток если не ликвидирован, то сглажен трудами Брайана Белендорфа (Brian Behlendorf) со товарищи и майнтайнерами прогрессивных дистрибутивов вкупе с примкнувшими к ним независимыми разработчиками. Аппаратные же претензии ZFS мы сейчас и рассмотрим.

Аппаратные потребности

Итак, ZFS предоставляет пользователю весьма много возможностей. И потому вправе предъявлять немало претензий к аппаратной части — процессору (изобилие возможностей ZFS создает на него достаточную нагрузку), оперативной памяти и дисковой подсистеме.

Впрочем, претензии эти отнюдь не сверхъестественные. Так, процессор подойдёт любой из относительно современных, начиная, скажем, с Core 2 Duo. Минимальный объём памяти определяется в 2 ГБ, с оговоркой, что применение компрессии и дедупликации требуют 8 ГБ и более.

Сама по себе ZFS прекрасно функционирует и на одиночном диске. Однако в полном блеске показывает себя при двух и более накопителях. В многодисковых конфигурациях рекомендуется разнесение накопителей на разные контроллеры: современные SSD способны полностью загрузить все каналы SATA-III, и равномерное распределение нагрузки на пару контроллеров может увеличить быстродействие.

К «железным» претензиям добавляются и притязания программные. В первую очередь, ZFS on Linux требует 64-битной сборки этой ОС, поскольку в 32-разрядных системах действует ограничение на адресное пространство

физической памяти. Кроме того, в конфигурации ядра должнв быть отключена опция CONFIG_PREEMPT.

Если вас привлекли достоинства ZFS и не устрашили её «железные» аппетиты, самое время опробовать её в деле. Что потребует знакомства с некоторыми специфическими понятиями.

Терминология

Центральным понятием ZFS является пул хранения данных (zpool). В него может объединяться несколько физических устройств хранения — дисков или дисковых разделов, причём первый вариант рекомендуется. Но не запрещёно и создание пула из одного диска или его раздела.

Каждый пул состоит из одного или нескольких виртуальных устройств (vdev). В качестве таковых могут выступать устройства без избыточности (то есть всё те же диски и/или их разделы), или устройства с избыточностью — зеркала и массивы типа RAID-Z:

Зеркальное устройство (mirror) — виртуальное устройство, хранящее на двух или более физических устройствах, но при чётном их количестве, идентичные копии данных на случай отказа диска;

RAID-Z — виртуальное устройство из нескольких устройств физических, предназначенное для хранения данных и их контрольных сумм с однократным или двойным контролем чётности. В первом случае теоретически требуется не менее двух, во втором — не менее трёх физических устройств.

Если пул образован устройствами без избыточности (просто дисками или разделами), то одно из vdev, соответствующее ему целиком, выступает в качестве корневого устройства. Пул из устройств с избыточностью может содержать более одного корневого устройства — например, два зеркала.

Пулы, образованные виртуальными устройствами, служат вместилищем для наборов данных (dataset). Они бывают следующих видов:

- файловая система (filesystem) — набор данных, смонтированный в определённой точке и ведущий себя подобно любой другой файловой системе;
- снапшот (snapshot) — моментальный снимок текущего состояния файловой системы, доступный только для чтения;
- клон (clone) — точная копия файловой системы в момент его создания; создаётся на основе снимка, но, в отличие от него, доступен для записи;
- том (volume) — набор данных, эмулирующий физическое устройство, например, раздел подкачки.

Наборы данных пула должны носить уникальные имена такого вида:

```
pool_name/path/(dataset_name)(@snapshot_name)
```

Пулы и наборы данных в именуются по правилам пространства имён ZFS, впрочем, довольно простым. Запрещёнными символами для всех являются

символы подчёркивания, дефиса, двоеточия, точки и процента. Имя пула при этом обязательно должно начинаться с алфавитного символа и не совпадать с одним из зарезервированных имён — log, mirror, raidz или spare (последнее обозначает имя устройства «горячего» резерва). Все остальные имена, в соответствие с демократическими традициями пространства имён ZFS, разрешены.

А вот об именах физических устройств, включаемых в пул, следует сказать особо.

Модели именования устройств

В современном Linux’е использование для накопителей имён «верхнего уровня», имеющих вид /dev/sda, не является обязательным, а в некоторых случаях и просто нежелательно. Однако правила менеджера устройств udev позволяют определять и другие модели идентификации накопителей:

- метке тома (/dev/disk/by-label);
- идентификатору диска (/dev/disk/by-id);
- пути к дисковому устройству (/dev/disk/by-path);
- универсальному уникальному идентификатору, Universally Unique IDentifier (/dev/disk/by-uuid).

А с полным списком вариантов идентификации блочных устройств можно ознакомиться, просмотрев имена подкаталогов в каталоге /dev/disk, их содержимое — это символические ссылки на имена «верхнего уровня».

С идентификацией по метке тома и по UUID, вероятно, знакомо большинство читателей. И к тому же в пространстве имён ZFS они не используются. А вот с идентификацией by-path и by-id нужно познакомиться поближе.

Модель именования by-path использует имена устройств, привязанные к их положению на шине PCI и включающие номер шины и канала на ней. Имя дискового устройства выглядит примерно так:

```
pci-0000:00:1f.2-scsi-0:0:0:0
```

Дисковые разделы маркируются добавлением к имени устройства суффикса part#.

Модель именования by-path идентифицирует устройства вполне однозначно, и особенно эффективна при наличии более чем одного дискового контроллера. Однако сами имена и устройств, и разделов описываются довольно сложной для восприятия последовательностью. Да и в большинстве «десктопных» ситуаций модель эта избыточна.

Модель идентификации by-id представляет имена носителей информации в форме, наиболее доступной для человеческого понимания. Они образованы из названия интерфейса, имени производителя, номера модели, серийного номера устройства и, при необходимости, номера раздела, например:

ata-SanDisk_SDSSDX120GG25_120823400863-part1

Таким образом, все компоненты имени устройства в модели by-id определяются не условиями его подключения или какими-то правилам, а задаются производителем и жёстко прошиты в «железе». И потому эта модель является наиболее однозначной для именования устройств. А также, что немаловажно, строится по понятной человеку логике.

Какую из моделей именования устройств выбрать для данного пула — зависит от его назначения и масштабов. Имена «верхнего уровня» целесообразно применять для однодисковых пулов (особенно если в машине второго диска нет и не предвидится). Они же, по причине простоты и удобопонятности, рекомендуются для экспериментальных и разрабатываемых пулов. И очень не рекомендуются — во всех остальных случаях, так как зависят от условий подключения накопителей.

Этого недостатка лишена модель by-id: как пишет Брайан, при её использовании «диски можно отключить, случайно смешать и подключить опять произвольным образом — и пул будет по-прежнему корректно работать». Как недостаток её рассматривается сложность конфигурирования больших пулов с избыточностью. И потому она рекомендуется для применения в «десктопных» и «квартирных» (типа семейного сервера) условиях.

Для больших (более 10 устройств) пулов из дисков, подключённых к нескольким контроллерам, рекомендуется идентификация by-path. Однако в наших целях она громоздка и избыточна.

Наконец, ZFS on Linux предлагает и собственную модель идентификации — /dev/disk/zpool, в котором именам by-path ставятся в соответствие уникальные и осмысленные «человекочитаемые» имена, даваемые пользователем. Модель эта рекомендуется для очень больших пулов, каковых на настольной машине ожидать трудно.

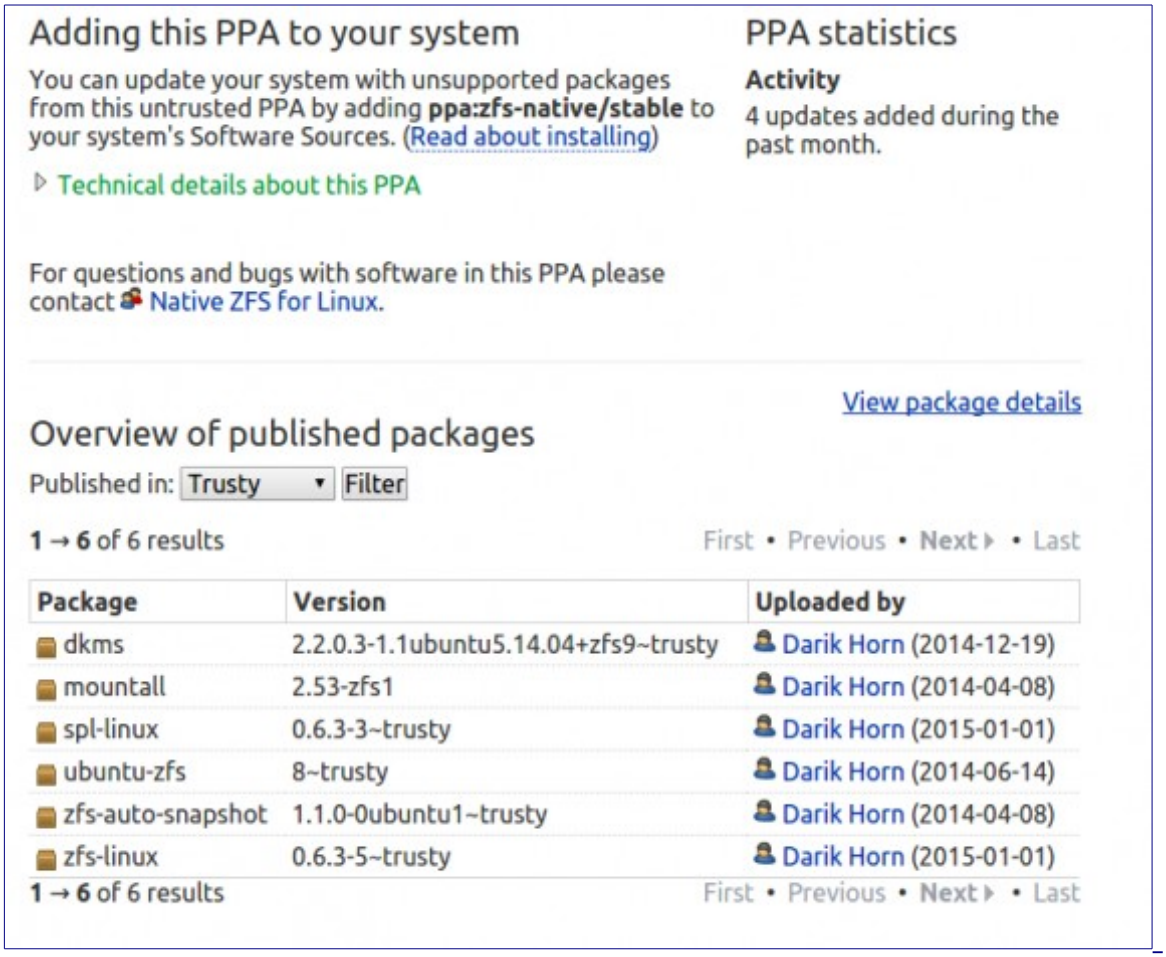
Так что дальше я буду использовать имена «верхнего уровня», говоря об абстрактных или экспериментальных ситуациях, и об именах by-id, когда речь зайдёт о практических примерах применения ZFS.

Включение поддержки ZFS в Mint

Для практического использования ZFS on Linux перво-наперво необходимо обеспечить её поддержку в вашем дистрибутиве — ибо по причинам, описанным ранее, сама собой она не поддержится ни в одном Linux’е.

Как это сделать, зависит от дистрибутива. В Сети можно найти подробные инструкции для Ubuntu, которые легко распространяются на все производные от неё системы, в том числе и на Mint.

Как уже было сказано, пакеты поддержки ZFS представлены в PPA-репозитори, где они реализованы в виде сценариев dkms, предполагающих сборку модулей под текущую версию ядра. Пакеты эти объединены в метапакет zfs-native, существующий в двух варианта — ZFS Stable Releases и ZFS Daily Releases, то есть стабильной и тестовой сборках, соответственно.



Adding this PPA to your system

You can update your system with unsupported packages from this untrusted PPA by adding **ppa:zfs-native/stable** to your system's Software Sources. (Read about installing)

▷ Technical details about this PPA

For questions and bugs with software in this PPA please contact Native ZFS for Linux.

PPA statistics

Activity

4 updates added during the past month.

View package details

Overview of published packages

Published in: Trusty ▾ Filter

1 → 6 of 6 results — First • Previous • Next ▸ • Last

Package	Version	Uploaded by
dkms	2.2.0.3-1.1ubuntu5.14.04+zfs9~trusty	Darik Horn (2014-12-19)
mountall	2.53-zfs1	Darik Horn (2014-04-08)
spl-linux	0.6.3-3~trusty	Darik Horn (2015-01-01)
ubuntu-zfs	8~trusty	Darik Horn (2014-06-14)
zfs-auto-snapshot	1.1.0-0ubuntu1~trusty	Darik Horn (2014-04-08)
zfs-linux	0.6.3-5~trusty	Darik Horn (2015-01-01)

1 → 6 of 6 results — First • Previous • Next ▸ • Last

Для использования ZFS в Ubuntu для начала нужно подключить нужный PPA-репозиторий. Поскольку все последующие действия потребуют прав суперпользователя, перво-наперво обретаем их на длительное время командой

```
$ sudo -i
```

с вводом пользовательского пароля. А затем собственно подключаем репозиторий:

```
# add-apt-repository ppa:zfs-native/stable
```

Или, при желании поэкспериментировать --

```
# add-apt-repository ppa:zfs-native/daily
```

Обновляем кэш:

```
# apt update
```

Теперь строим дерево зависимостей — в Mint 17.1 Rebecca это обязательный шаг, хотя ранее я обходился без него:

```
# apt build-dep ubuntu-zfs
```

И собираем необходимые пакеты:

```
# apt install ubuntu-zfs
```

Поскольку в репозитории они существуют не в бинарном виде, а в виде исходников, приведённая команда потянет за собой сборочный инструментарий. И сама сборка пакетов займёт определённое время. Но рано или поздно она закончится, и можно будет скомандовать

```
# modprobe zfs
```

и проверить результат командой

```
# lsmod | grep zfs
```

вывод которой будет выглядеть примерно так:

```
zfs                  1158757  4
zcommon                51283  1 zfs
znvpair                81997  2 zfs,zcommon
zavl                   15011  1 zfs
zunicode              331226  1 zfs
spl                    88617  5 zfs,zcommon,znvpair,zavl,zunicode
```

После чего остаётся создать точку монтирования для пула ZFS — в моём случае таким образом:

```
# mkdir /home/data
```

Дать ей атрибуты принадлежности обычному пользователю:

```
# chown -R alv:alv /home/data
```

Теперь можно приступать к применению ZFS в мирных практических целях.

Создаём простой пул

Освоив ранее основные понятия, мы научились понимать ZFS. Для обратной же задачи — чтобы ZFS понимала нас — нужно ознакомиться с её командами. Главные из них — две: zpool для создания и управления пулами, и zfs для создания и управления наборами данных. Немного, правда? Хотя каждая из этих команд включает множество субкоманд, с которыми мы со временем разберёмся.

Очевидно, что работу с ZFS следует начинать с создания пула хранения. Начнём с этого и мы. В простейшем случае однодисковой конфигурации это делается так:

```
# zpool create tank dev_name
```

Здесь create — субкоманда очевидного назначня, tank — имя создаваемого пула (оно обычно даётся в примерах, но на самом деле может быть любым — с учётом ограничений ZFS, я использую имя data), а dev_name — имя устройства, включаемого в пул. Каковое может строиться по любой из описанных ранее моделей. И, чтобы не повторяться, напомню: все команды по

манипуляции с пулами и наборами данных в них выполняются от лица администратора.

В случае, если в состав пула включается один диск, и второго не предвидится, можно использовать имя устройства верхнего уровня — например, sda для цельного устройства (обратим внимание, что путь к файлу устройства указывать не нужно). Однако реально такая ситуация маловероятна: загрузка с ZFS проблематична, так что как минимум потребуется раздел с традиционной файловой системой под /boot (и/или под корень файловой иерархии), так что команда примет вид вроде следующего:

```
# zpool create data sda3
```

Однако если можно ожидать в дальнейшем подсоединения новых накопителей и их включения в существующий пул, то лучше воспользоваться именем по модели by-id, например:

```
# zpool create data ata-Crucial_CT512MX100SSD1_14330CEEA98C-part3
```

Очевидно, что в случае однодискового пула ни о какой избыточности говорить не приходится. Однако уже при двух дисках возможны варианты. Первый — создание пула без избыточности:

```
# zpool create data dev_name1 dev_name2
```

где dev_name1 и dev_name1 — имена устройств в принятой модели именования.

В приведённом примере будет создано нечто вроде RAID’а нулевого уровня, с расщеплением (stripping) данных на оба устройства. Каковыми могут быть как дисковые разделы, так и диски целиком. Причём, в отличие от RAID0, диски (или разделы) не обязаны быть одинакового размера.

После указанной команды никаких сообщений не последует. No news — good news, говорят англичане; в данном случае это означает, что пул был благополучно создан. В чём можно немедленно убедиться двумя способами. Во-первых, в корневом каталоге появляется точка его монтирования /data. А во-вторых, этой цели послужит субкоманда status:
zpool status data

которая выведет нечто вроде этого:

```
pool: data

state: ONLINE

scan: none requested

config:



NAME        STATE     READ WRITE CKSUM

mypool      ONLINE       0     0     0
```

```
sdd       ONLINE       0     0     0

sdf       ONLINE       0     0     0


errors: No known data errors
```

А с помощью субкоманды list можно узнать объём новообразованного пула:

```
# zpool list data

NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT

data  18,9G    93K  18,9G     0%  1.00x  ONLINE  -
```

Легко видеть, что он равен сумме объёмов обеих флэшек, если «маркетинговые» гигабайты пересчитать в «настоящие».

К слову сказать, если дать субкоманду list без указания аргумента — имени пула, то она выведет информацию о всех пулах, задействованных в системе. В моём случае это выглядит так:

```
# zpool list

NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT

exp   18,9G    93K  18,9G     0%  1.00x  ONLINE  -

data  199G  20,8G   178G    10%  1.00x  ONLINE  -
```

Обращаю внимание, что даже чисто информационные субкоманды вроде list и status требуют прав администратора.

Разумеется, два пула в одной, да ещё и настольной, машине — излишняя роскошь. Так что пул, созданный в экспериментальных целях, подлежит уничтожению, что делается с помощью субкоманды destroy:

```
# zpool destroy exp
```

После чего он пропадёт из списка пулов. А что можно сделать с пулом до его уничтожения, увидим со временем.

«Избыточные» пулы

Избавившись от ставшего ненужным пула, рассмотрим второй вариант — создание пула с зеркальным устройством. Создаём его из двух накопителей одинакового объёма:

```
# zpool create -f exp2 mirror sdf sdg
```

Проверка показывает, что итоговый пул, как и следовало ожидать, равен объёму одного накопителя:

```
# zpool list mypool
```

```
NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT

exp2  3,72G  91,5K  3,72G     0%  1.00x  ONLINE  -
```

При различии объёмов больший диск будет «обрезан» до объёма меньшего.

Полное зеркалирование любыми, по моему мнению, в настольных условиях — роскошь непозволительная: банальные бэкапы данных проще и надёжнее. Тем не менее, не исключаю, что некоторая избыточность на уровне проверки контрольных сумм может оказаться не лишней, да и не столь накладна. Так что давайте посмотрим и на третий вариант пула из более чем одного устройства — RAID-Z.

Теоретически виртуальное устройство с одинарным контролем чётности, как уже говорилось, можно создать при наличии двух устройств физических. Однако практически это оказывается накладно, особенно если устройства не одинакового размера. Поэтому задействуем под него три накопителя:

```
# zpool create exp3 raidz sdd sdf sdg
```

что даст нам следующую картину:

```
# zpool list exp3

NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT

exp3  11,1G   205K  11,1G     0%  1.00x  ONLINE  -
```

Впрочем, как мне кажется, в настольных условиях не стоит выделки и эта овчинка.

Пул кэшируемый

И, наконец, последний вариант организации пула из более чем одного устройства — создание пула с кэшированием. Для чего создаём из двух устройств простой пул без избыточности и подсоединяем к нему устройство для кэша:

```
# zpool create exp4 sdd sdf cache sdg
```

Очевидно, что устройство для кэширования не должно входить в пул любого рода — ни в простой, ни в избыточный. Что мы и видим в выводе субкоманды list:

```
# zpool list exp4

NAME    SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT

exp4  18,9G    82K  18,9G     0%  1.00x  ONLINE  -
```

где никаких следов его обнаружить не удаётся. Если же появляются сомнения, а подключилось ли оно на самом деле, обращаемся к субкоманде status, которая покажет беспочвенность наших опасений.

Как я уже говорил в обзоре возможностей ZFS, подключение устройства кэширования имеет смысл при наличии большого традиционного винчестера

(или винчестеров) и относительно небольшого SSD, которое и играет роль дискового кэша.

О некоторых опциях команды zpool

Команда zpool поддерживает ещё множество субкоманд, предназначенных для экспорта и импорта пула, добавления к нему устройств и изъятия оных, и так далее. Но сейчас я расскажу о некоторых опциях, которые могут оказаться необходимыми при создании пула.

Одна из важный опций — -f: она предписывает принудительное выполнение данной операции и требуется, например, при создании пула из неразмеченных устройств.

Полезной может оказаться опция -n. Она определяет тестовый режим выполнения определённой субкоманды, то есть выводит результат, например, субкоманды zpool create без фактического создания пула. И, соответственно, сообщает об ошибках, если таковые имеются.

Интересна также опция -m mountpoint. Как уже говорилось, при создании пула по умолчанию в корне файловой иерархии создаётся каталог /pool_name, который в дальнейшем будет точкой монтирования файловых систем ZFS. Возможно, что это окажется не самым лучшим местом для их размещёния, и, как мы увидим в дальнейшем, это несложно будет изменить. Но можно задать каталог для пула сразу — например, /home/data: это и будет значением опции -m. Никто не запрещает определить в качестве такового и какой-либо из существующих каталогов, если он пуст, иначе автоматическое монтирование файловых систем пула в него окажется невозможным.

Наконец, нынче важное значение приобретает опция ashift=#, значением которой является размер блока файловой системы в виде степеней двойки. По умолчанию при создании пула размер блока определяется автоматически, и до некоторого времени это было оптимально. Однако затем, с одной стороны, появились диски так называемого Advanced Format, в других размер блока равен 4 КБ. С другой стороны, получили распространение SSD-накопители, обычно также имеющие четырёхкилобайтный блок. В этих условиях автоматика ZFS может работать некорректно, что приводит к падению производительности пула.

Для предотвращения означенного безобразия и была придумана опция ashift. Значение её по умолчанию — 0, что соответствует автоматическому определению размера блока. Прочие же возможные значения лежат в диапазоне от 9 для блока в 512 байт (29 = 512) до 16 для 64-килобайтного блока (216 = 65536). В интересующем нас случае четырёхкилобайтного блока оно составляет 12 (212 = 4096). Именно последнее значение и следует указать явным образом при создании пула из винчестеров AF или большинства SSD-накопителей.

Создание файловых систем

Пулы хранения представляют собой вместилища для наборов данных, для манипуляции которыми предназначена вторая из главнейших команд — zfs. Самыми важными наборами данных являются файловые системы, к рассмотрению которых мы и переходим.

Для создания файловых систем предназначена субкоманда create команды

zfs, которая требует единственного аргумента — имени создаваемой ФС и обычно не нуждается ни в каких опциях:

```
# zfs create pool_name/fs_name
```

Внутри пула можно создавать сколь угодно сложную иерархию файловых систем. Единственное условие — родительская файловая система для системы более глубокого уровня вложенности должна быть создана заблаговременно. Ниже я покажу это на конкретном примере создания файловых систем внутри каталога /home — наиболее оправданное место для размещёния наборов данных ZFS.

Начну я немножечко издалека. При стандартной установке Mint не обойтись без создания учетной записи обычного пользователя, и, следовательно, в каталоге /home будет присутствовать по крайней мере один подкаталог — /home/username.

Смонтировать же файловую систему ZFS в непустой каталог невозможно, и, значит, мы не можем сразу прибегнуть к опции -m для определения «постоянной прописки» создаваемого пула.

Поэтому для начала делаем для пула «прописку» во временной точке — пусть это будет традиционный /tank:

```
# zpool create -o ashift=12 tank ata-SanDisk_SDSSDX120GG25_120823400863-part3 ata-SanDisk_SDSSDX120GG25_120823402786-part3
```

Теперь создаём файловую систему для будущего домашнего каталога:

```
# zfs create tank/home
```

А внутри же неё — необходимые дочерние ветви, как то:

```
# zfs create tank/home/alv
```

которая потом заменит мой домашний каталог — в нём я не держу ничего, кроме конфигурационных файлов;

```
# zfs create tank/home/proj
```

это файловая система для моих текущих проектов, и так далее.

Как и было обещано разработчиками ZFS, процедура ничуть не сложнее, чем создание обычных каталогов. Благодаря этому файловые системы можно легко создавать по мере надобности, для решения какой-либо частной задачи. И столь же легко уничтожать их, когда задача эта выполнена. Что делается таким образом:

```
# zfs destroy pool_name/fs_name
```

Использовать субкоманду destroy следует аккуратно: никакого запроса на подтверждение при этом не будет. Правда, и уничтожить файловую систему, занятую в каком-либо текущем процессе, можно только с указанием опции -f, а файловую систему, содержащую системы дочерние, не получится убить и

таким образом.

Ни в какой специальной операции монтирования новообразованные файловые системы не нуждаются — оно происходит автоматически в момент их создания, о чём свидетельствует следующая команда: $ mount | grep tank tank/home on /tank/home type zfs (rw,atime,xattr) tank/home/alv on /tank/home/alv type zfs (rw,atime,xattr) tank/home/proj on /tank/home/proj type zfs (rw,atime,xattr) ...

Для обеспечения монтирования файловых систем ZFS при рестарте машины не требуется и никаких записей в файле /etc/fstab: это также происходит само собой, совершенно нечувствительно для пользователя. Правда, если для файловой системы ZFS определить свойство mountpoint=legacy, то с ней можно управляться и традиционным способом.

Как и для обычного каталога, объём каждой файловой системы ничем не лимитирован, и в момент создания для любой из них потенциально доступно всё пространство пула, которое равномерно уменьшается по мере разрастания файловых систем. На данный момент в моей системе это выглядит так.

Казалось бы, для тех же целей можно ограничиться обычными каталогами. Однако в наборах данных ZFS мы имеем дело с полноценными файловыми системами, для которых могут быть установлены индивидуальные свойства, аналогичные опциям монтирования файловых систем традиционных. Чем мы сейчас и займёмся.

Файловые системы: устанавливаем свойства

При создании файловая система ZFS получает по умолчанию определённый набор свойств, во многом сходный с атрибутами традиционных файловых систем, определяемыми опциями их монтирования. Полный их список можно получить командой

```
# zfs get all fs_name
```

Свойств этих очень много, однако далеко не все они представляют для нас интерес. Важно только помнить, что любое из свойств каждой файловой системы можно поменять с помощью субкоманды set и её параметра вида свойство=значение. Причём изменение свойств для материнской системы рекурсивно распространяется на все дочерние. Однако для любой последней свойства можно изменить в индивидуальном порядке. Что я сейчас и проиллюстрирую на примерах.

Так, абсолютно лишним представляется свойство atime, то есть обновление времени последнего доступа к файлам. Оно, во-первых, снижает быстродействие, с другой — способствует износу SSD-накопителей (правда, нынче и то, и другое чисто символичны). Так что отключаем это свойство для всех файловых систем:

```
# zfs set atime=off tank/home
```

Аналогичным образом расправляемся и со свойством xattr:

```
# zfs set xattr=off tank/home
```

А вот дальше можно заняться и индивидуализацией. Как я уже говорил, в момент создания файловые системы ZFS «безразмерны». Если это не подходит, для них можно установить квоты. Однако я этого делать не буду — в моём случае это приводит к потере половины смысла ZFS. А вот зарезервировать место для критически важных каталогов, дабы его не отъела, скажем, мультимедиа, известная своей прожорливостью, будет не лишним. И потому я для файловых систем tank/home/proj и tank/home/alv устанавливаю свойство reservation. Для файловой системы проектов оно будет максимальным:

```
# zfs set reservation=10G tank/home/proj
```

Для остальных ограничусь более скромным гигабайтом резерва.

Далее, поскольку данные в файловой системе tank/home/proj для меня действительно важны, и шутить с ними я склонен даже гораздо меньше, чем с дамами, предпринимаю дополнительные меры по их сохранности путём удвоения числа копий (по умолчанию оно равно 1):

```
# zfs set copies=2 tank/home/proj
```

А для данных не столь важных — тех, что часто проще скачать заново, нежели отыскать на локальной машине, можно выполнить и обратную операцию — отказаться от подсчёта контрольных сумм:

```
# zfs set checksum=off tank/home/media
```

Для файловых систем, содержащих хорошо сжимаемые данные (например, для моего домашнего каталога, где лежат одни dot-файлы), можно включить компрессию:

```
# zfs set compression=on tank/home/alv
```

Я этого не делал: экономия места получается грошовая, а нагрузка на процессор и расход памяти, как говорят, очень приличные. Однако это свойство целесообразно включать в системах с огромными логами, если выделить под них файловую систему в пуле ZFS.

При желании для некоторых файловых систем (например, того же домашнего каталога) можно отключить такие свойства, как exec, setuid, devices — легко догадаться, что результат будет аналогичен указанию опций монтирования noexec, nosuid, nodev для традиционных файловых файловых систем. И, разумеется, для файловых систем, изменение которых нежелательно, можно придать свойство readonly.

Все необходимые свойства файловых систем желательно установить до их наполнения контентом, ибо многие из них (например, компрессия) обратной силы не имеют.

Перемонтирование

После создания файловых систем и задания всех необходимых их свойств наступает психологический момент для перемонтирования их по месту «постоянной прописки» — то есть в каталог /home. Что потребует некоторых подготовительных действий.

Поскольку предполагается, что все новообразованные файловые системы должны быть полностью доступны обычному пользователю (то есть мне, любимому), перво-наперво следует изменить атрибуты из принадлежности — ведь создавались они от имени администратора и принадлежат юзеру по имени root. Для чего даю команду:
chown -R alv:users /tank/home/*

Теперь нужно скопировать конфиги из каталога /home/alv в /tank/home/alv:

```
# cp -Rp /home/alv/.* /tank/home/alv/
```

не забыв про опцию -p для сохранения атрибутов.

Все предыдущие операции можно было выполнять, получив права администратора с помощью команды sudo. Причём где угодно — в текстовом виртуальном терминале или в терминальном окне Иксового сеанса. Теперь же потребуется переавторизоваться в «голой» консоли.

Монтирование файловых систем ZFS в каталог с любым содержимым невозможно, так что требуется очистить каталог /home от следов прежней жизнедеятельности пользователя таким образом:

```
# rm -Rf /home/alv
```

При хоть одном активном пользовательском процессе в ответ на это последует сообщение об ошибке. Так что, возможно, перед этим придётся убить все реликтовые процессы, запущенные в Иксах от имени пользователя. Сначала выявляем их командой

```
# ps aux | grep alv
```

обращая внимание на идентификаторы процессов (PID). А затем последовательно мочим их в сортире:

```
# kill -9 ####
```

Альтернативный способ — разрузка системы в recovery mode с выбором пункта меню root, что в Mint эквивалентно однопользовательскому режиму. В этом случае никаких пользовательских процессов не будет по определению, и перенос файлов из /home/username в /tank/home/username можно выполнить напрямую.

Выполнив все указанные действия, определяем для набора данных tank/home свойство mountpoint, что и осуществит перемонтирование:

```
# zfs set mountpoint=/home tank/home
```

Теперь остаётся только с помощью команды ls убедиться, что в /home появились новые подкаталоги с нужными атрибутами:

```
drwxr-xr-x 26 alv users 48 Sep 23 14:27 alv/
drwxr-xr-x 18 alv users 18 Sep 22 02:28 proj/
...
```

А команда

```
# mount | grep /home
```

покажет нам новые точки монтирования файловых систем:

```
tank/home on /home type zfs (rw,noatime,noxattr)
tank/home/alv on /home/alv type zfs (rw,noatime,noxattr)
tank/home/proj on /home/proj type zfs (rw,noatime,noxattr)
...
```

На этом дело с подготовкой файловых систем ZFS к практической работе можно считать законченным: при перезагрузке машины все они будут благополучно смонтированы в автоматическом режиме.

Подключение пула ZFS, созданного в другой системе

Здесь речь пойдёт о том, как подключить к некоей Linux-системе (конкретно, Ubuntu) пул ZFS, ранее созданный в другой системе — теоретически это могут быть Solaris, FreeBSD или более иной дистрибутив Linux, для которого предусмотрена поддержка ZFS on Linux. Но практически я опробовал только последний вариант, о чём и расскажу.

Перво-наперво нужно перезагрузиться в ту систему, в которой создавался пул (в моём случае это была openSUSE), и экспортировать его командой:

```
# zpool export data
```

где data — имя пула с точкой монтирования /home/data.

Следующий шаг — вернуться в Ubuntu и создать в ней аналогичную точку монтирования для пула ZFS — в моём случае таким образом:

```
$ sudo mkdir /home/data
```

Дать ей атрибуты принадлежности обычному пользователю:

```
$ sudo chown -R alv:alv /home/data
```

И импортировать созданный в openSUSE пул ZFS:

```
$ sudo zpool import -f data
```

Не забыв об опции -f, предписывающей принудительной выполнение импорта. Без неё ответом на эту команду будет сообщение об ошибке.

Теперь в каталоге /home/data можно видеть те же самые файловые системы ZFS, которые были созданы в родительской для пула системе, вместе со всеми размещёнными в них данными. С которыми можно начинать работать.

Сказанное справедливо, если идентификаторы пользователя в обеих системах совпадают — в моём случае это именно так. Однако в случае общем это совсем не обязательно — и тогда надо озаботиться каким-либо способом обеспечения совместного доступа к ним из разных систем. Впрочем, к ZFS, о которой мы сейчас разговариваем, это не имеет ни малейшего отношения.

Индивидуализация системы

Последний очерк посвящается апофеозу знакомства с дистрибутивом Mint и его Cinnamon — сборке индивидуализированной системы на его основе. Как сказал то ли один из премьер-министров Великобритании, то ли некий президент Франции (кому только это изречение ни приписывали?):

Тот, кто в (информационной) юности не мечтал собрать свой дистрибутив Linux'а, не имеет сердца. Тот, кто продолжает мечтать об этом в (информационной) зрелости — не имеет разума.

Однако применительно к Linux Mint вполне возможно сочетать сердечность и разумность даже в преклонные годы, ибо собрать свою индивидуализированную систему на базе этого дистрибутива и просто, и полезно.

Свой Mint: введение

Во время первого знакомства с с Mint у меня сложилось впечатление, что в его инсталляции настолько мало лишних программ, что не стоило и заморачиваться с их удалением. Однако при дальнейшем рассмотрении оказалось, что лишних (для меня) приложений вполне достаточно — например, вся мультимедиа. С другой стороны, ряд привычных для меня программ (например, те же мультимединые) приходилось доустанавливать. И появилась мысль изготовить свой установочный диск этого дистрибутива. На котором не было бы ничего лишнего (повторяю, для меня). И, напротив, были бы все приложения, которые мне так или иначе пришлось доустанавливать.

В сети можно найти упоминания о нескольких инструментах для изготовления собственного дистрибутива на базе Ubuntu и её производных (а Mint, как известно, принадлежит к их числу):

- oem-config-remaster — утилита командной строки, позволяющая сделать снапшот установленной системы;

- remastersys — утилита для резервного копирования установленной системы и создания на её базе Live-носителя; именно таким образом собирается, например, дистрибутив Matuntu — отечественный вариант Ubuntu с MATE в качестве рабочей среды;

- Ubuntu Builder-- программа с графическим интерфейсом, позволяющая скомпоновать свой дистрибутив попакетно.

Однако первые две показались мне сложноватыми и избыточными для моих целей, а последняя, похоже, прекратила своё развитие — версии её для Ubuntu Trusty, на которой основываются текущие релизы Mint, в PPA-репозитории не обнаружилось.

И в итоге я остановился на программе Ubuntu Customization Kit (далее — UCK): гугление показало, что это примерно то, что мне нужно, ибо специально предназначается для коррекции состава пакетов в первую очередь.

UCK: обзор

Программа UCK в виде одноимённого пакета имеется в официальном репозитории, и потому может быть установлена любым из стандартных

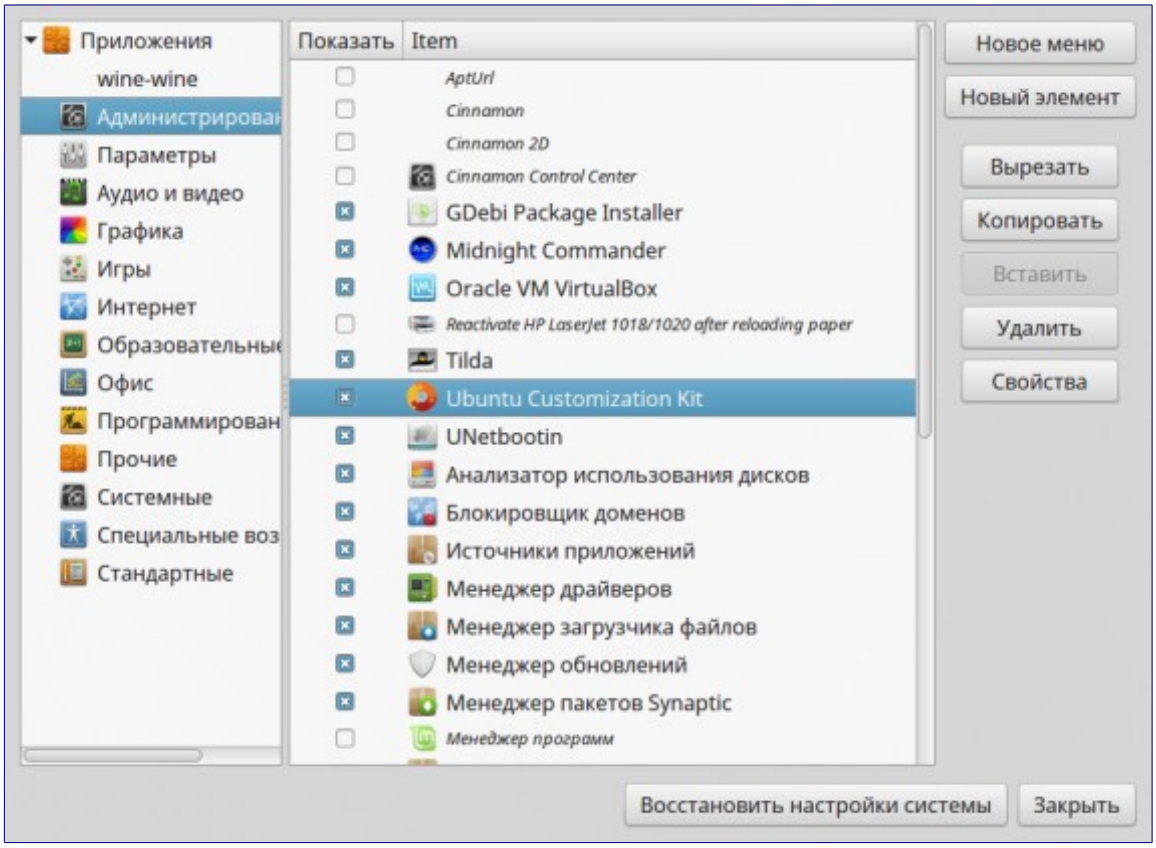
способов. После этого в секции Администрирование главного меню Cinnamon появляется пункт, который, как ни странно, называется Ubuntu Customization Kit, через который эту программу и можно запустить. Но прежде чем смотреть, что она после этого делает, попробуем представить, что у неё внутре.

Всякий, кому приходилось заниматься модификацией существующего установочного образа какого-либо дистрибутива, представляет себе основные этапы этого процесса:

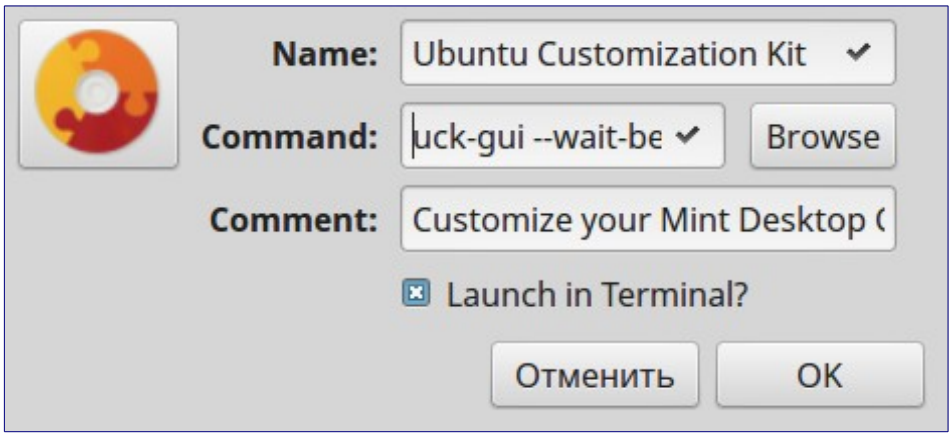
- монтирование образа как loop-устройства;
- развёртывание её файловой системы — нынче все дистрибутивы используют какой-либо механизм компрессии, в частности в убунтоидах это SquashFS;
- монтирование в loop-систему как связанных (bind) таких служебных, но абсолютно необходимых каталогов материнской системы, как /dev, /sys и, на всякий случай, /proc;
- выполнение операции chroot в loop-каталог, становящийся таким образом корневым;
- выполнение в chroot-окружении необходимых действий по удалению ненужных пакетов и установке необходимых;
- выход из chroot-окружения и обратная запаковка loop-каталога;
- размонтирование loop-устройства и создание из него загрузочного iso-образа с помощью isolinux.

Интуитивно понятно, что UCK не делает ничего иного, кроме перечисленного — он просто автоматизирует описанный процесс, делая его этапы почти незаметными для применителя.

Основные исполняемые файлы пакета uck (а их 16 штук) собраны, понятное дело, в каталоге /usr/bin и имеют префикс uck-*. Все они являются самыми обычными шелл-скриптами, причём по их именам легко догадаться о назначении каждого. «Головным», то есть запускающим весь процесс скриптом является /usr/bin/uck-gui — именно он вызывается через пункт меню Администрирование -> Ubuntu Customization Kit:



А через редактор меню можно посмотреть на его свойства:



По скриншоту можно догадаться о присутствии в строке запуска опции --wait-before-exit, отвечающей за ожидание нажатия Enter перед выходом из программы после успешного завершения её работы.

Кстати говоря, при неуспешном завершении, то есть возникновении ошибки по любой причине (например, нехватке места на целевом устройстве), ничего не происходит, кроме остановки работы. Никаких возвратов назад не предусмотрено, остаётся только закрывать главное терминальное окно программы (о котором далее) и начинать всё сначала. Так что, учитывая длительность распаковки SquashFS и необходимость удаления образовавшихся в процессе файлов (а их — многие десятки тысяч, что на файловой системе, например, XFS затягивается очень надолго), лучше действовать аккуратно и по возможности не ошибаться.

Команда uck-gui запускается от лица обычного пользователя — пароль для доступа к административным привилегиям запрашивается только тогда, когда они на самом деле потребуются. Кроме указанной --wait-before-exit, она имеет ещё как минимум две опции. Первая, -m, обеспечивающая кеширование

модифицированных частей образа, работает, как сказано в man (1) uck-gui, не всегда, и потому в стандартной ситуации не используется.

Вторая опция также штатно не задействована, но она может оказаться важной для применителя. Это опция remaster-dir, определяющая рабочий каталог для UCK, отличный от умолчального ~/tmp. Через редактор меню Cinnamon я переопределил этот каталог как ~/data/my-mint, поэтому итоговая команда для запуска UCK через меню приобрела такой вид:

```
uck-gui --wait-before-exit /home/data/my-mint
```

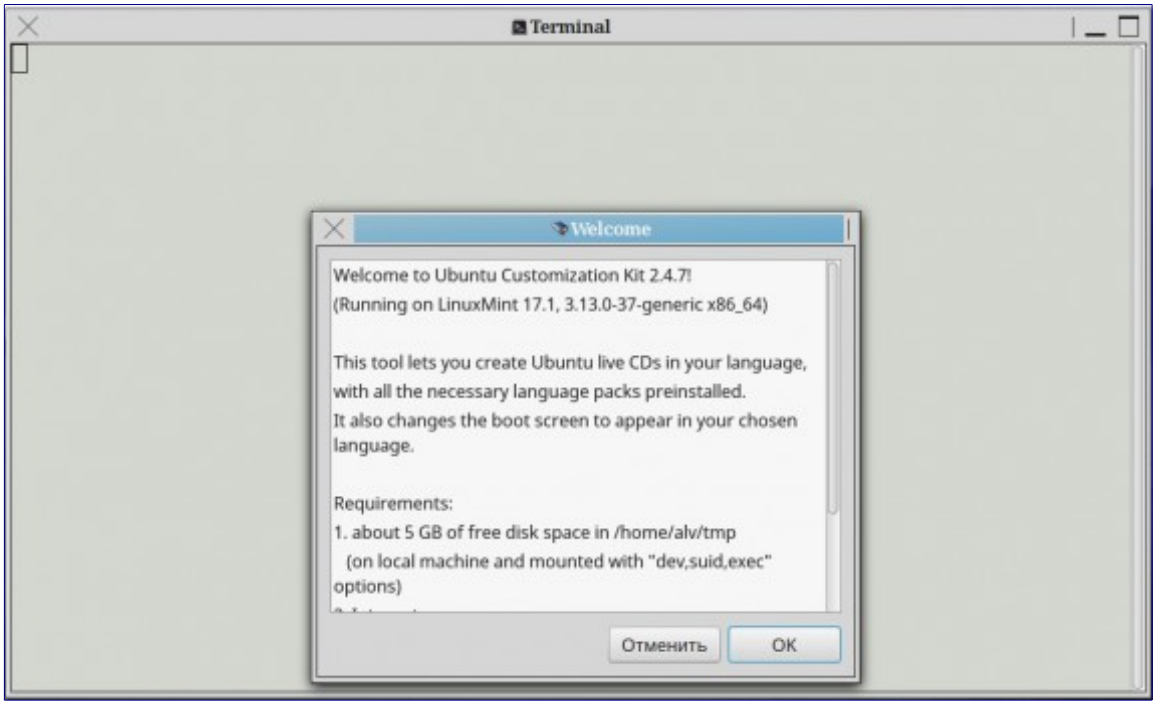
Кроме запуска процесса, сценарий uck-gui отвечает, в том числе, и за выбор типа десктопа — unity, gnome, kde, или others. Однако попытки вносить здесь какие-либо изменения (например, пополнение списка доступных десктопов) никакого результата за собой не повлекут. То есть добавленные десктопы появятся в меню их выбора, но ничего не изменится.

Потому что на самом деле кроме исполняемых скриптов в каталоге /usr/bin, основным компонентом UCK является также каталог /usr/lib/uck/. А в нём, кроме всего прочего — файл /usr/lib/uck/customization-profiles/localized_cd/customize, представляющий собой исполняемый шелл-сценарий, который отвечает в том числе и за вызов терминальной программы. Запомним его — в некоторых случаях он подлежит ручному редактированию.

Впрочем, всё сказанное проще продемонстрировать на примере конкретной сборки своего варианта дистрибутива, чем мы сейчас и займёмся.

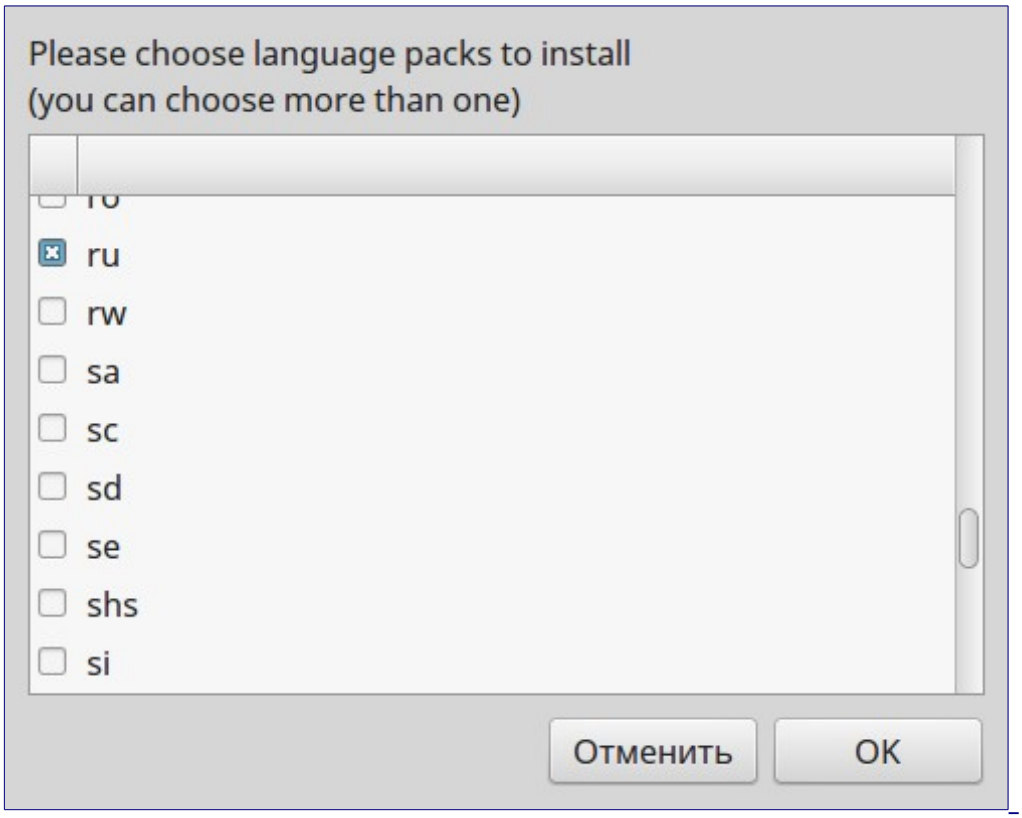
UCK: процесс

Сразу после запуска UCK перед нами появляется пустое терминальное окно и на фоне его — приглашение программы:

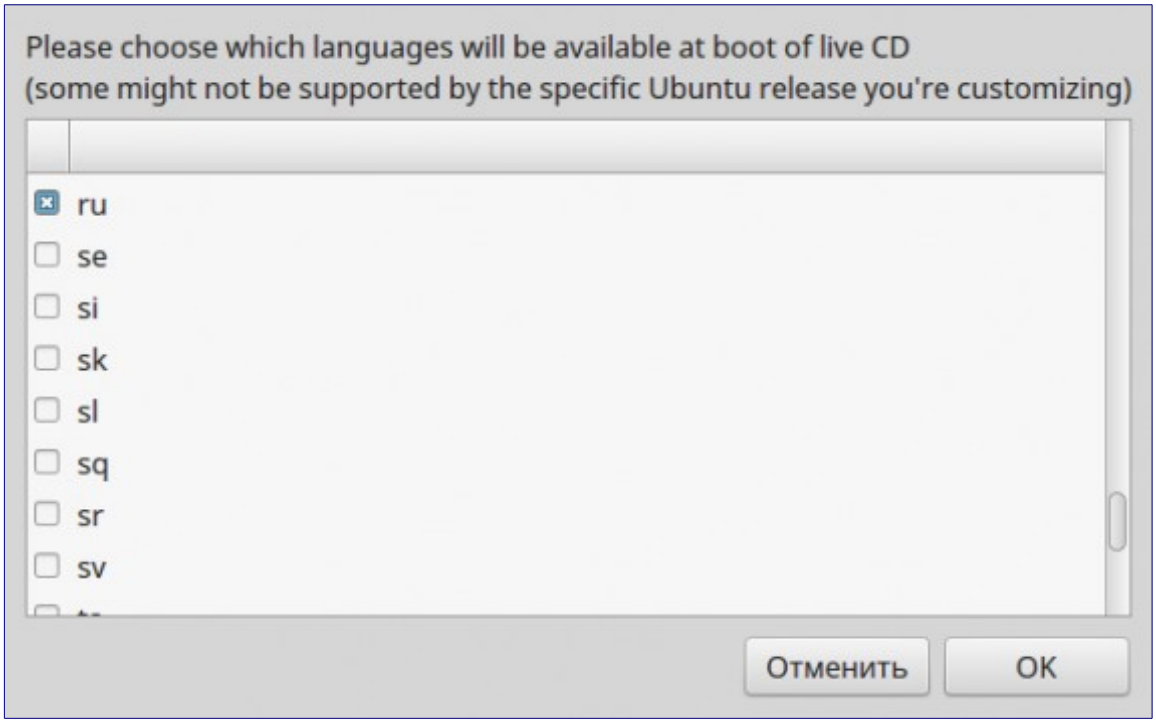


В нём указаны системные требования, не являющие собой ничего сверхъестественного: 5 ГБ свободного пространства в каталоге ~/tmp (или его аналоге, определённом выше) и доступ в Интернет. Так что можно смело нажимать OK и приступать к кастомизации, которая начинается

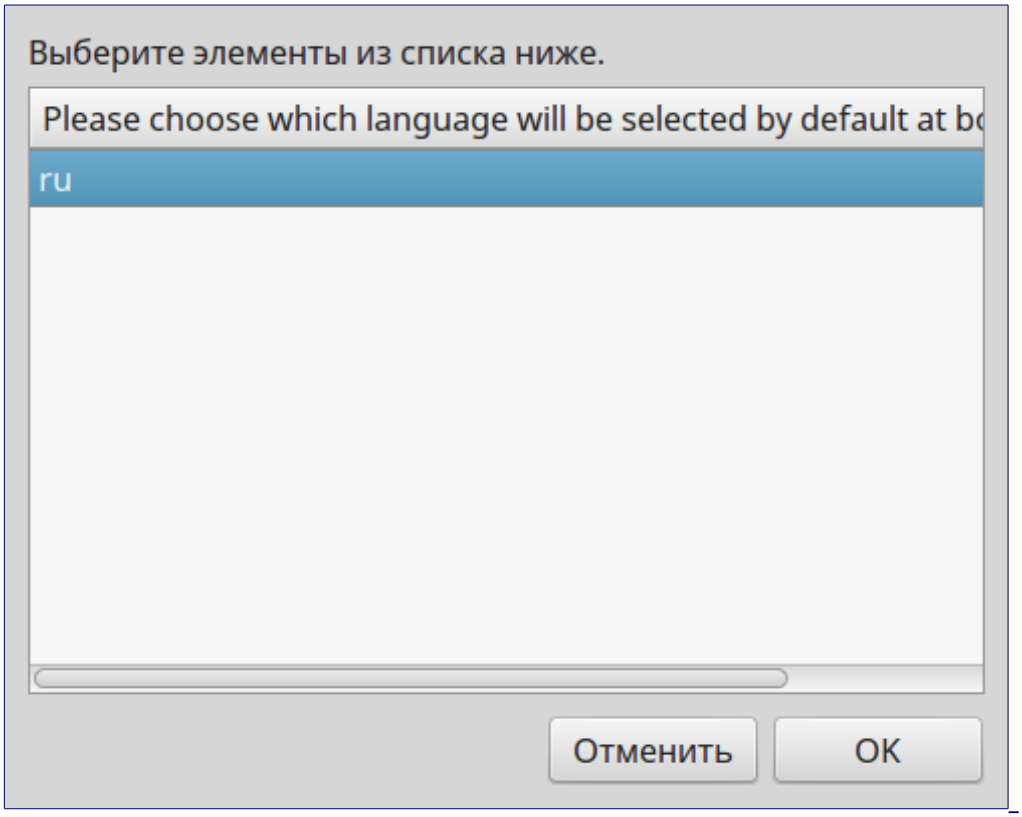
с выбора локали — сначала для инсталлятора:



Затем — для Live-носителя:



И наконец — для и инсталлированной системы:



Список всех доступных содержится в файле /usr/lib/uck/langlist. Причём, при желании иметь русифицированную систему, можно ограничиться отметкой боксика ru во всех трёх случаях. Впрочем, в Live-среде локаль всё равно останется en_US (о чём программа честно предупреждает). Однако язык инсталлтора будет русским. А в установленной системе локаль по умолчанию определится как надо:

```
~ $ locale

LANG=ru_RU.UTF-8

...
```

И к ней автоматически добавятся такие:

```
~ $ locale -a

C

C.UTF-8

en_AG

en_AG.utf8

en_AU.utf8

en_BW.utf8

en_CA.utf8

en_DK.utf8
```

en_GB.utf8

en_HK.utf8

en_IE.utf8

en_IN

en_IN.utf8

en_NG

en_NG.utf8

en_NZ.utf8

en_PH.utf8

en_SG.utf8

en_US.utf8

en_ZA.utf8

en_ZM

en_ZM.utf8

en_ZW.utf8

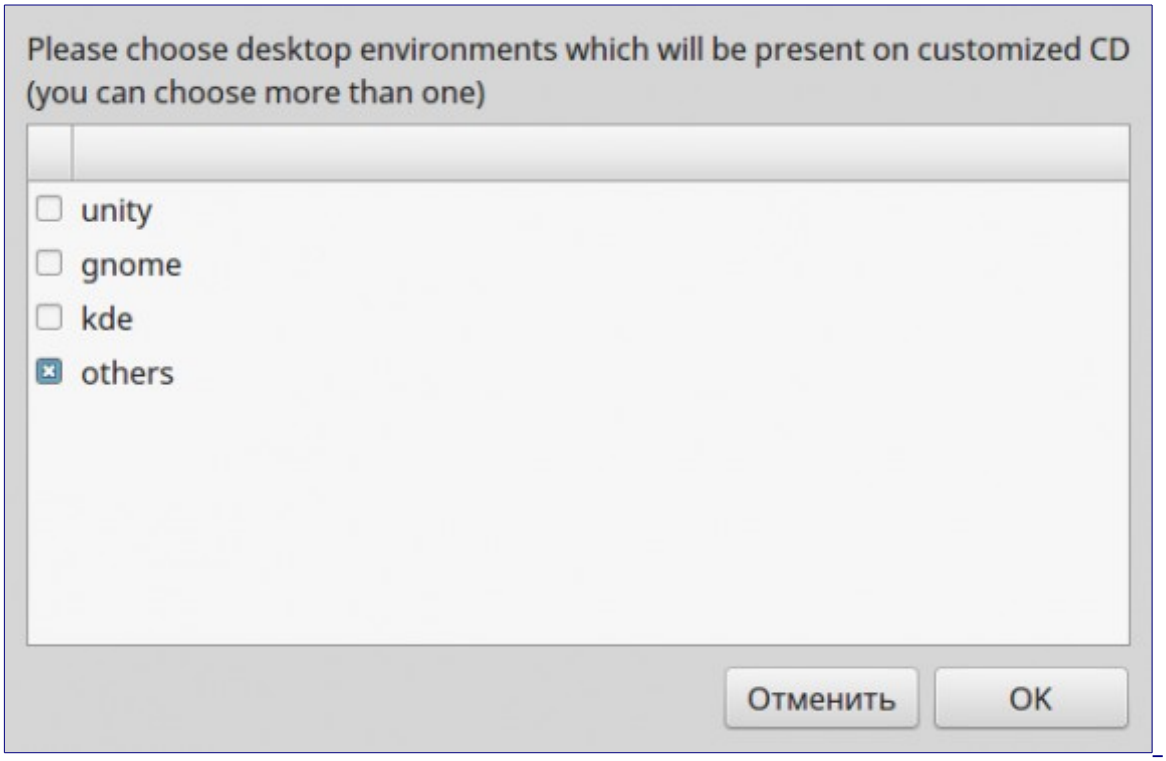
POSIX

ru_RU.utf8

ru_UA.utf8

Правда в Mint, в отличие от Ubuntu, ни одна из многочисленных английских локалей не будет всплывать в самый неподходящий момент.

Следующий шаг кастомизации — выбор рабочей среды:



На самом деле он определяет не среду, а терминальную программу, в которой будет жить chroot-окружение. При выборе первых трёх вариантов её будет соответствующий штатный терминал (Konsole, GNOME Terminal или Xfce Terminal), в случае варианта четвёртого будут просто перебраны они же плюс LXTerminal из LXDE. Соответственно, запустится тот, что найдётся первым, так что выбор пункта other подойдёт в подавляющем большинстве случаев, вне зависимости от десктопа, используемого на потрошимом Live-носителе.

На самый крайняк предусмотрен запуск XTerm, который, казалось бы, имеется в любом дистрибутиве. Но вот в Mint'e ни в одной официальной редакции его как раз нет. Поэтому при сборке системы с десктопом MATE потребуется редактирование того самого файла /usr/lib/uck/customization-profiles/localized_cd/customize, который я ранее предлагал запомнить. То есть в его секцию function run_console() следует вписать такие строки:

```
        if [ «$CONSOLE_APP» = «» ]; then

                CONSOLE_APP=`which mate-terminal`

                CONSOLE_APP_OPTIONS=(-t «UCK customization console» -e /bin/bash)
```

Я на всякий случай дописал в него также определения для терминальных программ Sakura и Terminator, которые временами использую.

```
        if [ «$CONSOLE_APP» = «» ]; then

                CONSOLE_APP=`which terminator`

                CONSOLE_APP_OPTIONS=(-t «UCK customization console» -e /bin/bash)

        fi

        if [ «$CONSOLE_APP» = «» ]; then
```

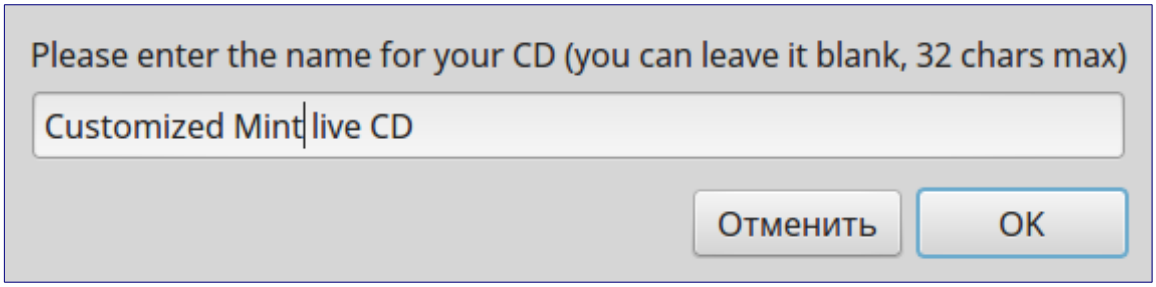
```
        CONSOLE_APP=`which sakura`

        CONSOLE_APP_OPTIONS=(-t «UCK customization console» -e /bin/bash)

    fi
```

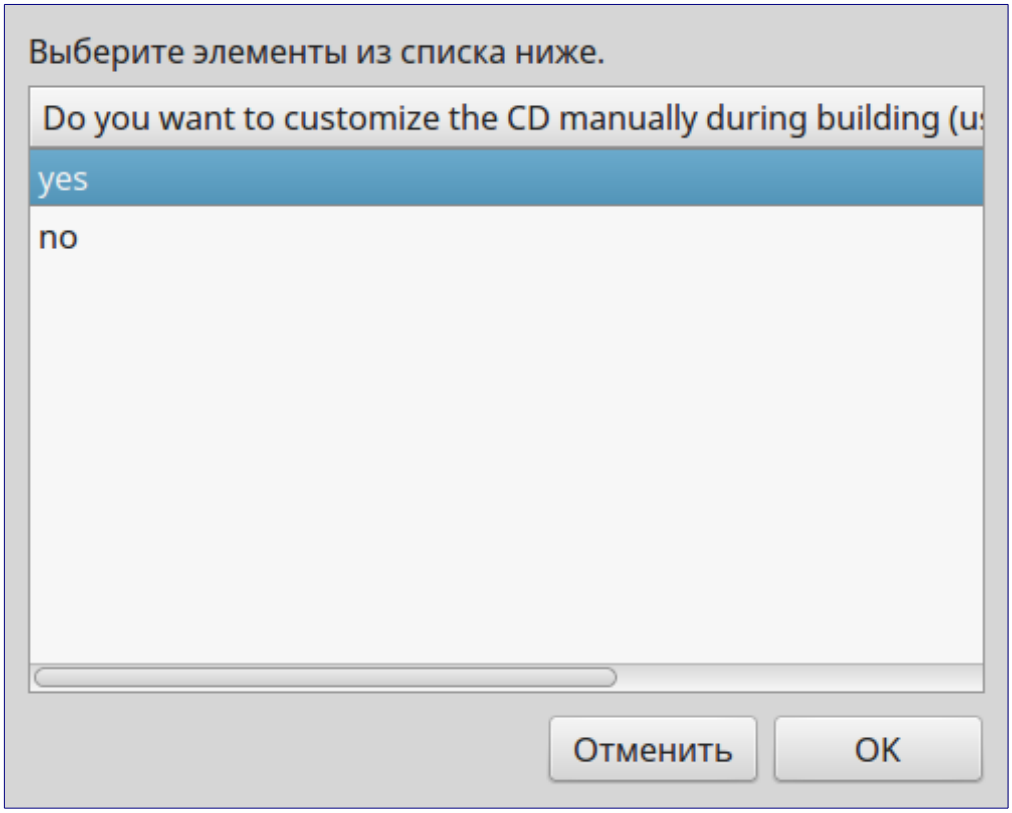
Теоретически тут можно переопределить и командную оболочку (например, /bin/zsh), но я этим заморачиваться не стал.

Теперь потребуется выбрать образ диска, который будет подвергнут потрошению — через обычное окно открытия файла. После чего будущему образу предлагается дать имя:

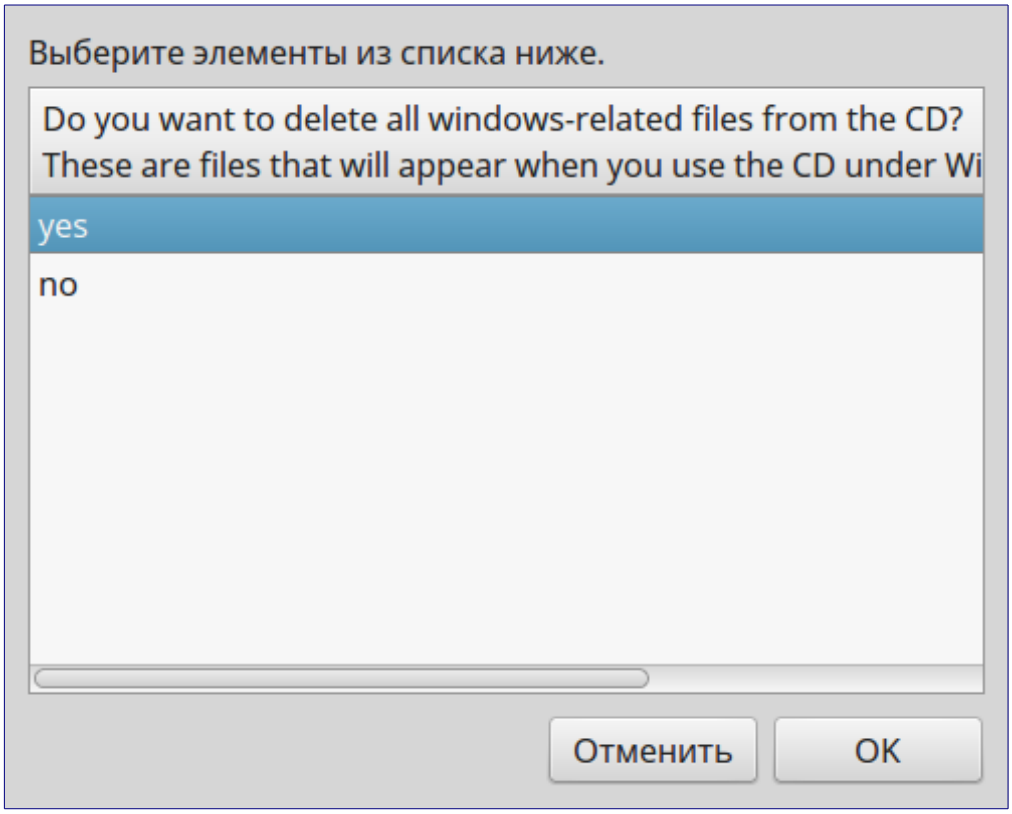


Впрочем, имя это, насколько я понял, в дальнейшем нигде не используется, так что особо напрягать свою фантазию не нужно.

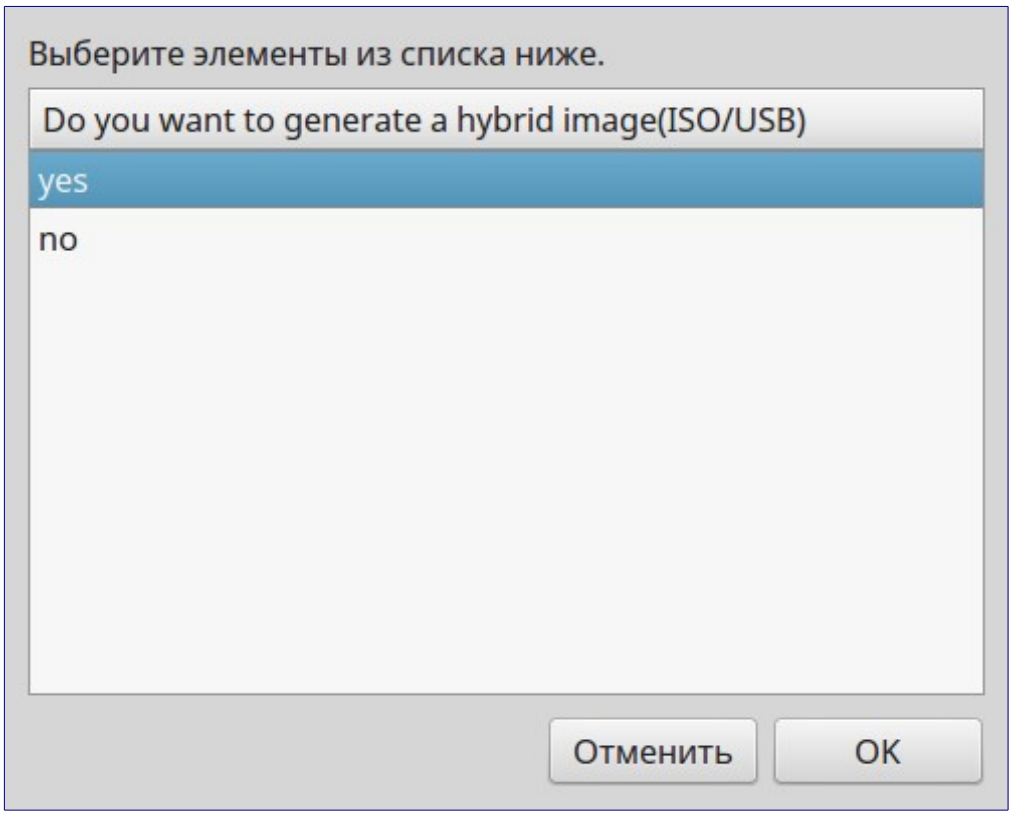
Далее следует серия вопросов — о желании кастомизировать будущий образ мануально:



Об удалении Windows-related файлов, типа autorun.inf (которых, впрочем, на установочном носителе Mint и так нет):

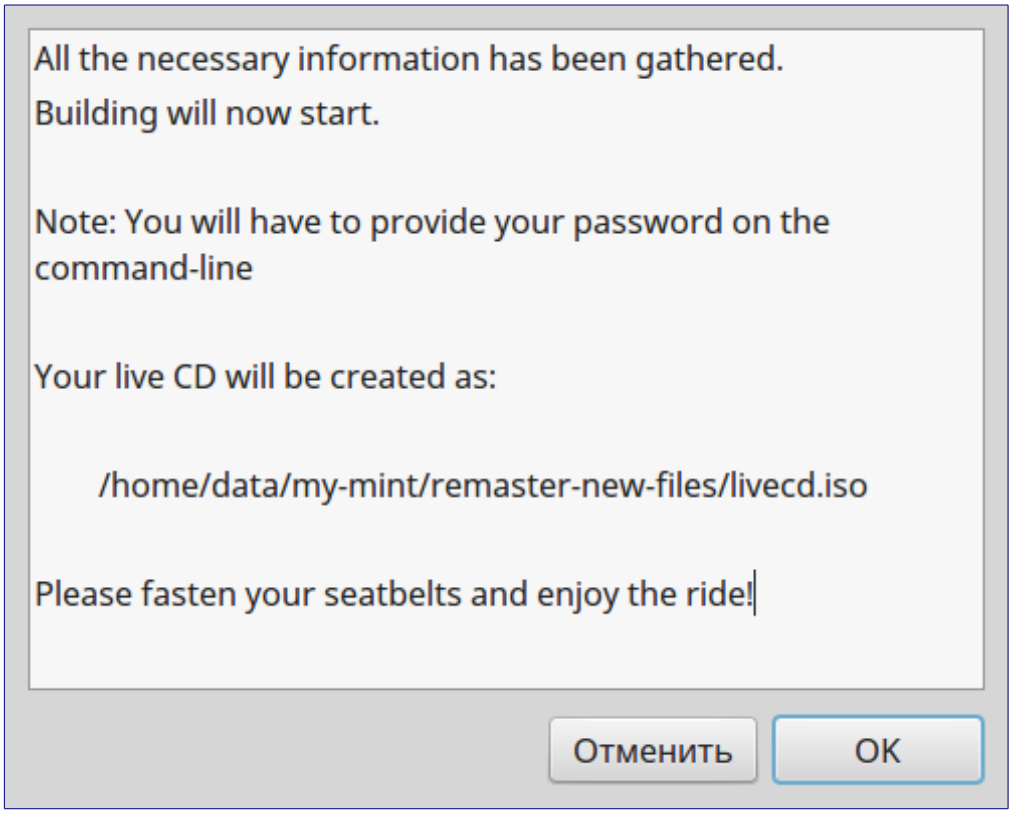


О создании гибридного образа — то есть пригодного для записи как на OD, так и на USB:

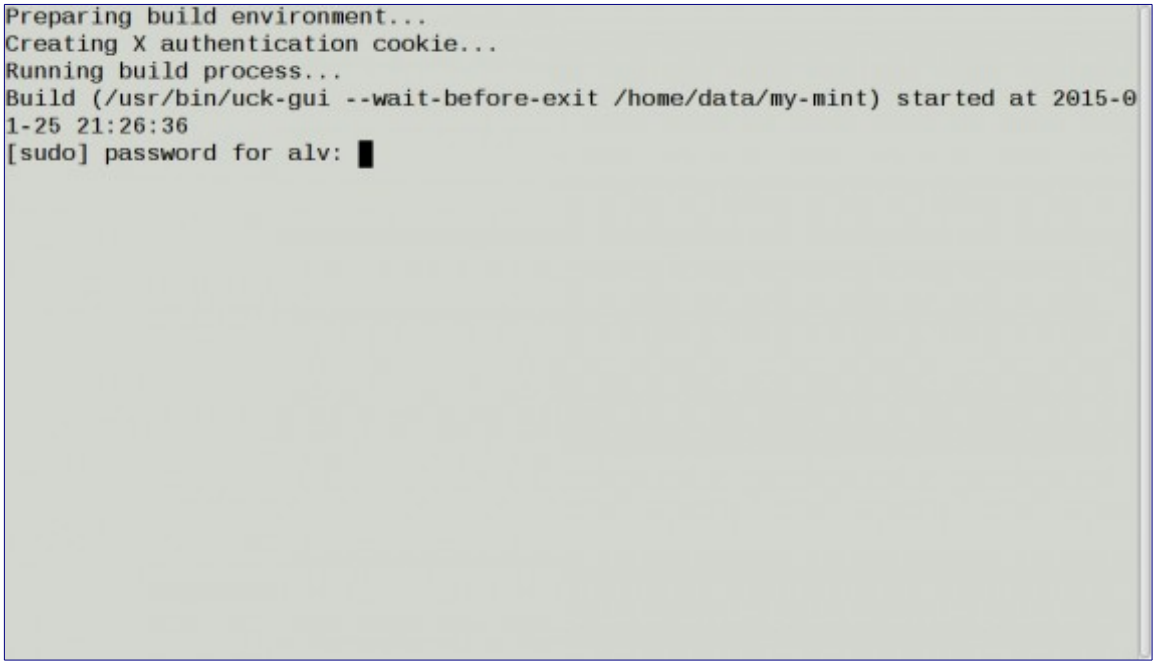


На все эти вопросы я, по понятным причинам, отвечал положительно, иначе и говорд городить бы не стоило.

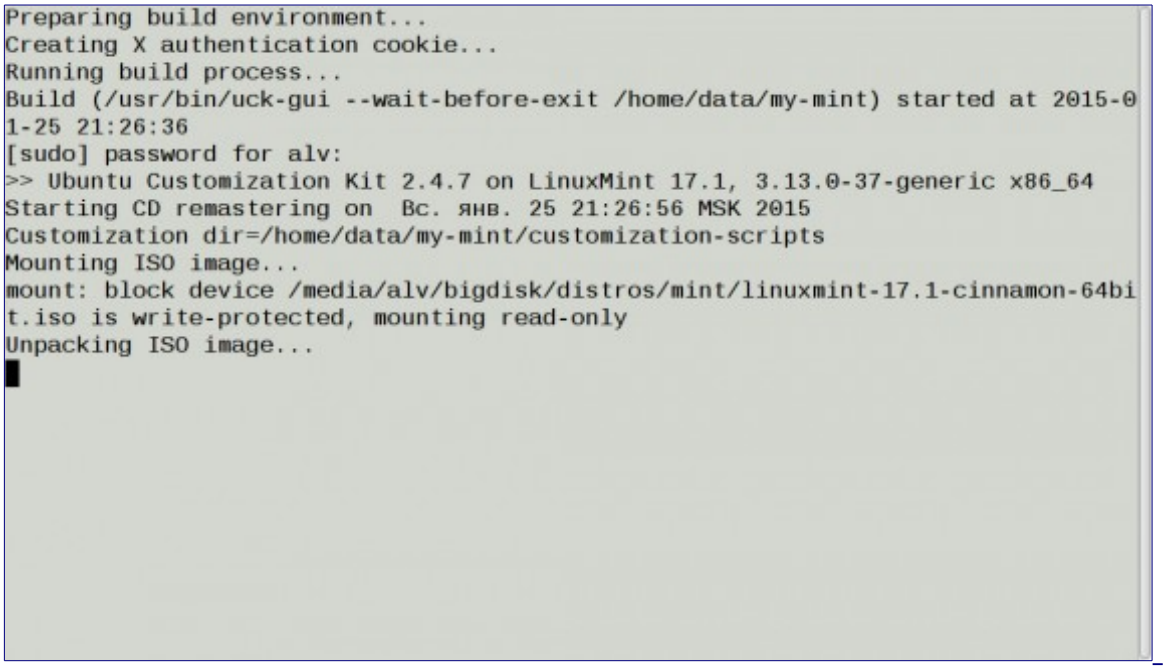
Далее сообщается, что вся необходимая информация собрана, по введении пароля можно приступать к сборке диска, который будет помещён в path2/remaster-new-files под именем livecd.iso (спрашивается, зачем придумывать ему осмысленное имя?):



А для начала процесса в исходном терминальном окне надо ввести пароль для доступа к административным привилегиям:



Вслед за этим происходит монтирование исходного образа, его разворачивание и декомпрессии SquashFS, которая занимает немало времени:



Когда же она закончится, каталог, определённый в качестве remaster-dir, будет выглядеть так:

```
ls [remaster-dir]

build.log

customization-scripts/

remaster-apt-cache/

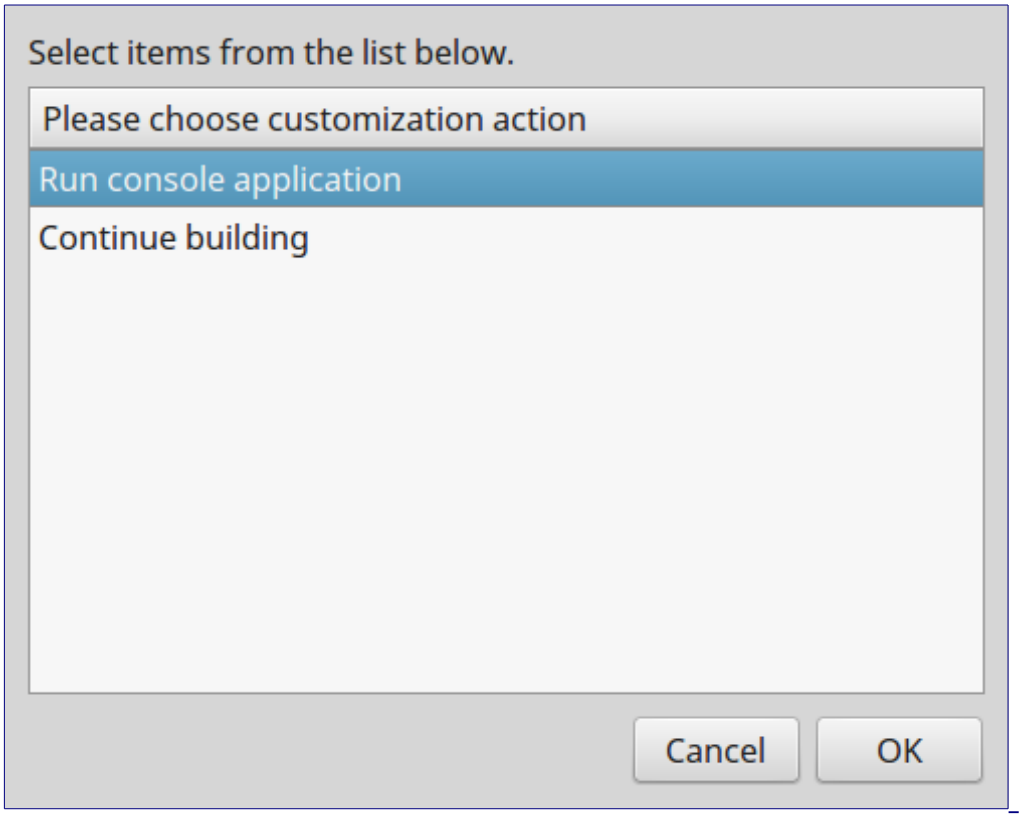
remaster-new-files/

remaster-root/

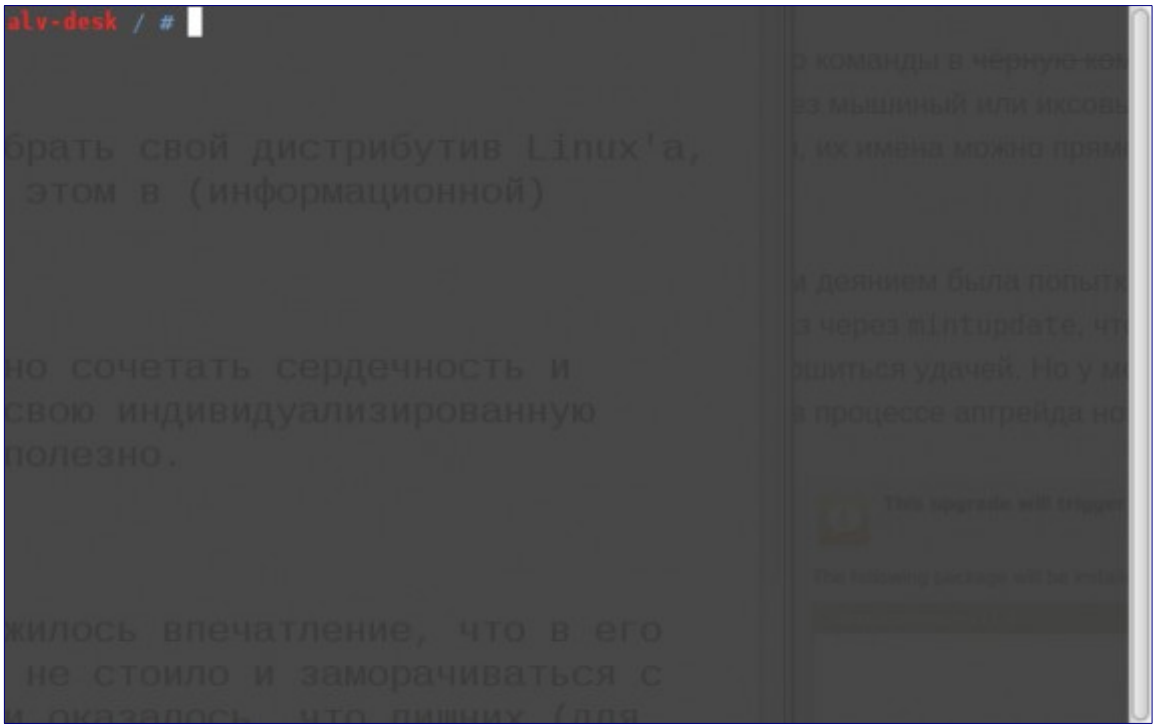
remaster-root/home
```

Очевидно, что build.log содержит протоколирование хода процесса, а в каталоге customization-scripts/ собраны скрипты кастомизации, сгенерированные посредством сценариев из /usr/lib/uck/. В каталоге remaster-apt-cache/ будет помещён локальный кеш устанавливаемых пакетов, а сами они в подкаталоге remaster-apt-cache/archives — аналоге /var/cache/apt/archives установленной системы. Таким образом, скачанные пакеты не засоряют ни корень развёрнутой из Live-образа системы (он расположен в каталоге remaster-root/), ни, тем более, каталог для сборки уже непосредственно нового образа — remaster-new-files/. В последнем после успешного завершения всего предприятия этот самый образ, под именем livecd.iso, и окажется. Ну а remaster-root/home, ясное дело, является домашним каталогом администратора (аналог /root обычной файловой иерархии).

Далее предлагается выбрать «заказное» действие — Run console application или Continue building:



Выбор первого пункта очевиден. Он влечёт за собой выполнение той самой команды chroot, о которой я говорил раньше, и запуск того самого терминала, который был неявным образом определён на стадии так называемого выбора десктопа:

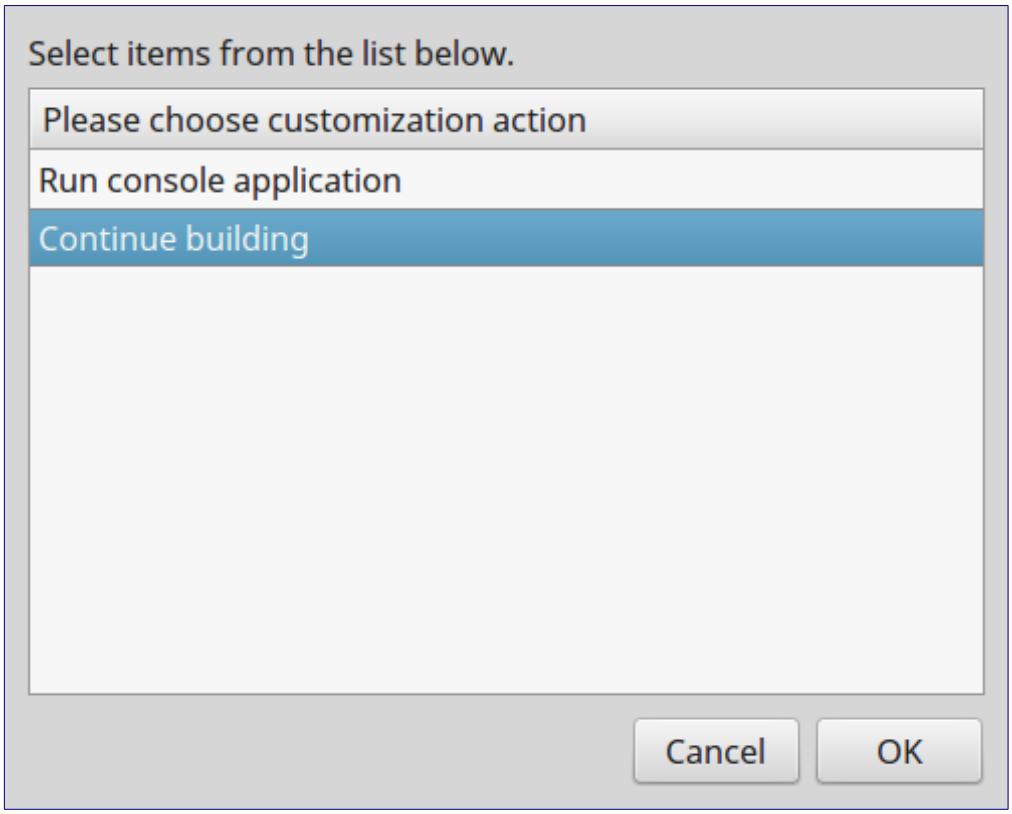


Обращаю внимание — командная оболочка в терминале — Bash, запущенная от лица администратора. То есть в дальнейшем для установки/удаления пакетов и прочих подобных мероприятий команда sudo не понадобится.

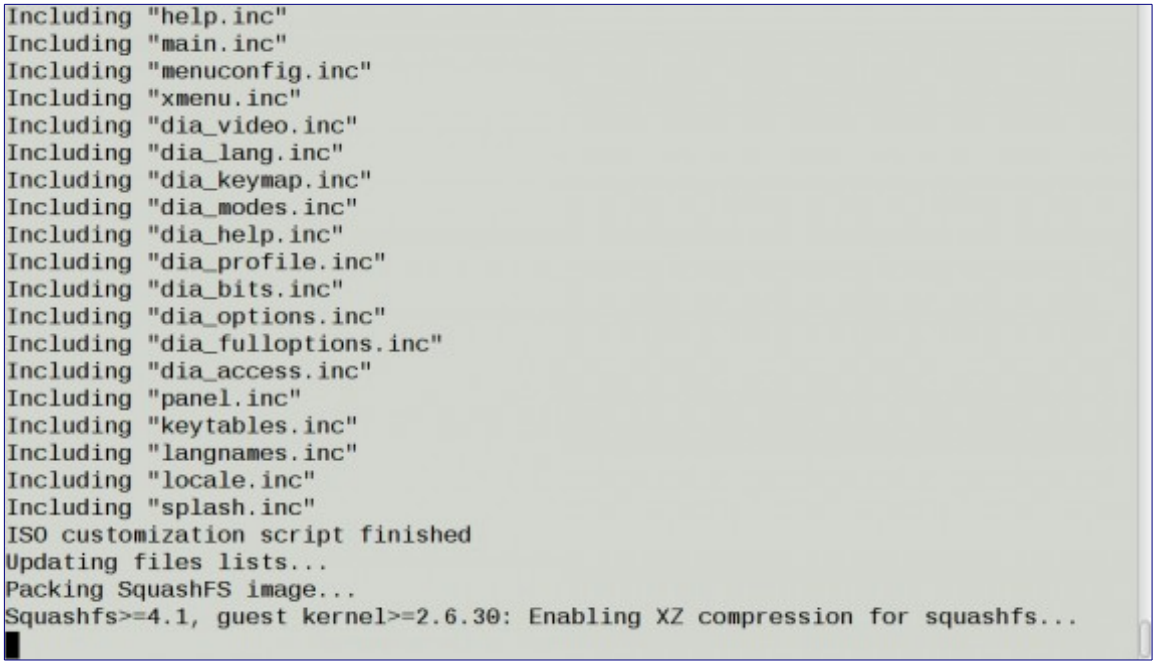
Если предыдущие стадии завершились успешно, то начинается самое важное: собственно потрошение исходного образа. Тут требуется аккуратность и последовательность, нарушение которой влечёт ошибки,

которые, как я уже говорил, крайне нежелательны. Так что в следующем миниочерке процедура потрошения будет рассмотрена подробно. А пока завершу описание основного процесса.

По завершении потрошения опять возникает панель с предложением выбрать «заказное» действие — и теперь столь же очевиден выбор второго из них:

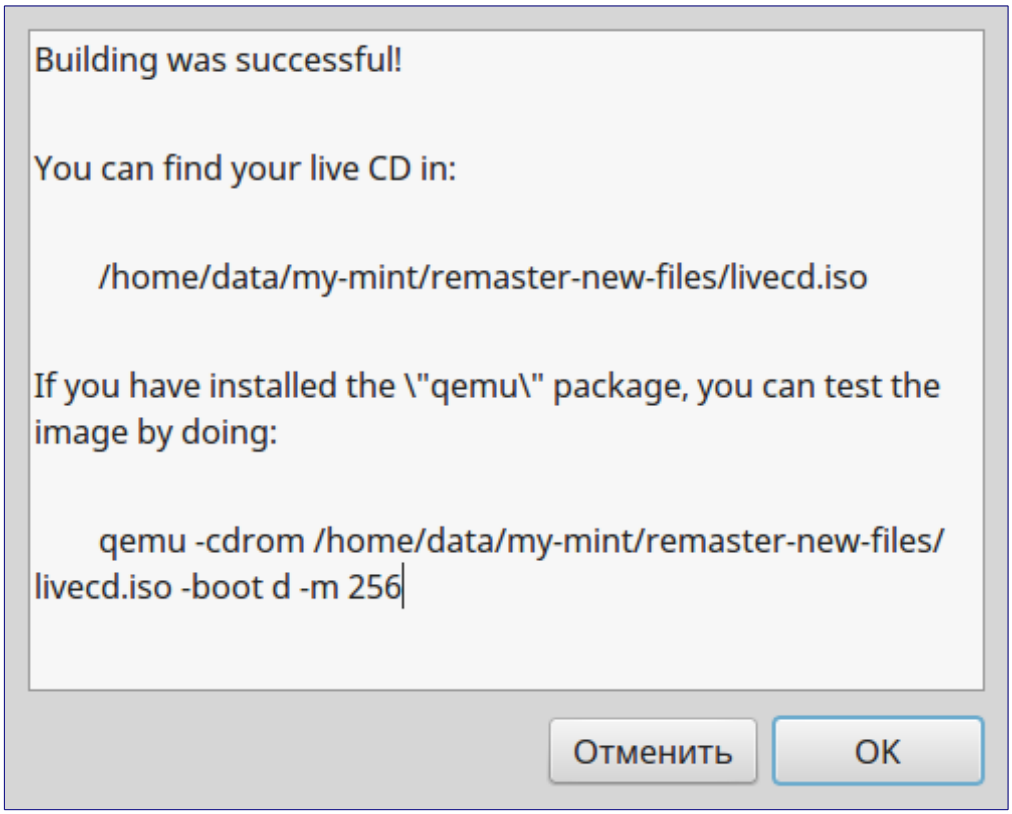


После чего начнается исполнение сценариев кастомизации, плавно переходящее в компрессию системы в виде SquashFS — это будет самым долгим делом во всём процессе:

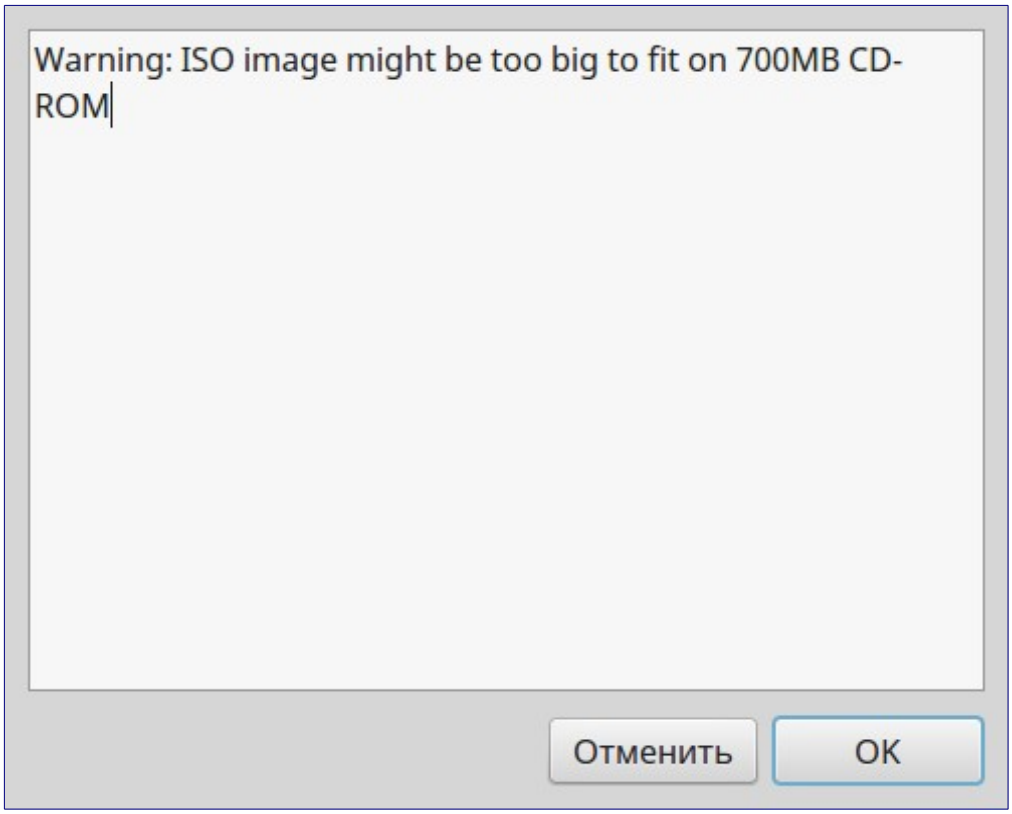


Однако всё когда-нибудь кончается — и упаковка SquashFS закончится сообщением об успехе операции и напоминанием о том, где и под каким

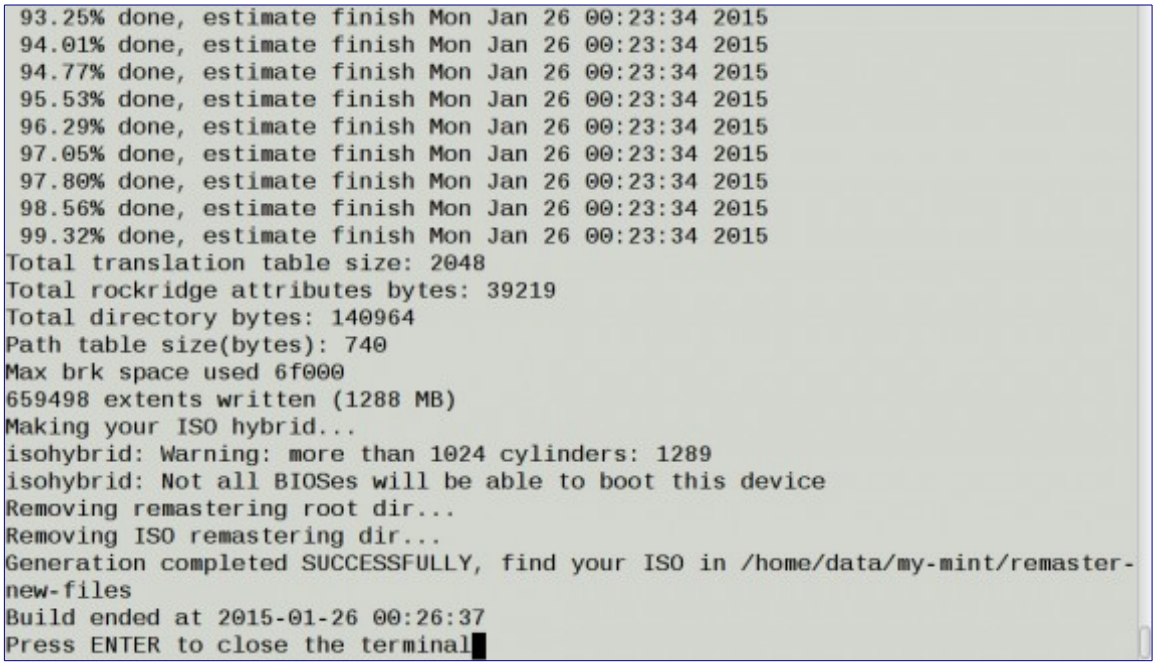
именем можно найти её результат:



Приняв это к сведению, я узнал, что собранный образ не вместится на стандартный семисотмегабайтный CD:



Что, как выяснилось, соответствовало действительности, о чём скажу в следующем миниочерке. А пока оставалось только нажать OK, что повлекло закрытие окна сборки. А в первом терминале — нажать Enter, в результате закроется и он:

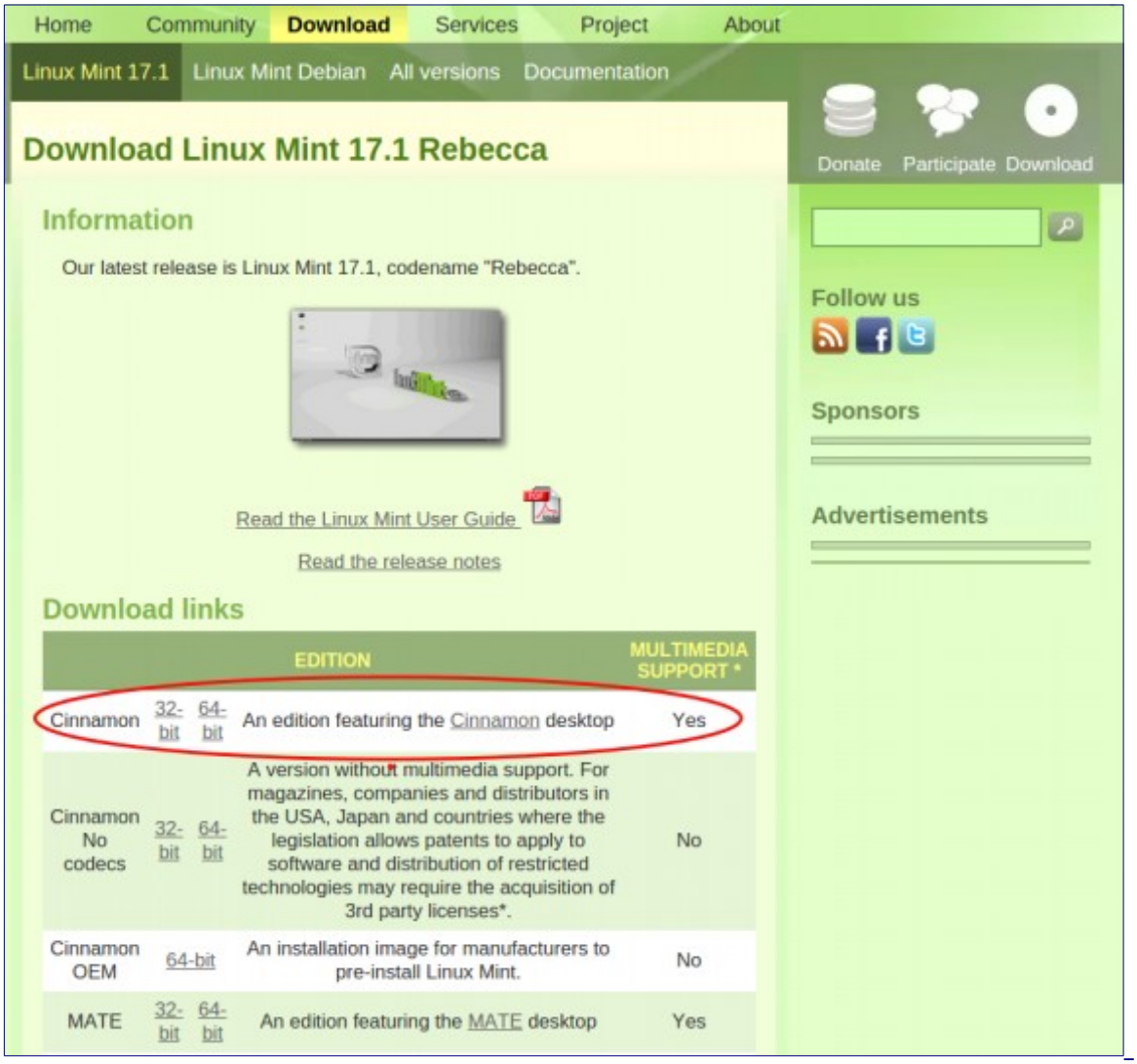


А итоговый образ, как уже говорилось, можно найти в каталоге /home/data/my-mint/remaster-new-files под именем livecd.iso.

«Потрошение» образа

А теперь я вернусь назад и расскажу о своём опыте «потрошения» исходного образа Cinnamon-редакции Mint. Не как пример для подражания или, тем более, копирования, но как вариант возможных действий. И себе на память — в качестве шпаргалки, тоже пригодится.

«Потрошению» подвергся образ Cinnamon-редакции Mint 17.1 Rebecca в 64-битном варианте:



Для начала я подключил PPA-репозитории, которые предполагал использовать:

```
# add-apt-repository -y ppa:mystic-mirage/komodo-edit && add-apt-repository -y ppa:zfs-native/stable && add-apt-repository -y ppa:andrew-crew-kuznetsov/crew
```

Напоминаю, что командная оболочка запущена от имени администратора, что символизируется видом приглашения командной строки в виде решётки #.

Подключение репозиториев можно проделать и с помощью mintsources, запустив его из командной строки. И тут только не следует тянуть с этим делом — после удаления ненужных (мне) программ он может работать с ошибками. Если же прибегнуть к стандартному add-apt-repository, то этой проблемы не возникает, и подключение PPA-репозиториев можно отложить.

Как обычно, подключение репозиториев надо завершить обновлением кеша пакетов:

```
# apt update
```

И теперь, казалось бы, самое время выполнить общее обновление системы. Однако — увы: у меня оно ни разу не проходило ни через mintupdate, ни через apt upgrade ни в каких вариантах: ни с отключением Менеджера обновлений в хост-системе в первом случае, ни с ручной фиксацией версий пакетов, не обновляемых mintupdate — во втором. Так что попытки эти я оставил, переходя сразу к удалению пакетов, которые полагаю лишними — Libreoffice,

GIMP, все мультимедийные, Thunderbird и всякие мелочи, типа Tomboy:

```
# apt purge tomboy gimp thunderbird libreoffice* banshee brasero totem vlc gimp-data libgimp2.0
```

Далее настала очередь шрифтов — удалению подверглись кхмерские, таиландские и прочие шрифты, столь необходимые в наших широтах:

```
# apt purge fonts-kacst fonts-kacst-one fonts-khmeros-core fonts-lao fonts-lklug-sinhala fonts-nanum fonts-sil-abyssinica fonts-sil-padauk fonts-takao-pgothic fonts-thai-tlwg fonts-tibetan-machine fonts-tlwg-garuda fonts-tlwg-kinnari fonts-tlwg-loma fonts-tlwg-mono fonts-tlwg-purisa fonts-tlwg-sawasdee fonts-tlwg-typewriter fonts-tlwg-typist fonts-tlwg-typo fonts-tlwg-umpush fonts-tlwg-waree ttf-indic-fonts-core ttf-punjabi-fonts
```

И напоследок — удаление драйверов видеокарт, последние представители которых были списаны в утиль много лет назад:

```
# apt purge
```

```
xserver-xorg-video-cirrus xserver-xorg-video-mga xserver-xorg-video-neomagic xserver-xorg-video-openchrome xserver-xorg-video-qxl xserver-xorg-video-s3 xserver-xorg-video-savage xserver-xorg-video-siliconmotion xserver-xorg-video-sis xserver-xorg-video-sisusb xserver-xorg-video-tdfx xserver-xorg-video-trident xserver-xorg-video-ati xserver-xorg-video-mach64 xserver-xorg-video-vmware xserver-xorg-video-vmware
```

Массовое удаление пакетов по традиции следует завершать командой

```
# apt autoremove
```

Она удаляет «осиротелые» зависимости, от которых больше ничего не зависит. Впрочем, в моём случае таковых не оказалось.

Теперь настало время ~~собирать камни~~ устанавливать пакеты. Каковых оказалось не так много:

```
# apt install zsh gprename hunspell-ru-ie-yo mc mdadm uck shutter komodo-edit komodo-edit-ru gnumeric abiword gnome-mplayer asunder lame flac terminator tilda yagf system-config-lvm f2fs-tools nilfs-tools btrfs-tools ubuntu-zfs google-earth-stable virtualbox-4.3
```

Вдаваться в объяснения, почему именно эти пакеты, полагаю здесь неуместным. Ну а порядок их перечисления в строк — абсолютно случайный, и образовался по мере воспоминания.

Напоследок была установлена новая Opera, предварительно скачанная и помещённая в каталог path2/remaster-root/tmp:

```
# apt deb path2/remaster-root/tmp
```

После установки пакетов остались последние штрихи — обеспечить «нескучные обои» некоторые мелочи. Первая задача была решена лобовым копированием с хост-системы (и в её среде) каталога с моими любимыми фоновыми картинками в каталог path2/remaster-root/usr/share/backgrounds. И созданием (уже в chroot-окружении) символической ссылки:

```
# ln -s ../linuxmint-alv/Mount_of_the_Rising_Sun.jpg \ default_background.jpg
```

Далее, с хост-системы же были перенесены настроечные файлы:

- мои конфиги для Zsh — ~/.zshrc и ~/.zshenv — в каталог /etc/zsh под именами newuser.zshrc.recommended и zshenv, соответственно;

- файла customize, описывающего доступные для UNC терминалы — в каталог path2/remaster-root/usr/lib/uck/customization-profiles/localized_cd/ (с тем же именем);

- файлов словаря для русского спеллинга в Komodo Edit (ru_RU.aff и ru_RU.dic) — в калалог path2/remaster-root/usr/lib/komodo-edit/mozilla/dictionaries.

И теперь оставалось только выйти из chroot-окружения и дождаться окончания сборки нового образа. Объём которого составил 1484 МБ — против 1478 МБ образа исходного. Но я и не ставил своей целью его уменьшение.

Получившийся образ был скопирован куда надо, получил нормальное имя alv-rebecca.02.iso и контрольную сумму командой

```
$ md5sum alv-rebecca.02.iso > alv-rebecca.02.md5
```

После этого образ подвергся проверке сначала в Live-режиме, а затем был установлен в виртуальной машине — проверять его на реальном железе мне в данный момент негде. Тем не менее, поскольку все предыдущие опыты такого рода заканчивались успешно, я решил поделиться своим образом с народом — вдруг кому пригодится. Так что скачать его можно с Яндекс.Диска.

Заключение

В заключение этого очерка скажу пару слов о том, зачем всё это делалось и делается — хотя, возможно, с этого следовало начать. Ибо вопрос этот возникает возникает довольно часто. Правда, я обычно отвечаю на него не вполне политкорректно:

Если вы не знаете, зачем это — значит, вам это не нужно.

Но можно попробовать ответить и иначе. Собирал я образы, разумеется, в первую очередь для себя, любимого, дабы иметь возможность устанавливать Mint с его Cinnamon в той комплектации, которая устраивает меня на 146%.

Во-вторых, с этого образа система будет устанавливаться на машины моих товарищей, не имеющих, по большей части, сложившихся предпочтений по части прикладного софта.

В-третьих, образ может быть использован для не совсем стандартных инсталляций, например, с подключением softRAID, что штатно его инсталлятором не поддерживается, на файловую систему Ext4 с отключённым журналирование (что до недавнего времени пропагандировалось Google как самое быстрое решение), для подключения существующего пула ZFS или создания нового.

В четвёртых, просто для экспериментов с дисковыми разделами и

файловыми системами, в том числе и с относительно экзотическими, такими, как f2fs или nilfs2.

И, наконец, в-пятых и последних — этот образ может послужить основой для специализированных сборок под некоторые задачи, например, связанные с цифровой картографией.

Вместо заключения. Порядок действий при настройке

Обычно в заключении к книгам, подобных этой , принято писать общие слова о достоинствах предмета описания (а заодно и автора оного). Отступлю от этой традиции и довольно конкретно опишу, в каком порядке лучше всего выполнять манипуляции по настройке дистрибутива Mint и среды Cinnamon, опираясь на всё изложенное выше в этих очерках. Разумеется, здесь в конспективном виде приводится мой личный алгоритм действий — как всегда, не в качестве образца для подражания, а как напоминание о необходимости выработки алгоритма собственного.

Первое, что я делаю после установки почти любого дистрибутива, с почти любой рабочей средой (и Mint с его Cinnamon не исключение) — переименовываю стандартные подкаталоги в своём домашнем каталоге. Если устанавливать систему с русской локалью, по умолчанию список этих подкаталогов будет таким:

```
$ ls -1 ~/

Видео

Документы

Загрузки

Изображения

Музыка

Общедоступные

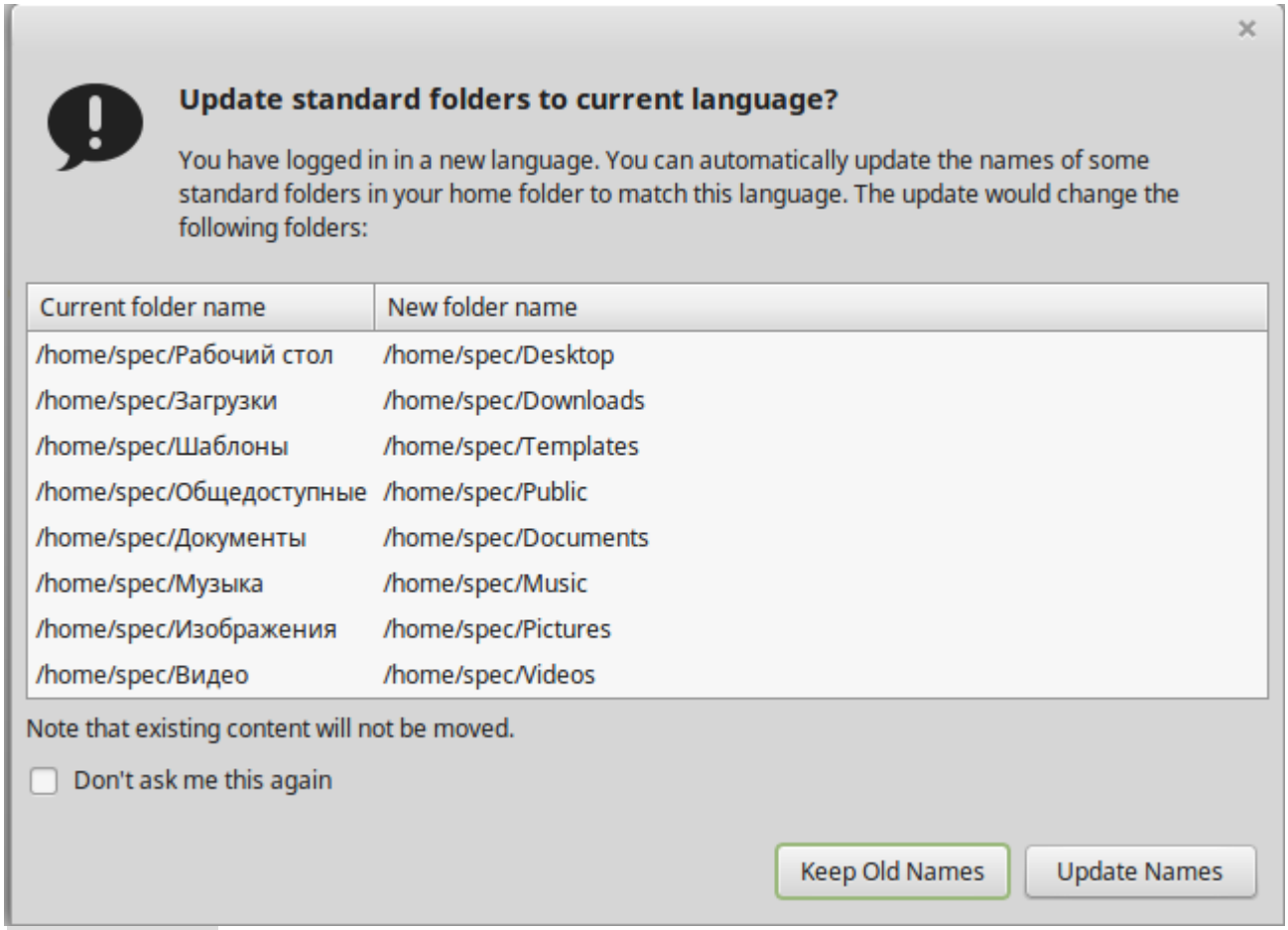
Рабочий стол

Шаблоны
```

Что, конечно же, соответствует пресловутому стандарту freedesktop.org . Но, по понятным причинам, крайне не удобно при работе в CLI. Зато легко исправляется такой командой:

```
$ LANG=C xdg-user-dirs-gtk-update --force
```

Вместо LANG=C можно указать LANG=POSIX или LANG=en_US , это одно и то же. И в любом случае за этим последует панель с запросом подтверждения: С ним следует согласиться, предварительно поставив «птицу» в боксике Don't ask methis again , чтобы в дальнейшем избежать повторения вопроса. После чего список подкаталогов в домашнем каталоге мгновенно приобретёт такой вид:




```
$ ls -1 ~/

Desktop/

Documents/

Downloads/

lost+found/

Music/

Pictures/

Public/

Templates/

Videos/
```

Который позвволяет более не задумываться о переключении раскладки клавиатуры во время навигации по файловой системе.

Следующие мои действия — создание комфортной среды обитания меня, любимого, что начинается с установки любимой оболочки Zsh:

```
$ apt install zsh
```

Затем копирование (с внешнего носителя или из сети, например, с Яндекс.диска) моих конфигов для неё в домашний каталог:

```
$ cp path2/{.zshrc,.zshenv} mytmp ~/
```

И, наконец, превращение Zsh в login shell:

```
$ chsh -s /bin/zsh
```

Как я уже гворил, в последнее время без выпадающего терминала я чувствую себя как без рук. И потому следующий шаг — установка оного:

```
$ apt install tilda
```

И внесение программы Tilda в список для Автозапуска в секции Параметры Ситемных настроек среды Cinnamon.

Раньше одним из первых моих действий по настройке среды была смена шрифтов интерфейса. Однако с появлением в Mint семейства гарнитур Notos оказалось, что в штатной теме оформления Cinnamon, именуемой Mint-X, меня в общем всё устраивает по умолчанию, а всякими частными украшательствами, типа подбора нескучных обоев и скринсейвером, я занимаюсь на досуге, под настроение.

А вот что для меня важно в среде обитания — это настройка клавиатуры, в которую для меня входят:

- включение привычных хоткеев для переключения между рабочими областями и управления окнами, в том числе их тайлингом;
- установка привычного варианта русской раскладки (Typewriter Legacy);
- выбор подходящих переключателей раскладки, клавиши Compose и некоторых других параметров совместимости.

Первоочередность установки языково-зависимых параметров клавиатуры вызвана тем, что обустройство в новой системе я почти всегда сочетаю с описанием этого процесса, которое выполняю на языке родных осин. И — в любимом текстовом редакторе, роль которого последнее время выполняет Komodo Edit, каковой устанавливается в это же время:

```
$ apt install komodo-edit komodo-edit-ru
```

И из внешнего источника в каталог ~/.komodoedit копируются необходимые конфиги и макросы. В том числе и словарные файлы для проверки русской орфографии, что требует предварительного подключения следующих репозиториев:

```
$ sudo add-apt-repository -y ppa:mystic-mirage/komodo-edit
```

```
$ sudo add-apt-repository -y ppa:andrew-crew-kuznetsov/crew
```

```
$ sudo apt update
```

Сочиняемые описания обычно нужно иллюстрировать скриншотами. А поскольку штатное средство для этого, имеющееся в Cinnamon, ниже всякой критики, то заодно устанавливается и соответствующий инструмент:

```
$ apt install shutter
```

Все эти вопросы были подробно рассмотрены в соответствующих очерках, и здесь я вдаваться в детализацию не буду.

Создав таким образом подходящую среду как для дальнейших действий, так и для их описания, можно заняться разборками с файловыми системами и системами размещения данных — в моём домашнем каталоге хранятся только файлв настроек приложений, рабочие данные я размещаю вне его.

Тут первое, что можно (если нужно) сделать — это отключить журналирование для файловой системы ext4, на которой у меня располагаются корень файловой иерархии и его ветвь ~/ (то есть, точнее, /home/alv). Делается это такими командами:

```
$ sudo tune2fs -O ^has_journal /dev/sda1
```

```
$ sudo tune2fs -O ^has_journal /dev/sda2
```

А успех операции проверяется командой

```
$ sudo tune2fs -l /dev/sda# | grep journal
```

которая должна просто вернуть приглашение командной строки.

Вопрос о том, нужно ли отключать журналирование для ext4, я здесь обсуждать не буду — ответ на него каждый должен дать себе сам.

Далее возможны варианты в зависимости от того, что предполагается использовать для хранения данных — LVM, softRAID или ZFS. Первый случай ни к каким подготовительным действиям не обязывает — к созданию физических томов и их групп, а также логических групп и томов поверх них можно приступать сразу, с помощью соответствующих консольных утилит. Однако, если есть желание воспользоваться интегрирующей их графической оболочкой, сооветствующий пакет надо установить:

```
$ apt install system-config-lvm
```

Если для размещения данных был выбран softRAID, то пакет для работы с ним должен быть установлен в обязательном порядке, в стандартной установке Mint таковой отсутствует:

```
$ apt install mdadm
```

Больше всего телодвижений потребует использование ZFS. И тут проще всего начать с получения «бессрочных» административных прав:

```
$ sudo -i
```

После чего подключить нужный репозиторий и обновить кеш:

```
# add-apt-repository ppa:zfs-native/stable
```

```
# apt update
```

Затем построить дерево зависимостей:

```
# apt build-dep ubuntu-zfs
```

Далее — собрать пакеты поддержки:

```
# apt install ubuntu-zfs
```

И, наконец, загрузив модули, необходимые для работы ZFS:

```
# modprobe zfs
```

проверить результат командой

```
# lsmod | grep zfs
```

Теперь наступает этап массового удаления ненужных пакетов и установки нужных. Останавливаться на нём я не буду — во-первых, это дело индивидуальных вкусов, а во-вторых, последнее время я устанавливаю Mint с самосборных образов, изготовленных посрджеством UCK, как было описано в последнем очерке. В этих образах ничего ненужного уже нет, а (почти) всё нужное присутствует.

Так что — за работу, товарищи!

Приложение. Перечень ресурсов

Эта книга была бы неполной без Перечня ресурсов, связанных с Mint и Cinnamon. Я не ставил себе целью собрать все сайты и блоги, в которых упоминаются этот дистрибутив и эта среда, даже русскоязычные. И потому, возможно, в моём Перечне ресурсов есть зияющие прорехи.

С другой стороны, все дистрибутивы семейства Ubuntu (а Mint в широком смысле слова принадлежит к этому клану) связаны друг с другом настолько тесно, что заставляют вспомнить, что говорили по этому поводу резонные люди из Одессы. Так что в Перечень попали некоторые ресурсы, имеющие к Mint и Cinnamon косвенное отношение. По иронии судьбы, это как раз самые интересные ресурсы из в русскоязычных ресурсов.

И ещё дисклаймер: делался этот Перечень в первую очередь для себя, любимого — на предмет упрощения работы над книгой. Так что несёт на себе отчётливый след вкусовщины, дедовщины и прочего субьективизма.

Англоязычные ресурсы

Англоязычные ресурсы начинаются, разумеется, с официального сайта Linux Mint, и тесно связанных с ним Linux Mint Blog'а (новости о дистрибутиве вообще), сайта Segfault (новости разработки) и официального форума. На последнем имеется русскоязычный раздел.

Здесь же, рядом, имеется официальная страничка Cinnamon, но похоже, что сама по себе она прекратила своё развитие, все новости относительно этой среды — на Mint Blog'е и Segfault'е. Однако разделы с темами, апплетами, десклетами и расширениями исправно пополняются.

Установочные образы всех поддерживаемых версий — по ссылкам отсюда. В наших палестинах, возможно, имеет смысл обратиться к зеркалу Яндекса.

Официальные репозитории — основной (с многочисленными зеркалами на каждом километре, здесь и по всему свету, за исключением России) и extra (кажется, без зеркал вообще).

Источники дополнительных пакетов — всё тот же бездонный Launchpad и несколько более мелководный GetDeb.

Linux Mint на Github'е — исходники Cinnamon и всех дистрибутив-специфических компонентов. Отсюда можно забрать и исходники Cinnamon одним архивом.

Всякого рода красивости (темы, обои, пиктограммки) в большом количестве можно найти на LinuxMint-Art. И в ещё большем — на NoobsLab.

Русскоязычные ресурсы

О существовании русскоязычного раздела на официальном форуме проекта я уже упоминал. Однако самые интересные форумы, как я уже говорил, имеют к Mint довольно косвенное отношение.

Первый в этом ряду — форум Росинка. Некогда он образовался для поддержки одноимённого проекта — отечественной сборки Linux Mint. Сам проект то ли заморожен, то ли прекратил развитие (последняя активность датируется октябрём 2013, когда вышла альфа-версия сборки на основе Mint 15, в свою очередь датированной маем того же года). Но форум его живёт — на нём обсуждаются вопросы, связанные с дистрибутивами семейства Mint, Ubuntu и соплеменными.

Второй — форум Matuntu. Он образовался совсем недавно, летом 2014 года, однако связанный с ним проект — отечественная сборка Ubuntu с рабочей средой MATE — возник ещё в те времена, когда об официальной поддержке этого десктопа в этом дистрибутиве никто и не заикался. Ныне же и проект, и форум активно развиваются. Причём на форуме обсуждается не только титульный дистрибутив, но весь спектр родственных систем, включая и Mint с Cinnamon.

Впрочем, существует и русскоязычный сайт и форум, носящий имя титульного дистрибутива. И не просто русскоязычный, а даже с русским доменным именем — Линукс Минт. Имеется сайт, также с форумом, с более традиционным именем — Официальное русскоязычное сообщество Linux Mint.

От коллективного творчества переходим к индивидуальному. Первое, что здесь имеет самое непосредственное отношение к Mint и Cinnamon — это сайт MintMem, который ведёт Валерий Желябовский aka Brego. Где, кроме того, можно найти материалы по замечательному текстовому редактору Komodo Edit.

Следующий ресурс, носящий название Авторские статьи об Open Source — также результат индивидуального творчества, и принадлежит Василию Алексеенко aka vasilisc. Правда, основная его тематика — Ubuntu и сородичи. Но, как я уже говорил, подчас очень трудно сказать, где кончается Ubuntu и начинается Mint.

На этом я пока поставлю… надеюсь, что не точку, а всего лишь запятую. В расчёте на то, что Рунет будет прирастать ресурсами по Mint и Cinnamon. Но, поскольку книга рассчитана в том числе и на совсем начинающих применителей, завершу это приложение небольшим списком русскоязычных ресурсов общего характера.

Ресурсы по Linux и UNIX вообще

Здесь в первую очередь следует назвать Linux по-русски — сайт Виктора Костромина, который он ведёт с 1999 года. И который, с одной стороны, является самым полным каталогом ссылок на русскоязычные ресурсы по нашей теме, а сдругой стороны, содержит большое количество авторских материалов, оригинальных и переводных.

Ориентироваться в океане свободного софта помогут ресурсы Linsoft и Zen Way — структурированные и аннотированные каталоги программ для Linux сотоварищи. А отыскать интересующий пакет в сборке под конкретный дистрибутив можно на Linux Packages Search.

Новостных Linux-и FOSS-ресурсов — бессчётное множество, наверное, больше, чем по настоящему интересных новостей по этой теме. Поэтому выбор их для того, чтобы быть в курсе событий — дело сугубо субъективное. Для меня он определился давно: Opennet — один из старейших порталов Рунета, который ведёт Максим Чирков, и Nixp — новостной сайт с элементами социальной сети, организованный Дмитрием Шуруповым.

Общие вопросы, связанные Linux и FOSS, обсуждаются на многих форумах, но среди них я поставил бы на первое место Unixforum.org, существующий уже более 10 лет. Кроме того, рассматриваемая тематика часто обсуждается в социальной сети Juick — здесь можно также быстро, часто в реальном времени, получить ответы на не вполне тривиальные вопросы.

И, наконец, тематике Linux и FOSS целиком посвящён «бумажный» журнал Linuxformat, материалы которого со временем становятся свободно доступными в онлайне в виде PDF-версии и WiKi. Впрочем, и подписку на него тоже ещё никто не запретил.

Оглавление

Аннотация....1
Об этой книге....1
Введение в Linux Mint....2
  Лирическое вступление....2
  Что такое Mint....3
  Немного истории....4
  Возвращаясь к современности....5
Введение в Cinnamon....5
  История....5
  Особенности....7
  Распространение....8
  Поддержка....8
Linux Mint: установка....10

Mint и Cinnamon: обзор среды....28
Cinnamon. Общий вид....29
Управляющая панель....32
Главное меню....34
Вызов приложений....38
Управление окнами....40
Тайлинг окон....44
Управление тайлингом....46
Тайлинг на практике....50
Интерфейс Cinnamon: краткий итог....51
Cinnamon и системные настройки....51
Системные настройки: вступление....51
Секция Оформление....52
Cinnamon и собственные темы....63
Секция Параметры....64
Автозапуск....65
Апплеты, десклеты и расширения....67
Панель и рабочий стол....73
Окна и их тайлинг....76
Рабочие области и Горячие углы....78
Конфиденциальность, Общие, Уведомления....83
Детали учётной записи....86
Прочие параметры....87
Секция Оборудование....89
О системе и дисплее....90
Клавиатура....91
Раскладки и переключатели....94
Мышь и сенсорная панель....104
Звук....106
Принтеры и цвет....107
Секция Администрирование....111
Окно входа в систему....111
Пользователи и группы....113
Mint: фирменный инструментарий....117
Вступление....117
Менеджер программ mintinstall....118
Интермедия об аккаунте в сообществе....130
Менеджер репозиториев software-sources....132
Менеджер обновлений mintupdate....138
Средства визуализация пакетов....145
Средство резервного копирования mintbackup....147
Программа записи USB mintstick....151
Языковые настройки — mintlocale....152
Менеджер драйверов и интегрированное видео AMD....155
Прочие настройки....158
Редактор dconf....158
Mint и консоль....162
GRUB2: восстановление загрузчика....167

Основы командного интерфейса....168
Введение в CLI....168
Командная строка....169
Автодополнение....171
Опции....172
Пример: опции команды ls....175
Аргументы....176
Пути к файлам....177
Кое-что об исключениях....178
Псевдонимы....179
Переменные....182
Навигация и редактирование....185
Управляющие последовательности....186
История команд....187
Поиск в истории....189
Регулярные выражения....189
Экранирование....191
О кавычках....192
Вводные слова о командных конструкциях....194
Совместное выполнение команд....195
Перенаправление....197
Конвейеры....198
Понятие о фильтрах....200
Сценарии оболочки....200
Понятие о функциях....203
Настройка шелла....204
Утилиты CLI....205
Самая главная команда....206
Команда sudo....211
Создание файлов и каталогов....215
Аттрибуция файлов....218
Навигация по файловой системе....223
Информация о файлах....225
Манипулирование файлами....229
Архивация....233
... и компрессия....236
Утилита find и xargs при ней....240
Команды обработки текстов: введение....246
Просмотр файлов....247
Сравнение, объединение и деление файлов....250
Поиск в файлах: grep сотоварищи....257
Поиск в файлах: утилита search....259
Sed: средство потокового редактирования....260
Текстовый редактор nano....262
Mint и Zsh....268
Zsh как login shell....269
Документация....274
Настройка приглашения....275

Темы приглашений....278
Приёмы навигации....283
Автодополнение....284
История команд....286
Рекурсивный поиск....287
Перенаправление расширенное и множественное....289
Просто псевдонимы и псевдонимы глобальные....291
Псевдонимы-суффиксы....293
Конфигурирование....295
Пакеты и репозитории....302
Пакеты, зависимости, библиотеки....303
Формат пакетов....305
Репозитории: введение....306
Устройство репозиториев Ubuntu....308
Особенности репозитория Mint....310
О PPA-репозиториях....311
Подключение PPA-репозиториев....312
Управление пакетами....318
Терминологическое введение....318
Средства для работы с пакетами. Обзор....320
Установщик пакетов dpkg....321
Установщик пакетов GDebi....325
Утилита apt. Реализация для Linux Mint....328
Введение....328
Общее описание....329
Применение....331
Информация о пакетах....333
Работа с бинарными пакетами....340
Обновление системы....345
Работа с пакетами исходных текстов....347
Итог....347
Примечание: кратко об apt из APT....348
Управление пакетами: Synaptic....349
Введение....349
Обзор....350
Настройка....365
Удаление пакетов: нетрадиционный метод....371
Пользовательские приложения....372
Файловый менеджер Nemo....373
Обзор возможностей....373
Nemo и его терминал....386
Некоторые расширения Nemo....389
Nemo и Dropbox....393
Nemo и Яндекс.Диск....399
Программы эмуляции терминала....401
GNOME Terminal....402
Terminator....413
Выпадающий терминал Tilda....425

Текстовые редакторы....430
Введение....430
Текстовый редактор Gedit....431
Альтернативные редакторы: введение....436
Текстовый редактор Geany....439
Вступление....439
Файл....441
Правка....444
Поиск....446
Вид....448
Документ....449
Проект....451
Сборка....451
Инструменты....451
Справка....453
Инструментальная панель....454
Поле редактирования, боковая панель и окно сообщений....455
Настройка....460
Geany и его плагины....472
Geany и его макросы....476
Geany и multiterm....483
Geany и управление файлами....485
Geany: управление проектами....488
Текстовый редактор Komodo Edit....494
Обзор....495
Русификация....497
Управление проектами....501
Макросы....502
Настройки....503
Вывод графики....509
И последнее о Komodo Edit....512
Заключение по редакторам....512
Графика....512
Средства сканирования: Simple Scan....513
Создание экранных снимков....517
Вступление....518
GNOME Screenshot....518
Shutter....520
Вьюверы изображений....526
Офисные приложения....531
Вступление....531
Abiword....532
Gnumeric....540
Мультимедиа....545
Медиа-потребительство....546
Оцифровка аудио-компактов....548
softRAID, LVM, ZFS....556
Общее введение....556

Инструменты дисковой разметки и форматирования....557
Виды дисковой разметки....558
CLI: инструменты разметки....559
Утилита fdisk....560
Утилита cfdisk....564
Утилиты форматирования....566
Про графические морды и особенно про GNOME Disks....566
Mint и softRAID....577
Mint и LVM....581
Mint и ZFS....593
Общее введение....593
Дисковая разметка....593
Файловые системы....594
Из истории систем размещёния....595
Начало истории ZFS....596
Из истории юриспруденции....598
Появление героини....599
Обзор возможностей....600
Аппаратные потребности....602
Терминология....602
Модели именования устройств....603
Включение поддержки ZFS в Mint....605
Создаём простой пул....607
«Избыточные» пулы....610
Пул кэшируемый....610
О некоторых опциях команды zpool....611
Создание файловых систем....612
Файловые системы: устанавливаем свойства....613
Перемонтирование....615
Подключение пула ZFS, созданного в другой системе....616
Индивидуализация системы....617
Свой Mint: введение....618
UCK: обзор....618
UCK: процесс....621
«Потрошение» образа....633
Заключение....636
Вместо заключения. Порядок действий при настройке....637
Приложение. Перечень ресурсов....642
Англоязычные ресурсы....642
Русскоязычные ресурсы....643
Ресурсы по Linux и UNIX вообще....643