федеральное государственное образовательное
учреждение Высшего профессионального образования
«Южный федеральный университет»

Дацюк О.В.

“Введение в Unix/Linux.”

Учебное пособие.

Ростов-на-Дону.
2009 г.

1. Введение в UNIX подобные операционные системы.

1.1.История и эволюция UNIX систем.

История развития операционных систем началась в 50-ые годы двадцатого столетия. В те времена производители ЭВМ, не уделяли должного внимания программному обеспечению, поставляющемуся вместе с приобретаемым оборудованием и покупатели уже сами разрабатывали необходимое программное обеспечение для своих нужд, либо модифицировали то, которое имелось. Отсутствие готовых операционных систем объяснялось, не тем что производители не хотели тратить время на их разработку, а тем что развивалась и модифицировалась, архитектура ЭВМ, каждое новое поколение компьютеров имело кардинальные отличия от предыдущего, на аппаратном уровне. Наиболее важным достижением в 60-е годы была создание архитектуры, на которой можно было организовать работу нескольких программ в режиме разделения времени(*time-sharing*). Это послужило толчком к созданию одной из первых ОС, *Multics*.

В ОС *Multics* была реализована единая централизованная файловая система. Дополнительно к тому, что в *Multics* одной из первых была реализована иерархическая файловая система, имена файлов могли быть практически произвольной длины и содержать любые символы. Файл или директория могли иметь несколько имен (короткое и длинное); также были доступны для использования символьные ссылки (*symlink*) между директориями. Еще одним из основных ведений была большая виртуальная сегментно-страничная память. В системе Multics была впервые реализована и следующая инновационная идея, а именно — динамическое связывание (*dynamic linking*) исполняемой программы с библиотеками кода. Благодаря динамическому связыванию, исполняемый процесс мог запрашивать у системы о подключении дополнительных сегментов к собственному адресному пространству, а именно сегментов, содержащих полезный код для исполняемой программы.

Компания *Bell Labs* вышла из проекта в 1969-м году. Несколько человек, разрабатывающих систему *Multics* в этой компании и создали систему *UNIX*. Позднее созданная система *UNIX* показала внешнее сходство с системой *Multics*, в том числе, названия используемых команд. Однако, философия дизайна *UNIX* была совершенно другой, ориентированной на создание системы как можно меньших размеров, и настолько простой, насколько это будет возможно, в чем и было ее основное отличие от системы *Multics*.

Название «*UNIX*» (изначально «*Unics*») было образовано от «*Multics*». Буква *U* в названии *UNIX* означала «*Uniplexed*» («односложная») в противоположность слову «*Multiplexed*» («комплексная»), лежащему в

основе названия системы *Multics*, для того чтобы подчеркнуть попытку создателей *UNIX*-а отойти от сложностей системы *Multics* для выработки более простого и работоспособного подхода.

Система *UNIX* разрабатывалась в несколько этапов. Началось все с того, что фирма *Bell* принимавшая участие в разработке ОС *Multics*, разочаровалась в данном проекте и вышла из него, отозвав всех своих программистов. На компьютер *GE-635*, где разрабатывалась *Multics*, установили *GECOS* – операционную систему, намного более простую, чем *Multics*. ОС *GECOS* позволяла работать с файлами и базами данных, но для сложных вычислительных программ совершенно не годилась. Конечно такая система, не удовлетворяло программистов *Bell*, особенно тех кто принимал участие в проекте *Multics*, их не оставляло желание разработать гибкую и пригодную для программирования систему. Среди них были Кен Томпсон, Деннис Ричи, Джои Осанна и Рад Кеннедей, которым ограничения *GECOS* сильно мешали. В конце весны 1969 г. Томпсон и Ричи обратились к руководству с просьбой предоставить им эксклюзивный мощный компьютер для работы над новой системой с разделением времени. В своем проекте программисты планировали собрать все лучшее, что было в *Multics*, сделать систему максимально гибкой и функциональной. Компания отказалась выделить отдельный компьютер, фирма *Bell* не могла по коммерческим соображениям отдать под некоммерческий проект машину стоимостью миллион долларов.

В свое распоряжение Кен Томпсон получил старенький компьютер *PDP-7*, стоявший в углу одной из лабораторий и редко использовавшийся. Кен тогда как раз закончил работу над игрой *Space Travel* - симулятором солнечной системы, по которой можно было летать на маленьком космическом корабле – и сразу захотел портировать ее со своего рабочего *GE-635* на эту машину. Во-первых, потому, что играть на *PDP-7* было намного дешевле, чем на *GE-635*, во-вторых – дисплей у *PDP-7* больше подходил для видеоигр. Правда, компьютер фирмы *DEC* не поддерживал многих функций, реализованных в игре. И для того чтобы запустить *Space Travel* на *PDP*, нужно было не только перенести исходный код, а с нуля написать всю программную среду, в которой будет работать программа. Именно этим и занялись Кен Томпсон и Деннис Ричи летом 1969 года.

Томпсон решил воплотить все самые удачные идеи, которые появились при разработке *Multics*, а именно: иерархическая древовидная структура файловой системы, концепции файла и процесса, командный интерпретатор для пользователя, многопользовательский режим работы (могли работать два пользователя одновременно) и много чего еще. Работа шла таким образом: на имевшемся до этого компьютере *General Electric 635* писали ассемблерный код и потом с помощью перфоленты переносили на *PDP-7*, на которой впоследствии отлаживали. Так было получено простенькое ядро будущей системы, текстовый редактор, несколько утилит и собственный Ассемблер. При этом оси требовалось всего 12 килобайт оперативной памяти (столько

весило ядро системы), 8 килобайт занимали программы и утилиты, а максимально допустимый размер файла составлял 64 килобайта. После этого можно было полностью продолжать работу уже на самой PDP-7 в создаваемой операционке. Первоначальное название, которое было придумано для новоиспеченного продукта, – *UNICS* (*Uniplexed Information and Computing System*). И немного позднее было сокращено до привычного нам *UNIX*. Официальной датой рождения *UNIX* и началом так называемой «эры *UNIX*» стало 1 января 1970 года.

В 1971 году лаборатории *Bell Labs* потребовалась система обработки текстов, и в качестве платформы для нее был выбран полюбившийся всей конторе *UNIX*. В этом же 71 году торговая марка *UNIX* была запатентована *Bell Labs* для серии машин *DEC PDP -11/20*. В это время Томпсон работал над компилятором языка *Fortran*, но то, что в итоге у него получилось, было названо языком B, который немного позднее превратился во всем нам хорошо известный C. В 1973 году *UNIX* был переписан на язык C, что сделало систему полностью переносимой. А в 1974 исходные коды *UNIX* стали распространяться в университетах за символическую плату, что обеспечило дальнейшую популярность этой оси, а также начало вовлекать в разработку все новых и новых разработчиков. Нарастающая популярность *UNIX* заставила Калифорнийский университет в Беркли предложить свой вариант *UNIX - BSD* (*Berkeley Software Distribution*) , на базе которого по заказу *DARPA* (Агентство перспективных проектов военного ведомства США) компания *BBN* реализовала в системе *BSD* 4.1 протоколы *TCP/IP*. Так возникла сеть Интернет. Небольшая цена, понятный и доступный для изучения код на C, гибкость и переносимость, возможность настроить ось под любую конфигурацию сделали ее привлекательной для большого количества не только профессионалов, но и любителей. Таким образом, были разработаны великий текстовый редактор *vi* (Билл Джой), возможность работы с виртуальной памятью (Поркер и Бабаоглу) и множество других утилит, без которых невозможно представить себе *UNIX*-подобные системы.

Необходимо вспомнить и разработанную в Массачусетском технологическом институте систему *X-Window* (1984 г.). Основанная на *TCP/IP*, она обеспечивает мобильный графический интерфейс, к которому прилагается концепция "клиент - сервер", наиболее революционная для своего времени. Сегодня *UNIX* и *X-Window* почти неразделимы.

Долгим и тернистым был *UNIX* на рынок программных средств. Считается , что только с 1 января 1984 г. дочерняя компания *AT&T Bell Labs* (позднее переименованная в *USL - UNIX System Laboratories*) вышла на рынок с UNIX в качестве торгового продукта.

Под благовидным предлогом стандартизации *UNIX AT&T* ввел *SVID* (*System V Interface Definition*) и этим ходом вновь отождествил *UNIX* со своей *System V* (1983 г.).

Другим важным событием стало соглашение *AT&T* с ведущими *UNIX* - производителями *Sun* и *Microsoft* в 1987 г. о так называемой унификации *UNIX*. Проект предусматривал создание четвертого издания *System V* (SVR4), которая объединяла характеристики *Xenix Microsoft* (другое название *UNIX* для микрокомпьютеров, основанной на седьмом издании и испытавшей сильное влияние *System V*), *sunOS* (система *UNIX* фирмы *Sun Microsystems*, основанной на BSD) и *System* V 3.2. В 1987 г. *AT&T* в первый раз лицензировала имя *UNIX*.

В мае следующего года несколько ведущих компаний, среди которых были *Apollo, Bull, HP, IBM* и *Siemens*, учредили Фонд открытых систем OSF (*Open Software Foundation*) - организацию с целью разработки и распространения открытых программных систем. Она финансировала разработки программного обеспечения в соответствии с наиболее современными требованиями к открытым системам, специфицированными в соответствующих стандартах для разработки: системный интерфейс *OSF/AES*, графический потребительский интерфейс *OSF/Motif*, распределенные системы *OSF/DCE* и т.д.

Основой проекта является выбор *UNIX* - технологии. После внимательного изучения выбрали ядро Match, разработанное в университете Карнеги-Меллон (100000 строк исходного кода; код первой *UNIX* , код первой *UNIX* был на порядок скромнее), а все остальное (*OSF* -1 имеет в общем около 800000 строк кода) предлагалось взять из *IBM AIX* третьей версии.

Выбор *AIX* не понравился *AT&T*. Концерну было предложено принять членство *OSF*, но он поставил неприемлемое условие заменить *AIX* на *SVR4*. В результате *AT&T, Sun, UNISYS, XEROX* и др. создали в марте 1989 г. организацию *UNIX International (UI),* которая дала широкую дорогу *USL*. Разработка и лицензирование программ оставались привилегией *USL*, но общий контроль был сохранен за *AT&T*. Взаимные столкновения и несовместимость продуктов *OSF* и *UI* явилась кульминационным пунктом в так называемых UNIX-войнах .

В 1993 г компания *AT&T* продала свою долю прав на *UNIX* фирме *Novell*. За это время фирма *Novell* выпустила собственные версии *UNIX* на базе *System V* версии 4, получившие название *UNIXWare*. *UNIXWare* предназначена для взаимодействия с системой *NetWare* разработки Novell.

В декабре 1995 г. компания *Santa Crouze Operation* купила у *Novell* патенты на все ее *UNIX* - продукты, включая ОС *UNIXWare* и исходный код *UNIX System V*.

На сегодняшний день существует несколько операционных систем, которые имеют полное право использовать именно слово *UNIX* в своем название. Они удовлетворяют либо стандарту *BSD*, либо стандарту *System V*, все эти системы являются коммерческими и стоят достаточно больших денег.

1.2.История создания Unix-подобных систем.

Началась история развития свободно распространяемого программного обеспечения (ПО), которому, несомненно, относятся такие *UNIX-подобные* системы как *Linux*, в 1983 году. Задолго до появления Linux, Ричардом Мэттью Столлманом, был основан проект *GNU*(***G****NU's* ***N****ot* ***U****NIX)*. Проект *GNU* был ответом на усиливающуюся тенденцию распространения коммерческого ПО и закрытие исходных текстов программ авторским правом. Такая тенденция не устраивала Ричард Мэттью Столлман, он привык, что может получать доступ к исходным кодам программ и вносить в него правки по своему усмотрению. Столлман организовал Фонд свободного программного обеспечения (*Free Software Foundation*) чтобы в рамках данного проекта начать создавать свободно распространяемое ПО, где каждый желающий мог бы получить исходную программу и внести в нее те изменения, которые посчитает необходимым. Именно тогда началось жесткое противостояние сообщества свободных программ и индустрии коммерческого ПО. Оно продолжается, и сегодня и наверняка будет длиться еще долго. Многие считают, что *GNU* это бесплатное ПО, но это не совсем так, *GNU* свободно распространяемое программное обеспечение, просто слово *free* в английском языке имеет два значения – свободный и бесплатный.

Разработка системы *GNU* началась с написания приложений, а не ядра ОС. Было решено, что ядро должно отвечать принципам *UNIX* системы, исходя из этого предположения, начался этап разработки приложений, для будущей операционной системы. Был написан редактор *emacs*, компилятор языка С, библиотека важнейших функции *libc*, оболочка пользователя, получившая названия *bash*(*Bourne Again Shell*) и множество других утилит, обновленные версии которых входят в дистрибутивы систем *GNU* и в настоящее время.

В программах *GNU* использовались системные вызовы *POSIX*, реализованные почти во всех системах *UNIX*, из-за чего все приложения *GNU* могли работать в любой *UNIX*-подобной операционной системе. Но оставалась проблема, отсутствия свободного *UNIX*-совместимого ядра, разработчики *GNU* должны были пользоваться запатентованными платными ядрами.

В 1991 году Линус Торвальдс, финский студент, захотел написать *UNIX-совместимое* ядро для своего персонального компьютера *i386*. До появления *i386* архитектуры было физически невозможно разрабатывать *UNIX* подобные архитектуры для персональных компьютеров. За основу Линус взял ОС *MINIX*, учебной операционной системы, для демонстрации архитектуры и возможностей *UNIX* систем. Линус хотел получить полноценное функциональное ядро. Название своему ядру он дал *freax*, но хозяин *ftp* сервера заменил это на *Linux* – гибрид имени и слова *UNIX*.

Совместимость с *UNIX* означала поддержку стандарта *POSIX*, а поддержка *POSIX*, автоматически позволяло использовать приложения *GNU*.

Огромную роль в создании *Linux* сыграли глобальные компьютерные сети *Usenet* и *Internet*. Линус Торвальдс обсуждал свою работу и возникающие трудности с другими разработчиками в телеконференциях. Решением, которое позволило ускорить разработку стала публикация исходных текстов ещё малоработоспособной первой версии ядра под свободной лицензией *GNU GPL*. Благодаря этому и получавшей всё большее распространение сети Internet очень многие заинтересованные программисты получили возможность самостоятельно компилировать и тестировать ядро, участвовать в обсуждении и исправлении ошибок, а также присылать исправления и дополнения к исходным текстам Линуса. Работа над ядром пошла быстрее и эффективнее.

В 1992 году версия ядра *Linux* достигла 0.95, а в 1994 году вышла версия 1.0, что свидетельствовало о том, что разработчики, наконец сочли, что ядро в целом закончено и все ошибки (теоретически) исправлены. В настоящее время разработка ядра *Linux* — дело уже гораздо большего сообщества, чем во времена до версии 1.0. Изменилась и роль самого Линуса Торвальдса: теперь он не главный разработчик, а наиболее авторитетный член сообщества, по традиции оценивающий качество исходных текстов, которые должны быть включены в ядро, и дающий своё добро на их включение. Тем не менее, общая модель свободной разработки сообществом сохраняется.

Сочетание *GNU* и *Linux* давало возможность создать свободную операционную систему, но само по себе ещё не составляло такой системы, потому что *Linux* и различные утилиты *GNU* оставались разрозненными программными продуктами, написанными разными людьми, не всегда принимавшими в расчёт то, что делали другие. Основным же свойством любой системы является согласованность её компонентов. Скомпилированное ядро *Linux* с небольшим комплектом скомпилированных уже на *Linux* утилит *GNU* представлял собой операционную систему, в которой можно было уже выполнять какие-то прикладные задачи. Конечно, первое, чем можно было заниматься в *GNU/Linux* — писать программы на Си. На самом деле, всякий раз, когда говорится «операционная система *Linux*», понимается «ядро *Linux* и утилиты *GNU*». Фонд свободного ПО рекомендует использовать название GNU/Linux. Проект *GNU* имеет и свое собственное ядро *Hard*, название системы на основе этого ядра *GNU/Hard*. Проект разработке ядра *Hard* все еще не завершен, хотя многие *GNU* приложения на нем уже функционируют нормально.

Когда задача получить компьютер с постоянно работающей на нём системой *GNU/Linux* стала востребованной и довольно распространённой, разработчики в хельсинкском и техасском университетах создают собственные наборы дискет, с которых скомпилированное ядро и основные

утилиты можно записать на жёсткий диск, после чего загружать операционную систему прямо с него. Эти наборы дискет стали первыми прототипами современных дистрибутивов *Linux* — комплектов программного обеспечения, на основе которых можно получить работающую операционную систему на своём компьютере.

Назвать набор дискет дистрибутивом можно было достаточно условно. Скопировать программы на жесткий диск, чтобы получить работающую ОС, задача не тривиальная даже для продвинутого пользователя. Полноценной ОС, требуется программа инсталлятор, и программы конфигурирования ОС. Именно наличие таких средств отличает современные дистрибутивы *Linux*. Еще одной важной чертой дистрибутивов ОС *Linux* это система автоматического обновления, в *Linux* нельзя установить программы раз и на всегда, их еще необходимо обновлять. Свободное программное обеспечение очень динамично развивающаяся область.

Первым дистрибутивом получившее широкое распространение стал *Slackware*, собранный Патриком Фолькердингом. Он появился уже в 1994 году.

Первой коммерческой фирмой успешно развивающей *Linux* дистрибутивы стала *Red Hat*, она создавала дистрибутивы рассчитанные на широкий круг пользователей. Обычно дистрибутив *Linux* системы в те времена, использовал программист, и именно в расчете на них создавалось большинство дистрибутивов. *Red Hat* создавала свой дистрибутив не только для программистов профессионалов, но и для обыкновенных пользователей и системных администраторов, для которых компьютер – в первую очередь офисное рабочее место или рабочий сервер. *Red Hat* уделяла много внимания разработке приложений с графическим интерфейсом, для выполнения типичных задач по настройке и конфигурированию системы. Благодаря фирме *Red Hat*, очень широкое распространение получил формат пакетов *RPM*(*Red Hat Package Manager*).

Практически одновременно с *Red Hat* появился проект *Debian*. Его задача была примерно той же — сделать целостный дистрибутив *Linux* и свободного программного обеспечения *GNU*, однако этот проект был задуман как принципиально некоммерческий, проводимый в жизнь сообществом разработчиков, нормы взаимодействия в котором полностью соответствовали бы идеалам свободного ПО. Сообщество разработчиков *Debian* — международное, участники которого взаимодействуют через *Internet*, а нормы взаимодействия между ними определяются специальными документами — полиси(*policy*).

В настоящий момент существует множество дистрибутивов *Linux* систем. Наиболее известные коммерческие дистрибутивы *Red Hat* и *Suse Enterprise Linux*. Проект *SUSE* изначально разрабатывался в Германии, но сейчас его владельцем является американская корпорация *Novell, Inc*.. Был основан на дистрибутиве *Slackware*, однако был значительно переделан и

представляет собой обособленный дистрибутив, отличается от последнего удобством, а также системой администрирования и управления пакетами *YaST*. Со временем *SUSE* включила в себя много аспектов *Red Hat Linux* (например, использование системы RPM и */etc/sysconfig*). Есть свободно распространяемая версия *openSUSE*.

Наиболее популярными свободными дистрибутивами являются дистрибутивы основанные на *Debian* это *Unbuntu, Kubuntu* и т.д.

Контрольные вопросы и задания.

1. Какая система послужила прототипам для ОС UNIX?
2. Почему ОС разрабатываемая в *Bell Labs* получила название UNIX?
3. Какая одна из особенностей ОС *UNIX*, позволила быстро переносить систему с одной архитектуры на другую?
4. На какой системе был реализован стек протоколов *TCP/IP*?
5. Основные идеи проекта *GNU*?
6. Как проект *GNU* связан с *Linux*?
7. Что представляет собой дистрибутивы *Linux*?

2. Особенности архитектуры.

2.1.Функциональные свойства UNIX-подобных систем

Одной из главных особенностей архитектур *UNIX* систем является ее хорошая стандартизация. В мире существует множество *UNIX* систем, *AIX, Solaris,HP-UX, FreeBSD, GNU/Linux* поэтому очень часто в литературе используется термин UNIX-подобные системы, для обозначения всего спектра операционных систем построенных по принципам близким к *UNIX* стандартам. Основой всех *UNIX-подобных* систем является принципиально одинаковая архитектура и ряд стандартных интерфейсов. Опытный администратор без особого труда сможет обслуживать другую версию операционной системы, а для пользователей переход на другую систему и вовсе может оказаться незаметным.

Другой отличительной чертой *UNIX* подобных систем является многопользовательский режим работы. Системы *UNIX* изначально проектировались и создавались, с таким расчетом, чтобы позволить работать с одной ЭВМ нескольким пользователям одновременно. Для этого раньше к машинам подключали терминалы, и любой человек мог сесть за него и начать работать с системой. Когда начали развиваться сетевые технологии, то терминалы как устройства были заменены, виртуальными терминалами. Были разработаны несколько сетевых протоколов, которые позволяли удаленно подключаться к *UNIX-системам* с любого компьютера в сети и

работать с ними, словно сидишь за терминалом непосредственно подключенном к компьютеру. Самыми известными протоколами являются *telnet, rlogin, ssh*.

Еще одной особенностью *UNIX* является то, что архитектура сразу создавалась как многозадачная. Поэтому до появления процессоров архитектуры i386 не было и речи о переносе системы на архитектуру PC. Многозадачность подразумевает, что в системе одновременно выполняется множество процессов(задач). Специальная организация процессов позволяет им выполняться в своем собственном защищенном адресном пространстве, это гарантирует безопасность и независимость процессов. Взаимодействие и управление процессами происходит через стандартизированный набор системных вызовов одинаковый для большинства систем.

В операционной системе *UNIX* существуют два основных типа объекта с которыми приходиться взаимодействовать пользователю это файлы и процессы. Эти объекты как раз и определяют архитектуру операционной системы.

Все данные пользователя хранятся в файлах; доступ к периферийным устройствам осуществляется посредством чтения и записи специальных файлов; во время выполнения программы, операционная система считывает исполняемый код из файла в память и передает ему управление.

Вот спиоск основных особенностей *UNIX*

- поддержка многопользовательского режима;
- поддержка многозадачного режима;
- единая иерархическая файловая система;
- развитые средства работы в компьютерных сетях.
- широкое использование текстовых файлов для хранения настроек
- широкое использование утилит обработки текста для выполнения повседневных задач под управлением скриптов.
- “Подъем” ОС после загрузки ядра путём исполнения скриптов стандартным интерпретатором команд.
- широкое использование конвейеров (*pipe*).

2.2.Файловая система UNIX

При подключении пользователя к удаленной системе он попадает в свой домашний каталог, являющийся частью единой файловой системы.

UNIX-подобные операционные системы поддерживают древовидную иерархическую структуру, файлов и каталогов. При такой структуре представления данных на диске, каждый файл расположен в определенном хранилище данных – каталоге, каждый каталог вложен в какой-то другой

каталог. В результате получается дерево, вершинами которого являются не пустые каталоги, а листьями файлы и пустые каталоги. Корень такого дерева называется корневым каталогом и обозначается специальным символом / (прямой слэш). Каждому элементу файловой системы соответствует имя, определяющее его положение в дереве файловой системы. Полным путем к файлу называется список всех вершин дерева файловой системы, начиная с корня, записанных слева направо и разделенных специальными символами разделителями /, которые необходимо пройти, чтобы добраться до файла. Полным именем файла называется полный путь к файлу плюс его имя.

Например: */export/home/oleg/cpp/prog1.c*

Кроме понятия полный путь, в UNIX используется понятие относительного пути – это путь к каталогу или файлу от текущего каталога. Если мы находимся в домашнем каталоге пользователя *oleg* - */export/home/oleg*, то относительный путь к тому же файлу будет *cpp/prog1.c* . Важно, чтобы первое имя в относительном пути (не начинающегося с символа /) было видно из текущего каталога.

Физически каталоги и файлы могут находиться на разных дисках или даже на разных компьютерах, но все равно они будут частью единой файловой системы. На Рис. 1 схематично представлена файловая система UNIX. Полное имя файла hosts.txt находящегося в каталоге с именем *etc*, выглядит так */etc/hosts.txt*, а полный путь к нему */etc*, в конце пути обычно знак разделителя не ставится.

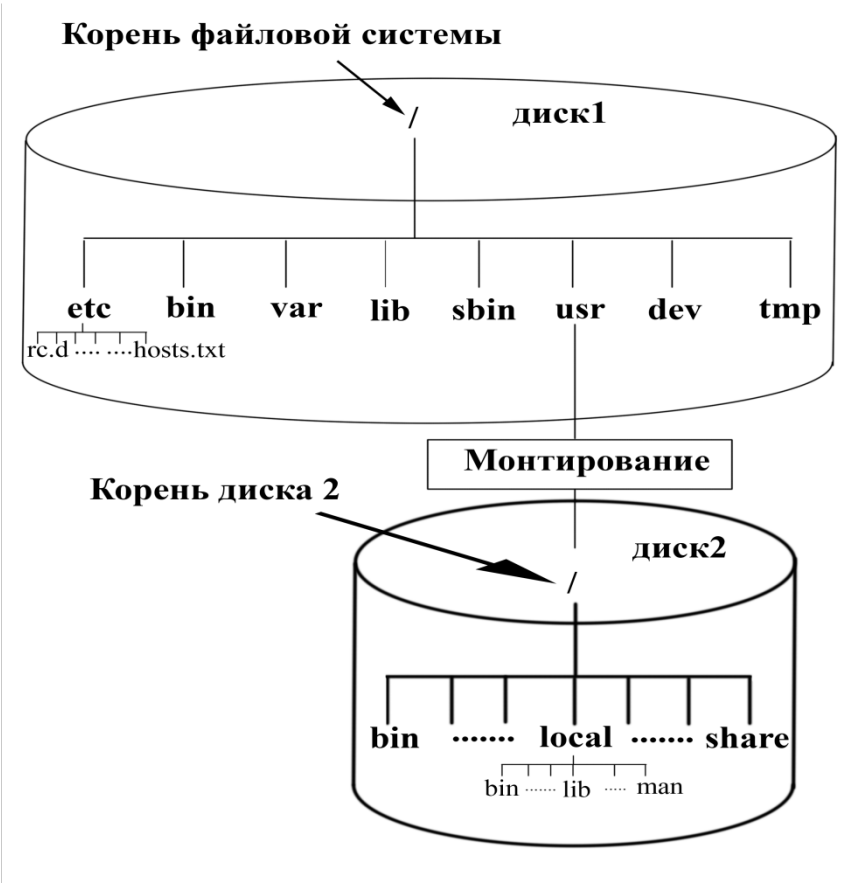


Рис. 1 Схема файловой системы UNIX

2.3.Процессы в ОС UNIX

В Unix системах под процессом понимается объект операционной системы, выполняющий код программы и имеющий свой собственный контекст: стек, набор страниц памяти, таблицу открытых файлов и уникальный номер (*PID* – сокращение от process *ID*), присвоенный ему системой. Из определения видно, что понятие процесс не тождественно понятию программа. Несколько процессов могут исполнять одну и ту же программу в одно и тоже время. Например, в системе есть одна копия программы *tcsh*, но может существовать много процессов исполняющих код этой программы, т. к. она запускается каждый раз при подключении нового пользователя к системе. Конфликтов между разными процессами, выполняющими код одной программы, не возникает, потому что каждый процесс имеет собственный контекст.

В момент загрузки системы запускается самый первый процесс с номером 1. Все остальные процессы являются потомками первого процесса. Для порождения процессов в ядре ОС *UNIX* имеется специальный механизм *fork*(ветвление). Процесс, порожденный системным вызовом fork, называется дочерним процессом, а процесс его породивший - родительским процессом.

Все объекты ОС *UNIX*, как статические – файлы, так и динамические – процессы, являются собственностью какого-либо пользователя. Собственником системных объектов является суперпользователь **root**. Этот механизм позволяет организовать надежную и эффективную защиту объектов ОС *UNIX*. На практике это означает, что никто не может удалить процесс или файл какого-либо пользователя, кроме него самого или суперпользователя.

2.4. Команды в ОС UNIX

Работа с ОС UNIX в удаленном режиме выполняется главным образом в режиме “командной строки”. Пользователь вводит команду и ожидает завершения ее выполнения. Выполнение команды состоит из нескольких этапов:

shell выдает приглашение и ожидает ввода команды;

пользователь вводит команду и нажимает *Enter(Return)*;

shell анализирует введенную команду;

если введена внутренняя команда *shell*, то *shell* исполняет ее,

если введена внешняя команда, то *shell* пытается найти

исполнимый файл с программой и запускает ее на выполнение.

Поиск выполняется внутри каталогов, перечисленных в переменной *PATH*. Приглашением является набор символов, который выводит командная оболочка, ожидая ввод командной строки. Вид приглашения устанавливается пользователем.

Синтаксис командной строки:

команда [опции] [аргумент] ... [аргумент]

команда – строка символов, совпадающая с именем программы или встроенной функцией оболочки, которую необходимо выполнить;

опции (ключи) – специальные параметры программы, которые влияют на процесс ее выполнения и результат работы команды. Параметры могут состоять из одного символа перед которым ставится знак минус (*-l*) или из слова целиком, перед которым ставятся два знака минус подряд *(--help*). Параметры состоящие из одного символа могут объединятся например *–al*, что эквивалентно *–a –l*. Для разделения компонентов внутри командной строки по умолчанию используется пробел;

аргумент – параметр, передаваемый команде.

В скобках указываются необязательные элементы команды. Таким образом, обязательным элементом является только имя команды.

В обычном режиме выполнения команды блокируют терминал до завершения команды. В тех случаях, когда команда выполняется длительное время, используют фоновый режим выполнения команды:

команда [опции] [аргументы] &

В этом случае, происходит немедленное освобождение терминала, а команда выполняется в фоновом режиме.

С каждым процессом в ОС UNIX связывается три стандартных файла: файл ввода (*stdin*), файл вывода (*stdout*) и файл диагностики (*stderr*). По умолчанию файл вывода и диагностики связывается с монитором, в файл ввода с клавиатурой. Однако выдачу результата можно перенаправить в любой файл с помощью знаков **>** или >> :

команда [опции] [аргументы] > outfile или

команда [опции] [аргументы] >> outfile.

Если файл с именем outfile уже существовал то в первом случае он будет перезаписан заново, а во втором, выдача будет добавлена в конец файла. В тех случаях, когда выдача результата не требуется, то используют перенаправление в спецфайл */dev/null*. Выдача в “никуда”.

Для перенаправления файла ввода используется символ <. Например, можно заранее подготовить письмо в файле letter, а затем отправить его по электронной почте пользователю drug:

mail drug < letter

Часто возникает необходимость, перенаправить вывод одной команды на вход другой. Для этого используются конвейеры. Конвейером называется операция, когда выход одной команды перенаправляется на вход другой команды. Используется специальный оператор | (конвейер) между двумя

командами для перенаправления *stdout* первой команды на *stdin* второй команды.

команда1 [опции] [аргументы] | команда2 [опции] [аргументы]

ps -el | wc -l - команда **ps** выводит список процессов в системе, и передает его на вход команды **wc –l**. Команда **wc –l** подсчитывает число строк, полученных от команды **ps**. В результате получим число выполняющихся процессов в системе.

Очень мощным средством является механизм подстановки результата работы команды. Для этого команда заключается между знаками "слабое ударение" (`...`). Результат можно подставлять, либо в качестве аргумента в другую команду, либо присваивать какой-либо переменной:

banner `date | cut -c12-19`

DATE=`date | cut -c12-19`

Как правило, в строке набирается одна команда, но можно набрать и несколько команд, для этого их необходимо разделить символом ‘;’ :

cmd1;cmd2;cmd3 - последовательное выполнение

2.4.1.Условное выполнение команд

cmd1 && cmd2 - 2-я команда выполнится, только в случае успешного завершения 1-ой команды

cmd1 || cmd2 - 2-я команда выполнится только, если выполнение 1-ой команды завершилось с ошибкой

Группирование процессов выполняется заключением нескольких команд в круглые скобки **(k1;k2;k3)**. Команды отделяются друг от друга точкой с запятой. Группирование обычно применяется при условном выполнении команд:

k1 && (k2;k3)

Однако эта процедура обладает еще одним свойством – для выполнения сгруппированных команд запускается отдельный shell. Если для группирования команд использовать фигурные скобки, то команды будут выполняться в той же самой оболочке.

2.4.2.Останов выполнения команды

Выполнение интерактивной команды прерывает комбинация клавиш “*CTRL C*”. Часто для этой цели пытаются использовать комбинацию “*CTRL Z*”, однако эта комбинация только приостанавливает выполнение, а не завершает его. Для завершения фоновой команды используется специальная команда **kill**:

kill -9 PID

Здесь **PID** – идентификатор процесса, а -9 сигнал завершения.

Большинство команд в операционной системе *UNIX* помимо выдачи на стандартные потоки вывода и ошибок возвращает еще и код завершения команды. По этому коду завершения *shell* определяет, как завершилась работа программы. Не официально принято, что при правильном завершении, команда возвращает нуль. Если код возврата отличен от нуля, то это означает ошибку при выполнении команды.

2.5.Наиболее часто используемые команды

В системе UNIX несколько тысяч команд, однако, половина из них является командами системного администратора, а другая выполняет вспомогательные функции. В обычной работе пользователю достаточно хорошо знать несколько десятков команд. В этом пособии мы рассмотрим кратко небольшой набор наиболее употребительных команд. В первую очередь потребуются команды для работы с каталогами и файлами. Как и ранее, в квадратных скобках будем указывать необязательные параметры.

Команды для работы с файлами и директориями.

Допустимые имена файлов

В именах файлов и каталогов можно использовать любые печатные символы, однако не рекомендуется использовать символы, имеющих специальное назначение:

slash (/)	разделитель имен каталогов в полном имени файла
backslash (\)	символ экранирования специального значения символов
ampersand (&)	символ исполнения команды в фоновом режиме
angle brackets (< and >)	символы перенаправления ввода вывода
question mark (?)	означает любой символ в шаблоне
dollar sign ($)	означает подстановку значения переменной
left right bracket ([])	определяют диапазон символов
asterisk (*)	строка произвольной длины из любых символов
vertical bar (|)	оператор конвейера
space ( )	разделитель в командной строке

UNIX не запрещает использовать эти символы в именах файлов, но необходимо экранировать их специальное назначение символом \ или заключать их в одинарные кавычки '*'. Не требуется обязательное использование тех или иных расширений, однако некоторые программы

ориентированы на работу с файлами с определенным расширением, поэтому не рекомендуется использовать их произвольным образом.

Соглашения о расширениях:

.c	файл с программой на С;
.h	include файл;
.f	файл с программой на фортране;
.o	объектный файл;
.a	статическая библиотека;
.so	динамическая библиотека;
.html	HTML документ;
.tar	архивный файл;
.gz	сжатый файл.

Некоторые команды допускает использование в качестве аргумента списков имен файлов. Эти списки удобно формировать с помощью шаблонов:

*	соответствует любой строке;
?	соответствует любому символу;
[c1- c2]	любая литера из диапазона символов c1-c2
[!c1-c2]	любой символ, кроме заданного диапазона.

Имена, начинающиеся с точки (.) присваиваются служебным и конфигурационным файлам и каталогам.

2.5.1.Команды для работы с деревом каталогов

pwd - напечатать полное имя текущей директории.

cd [dirname] - перейти в указанный каталог

Здесь dirname имя каталога, которое может состоять из собственно имени и пути к нему. Путь может быть абсолютным, если он начинается с символа /, и относительным, если начинается с любого другого символа.

Примеры перемещения по дереву каталогов:

cd /export/home/oleg – переход в домашний каталог пользователя oleg;

cd / - переход в корневой каталог файловой системы

cd prog/cc - переход из текущего каталога в каталог cc, находящийся в каталоге prog

cd ../gosha/bin - возврат на шаг назад и переход в каталог bin пользователя gosha

cd ~/bin - переход в свой каталог bin

cd - переход в свой домашний каталог

специальные имена каталогов:

. (точка) – текущий каталог

.. (две точки) – родительский каталог по отношению к текущему

~ (тильда) - домашний каталог

- (тире) - возврат в предыдущий каталог.

ls [опции] [имена] – выводит содержимое каталога или атрибутов файлов.

имена – это имена директорий или файлов. Если имена не указаны, то выводится содержание текущей директории.

Наиболее часто используются опции

-a - вывести все файлы (даже если имена начинаются с точки)

-l – вывести подробную информацию о файлах и папках(права доступа, имя владельца и группы, размер в блоках по 512 байт, время последней модификации, имя файла или директории)

-t - имена файлов сортируются не по алфавиту, а по времени последнего изменения

-R - рекурсивно пройти по всем подкаталогам, выводя по ним информацию

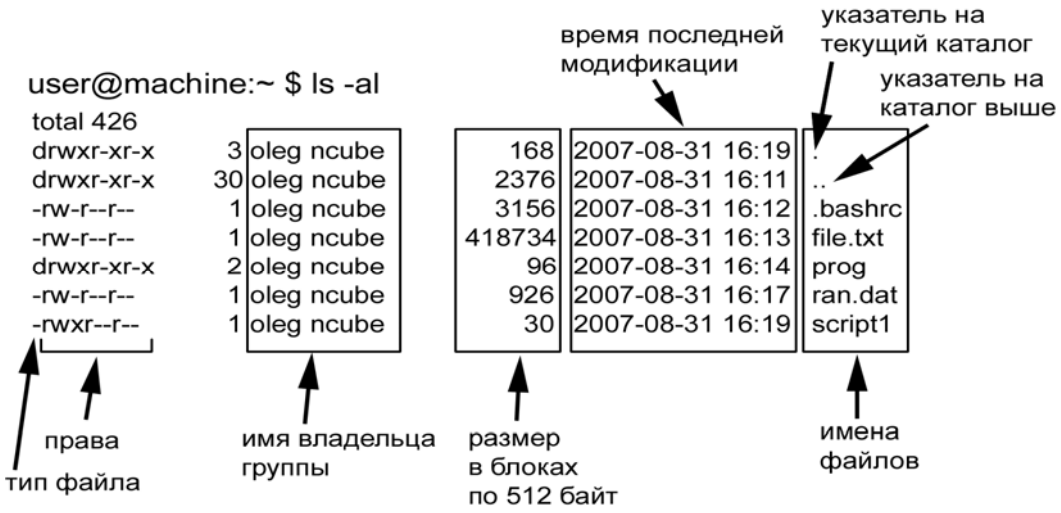


Рис. 2 Результат выполнения команды ls –al.

На Рис. 2 приведен результат выполнения команды **ls –al**. Выводится полный список файлов, и полная информация о каждом файле.

Первый символ в строке обозначает тип файла. Если строка начинается с символа "**–**", то это обычный файл, знаком "**d**" – обозначается что файл является директорией (в unix директории или каталоги являются специальными файлами), "**c**" – специальный файл, связанный с устройством выдающим информацию виде потока байт (т.н. *raw device*), "**b**" - специальный файл связанный с блочным устройством, "**l**" – означает что файл является ссылкой на другой файл. Далее следуют права доступа к файлу, их рассмотрим подробнее.

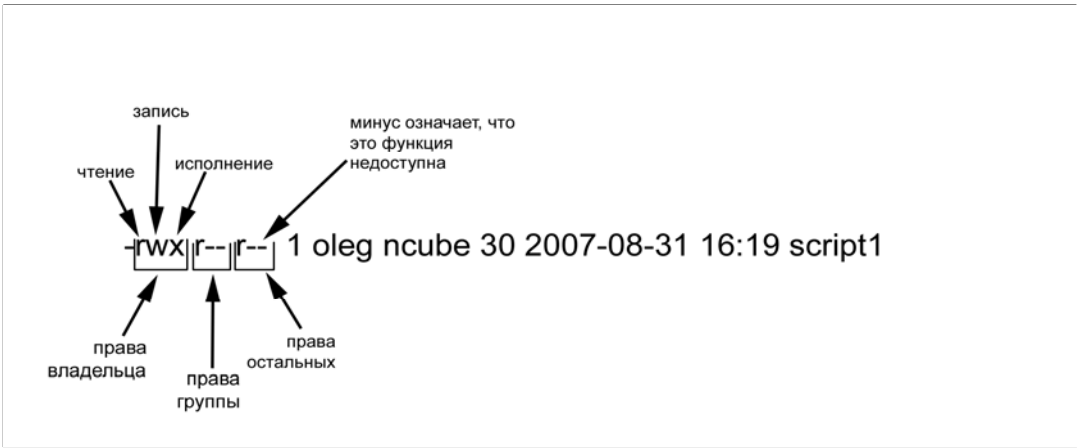


Рис. 3 Права доступа к файлу

Для файлов в UNIX устанавливается три группы прав:

- права собственника файла
- права членов группы, в которую входит собственник
- права для всех остальных пользователей системы.

Для каждой из этих групп назначаются права на чтение "**r**", на запись "**w**", на запуск "**x**". Для директории символ "x", означает, что данная группа пользователей обладает правом входа с помощью команды **cd**. Если какие-то права запрещены, то вместо соответствующей буквы ставится знак "**–**".

chmod [опции] права имена – команда изменения прав доступа к файлу.

права – строковое или цифровое представления прав доступа.

имена – имена файлов или директорий, для которых, необходимо изменить права доступа.

опции – два значения

-f - не прерывать выполнение при ошибке

-R – выполнять рекурсивно

Права для файла можно задавать в виде символьной строки или цифрового кода. При задании прав виде строки права для владельца обозначаются - **u(user)**, группы - **g(group)**, остальные – **o(other)**, или если изменяются права для всех то - **a(all)**. Для разрешения используется знак "+", для, запрещения "**-**".

Примеры:

Чтобы пользователи, входящие в одну группу с вами и все остальные имели возможность изменять в файл *script1* следует ввести команду

chmod g,o+w script1

Команда

chmod a+x script1 – позволит всем пользователям зарегистрированным в системе запускать файл script1 на исполнение

Помимо знаков "+" и "**-**" можно пользоваться знаком "=". Отличие заключается в том, что использование "+" или "–" позволяет регулировать отдельные права, только чтение или запись. При использовании "=" необходимо задавать полный набор прав, т. е. сразу указывать права для чтения, записи, и исполнения.

chmod g=rwx script1 - разрешит пользователям входящим в вашу группу читать, изменять и исполнять файл.

Для того чтобы указать права доступа в цифровом коде, надо задать три восьмеричных числа, которые будут регулировать права для каждого

типа пользователя (владелец, группа, остальные). Каждое из прав соответствуют определенным цифрам **r** (чтение) - 4, **w**(запись, изменение) - 2, **x**(исполнение) - 1. В команде chmod нужно указать сумму, которую образуют права для заданной группы.

Чтобы установить права **rwxr-xr--** для файла scrip1 следует набрать команду:

chmod 754 scritpt1

Другими словами для установки какого-то права нужно задать 1 в соответствующей позиции восьмеричной триады

2.5.2.Команды для работы с каталогами

mkdir [опции] имя_директории ... - создать новые каталоги опции

-m mode − задать права доступа

-p − создавать при необходимости родительские каталоги

rmdir имя_директории . . . - удалить каталоги

Каталоги должны быть пустыми

2.5.3.Команды для работы с файлами

touch [опции] имя_файла

Команда создает файл, если он не существовал или модифицирует время последнего изменения файла.

rm [опции] имя_файла ... – удаление файлов

опции

-i − интерактивное удаление (с требованием подтверждения)

-f − без выдачи сообщений

- r - рекурсивное удаление каталогов вместе с содержимым

Примеры

rm file1 file2 – удаление файлов *file1* и *file2*

rm data – удаление пустой директории.

rm –r data – удаление не пустой директории.

rm /tmp/file1 – удаление файла по полному имени

Для задания списка файлов можно использовать шаблоны, но пользоваться следует крайне осторожно. Команда

rm test* - удалит все файлы с именами, начинающимися на *test*

rm test * - удалит вообще все файла в каталоге

mv [опции] источник назначение – перемещение файлов и директорий

опции

-i – интерактивное перемещение (с требованием подтверждения)

-f – без выдачи сообщений

Команда **mv** выполняет множество функций в зависимости типа аргументов.

1. переименовывает файлы и каталоги, если оба аргумента являются либо файлами либо каталогами:

mv file1 file2 – в рабочем каталоге исчезнет файл *file1* и появится *file2*;

mv dir1 dir2 – в рабочем каталоге исчезнет каталог *dir1* и появится каталог с именем di*r*2 и с тем же самым содержимым;

2. перемещает файл или каталог в другой каталог с тем же именем или другим:

mv file1 dir2 – перемещает *file1* из рабочего каталога в каталог *dir2* с тем же именем;

mv file1 dir2/file2 – перемещает *file1* из рабочего каталога в каталог *dir2* с именем *file2*;

3. если источником является список файлов, а назначением является каталог, то можно использовать шаблоны:

mv file* ../dir2 – перемещает все файлы, имена которых начанается со строки *file*, в каталог одного уровня с рабочим;

Во всех операциях, объекты выступающие в качестве аргумента источник исчезают.

cp [опции] источник назначение– копирование файлов и директорий.

опции

-i – интерактивное копирование (с требованием подтверждения, если объект **назначение** уже существует)

-f – без выдачи сообщений

-r – рекурсивное копирование каталогов вместе с содержимым

-p – копирование с сохранением атрибутов файлов (прав доступа, времени модификации)

cp file1 file2 – будет создана копия файла *file1* в файле с именем *file2*;

cp file1 dir2 – будет создана копия файла *file1* в каталоге *dir2*;

cp -r dir1 dir2 – будет создана копия каталога *dir1* в каталоге *dir2*;

cp file1 file2 file3 /tmp – копирует файлы с именами *file1*, *file2*, *file3* в директорию tmp корневого каталога. Это можно выполнить командой:

cp file* /tmp

cat [опции] [файл][файл]...

Команда **cat** объединяет файлы и выдает их на стандартный поток вывода. Если аргумент файл отсутствует, то команды **cat** будет принимать входной поток из стандартного файла ввода (клавиатуры). Поскольку команда работает со стандартным файлом вывода (терминалом), то чаще всего она используется для просмотра на экране содержимого файла. Не рекомендуется выдавать бинарные файлы.

cat ls.txt – выводит содержимое файла с имененм ls.txt на терминал;

cat ls1.txt ls2.txt ls3.txt – по очереди выводит на терминал содержимое файлов *ls1.txt, ls2.txt, ls3.txt*.

cat ls1.txt ls2.txt ls3.txt > lsall.txt – объединяет "сливает" три файла в один. При этом старые файлы сохраняются. Если файл *lsall.txt* уже существовал, то он затрется новым содержимым. Можно дописать в конец файла, если использовать для перенаправления знак >> .

Команду cat можно использовать для создания файла:

cat > ls.txt – все набранное на клавиатуре будет записано в файл *ls.txt*. Оборвать ввод можно сочетанием клавиш Ctrl-D.

Команда **cat** выдает все содержимое на экран одной выдачей. Если файл большой, то на экране можно будет увидеть только последние строки. Для выдачи порциями следует использовать конвейер:

cat file1| more .

Для просмотра порциями текстовых файлов можно напрямую использовать команды:

more file.txt

less file.txt

Команда **less** содержит большой набор внутренних команд для перемещения по файлу, поиска контекста и даже редактирования:

пробел – перемещение вперед на один экран;

Return – перемещение вперед на одну строку;

Ctrl-b - перемещение назад на один экран;

g – переход в начало файла, #g – переход на заданную строку;

G – переход в конец файла;

h – доступных комаанд.

tail [опции] файл – просмотр конца файла. По умолчанию 10 последних строк. С помощью опций можно начать просмотр с любой позиции.

опции

-n number − просмотр с указанной строки

-r number − отображение в обратном порядке

-f − непрерывная выдача файла по мере его заполнения

Прерывание интерактивной выдачи комбинацией *Ctrl c*.

grep [опции] строка [файл][файл]... − поиск контекста “строка” в указанных файлах.

опции

-i − поиск без учета регистра

-n − отображать номера строк, содержащих контекст

-v − отображать строки, не содержащие контекст.

find [опции] каталог выражение – рекурсивный поиск файлов в указанном каталоге по различным атрибутам, таким как имя, размер, время модификации, права доступа.

Выражения:

-name filename - поиск файла с именем filename. Возможно использовние шаблонов, но тогда их надо брать в кавычки “test*”;

-size [+|-]Number – поиск файлов с заданным размером, превышающим его +, или меньшим - . Размер в блоках 512 байт;

-atime number − поиск файлов, к которым происходил доступ number суток назад;

-mtime number − поиск файлов, которые были модифицированы number суток назад;

-exec command {} \; − выполнить команду **command** над списком файлов, найденых командой **find**.

Пример:

find . -name “core.*” -exec rm {} \; − рекурсивно удалить все *core* файлы, начиная с текущего каталога.

Следует отметить, что многие действия, из перечисленных выше и связанных с манипуляциями с каталогами и файлами можно выполнять с помощью специальной программы – файлового менеджера *Midnight*

Commander. Он не требует графической оболочки, вызывается в терминальном окне командой

mc

С помощью этой программы можно перемещаться по дереву каталогов, просматривать содержимое каталогов и файлов, создавать каталоги (но не файлы), удалять, копировать, перемещать каталоги и файлы, вести поиск файлов, редактировать файлы. Для тех пользователей, которые не хотят осваивать и использовать редактор **vi** – эта программа является очень хорошим подспорьем.

Для эффективного использования дискового пространства и для уменьшения объемов пересылаемых данных рекомендуется использовать команды архивирования и сжатия. Для архивирования удобнее всего использовать команду **tar**, а для сжатия файла коман **gzip** или **bzip.** Последние версии команды tar позволяют совместить выполнение архивации и сжатия.

tar [опции] [файл] [файл]...

опции (основные)

-c – создание нового архива;

-r – добавление файлов в архив;

-t – просмотр содержимого архива;

-u – добавляются только обновленные файлы;

-x – извлечение файлов из архива;

-z – использовать в качестве фильтра команду **gzip**;

-j – использовать в качестве фильтра команду **bzip2**;

-z – использовать в качестве фильтра команду **gzip**;

Пример:

tar cvzf dir1.tar.gz dir1

В результате выполнения команды будет создан архивный файл **dir1.tar.gz**, в который будет помещено все содержимое каталога **dir1.** Архивный файл будет сжат командой **gzip**. После этого каталог *dir1* может быть удален.

Для распаковки архива следует использовать команду:

tar xvzf dir1.tar.gz

которая восстановит каталог **dir1**, при этом архивный файл не уничтожается автоматически. Напрямую работать с архивом как с директорией позволяет файловый менеджер *Midnigth Commander* (**mc**).

2.5.4.Команды управления процессами.

ps [опции] – выводит текущую информацию о процессах в системе.

опции

-e – выводит информацию о всех процессах в системе

-f – подробный вывод.

-e – выводит подробную информацию о всех процессах, кроме ядра

команда **ps**, выполненная без опций, выводит информацию только о процессах запущенных с терминала. Информация будет представлена виде четырех столбцов.

PID	номер процесса
TTY	имя терминала
TIME	используемое время процессора, в минутах и секундах
CMD	имя команды породивший этот процесс

Если использовать флаг **–f** , то информация будет представлена более подробно в виде восьми столбцов.

uname	имя пользователя породившего процесс
PID	номер процесса
PPID	номер процесса родителя
C	данные об использовании процесса(планировщик)
STIME	время создания процесса
TTY	имя терминала
TIME	используемое время процессора, в минутах и секундах
CMD	имя команды породивший этот процесс

Вторая, в большинстве случаев, более удобная информационная команда – это команда **top**. Она позволяет отслеживать наиболее активные процессы. С помощью этой команды можно проследить, какие процессы вызывают неадекватную загрузку системы, и удалить их командой **kill**.

kill сигнал **ид_процесса** – команда посылает процессу один из допустимых сигналов. Список допустимых сигналов можно посмотреть командой:

kill –l

Самый сильный сигнал завершения процесса – **KILL** или **-9**. Команда

kill -9 id_number – немедленно завершит процесс.

2.5.5.Команды управления окружением

Любая команда, запускаемая пользователем - будь то системная команда, компилятор или программа пользователя, выполняется в окружении, сформированном оболочкой. Это окружение формируется с помощью конфигурационных файлов. Механизм формирования окружения заключается, главным образом, в установке правильных значений для переменных окружения.

К сожалению, в этом вопросе в *UNIX*-подобных системах единого стандарта нет.

В каждую версию системы включается несколько командных интерпретаторов и пользователю предлагается самому выбирать ту или иную оболочку. С точки зрения выполнения команд это никак не проявляется – любая внешняя команда или программа пользователя будет одинаковым образом выполняться в любой оболочке. А вот с точки зрения формирования среды и внутренних команд среды различия имеются. Ранее упоминалось, что в UNIX-подобных системах поддерживаются два семейства *Shell*: *Bourne Shell (sh, ksh, bash)* и *C Shell (csh, zsh, tcsh)*.

Оболочки имеют немного отличающийся синтаксис встроенных команд и используют различные конфигурационные файлы. Наибольшее распространение на сегодняшний день получили *tcsh* и *bash*, ставший фактически стандартным в *Linux* системах.

2.5.6.Конфигурационные файлы:

bash - /etc/profile, ~/bash_profile, ~/bash_login, ~/.profile, ~/.bashrc

tcsh - /etc/csh.cshrc /etc/csh.login, ~/.login, ~/.cshrc ~/.tcshrc,

Таким образом, видно, что часть установок делает системный администратор, а часть каждый пользователь выполняет сам. Необходимые установки можно делать в любом из перечисленных файлах, но при этом следует иметь в виду, что некоторые конфигурационные файлы выполняются только один раз - при входе в систему, а другие каждый раз при запуске нового *shell*.

В оболочке *bash* действует следующий сценарий:

при интерактивном запуске выполняется файл */etc/profile*, а затем ищутся файлы в следующей последовательности *~/bash_profile*, *~/bash_login* *~/.profile* и на исполнение запускается первый из найденных;

при неинтерактивном выполняется только *~/.bashrc*

В оболочке *tcsh* при интерактивном запуске выполняются */etc/csh.cshrc* */etc/csh.login*, а затем ищутся файлы *~/.tcshrc*, *~/.cshrc* и выполняется первый

из найденых. Затем исполняется *~/.login*. При не интерактивном запуске выполняются /*etc/csh.cshrc* и один из *~/.tcshr*c или *~/.cshrc*

Переменные среды, устанавливаемые всеми оболочками, в основном совпадают, однако в каждой из оболочек имеются переменные, которые отсутствуют в других. Для того, чтобы посмотреть все установленные переменные среды служит команда:

env

Эта команда не является встроенной командой оболочки, поэтому она работает во всех оболочках. Перечислим наиболее важные переменные среды, которые могут потребовать корректировки:

TERM – тип терминала (например xterm, vt100). Требуется для правильной работы терминальных редакторов;

SHELL – оболочка, устанавливаемая при входе;

PATH – содержит список директорий, в которых ищутся команды. В том числе и программы пользователя. Например *PATH=/bin:/usr/bin:/usr/local/bin:~/bin:.:*

Просмотр слева направо, исполняется первая найденная команда. Каталог *~/bin* указывает на то, что команды(программы) следует искать в пользовательском каталоге *bin*, а . на то, что поиск выполнять в текущем каталоге. Проверить, какая команда будет исполнена можно с помощью команды: **which** команда

LD_LIBRARY_PATH - список директорий, в которых ищутся динамические библиотеки, подключаемые во время выполнения программы.

Пример: LD*_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:~/lib*

MANPATH – путь поиска man-страниц. Важна для правильной работы команды man команда, которая позволяет получить описание любой команды. Если команда есть, а описание по ней не выдается – значит неправильно установлена эта переменная.

DISPLAY – переменная, указывающая экран, для выдачи графической информации.

Примеры:

DISPLAY=:0.0 - если пользователь работает на консоли компьютера;

DISPLAY=имя_компьютера:0.0 или IP_adress:0.0 – если пользователь работает с удаленного компьютера. Здесь *имя_компьютера* – это имя того компьютера, с которого работает пользователь (или IP адрес).

В оболочке *tcsh* имеется переменная *REMOTEHOST*, которая позволяет автоматически устанавливать правильное значение для переменной *DISPLAY* при подключении с удаленного компьютера. В оболочке *bash* такой

переменной, к сожалению, нет и для того, чтобы работать с графическими приложениями необходимо устанавливать эту переменную.

Некоторые приложения требуют установки своих переменных. Например, на многопроцессорных(многоядерных) системах для выполнения распараллеленных с помощью *OpenMP* программ требуется задание переменной *OMP_NUM_THREADS*, указывающую максимальное число потоков, которое можно запускать на системе.

Команда **env** выдает довольно большой объем информации. Для того, чтобы посмотреть значение конкретной переменной можно воспользоваться командой:

echo $ИМЯ_ПЕРЕМЕННОЙ - здесь символ **$** означает извлечение значения переменной.

Установка переменных окружения по-разному выполняется в *bash* и *tcsh*.

Bash:

VAR=value - присваивание переменной VAR значения value;

export VAR – экспорт переменной в глобальное окружение.

Эти две операции можно совместить:

export VAR=value

Внимание! Около знака равенства не должно быть пробелов.

Пример:

export PATH=$PATH:/usr/local/mpi/bin – в переменную *PATH* добавили директорию с командами *MPI (/usr/local/mpi/bin*).

Tcsh:

setenv VAR value - установка переменной выполняется специальной командой.

Команда **setenv** устанавливает переменные окружения или глобальные переменные. В *tcsh* помимо переменных окружения имеется набор внутренних переменных оболочки. Они устанавливаются командой **set** .

Примеры:

setenv PATH $PATH:/usr/local/mpi/bin – добавление в переменную окружения *PATH* каталога /*usr/local/mpi/bin*

set prompt="`/usr/bin/uname -n`:`pwd`->% " -

установка переменной *prompt* , определяющей вид приглашения оболочки в *tcsh*: *rsuib:/export/home/victor->%*

2.6.Неинтерактивные редакторы.

Неинтерактивные текстовые редакторы, предназначены для пакетного редактирования файлов. Неинтерактивные редакторы широко распространены в *UNIX-подобных* системах, где используются для решения множество самых разнообразных задач. Основные функции, неинтерактивных редакторов в *UNIX* системах:

- Редактирование больших текстовых файлов
- Редактирование большого количества файлов.
- Редактирование вывода команд.

Для работы и выборки текста неинтерактивные редакторы используют так называемые регулярные выражения. Регулярные выражения (англ. *regular expressions*, сокр. *RegExp*, *RegEx*, жарг. регэ́кспы или ре́гексы) — способ синтаксического разбора текстовых фрагментов по формализованному шаблону. Образец (англ. *pattern*), задающий правило поиска, по-русски также иногда называют «*шаблоном*», «*маской*». Шаблоны являются развитием символов-джокеров (англ. *wildcard characters*).

Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста. Многие языки программирования имеют инструменты для работы с регулярными выражениями. Например, *Java*, *.NET Framework, Perl, PHP, JavaScript, Python, Tcl, Ruby* и др. Набор утилит (включая редактор *sed* и фильтр *grep*), поставляемых в дистрибутивах *UNIX*, одним из первых способствовал популяризации понятия регулярных выражений.

2.6.1.Регулярные выражения.

Выражение - это набор символов. Символы, которые имеют особое назначение, называются *метасимволами*. Регулярные выражения - это набор символов и/или метасимволов, которые имеют специальное значение. Основная функция регулярных выражений поиск текста по шаблону и работа со строками.

- Звездочка * означает любое количество символов в строке, предшествующих "звездочке", в том числе и нулевое число символов. Выражение "1133*" -- означает 11 + один или более символов "3" + любые другие символы: 113, 1133, 113312, и так далее.
- Точка . означает не менее одного любого символа, за исключением символа перевода строки (\n). Выражение "13." будет означать 13 + по меньшей мере один любой символ (включая пробел): 1133, 11333, но не 13 (отсутствуют дополнительные символы).

- Символ ^ означает начало строки, но иногда, в зависимости от контекста, означает отрицание в регулярных выражениях.
- Знак доллара **$** в конце регулярного выражения соответствует концу строки. Сочетание **^$** означает пустую строку. Символы **^$** иногда называют якорями, т.к. они задают место регулярного выражение в строке.
- Квадратные скобки **[...]** предназначены для задания подмножества символов. Квадратные скобки, внутри регулярного выражения, считаются одним символом, который может принимать значения, перечисленные внутри этих скобок.
    - Выражение "[xyz]" -- соответствует одному из символов *x*, *y* или *z*.
    - Выражение "[c-n]" соответствует одному из символов в диапазоне от *c* до *n*, включительно.
    - Выражение "[B-Pk-y]" соответствует одному из символов в диапазоне от *B* до *P* или в диапазоне от *k* до *y*, включительно.
    - Выражение "[a-z0-9]" соответствует одному из символов латиницы в нижнем регистре или цифре.
    - Выражение "[^b-d]" соответствует любому символу, кроме символов из диапазона от *b* до *d*, включительно. В данном случае, метасимвол ^ означает отрицание.
    - Объединяя квадратные скобки в одну последовательность, можно задать шаблон искомого слова. Так, выражение "[Yy][Ee][Ss]" соответствует словам *yes*, *Yes*, *YES*, *yEs* и так далее
- Обратный слэш \ служит для экранирования специальных символов, это означает, что экранированные символы должны интерпретироваться буквально, т.е. как простые символы. Комбинация "**\$**" указывает на то, что символ "**$**" трактуется как обычный символ, а не как признак конца строки в регулярных выражениях. Аналогично, комбинация "\\" соответствует простому символу "\".
- Экранированные "угловые скобки" - **\<...\>** -- отмечают границы слова. Угловые скобки должны экранироваться, иначе они будут интерпретироваться как простые символы. Выражение "**\<the\>**" соответствует слову "*the*", и не соответствует словам "*them*", "*there*", "*other*" и т.п.
- Знак вопроса **?** означает, что предыдущий символ или регулярное выражение встречается 0 или 1 раз. В основном используется для поиска одиночных символов.
- Знак "плюс" + указывает на то, что предыдущий символ или выражение встречается 1 или более раз. Играет ту же роль, что и символ "звездочка" (*), за исключением случая нулевого количества вхождений.

- Экранированные "фигурные скобки" - \{ \} - задают число вхождений предыдущего выражения. Экранирование фигурных скобок - обязательное условие, иначе они будут интерпретироваться как простые символы. Такой порядок использования, технически, не является частью основного набора правил построения регулярных выражений. Выражение **"[0-9]\{5\}**" - в точности соответствует подстроке из пяти десятичных цифр (символов из диапазона от 0 до 9, включительно). В "классической" (не совместимой с *POSIX*) версии **awk**, фигурные скобки не могут быть использованы. Однако, в **gawk** предусмотрен ключ *--re-interval*, который позволяет использовать (неэкранированные) фигурные скобки.
- Круглые скобки **( )** предназначены для выделения групп регулярных выражений.
- Вертикальная черта | выполняет роль логического оператора "ИЛИ" в регулярных выражениях и служит для задания набора альтернатив.

  Классы символов **POSIX. [:class:]**
- Это альтернативный способ указания диапазона символов.
- Класс **[:alnum:]** - соответствует алфавитным символам и цифрам. Эквивалентно выражению *A-Za-z0-9*.
- Класс **[:alpha:]** - соответствует символам алфавита. Эквивалентно выражению *A-Za-z*.
- Класс **[:blank:]** - соответствует символу пробела или символу табуляции.
- Класс **[:cntrl:]** - соответствует управляющим символам (*control characters*).
- Класс **[:digit:]** - соответствует набору десятичных цифр. Эквивалентно выражению 0-9.
- Класс **[:graph:]** (печатаемые и псевдографические символы) - соответствует набору символов из диапазона *ASCII* 33 - 126. Это то же самое, что и класс **[:print:]**, за исключением символа пробела.
- Класс **[:lower:]** - соответствует набору алфавитных символов в нижнем регистре. Эквивалентно выражению *a-z.*
- Класс **[:print:]** (печатаемые символы) -- соответствует набору символов из диапазона *ASCII* 32 - 126. По своему составу этот класс идентичен классу **[:graph:]**, описанному выше, за исключением того, что в этом классе дополнительно присутствует символ пробела.

- Класс **[:space:]** - соответствует пробельным символам (пробел и горизонтальная табуляция).
- Класс **[:upper:]** - соответствует набору символов алфавита в верхнем регистре. Эквивалентно выражению *A-Z*.
- Класс **[:xdigit:]** - соответствует набору шестнадцатеричных цифр. Эквивалентно выражению *0-9A-Fa-f*.

2.6.2.Редактор sed.

Sed - это неинтерактивный строчный редактор. Он принимает текст либо входного потока, либо из текстового файла, выполняет некоторые операции над строками и затем выводит результат в выходной поток или в файл. Обычно, в сценариях, sed используется в конвейерной обработке данных, совместно с другими командами и утилитами.

Над какими строками следует выполнять операции sed определяет, по заданному адресному пространству. Адресное пространство строк задается либо их порядковыми номерами, либо шаблоном. Например, команда 3d заставит sed удалить третью строку, а команда **/windows/d** означает, что все строки, содержащие "*windows*", должны быть удалены. Если в качестве номера строки используется знак **$**, то он имеет значения последней строки в файле. Выражение **1,$ d** означает удалить все строки в файле.

Командная строка редактора sed

[адрес1[,адрес2]] функция [аргументы]

Основные операции sed

Операция	Название	Описание
[адрес1[,адрес2]] a \Текст	append	Добавить "`Текст`" после указанной строки.
[адрес1[,адрес2]] i \Текст	insert	Вставить строку содержащую “Текст” перед указанной строкой
[адрес1[,адрес2]] c \Текст	change	Замена строк, на строку “Текст”
[адрес1[,адрес2]] /p	print	Печать [указанного диапазона строк]
[адрес1[,адрес2]] /d	delete	Удалить [указанный диапазон строк]
s/pattern1/pattern2/	substitute	Заменить первое встреченное соответствие шаблону pattern1, в строке, на pattern2
[адрес1[,адрес2]] /s/pattern1/pattern2/	substitute	Заменить первое встреченное соответствие шаблону pattern1, на pattern2, в указанном диапазоне строк
[адрес1[,адрес2]] /y/pattern1/pattern2/	transform	заменить любые символы из шаблона pattern1 на соответствующие символы из pattern2, в указанном диапазоне строк (эквивалент команды tr)

Примеры операций в sed

Операция	**Описание**
`8d`	Удалить 8-ю строку.
`/^$/d`	Удалить все пустые строки.

`1,/^$/d`	Удалить все строки до первой пустой строки, включительно.
`/Jones/p`	Вывести строки, содержащие "Jones" (с ключом -n).
`s/Windows/Linux/`	В каждой строке, заменить первое встретившееся слово "Windows" на слово "Linux".
`s/BSOD/stability/g`	В каждой строке, заменить все встретившиеся слова "BSOD" на "stability". g Операция выполняется над всеми найденными соответствиями внутри каждой из заданных строк(Без оператора g (global), операция замены будет производиться только для первого найденого совпадения, с заданным шаблоном, в каждой строке.)
`s/ *$//`	Удалить все пробелы в конце каждой строки.
`s/00*/0/g`	Заменить все последовательности ведущих нулей одним символом "0".
`/GUI/d`	Удалить все строки, содержащие "GUI".
`s/GUI//g`	Удалить все найденые "GUI", оставляя остальную часть строки без изменений.

2.6.3.Редактор шаблонов awk.

Awk - это полноценный язык обработки текстовой информации с Си подобным синтаксисом. Он обладает широкими возможностями в обработке текста, однако, остановимся лишь на некоторых из них - наиболее часто встречающихся при написании сценариев командной оболочки.

Awk рассматривает входной поток как список записей. **Awk** "разбивает" каждую запись на отдельные поля. По-умолчанию, записи отделяются друг от друга символом новой строки(запись это строка), поля - это последовательности символов, отделенные друг от друга пробелами табуляцией, однако имеется возможность назначения других символов, в качестве разделителя полей. **Awk** анализирует и обрабатывает каждое поле в отдельности. Это делает его идеальным инструментом для работы со структурированными текстовыми файлами.

Внутри сценариев командной оболочки, код **awk**, заключается в "строгие" (одиночные) кавычки и фигурные скобки.

AWK-программа состоит из операторов (правил), имеющих вид:

BEGIN {действие}

шаблон {действие}

шаблон {действие}

. . .

END {действие}

. . .

Каждая запись поочерёдно сравнивается со всеми шаблонами, и каждый раз, когда она соответствует шаблону, выполняется указанное действие. Если *шаблон* не указан, то *действие* выполняется для любой записи. Если не указано *действие*, то запись выводится. В **AWK** также

существует 2 предопределённых шаблона **BEGIN** и **END**. **BEGIN** выполняется до начала обработки входного потока. **END** — после обработки последней записи входного потока.

Действие в **awk** - последовательность операторов. Оператор есть одна из конструкций:

if (условие) оператор [else оператор]

while (условие) оператор

for (выражение; условие; выражение) оператор

break

continue

{ [оператор] ... }

переменная = выражение

print [список_выражений] [> выражение]

printf формат [, список_выражений] [> выражение]

next # пропустить оставшиеся шаблоны и перейти к следующей строке

exit # пропустить оставшиеся строки

Это далеко не полный список операторов возможных в **awk** программе. Операторы завершаются точкой с запятой, переводом строки или правой скобкой.

Выражения строятся из цепочек символов и чисел с помощью операций **+, -, *, /, %** и конкатенации (обозначается пробелом). В выражениях также можно использовать операции из языка

C: ++, --, +=, -=, *=, /=, %=.

Сравнения чисел, если оба числа, иначе - строк **<, <=, ==, !=, >=, >**,

Логические операции !, ||, &&

Операция "пробел"конкатенация.

Принадлежность:

~ - Содержится;

!~ - Не содержится.

Например:

$1 ~ /[Oo]lga/ - Указывает на строки, первое поле которых содержит Olga или olga.

$1 > = "s" - Указывает на строки, начинающиеся с символа s или следующих за ним по порядку: t, u, v...

Переменные инициализируются пустыми цепочками. Переменные могут интерпретироваться как числовые или строковые. Они принимают значения в зависимости от контекста, например:

x = 1;, x число;

x = " ";, x - строка;

Если вы хотите в скрипте **awk** использовать значение поля то применяются следующая конструкция **$1, $2**, ..., получившая название переменные поля.

$(i+1) - интерпретируется как поле, номер которого зависит от значения переменной **i**.

Допускается использование массивов. Массивы не объявляются, а принимают значения из контекста, например:

x[NR] = $0 - элементу массива **x**, индексированному **NR**, присваивается обрабатываемая строка.

x["apple"] - элементы массива могут индексироваться не числовым значением, т.е. строкой.

Специальные переменные

NF	Количество полей в текущей записи.
NR	Порядковый номер текущей записи.
FILENAME	Имя файла, из которого в данный момент производится ввод.

Встроенные функции

sin (expr)	синус expr
cos (expr)	косинус expr
exp (expr)	возведение в степень expr
log (expr)	натуральный логорифм expr
sqrt (expr)	извлечение корня expr
int (expr)	целая часть числа
length (s)	длина строки s
printf (fmt, ...)	форматирование (аналогично Си) по спецификации fmt.
substr (s, m, n)	подстрока в n символов строки s, начинающаяся с m.
getline ()	чтение следующей строки.
0	конец файла, иначе 1.
index (s1, s2)	номер позиции, с которой s1 совпадает с s2, иначе 0.
split (s, M, c)	строка s разбивается элементы массива M по разделителю c (по умолчанию FS=" "); функция возвращает число полей.

Вызов **awk**. Возможны два основных варианта:

awk [-Fc] 'prog.awk' [files]

Это простейший случай, когда программа (заключенная в кавычки " ' ") находится в теле команды, **"-Fc"** - флаг, меняющий стандартный разделитель

полей на "**c**" "**file**" - имя файла исходных данных, при его отсутствии - со стандартного входа. (Этот формат использован в начальных примерах).

cat имя файла | awk '/до/ {print}'

и

awk '/до/ {print}' < f-awk

awk [-Fc] -f prog.awk [files]

Флаг "**-f**" говорит о том, что **awk**-программу надо брать из файла, имя которого указано следом (имя может быть произвольным и расширение ".**awk**" добавлено здесь просто из эстетических соображений).

Пример

awk–F:'{print$1}' /etc/passwd - команда выводит первое поле из файла паролей. Первое поле файла паролей в UNIX содержит имя пользователя в системе и называется логин(login).

Создадим текстовый файл с именем *employees* и следующим содержанием.

409,John Baker,56000,civil engineering
678,Fred Smith,73000,physics
429,Julia Tanguay,47000,computer science
349,Peggy Bantin,67000,physics
268,Mario Hodgkins,55000,programming

В качестве разделителя в строке используется запятая. Если произвести обработку строк данного файла, с помощью **awk**, указав ему предварительно, что, символом разделителем является запятая, получим например, для последней строки:

Поля	268	Mario Hodkins	55000	programming
Параметрические переменные	$1	$2	$3	$4

Распечатка всех строк содержащих слова работников преподающих на факультете физики

awk -F, '/physics/ {print}' employees

Вывести имя и факультет где работает данный человек.

awk -F, '{print $2, $4}' employees

Вывести имя если оклад равен 55000

awk -F, '$3 == 55000' employees

Вывести работников физического факультета и программистов.

awk -F, '/physics|programming/ {print}' employees

Вывести записи о работниках у которых зарплата выше 55000 и они трудятся в департаменте физики.

awk '$3 > 55000 && $4 == "physics" ' employees

Вывести записи о всех работниках у которых зарплата больше 55000 и меньше 70000

awk -F, '$3 > 55000 && $3 < 70000' employees

Программа на языке awk выводящая список работников всех департаментов и их заработную плату, а также в конце выводится сумма расходов на зарплату для всех сотрудников.

```
awk -F, '
BEGIN {
  print " The name and salary of each employee is: "
   }
{
    name = $2
    salary = $3
   saltotal + = salary
    print name, salary
}
END {
  print "Total salaries for this dept are " saltotal
    }
' employees
```

2.7.Редактор vi.

Редактор vi - универсальный полноэкранный текстовый редактор в среде UNIX. Универсальность означает, что, во-первых, этот редактор есть во всех UNIX-подобных ОС и, во-вторых, этот редактор работает с практически любым видом терминала. Подобная универсальность обернулась несколько непривычным (для пользователей DOS) пользовательским интерфейсом: для управления редактором используются лишь ``обычные'' кнопки клавиатуры (алфавитно-цифровые символы и знаки препинания).

Командный режим	Когда вы запустите редактор **vi**, он будет работать в командном режиме. Вы можете ввести любую подкоманду, за исключением тех, которые предназначены только для режима ввода текста. Редактор возвращается в командный режим, когда завершается работа подкоманд и других режимов. Чтобы отменить выполнение подкоманды, нажмите клавишу **Esc**.

Режим ввода текста	Этот режим используется в редакторе **vi** для добавления текста. Вход в режим ввода текста осуществляется при помощи любой из следующих подкоманд: **a, A,i, I, o, O, cx** (где **x** – область действия подкоманды), **C,s, S** или **R**. После ввода одной из этих подкоманд вы можете вводить текст в буфер редактирования. Чтобы вернуться в командный режим, нажмите клавишу **Esc** (нормальный выход) или клавишу прерывания(**Ctrl-C**) (непредвиденный выход).
Режим последней строки	Подкоманды с префиксами : (двоеточие), / (слэш),? (вопросительный знак), ! (восклицательный знак) или !! (два восклицательных знака) читают строку, отображаемую в нижней части экрана. Когда вы вводите первый символ, редактор **vi** помещает курсор в нижнюю часть экрана, где вы можете ввести остальные символы команды. Нажмите клавишу **Enter**, чтобы запустить подкоманду, или нажмите клавишу прерывания (**Ctrl-C**) для отмены. Если используется префикс (!!), то курсор перемещается только после ввода обоих восклицательных знаков. Если для входа в режим последней строки используется двоеточие (:), то редактор **vi** придает специальное значение следующим символам, если они используются до команд, указывающих количества: % Все строки, независимо от положения курсора. $ Последняя строка - Текущая строка

Список подкоманд весьма обширен, поэтому мы изучим только некоторые наиболее распространенные подкоманды. Эти подкоманды используются для выполнения следующих действий:

- Перемещение курсора
- Редактирование текста
- Манипуляции с файлами
- Другие действия

Следующие подкоманды вводятся в командном режиме. Вы можете отменить работу команды до ее завершения, нажав клавишу Esc.

Перемещение курсора.	
Перемещение в пределах строки	
Стрелка влево, h или Ctrl-	Перемещение курсора на один символ влево.

H	
Стрелка вниз, j, Ctrl-J или Ctrl-N	Перемещение курсора на одну строку вниз (он остается в том же столбце).
Стрелка вверх, k или Ctrl-P	Перемещение курсора на одну строку вверх (он остается в том же столбце).
Стрелка вправо или I	Перемещение курсора на один символ вправо.
Перемещение в пределах строки к определенному месту строки.	
^	Перемещение курсора к первому непустому символу.
0	Перемещение курсора к началу строки.
$	Перемещение курсора к концу строки.
fx	Перемещение курсора к следующему символу x.
Fx	Перемещение курсора к последнему символу x.
tx	Перемещение курсора к столбцу, предшествующему следующему символу x.
Tx	Перемещение курсора к столбцу, идущему после следующего символа x.
;	Повторить последнюю подкоманду f, F, t или T
,	Повторить последнюю подкоманду f, F, t или T в противоположном направлении.
Перемещение к определенному слову.	
w	Перемещение курсора к следующему маленькому слову.
b	Перемещение курсора к предыдущему маленькому слову.
e	Перемещение курсора к концу маленького слова.
W	Перемещение курсора к следующему большому слову.
B	Перемещение курсора к предыдущему большому слову.
E	Перемещение курсора к концу большого слова.
Перемещение к определенной строке	
H	Перемещение курсора к верхней строке на экране.
L	Перемещение курсора к последней строке на экране.
M	Перемещение курсора к средней строке на экране.

G	Перемещение курсора к последней строке файла.
1G	Перемещение курсора к первой строке файла.
nG	Перемещение курсора к n-й строке файла.
+	Перемещение курсора к первому непустому символу следующей строки.
-	Перемещение курсора к первому непустому символу предыдущей строки.
Перемещение к определенному предложению, абзацу или разделу.	
(	Перемещение курсора в начало предыдущего предложения.
)	Перемещение курсора в начало следующего предложения.
}	Перемещение курсора в начало следующего абзаца или следующего раздела, если вы используете режим C.
]]	Перемещение курсора к следующему разделу.
[[	Перемещение курсора к предыдущему разделу.
Редактирование текста	
Метки	
′′	Перемещение курсора к предыдущему местоположению текущей строки.
′	Перемещение курсора в начало строки, содержащей предыдущее местоположение текущей строки.
mx	Маркирование текущей позиции буквой, указанной в параметре **x**.
`x	Перемещение курсора к месту, маркированному меткой, указанной параметром **x**.
'x	Перемещение курсора в начало строки, содержащей метку, указанную в параметре **x**.
Добавление текста в файл.	
aТекст	Вставка текста, указанного параметром **Текст**, после курсора. Выйдите из режима ввода текста, нажав клавишу **Esc**.
ATекст	Вставка текста, указанного параметром **Текст**, в конец строки. Выйдите из режима ввода текста, нажав клавишу **Esc**.
iТекст	Вставка текста, указанного параметром **Текст**, перед курсором. Выйдите из режима ввода текста, нажав клавишу **Esc**.
IТекст	Вставка текста, указанного параметром **Текст**, перед первым непустым символом строки.

	Выйдите из режима ввода текста, нажав клавишу **Esc**.
o	После текущей строки добавляется пустая строка. Выйдите из режима ввода текста, нажав клавишу **Esc**.
O	Перед текущей строкой добавляется пустая строка. Выйдите из режима ввода текста, нажав клавишу **Esc**.
Изменение текста в режиме ввода	
Ctrl-D	Возврат к предыдущей точке автоматического отступа.
^ Ctrl-D	Отмена автоматического отступа только для текущей строки.
0Ctrl-D	Возврат курсора к левому полю.
Esc	Завершение вставки и возврат в командный режим.
Ctrl-H	Удаление последнего символа.
Ctrl-Q	Ввод любого символа, если отключен режим xon.
Ctrl-V	Ввод любого символа.
Ctrl-W	Удаление последнего маленького слова.
\	Отмена особого смысла (квотирование) символов **erase** и **kill**.
Ctrl-?	Прерывание и завершение вставки или последовательности **Ctrl-D**.
Изменение текста в командном режиме	
C	Изменение оставшейся части строки (то же, что **c$**).
cc	Изменение строки.
cw	Изменение слова.
cwТекст	Изменение слова на текст, указанный параметром **Текст**.
D	Удаление оставшейся части строки (то же, что **d$**).
dd	Удаление строки.
dw	Удаление слова.
J	Соединение строк.
rx	Текущий символ заменяется символом, указанным в параметре **x**.
RТекст	Замена символов текстом, указанным в параметре Текст.

s	Замена символов (то же, что **cl**).
S	Замена строк (то же, что **cc**).
u	Отмена предыдущего изменения.
x	Удаление символа под курсором.
X	Удаление символа перед курсором.
<<	Сдвиг одной строки влево.
>L	Сдвиг всех строк от курсора до конца экрана влево.
~	Перевод буквы под курсором в противоположный регистр.
Копирование и перемещение текста	
p	Копирование текста из буфера обмена в позицию после курсора.
P	Копирование текста из буфера обмена в позицию до курсора.
″xp	Копирование текста из буфера обмена **x**.
«xd	Копирование текста в буфер обмена **x** (с удалением текста)
y	Помещает следующий объект (например, **w** – слово) в буфер обмена.
″xy	Помещает следующий объект в буфер **x**, где **x** – любая буква.
Y	Помещает строку в буфер обмена.
Восстановление и повторное применение изменений	
u	Откат последнего изменения.
U	Восстановление текущей строки, если курсор не покидал строку с момента последнего изменения.
.	Повтор последнего изменения или инкремент команды «np.
″np	Извлекается n-ое по счету удаление целой строки или блока строк.
Манипуляции с файлами	
Сохранение изменений в файле	
:w	Содержимое буфера редактирования записывается в исходный файл.
:w Файл	Содержимое буфера редактирования записывается в файл, указанный параметром

	Файл.
:w! Файл	Содержимое буфера редактирования перезаписывает содержимое файла, указанного параметром Файл.
Редактирование второго файла	
:e Файл	Редактирование указанного файла.
:e!	Повторное редактирование файла с отменой всех изменений.
:e + Файл	Редактирование указанного файла, начиная с конца.
:e + Число Файл	Редактирование указанного файла, начиная со строки с указанным номером.
:e #	Редактирование альтернативного файла. Альтернативным файлом обычно является предыдущий файл, который редактировался до обращения к другому файлу командой :**e**. Однако если изменения в текущем файле не были закончены, когда был вызван новый файл, то альтернативным становится новый файл. Эта подкоманда аналогична подкоманде **Ctrl-A**.
:r Файл	Читает файл в буфер редактирования путем добавления новых строк перед текущей строкой.
:r !Команда	Запускает указанную команду и помещает ее выходные данные в файл путем добавления новых строк под текущим положением курсора.
:ta Тег	Редактирование файла, содержащего указанный Тег, начиная с местоположения этого тега. Чтобы использовать эту подкоманду, вы сначала должны определить базу данных имен функций и их местоположения, используя команду ctags.
Ctrl-A	Редактирование альтернативного файла. Альтернативным файлом обычно является предыдущий файл, который редактировался до обращения к другому файлу командой **:e.** Однако если изменение текущего файла не было закончено, когда был вызван новый файл, то альтернативным становится новый файл. Эта подкоманда аналогична подкоманде **:e#**
Редактирование списка файлов	
:n	Редактирование следующего файла в списке, введенного с командной строки.
:n Файлы	Указывается новый список файлов для редактирования.
Поиск информации	

Ctrl-G	Отображается имя текущего файла, номер текущей строки, количество строк в файле и какая доля файла была пройдена до местоположения курсора.
Настройка экрана	
Ctrl-L	Очистка и перерисовка экрана.
Ctrl-R	Перерисовка экрана и удаление пустых строк, помеченных символом
zЧисло	Указывается количество строк на экране.
Ввод команд оболочки shell	
:sh	Вход в оболочку *shell* для выполнения одной или нескольких команд. Вы можете вернуться в редактор vi, нажав клавиши **Ctrl-D**.
:!	Команда Ввод указанной команды и возврат в редактор vi.
:!!	Повтор последней подкоманды :!Команда
Число!!Команда	Запуск указанной команды и замена строк, указанных параметром Число выходными данными команды. Если число не указано, по умолчанию принимается 1. Если команда принимает данные со стандартного ввода, указанные строки используются в качестве входных данных.
!Объект Команда	Запускается указанная команда, и указанный Объект заменяется выходными данными этой команды. Если команда принимает данные со стандартного ввода, указанный объект используется в качестве входных данных.
Прерывание и завершение работы редактора vi	
Q	Перевод редактора ex в командный режим.
ZZ	Выход из редактора vi с сохранением изменений.
:q	Выход из редактора vi. Если вы изменили содержимое буфера редактирования, редактор выводит предупреждающее сообщение и выход не производится.
:q!	Выход из редактора vi с удалением буфера редактирования (отмена внесенных изменений).
Esc	Завершение ввода текста или прерывание работы подкоманды до ее завершения.
Ctrl-?	Прерывание работы подкоманды.

2.8.Программирование bash.

В одной из предыдущих глав *shell* рассматривалась как командная оболочка, но ее задача не ограничивается функцией передаточного звена между, пользователем и системой, это мощное средство программирования. Программы на языке *shell* называются сценариями или скриптами. Почти всем в *UNIX-подобных* системах управляют сценарии, написанные на *shell*, даже загрузкой системы, управляют shellовские скрипты, все задачи администрирования также реализуются через них. Поэтому владение навыками программирования *shell* является залогом успешного решения задач администрирования системы, даже если вы не предполагаете заниматься написанием своих сценариев. Во время загрузки *Linux* выполняется целый ряд сценариев из */etc/rc.d*, которые настраивают конфигурацию операционной системы и запускают различные сервисы, поэтому очень важно четко понимать эти скрипты и иметь достаточно знаний, чтобы вносить в них какие-либо изменения.

Что же собой представляет скрипт? В простейшем случае это набор команд, записанных в файл. Создадим файл *script1* и поместим туда следующие строки.

Например

host:~> cat > script1

 # Hello world

echo “Hello world”

Ctrl-D

Строка **# Hello world** – комментарий, любая строка в скрипте начинающая с символа # рассматривается как комментарий и не выполняется оболочкой

echo “Hello world” - команда печати по умолчанию выводит строку **Hello world** на экран терминала.

Чтобы запустить script1 на исполнение, т.е. выполнить все перечисленные в нем команды необходимо в командной строке набрать следующую последовательность.

host:~> sh script1

Если файл сценария начинается с последовательности #!, которая называется *sha-bang*, то это указывает системе какой интерпретатор следует использовать для исполнения сценария. Это двухбайтовая последовательность, или специальный маркер, определяющий тип сценария, в данном случае - сценарий командной оболочки. Более точно, *sha-bang* определяет интерпретатор, который вызывается для исполнения сценария, это может быть командная оболочка (*shell*), иной интерпретатор или утилита.

Например

host:~> cat > script1_1

#!/bin/sh

 # Hello world

echo “Hello world”

Ctrl-D

Файл *script1_1* можно сделать исполняемым командой **chmod**, и запустить его после этого можно напечатав:

host:~> ./script1_1

Знак ./ перед именем скрипта, говорит оболочке, что поиск команды *script1_1* надо осуществлять в текущем рабочем каталоге, иначе оболочка пользователя будет осуществлять поиск файла *scrip1_1* в каталогах перечисленных в переменной *PATH* и если в переменной не установлен специальный путь “.” то оболочка скорее всего не сможет найти необходимый файл.

2.8.1.Переменные

Переменные - это одна из основ любого языка программирования. Они совершенно необходимы для абстрагирования, каких-либо величин с помощью символических имен. Физически переменные представляют собой ни что иное, как участки памяти, в которые записана некоторая информация.

Самый простой способ присвоения значения переменной - написать следующее:

Имя_переменной=значение

Например

wdir=/home/user/tmp – присваиваем переменной *wdir* значение */home/user/tmp* - путь к каталогу.

Для подстановки значения переменной в *shell* используется знак **$**. Знак **$** еще называется ссылкой на переменную. Необходимо всегда помнить о различиях между *именем* переменной и ее *значением*. Если **variable1** это имя переменной, то **$variable1** - это ссылка на ее *значение*. "Чистые" имена переменных, без префикса **$**, могут использоваться только при объявлении переменных, при присваивании переменной некоторого значения, при *удалении.*

Например

echo $wdir

выведет на терминал

/home/user/tmp

echo wdir

выведет на терминал

wdir

Заключение ссылки на переменную в двойные кавычки (" ") никак не сказывается на работе механизма подстановки.

echo “$wdir”

напечатает

/home/user/tmp

Этот случай называется "частичные кавычки", иногда можно встретить название "нестрогие кавычки".

Одиночные кавычки (' ') заставляют интерпретатор воспринимать ссылку на переменную как простой набор символов, потому в одинарных кавычках операции подстановки не производятся.

echo ′$wdir′

напечатает

$wdir

Этот случай называется "полные", или "строгие" кавычки.

Написание **$variable** фактически является упрощенной формой записи **${variable}.** Более строгая форма записи **${variable}** применяется в тех случаях, когда применение упрощенной формы записи порождает сообщения о синтаксических ошибках

Например.

echo $wdir1

echo ${wdir}1

Первая команда **echo** напечатает пустую строку, т.к. воспримет **$wdir1** как имя переменной, а она не установлена.

Вторая команда **echo** напечатает

/home/user/tmp1

т.к. имя переменной четко отделено от цифры 1 с помощью фигурных скобок**{}**.

Если в значении, которое присваивается переменной, встречаются пробелы необходимо это значение заключать в двойные кавычки.

Например.

Number=”один два три”

Двойные кавычки используются для предотвращения разбиения строки на слова. Заключение строки в кавычки приводит к тому, что она передается

как один аргумент, даже если она содержит пробелы, считающиеся в *shell* по умолчанию разделителями строки.

2.8.2.Специальные типы переменных.

Локальные переменные - переменные, область видимости которых ограничена блоком кода или телом функции.

Переменные окружения - переменные, которые затрагивают командную оболочку и порядок взаимодействия с пользователем. Эти переменные "экспортируются", т.е. передаются окружению, локальному по отношению к сценарию. Экспорт переменных выполняется командой **export**.

Позиционные переменные параметры аргументы, передаваемые скрипту из командной строки - **$0, $1, $2, $3...**, где **$0** - это название файла сценария, **$1** - это первый аргумент, **$2** - второй, **$3** - третий и так далее. Аргументы, следующие за **$9**, должны заключаться в фигурные скобки, например: **${10}, ${11}, ${12}.**

Специальные переменные **$*** и **$@** содержат все позиционные параметры (аргументы командной строки).

Пример.

host ~ > cat > scritpt1_2

#!/bin/sh

echo $3

echo $2

echo $1

Ctrl-D

Запустим скрипт

host ~ > ./script1_2 hello 1 world

Позиционной переменной **$1** присвоится значение **hello**, **$2=1** и **$3=world**

На экран терминала будет выведено

world

1

hello

Переменная статуса $?, каждая команда возвращает *код завершения* (иногда код завершения называют *возвращаемым значением*). В случае успеха команда должна возвращать 0, а в случае ошибки -- ненулевое значение, которое, как правило, интерпретируется как код ошибки. Практически все команды и утилиты *UNIX* возвращают 0 в случае успешного завершения, но имеются и исключения из правил. В *shell* существует

специальная переменная **$?**, которая хранит код возврата последней выполнившейся команды.

2.8.3.Арифметические операции в shell

Арифметические операции с числами, являются основой любого языка, программирования, предусмотрена такая возможность и в *bash*, но есть ограничение *shell* рассчитан на выполнение операций с целочисленной арифметикой.

let

команда, выполняющая арифметические операции с переменными.

#!/bin/bash

echo

let a=12 # Аналогично a=12

let a=a+5 # эквивалентно let “a = a + 5” увеличивает читаемость скрипта.

Команда **let** позволяет выполнять множество арифметических операций и поддерживает синтаксис принятый в языке C.

+	сложение
-	вычитание
/	деление
*	умножение
++	инкрементальное сложение a++ => a=a+1
--	a=a-1
/=	a=a/2
*=	a=a*

Часто вместо команды **let** используется **(( ))**.

((a++)) аналогично записи **let a++**.

expr

Универсальный обработчик выражений: вычисляет заданное выражение (аргументы должны отделяться пробелами). Выражения могут быть арифметическими, логическими или строковыми.

expr 3 + 5

возвратит 8

expr 5 % 3

возвратит 2

expr 5 * 3

возвратит 15

В арифметических выражениях, оператор умножения обязательно должен экранироваться обратным слэшем.

y=`expr $y + 1` Операция инкремента переменной, то же самое, что и **let y=y+1**, или **y=$(($y+1))**

2.8.4.Условные операторы.

Любой язык программирования обязан иметь условные операторы, необходимые для проверки условий, чтобы программа могла выбрать тот или иной алгоритм действий в зависимости от сложившейся ситуации. В *shell*, для проверки условий, используется команда **test**, различного вида скобочные операторы совместно с условным оператором **if/then/else**.

Оператор **if/then/else** проверяет является ли код завершения 0(0 “успех”), если да то выполняются команды следующие за командой **then**.

Выглядит оператор **if/then/else** следующим образом.

if условие

then

команда1

команда2

…

командаN

else

команда1

команда2

…

командаN

fi

Следует обратить внимание на то что, ключевое слово **then** набирается на следующей строке после **if**. Если же **then** набирается на той же строке что и **if** после условия обязательно необходимо поставить точку запятой “;”.

if условие; then

команда1

команда2

…

командаN

else

команда1

команда2

…

командаN

fi

Что представляет собой условие в операторе **if/then**? Условие это результаты работы команды, такой как test, которая предназначена для сравнения различных величин, скобочные операторы, либо код завершения команды.

Например

```
fnm="${HOME}/.bashrc"
fnd_str="unalias"
if grep -q $fnd_str $fnm
then
echo "$fnm содержит покрайней мере одну строку с $fnd_str"
else
echo "$fnm  не содержит строк с $fnd_str"
fi
```

В этом скрипте, условие для оператора **if/then/else** определяется, кодом завершения команды **grep**. Если команда **grep** находит, строку **unalias** в файле, *.bashrc*, то возвращается код успеха, и на терминал выводится соответствующая строка, иначе возвращается код неуспеха, и работает ветка **else**.

```
if test -z "$1"
then
echo "Аргументы командной строки не заданы."
else
echo "Первый аргумент командной строки $1."
fi
```

В данном примере непосредственно используется команда **test** для проверки заданы аргументы командной строки или нет. Использование команды **test** выглядит следующим образом.

test выражение

Команда **test** возвращает 0 (*True*) или 1 (*False*), в зависимости от определения выражения. С помощью команды **test** можно сравнивать строки на равенство и неравенство, а также на то, будет ли первая строка поставлена до или после второй при сортировке. Для всего этого используются, соответственно, операторы **=, !=, < и >**. Унарный оператор **-z** проверяет, не пуста ли строка, тогда как оператор `-n` или вообще отсутствие оператора возвращает *True* если строка не пуста. В примере мы используем **–z** для того чтобы определить, если значение параметрической переменной **$1** соответствующая первому аргументу командной строки пустая строка, то следовательно аргументы командной строки не заданы.

Команда **test** позволяет сравнивать два числовых значения с помощью операторов **-eq, -ne, -lt, -le, -gt**, **-ge**, означающих соответственно "*равно*", "*неравно*", "*меньше*", "*меньше либо равно*", "*больше*" и "*больше либо равно*".

Еще очень часто команда **test** используется для тестирования файлов.

Некоторые часто используемые тесты файлов

Оператор	Характеристика
-d	Папка
-e	Существует (также -a)
-f	Стандартный файл
-h	Символьная ссылка (также -L)
-p	Именованный канал
-r	Доступный вам для чтенения
-s	Не пустой
-S	Сокет
-w	Доступный вам для записи
-N	Был изменен со времени последнего прочтения

Выражение **test expr** и **[expr]** представляют из себя эквивалентные выражения и очень часто в скриптах используется более короткая и читаемая запись выражения в квадратных скобках вместо команды **test**.

```
fl=/etc/passwd
if [ -e $fl ]
then
```

```
  echo "Файл паролей существует."
fi
```

Конструкцию **if** можно записать было и через **test**.

```
fl=/etc/passwd
if test -e $fl
then
  echo "Файл паролей существует."
fi
```

Наконец, опции **-a** и **-o** соединяют выражения как логическое *И* и *ИЛИ* соответственно, а унарный оператор **!** делает смысл теста противоположным.

Например

```
test "a" != "$HOME" -a 3 -ge 4 ; echo $?
[ ! \( "a" = "$HOME" -o 3 -lt 4 \) ]; echo $?
```

Команда **test** предоставляет широчайшие возможности, но является несколько неудобной в использовании из-за необходимости делать переходы в коде с помощью \ и из-за различиях в сравнениях строк и чисел. К счастью, *bash* располагает двумя другими способами тестирования, которые покажутся более естественными для тех, кто знаком с синтаксисом C, C++ или Java.

Составная команда (( )) выполняет арифметические действия и устанавливает код возврата равный 1, если выражение равно 0, или код возврата равный 0, если выражение имеет ненулевое значение. Вам ненужно ставить \ перед операторами между ((и)). Арифметика работает с целыми числами. Деление на 0 вызывает ошибку, но переполнение этого не делает. Вы выполняете обычные для языка C арифметические, логические и побитовые операции. (( )) могут использоваться и для определения истинности логических выражений, как показано в примере ниже.

Например.

```
num1=5
num2=6
if (( num1 > num2 ))
 then
echo “$num1 > $num2”
else
echo “$num1 < $num2”
fi
```

Следует обратить внимание, что внутри скобок переменные могут использоваться без знака $.

Составная команда **[[ ]]** позволяет вам использовать более естественный синтаксис для тестирования имен файлов и строк. Вы можете объединять тесты, которые разрешены для команды **test** с помощью круглых скобок и логических операторов.

Например

[[(-d "$HOME") && (-w "$HOME")]]

2.8.5.Циклы.

Циклы это одна из управляющих конструкций, любого языка программирования, задача, которой организация многократного исполнения набора инструкций. Конечно же операторы циклов есть и в *shell*.

for (in)

Это одна из основных разновидностей циклов в shell, эта конструкция значительно отличается от аналога в языке C.

```
for arg in [list]
do
 команда(ы)...
done
```

На каждой итерации цикла, переменная-аргумент цикла *arg* последовательно, одно за другим, принимает значения из списка *list*.

```
for arg in "$var1" "$var2" "$var3" ... "$varN"
# На первом проходе, $arg = $var1
# На втором проходе, $arg = $var2
# На третьем проходе, $arg = $var3
# ...
# На N-ном проходе, $arg = $varN
```

Элементы списка заключены в кавычки для того, чтобы предотвратить возможное разбиение их на отдельные аргументы (слова).

Элементы списка могут включать в себя шаблонные символы.

Если ключевое слово **do** находится в одной строке со словом **for**, то после списка аргументов (перед **do**) необходимо ставить точку с запятой.

```
for arg in [list] ; do
#!/bin/bash
# Список планет.
```

for planet in Меркурий Венера Земля Марс Юпитер Сатурн Уран Нептун Плутон

do

echo $planet

done

Простой скрипт демонстрирующий работу цикла. На каждой итерации переменной *planet* будет присваиваться название одной из планет нашей солнечной системы.

В качестве списка **[list]** можно использовать и вывод команды.

Например

for i in $(ls); do

echo item: $i

done

Список возможных значений переменой цикла, формируется командой **ls**.

Если необходимо организовать цикл счетчиком, как принято во многих языках программирования то используется следующая конструкция.

#!/bin/bash

for i in `seq 1 10`;

do

echo $i

done

Особенность данного цикла заключается в использование команды **seq**, команда генерит последовательность значений цифр в виде строк 1 2 3 4 5 6 7 8 9 10. Соответственно переменная итератор i последовательно будет принимать значение 1,2,3…. и так далее.

while

Оператор **while** проверяет условие перед началом каждой итерации и если условие истинно (если код возврата равен 0), то управление передается в тело цикла. В отличие от циклов **for**, циклы **while** используются в тех случаях, когда количество итераций заранее не известно.

while[condition]
do
command...
done

Как и в случае с циклами **for/in**, при размещении ключевого слова **do** в одной строке с объявлением цикла, необходимо вставлять символ ";" перед **do.**

while [condition] ; do

В скрипте приводится пример использования **while** для организации цикла со счетчиком.

```
#!/bin/bash
i=0
LIMIT=10
while [ "$i" -lt "$LIMIT" ]
do
  echo -n "$i "        # -n подавляет перевод строки.
  i=`expr $i + 1`    # допускается i=$(($i+1)).
done
echo
exit 0
```

["$i" -lt "$LIMIT"] конструкция как и в условных операторах соответствует команде **test**, опция **lt** указывает на сравнение арифметических чисел, условие верно пока i меньше *LIMIT*, при каждой итерации цикла, переменная **i** увеличивает свое значение на единицу. Строка **i=`expr $i + 1`**, команда **expr** может осуществлять операции с различными типами переменных, она считывает командную строку, декодирует ее и выполняет необходимые операции.

Контрольные вопросы и задания.

1. Перечислите основные особенности архитектуры UNIX?
2. Чем отличается файловая система UNIX от файловой системы Windows?
3. Что такое абсолютный и относительный путь?
4. Что означает каталог .. ?
5. Что такое дерево процессов в UNIX?
6. Что такое PID?
7. Какой процесс в системе UNIX является основным?
8. Какие функции выполняет командный интерпретатор?
9. Как shell находит команды?
10. Что такое аргумент команды?

11. Что такое опции?
12. Что такое стандартные файлы потоков?
13. Как выполняется перенаправление вывода?
14. Что такое командный конвейер?
15. Что такое условное выполнение команд?
16. Как экстренно прервать выполнение команды?
17. Какой командой можно посмотреть рабочую директорию?
18. Какая команда служит для изменения текущей директории на другую?
19. Как посмотреть список файлов в папке?
20. Что значит –r-x---r--?
21. Какой командой можно поменять права доступа к файлу?
22. Какие права установит для файла команда изменения прав, если задать 644?
23. Как создать папку?
24. Как удалить папку и файлы?
25. Копирование файлов?
26. Найти в папке /usr все файлы по размеру больше 4МБ?
27. Как распаковать файл tgz?
28. Как создать архив?
29. Вывести список всех ваших процессов?
30. Какие файлы являются конфигурационными для bash?
31. Чем отличаются ~/.bash_profile и ~/.bashrc?
32. Что хранит переменная PATH?
33. Какой командой можно посмотреть переменные окружения?
34. Удалить все пустые строки в файле?
35. Как вывести из файла все строки, где встречаются только цифры?
36. Заменить в файле слов Windows на слово Linux?
37. Как вывести только логины пользователей из файла /etc/passwd?
38. Как вывести только имя файла и его размер?
39. Какие команды в vi позволяют перейти в режим редактирование текста?
40. Какой командой в vi можно удалить строки?
41. Как скопировать строки в vi?

42. Как завершить работу в vi?
43. Написать, скрипт который бы проверяла включены компьютеры или нет?
44. Написать скрипт подсчитывающий число команд в системе?

3. Графическая система UNIX-подобных систем.

В *UNIX-подобных системах* оконная система *X-Windows*, обеспечивает стандартные протоколы и предоставляет инструменты, для построения графического интерфейса пользователя. *X-Window* реализует базовые функции графической среды: отрисовку и перемещение окон на экране, взаимодействие с мышью и клавиатурой, но *X Window System* не определяет деталей интерфейса пользователя — этим занимаются менеджеры окон. Из-за этого внешний вид графических интерфейсов пользователя в среде *UNIX-подобных* системах может очень сильно различаться в зависимости от возможностей и настроек конкретного оконного менеджера.

X-Window System построена по принципу клиент сервер, что позволяет запускать графические приложения на другой машине в сети, а их интерфейс при этом будет передаваться по сети и отображаться на локальной машине пользователя (в случае если это разрешено в настройках). Система *X-Window System* была разработана в Массачусетском технологическом институте (MIT) в 1984 году. Нынешняя версия протокола — *X11* — появилась в сентябре 1987 года.

X-сервер обменивается сообщениями с различными клиентскими программами. Сервер принимает запросы на вывод графики (окон) и отправляет обратно пользовательский ввод (от клавиатуры, мыши или сенсорного экрана).

Эта клиент-серверная терминология — пользовательский терминал в качестве «*сервера*» и удалённые приложения в качестве «*клиентов*» — зачастую запутывает новых пользователей *X*, так как обычно эти термины имеют обратные значения. Но *X Window System* принимает точку зрения программы, а не конечного пользователя аппаратуры: локальный дисплей предоставляет услуги отображения графики программам, и потому выступает в роли сервера. Удалённые программы используют эти услуги, и потому играют роль клиентов.

Как должен выглядеть интерфейс пользователя приложения — кнопки, меню, заголовки окон и т. д. решаются на уровне оконных менеджеров.

Оконный менеджер управляет размещением и внешним видом окон приложений. Он может создавать интерфейс, подобный *Microsoft Windows* или *Macintosh* (например, так работают оконные менеджеры *Kwin* в *KDE* и *Metacity* в *GNOME*), или совершенно другой стиль (например, в фреймовых

оконных менеджерах, таких, как *Ion*). Оконный менеджер может быть простым и минималистичным (как *twm* — базовый оконный менеджер, поставляемый с *X*), а может предлагать функциональность, близкую к полноценной рабочей среде (например, *Enlightenment*).

Многие пользователи используют *X* вместе с полной средой рабочего стола, которая включает в себя оконный менеджер, различные приложения и единый стиль интерфейса. Наиболее популярные среды рабочего стола — *GNOME* и *KDE*. В стандарте *Single UNIX Specification* указана среда *CDE*. Проект *freedesktop.org* пытается обеспечить взаимодействие между различными средами, а также компоненты, необходимые для конкурентоспособного рабочего стола на основе *X*.

Референсная (или образцовая) реализация (англ. *reference implementation*) от фонда *X.Org Foundatio*n, называемая *X.Org Server*, является канонической реализацией *X Window System*. Поскольку она распространяется на условиях весьма либеральной лицензии, появились несколько её разновидностей (как свободных, так и проприетарных). Коммерческие поставщики *UNIX* зачастую берут демонстрационную реализацию и адаптируют её к собственному аппаратному обеспечению, обычно сильно модифицируя её и добавляя проприетарные расширения.

Вплоть до 2004 года проект *XFree86* был наиболее распространённым вариантом X для свободных *UNIX-подобных* операционных систем. *XFree86* возник как порт *X* на 386-совместимые персональные компьютеры. К концу 1990-х этот проект стал главным источником технических инноваций в *X Window System* и де-факто руководил разработкой *X*. Однако в 2004 году *XFree86* поменял условия лицензии и реализация *X.Org Server* (которая является форком XFree86, но со свободной лицензией) стала более распространённой.

Контрольные вопросы и задания.

1. Особенности графической системы *UNIX*?
2. Какие функции выполняет *X-сервер*?
3. В чем отличие *X-сервера* от менеджера окон?

4. Администрирование.

Администрирование *UNIX* зависит от конкретной системы, но при этом между различными версиями и дистрибутивами есть много общих черт, что позволяет администратору, хорошо знающему принципы и устройство системы без труда переходить от одной *UNIX* системы к другой. Наиболее распространенными и доступными для пользователей являются дистрибутивы *Linux*. Мы рассмотрим задачи администрирования на примере *OpenSUSE*. *OpenSUSE* разрабатывается, как вспомогательная система *Suse Linux Enterprise Server(SLES)* фирмы *Novell*. *SLES* хорошо поддерживается

компаниями производителями компьютерного оборудования, что распространяется и на открытую версию системы, соответственно уменьшаются проблемы с драйверами устройств. Почти все драйвера проходят проверку работоспособности под *SLES*.

Начнем знакомиться с задачами администрирования с установки системы.

4.1.Инсталляция.

Для инсталляции операционной системы *OpenSuSe* вам потребуется компьютер с как минимум 256 Мбайтами оперативной памяти и процессором не ниже Pentium IV. Дистрибутив ОС можно загрузить из интернета. Любой дистрибутив представляет собой образ инсталляционного DVD диска. Существует несколько способов инсталляции. Наиболее часто встречающийся, когда образ системы записывают на “болванку” и с нее проводят инсталляцию. Но этот способ не единственный, допустим, если на компьютере нет DVD привода, то можно инсталлировать по сети, тогда с помощью утилит, поставляемых в образе ОС, создается загрузочная флешка, и с помощью нее запускается инсталляция системы через сеть. Инсталляция системы через сеть может быть выполнена и без создания загрузочного носителя, а с помощью сетевой технологии PXE.

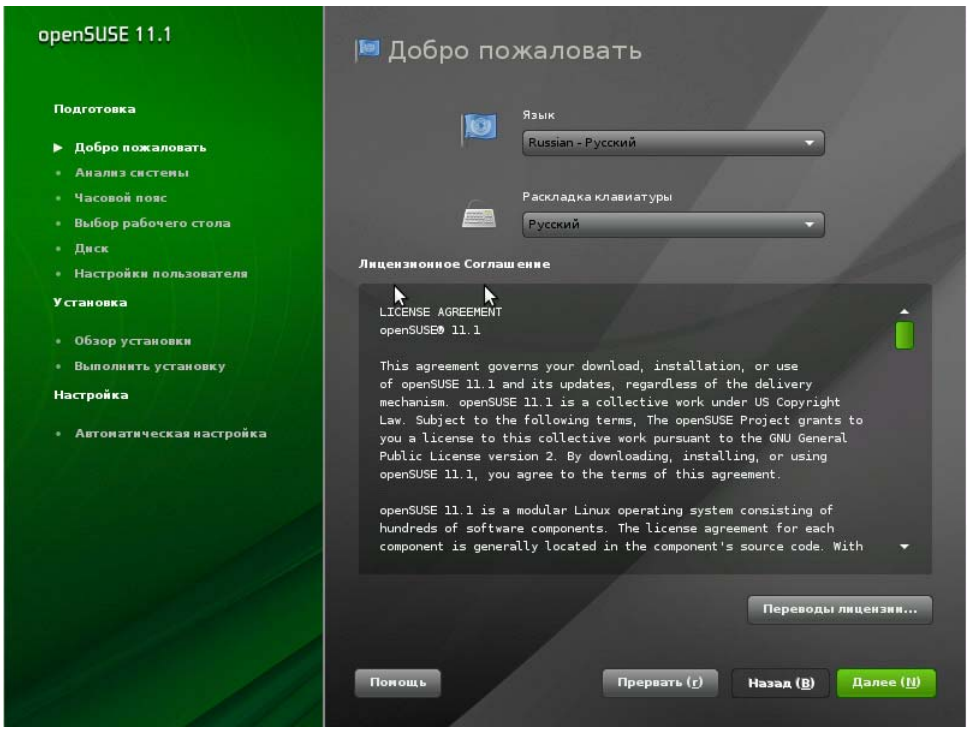
PXE (англ. *Preboot Execution Environment*, произносится пикси) - технология загрузки компьютеров с помощью сетевой карты без использования жёстких дисков, компакт-дисков и других устройств, применяемых при загрузке операционной системы. *PXE*-код, обычно находящийся в ПЗУ сетевой карты, получает из сети по протоколу *TFTP* (получив адрес *TFTP*-сервера по *DHCP*) исполняемый файл, после чего передаёт ему управление.

Остановимся на наиболее часто встречающейся ситуации, а именно, инсталляции с DVD устройства.

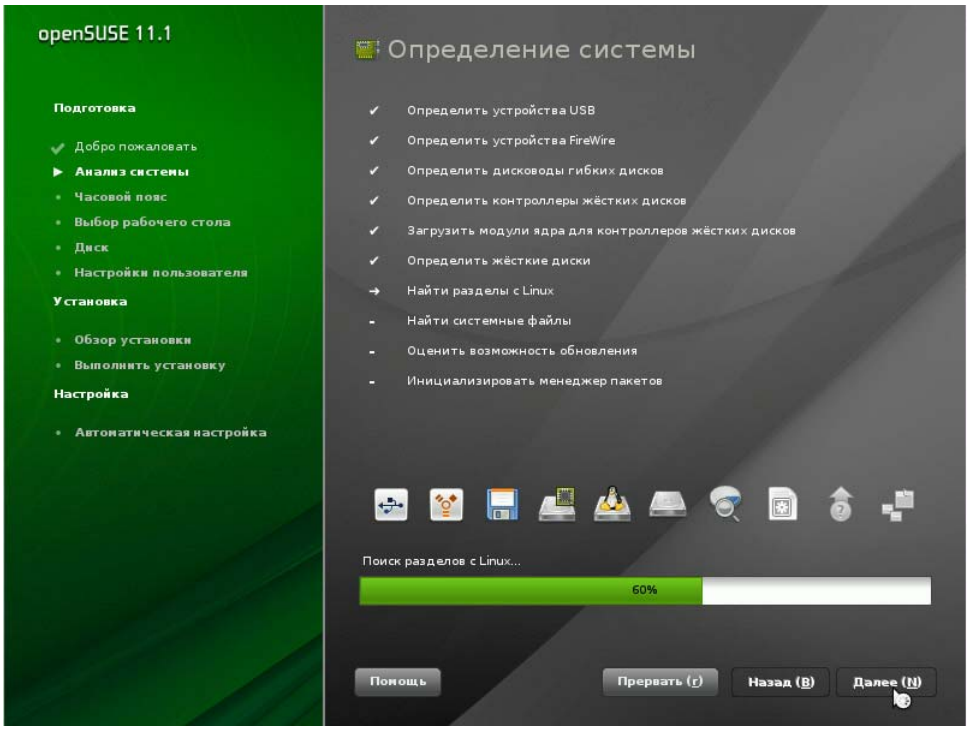
Загрузитесь с DVD. Вы увидите инсталляционное меню на английском языке.

Сейчас многие дистрибутивы являются мультиязычными. *OpenSUSE* не исключение, нажав кнопку *F2*, вы можете переключиться на русский язык.

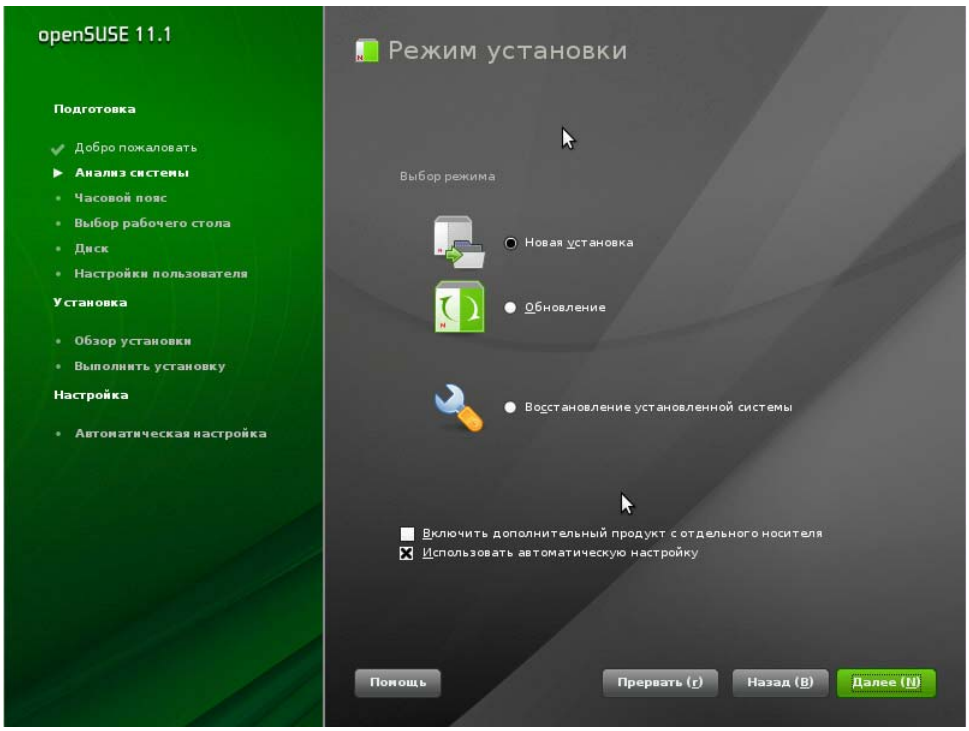
Пункт **Установка**, запускает процедуру загрузки инсталляции системы. Сначала, пользователю предоставят возможность ознакомиться с лицензионным соглашением.



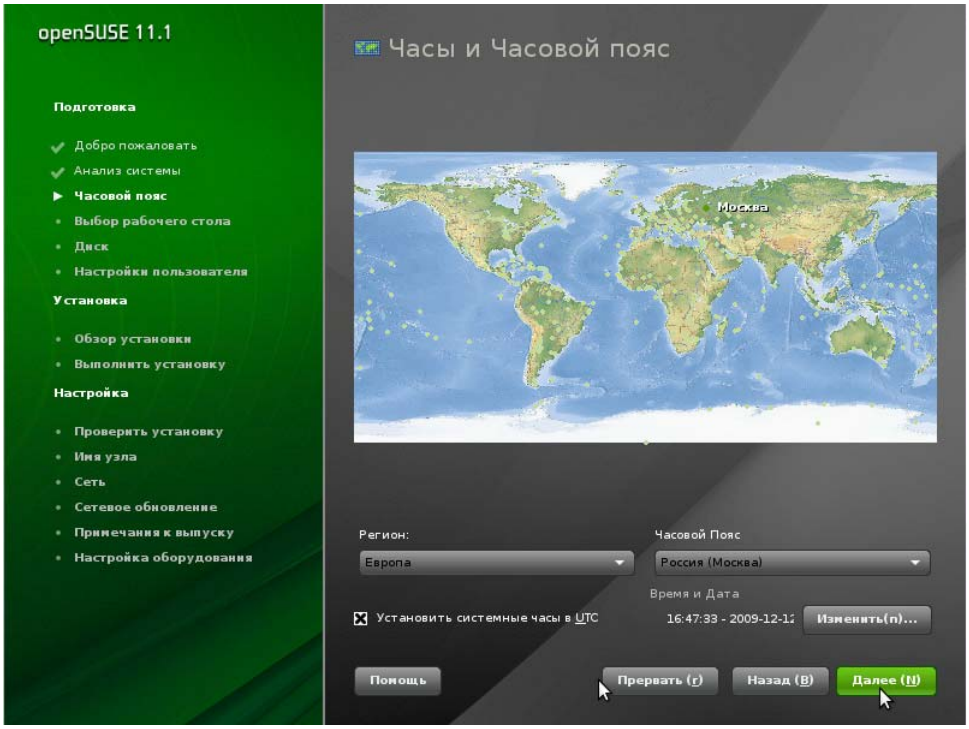
Далее инсталлятор проведет анализ конфигурации компьютера.



Существует два типа инсталляции системы. Первый установка новой системы, второй обновление старой.

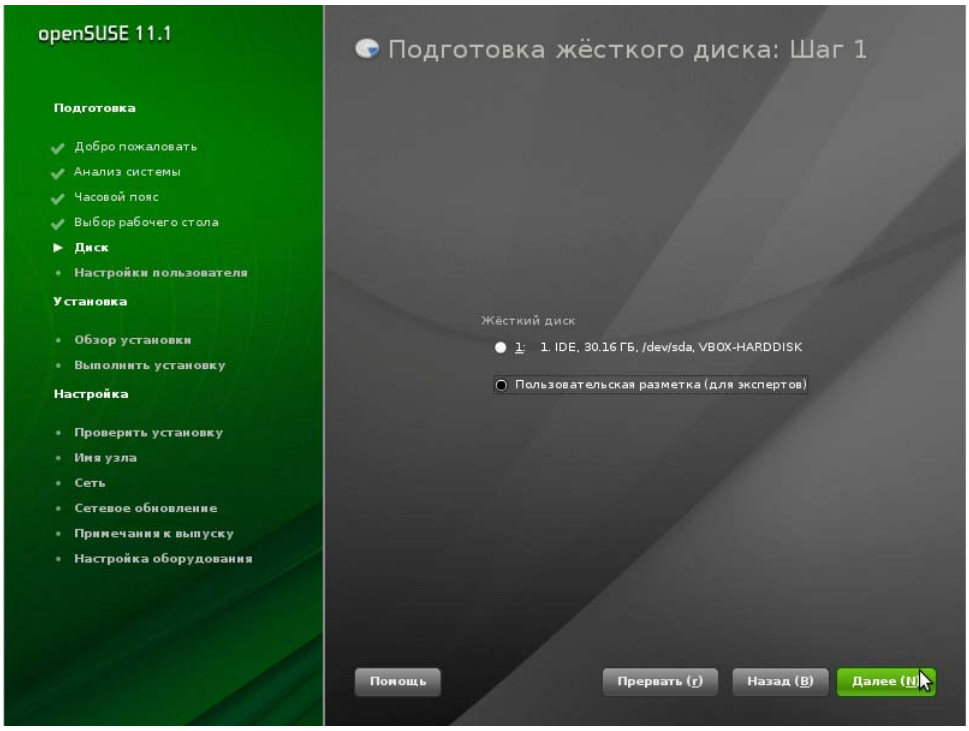


Необходимо задать часовой пояс и настроить часы. Надо обязательно, снять флажок **Установить часы в UTC** иначе при перезагрузке будет изменяться время.



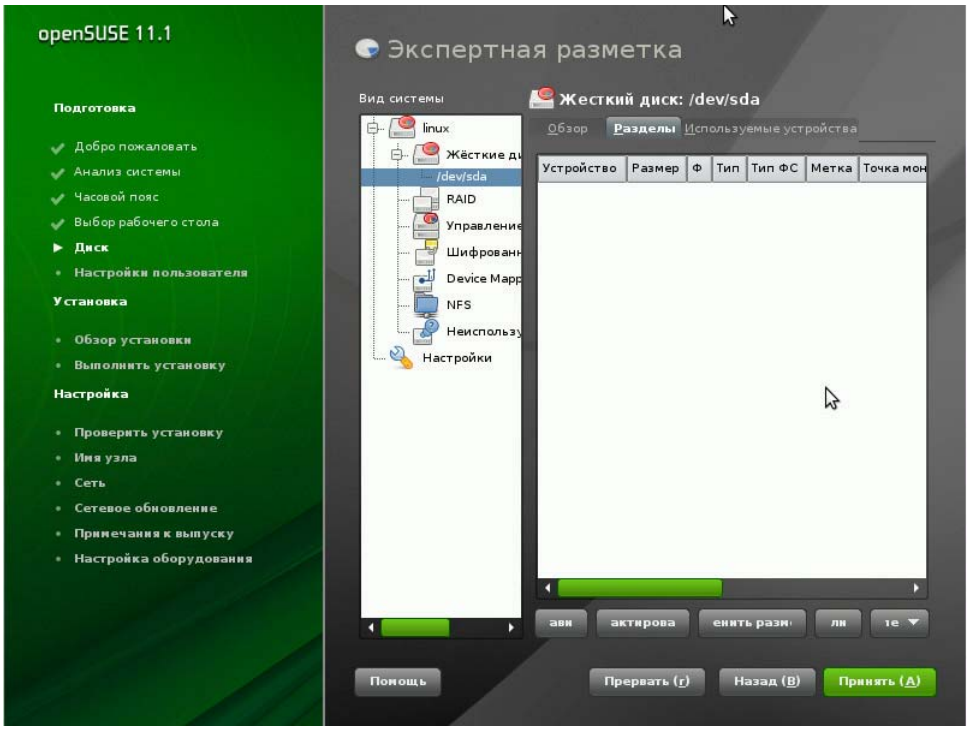
Далее необходимо выбрать графическую оболочку. На самом деле вы можете поставить все графические оболочки, какие представлены в дистрибутиве. И переключатся между ними по вашему желанию. Наиболее распространенные это *KDE* и *Gnome*.

Следующий шаг разметка диска. Есть несколько способов разметить диск. Первый, на диске создается один большой раздел и в него устанавливается ОС. Второй разбить диск на разделы. Любой диск может содержать четыре основных раздела, если же необходимо большее число разделов, то можно создать один раздел расширенный и в нем уже создавать сколько угодно разделов.



В *UNIX* системе есть предопределенные директории, которые должны присутствовать в любой системе, в них система записывает служебную информацию, либо программное обеспечение, обычно некоторые директории и их содержимое помещаются в отдельные файловые системы. Разделение

диска на разделы, позволяет обеспечить большую живучесть ос, уменьшить время работы с диском и более оптимально использовать дисковое пространство.



Конечно, нет необходимости создавать множество разделов, обычно в отдельный раздел выносится директория */usr* иногда */var*. И есть два раздела необходимых на любой системе это корневая файловая система / и раздел подкачки. И еще создается несколько пользовательских разделов между которыми распределяется основной объем дискового пространства.

Размер / около 2 Гбайт

Размер раздела подкачки *swap* двойной размер оперативной памяти. т.е если у вас объем оперативной памяти 1 Гбайт то объем раздела подкачки должен быть равен 2 Гбайта.

Размер /var около 5 Гбайт.

Размер /usr около 15 Гбайт.

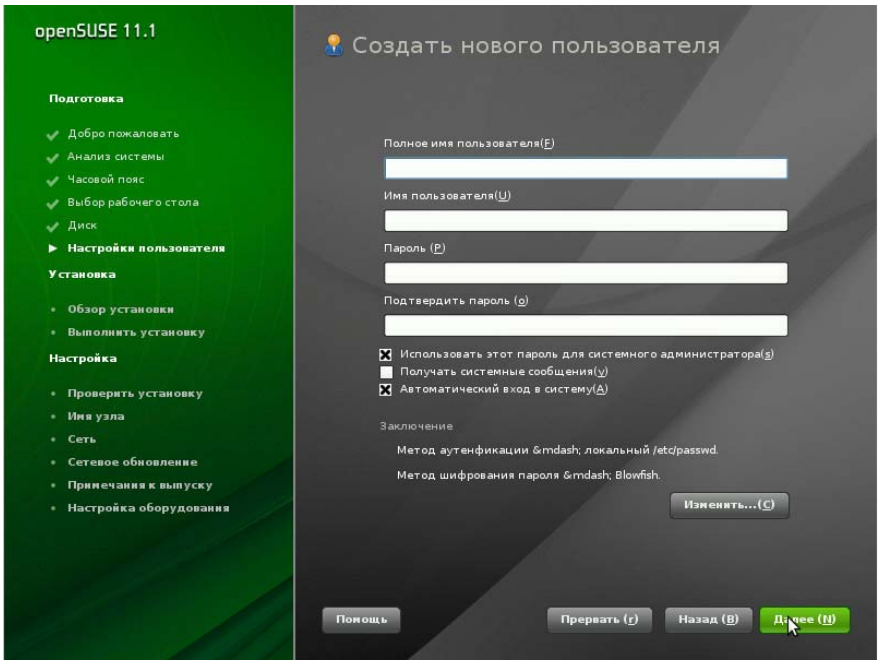
При создании почти всех разделов, кроме *swap*, необходимо указать тип файловой системы. В *UNIX* есть несколько типов файловой системы, наиболее распространенный это ext3.

Сравнительные параметры различных файловых систем.

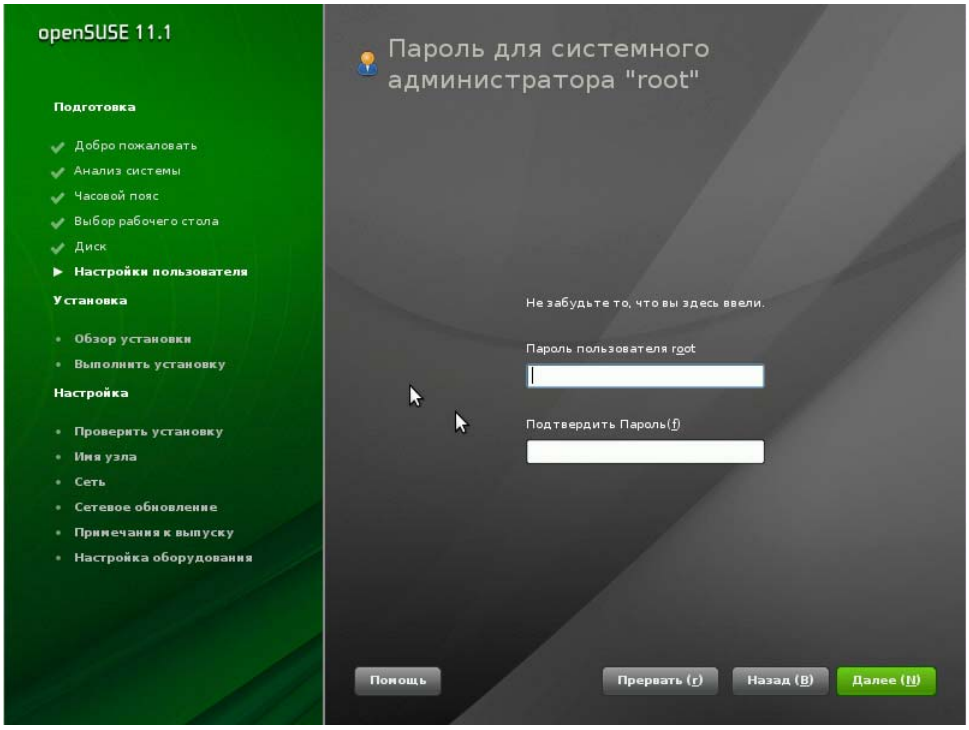
	Ext2	Ext3	Jfs	Reiserfs	xfs

журналирование	No	Yes (10MB)	Yes (AUTO)	Yes (32MB)	yes
изменение размера	Yes Unmount	Yes Unmount	Yes	Yes	Yes Mount
Макс. допустимый размер.(TB-Терабайт, PB пента байт, EB – Экзо байт.	File: 2TB FS:16TB	File:2TB FS:16TB	File:4PB FS:32PB	File:16TB FS:1EB	File:2TB FS:8EB
Представление информации.	I-Nodes (block)	I-Nodes (block)	I-Nodes (b-tree)	B-tree	I-Nodes (b-tree)

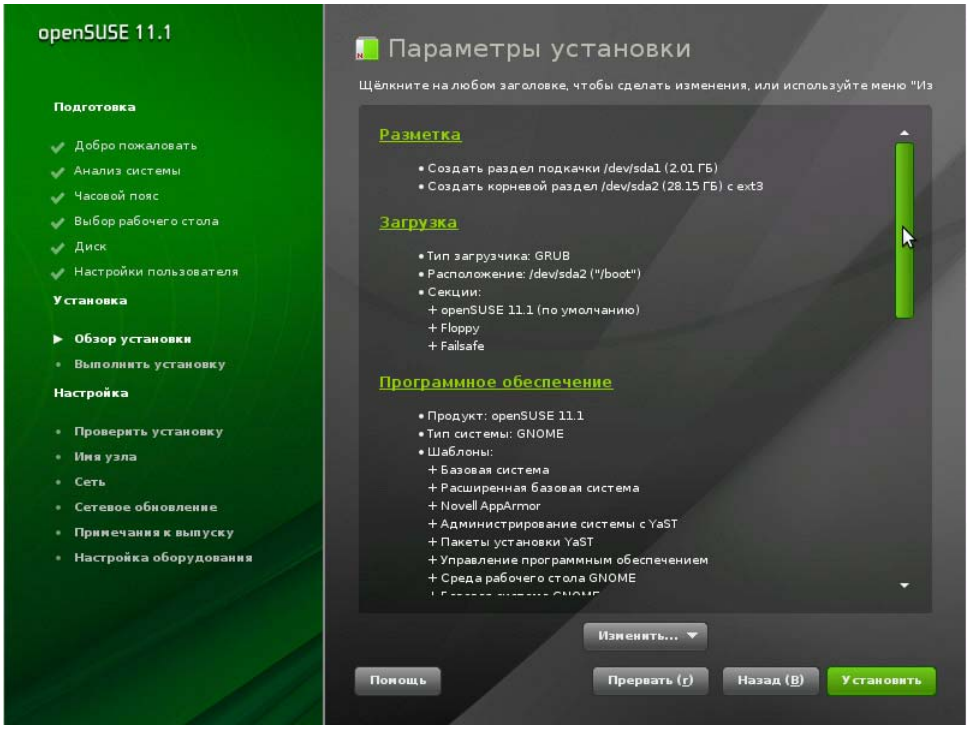
После разметки процедура инсталляции попросит создать пользователя, под которым вы хотите работать, если пользователей в системе будет несколько, то необходимо выключить авто вход в систему. Также рекомендуется снять, флажок чтобы пароль пользователя не использовался в качестве пароля для *суперпользователя*.



Если для пользователя и *root* вы используете разные пароли, флажок снят, то система выдаст вам окно, где необходимо вести пароль для *root*.



Перед тем как процедура инсталляции начнет записывать файлы на жесткий диск, вы увидите сводку параметров системы.

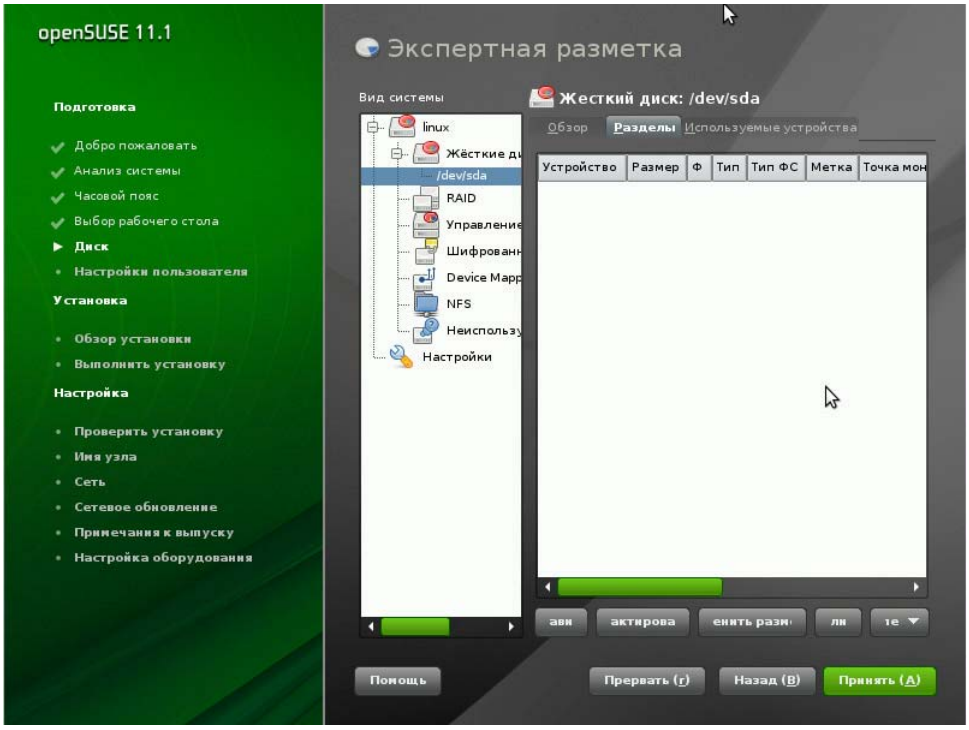


В разделе **Разметка**, можно внести изменения в разметку жесткого диска.

Раздел **Загрузка** позволит вам отредактировать параметры загрузчика.

Раздел **Программное обеспечение**, обязательно необходимо посмотреть и добавить нужное программное обеспечение. Программное

обеспечение можно добавить и после установки системы, но лучше сразу поставить как можно больше пакетов необходимых вам для работы.



В разделе пользователя, можно установить пароль для root.

После того как все необходимые изменения внесены, нажимайте кнопку установить.

Инсталлятор отформатирует созданные разделы и начнет устанавливать пакеты. Процесс идет автоматически, можете передохнуть и выпить чашечку чая. Время, которое потребуется на установку пакетов укажет индикатор, в системе версии 11.1 индикатор достаточно точно определяет необходимое для установки всех пакетов время.

После завершения установки пакетов, система завершит свою работу и пойдет на перезагрузку. Во время перезагрузки можно уже вытащить инсталляционный DVD, либо в меню выбрать опцию загрузка с жесткого диска.

На самом деле установка системы не завершена, перезагрузившись система, потребует настроить дополнительные параметры необходимые ей для работы.

- Установить имя узла.
- Настроить сетевые интерфейсы.
- Скачать и установить обновления программного обеспечения. (именно поэтому лучше сразу ставить все необходимые программные продукты при установки системы)
- Настроить звуковую и видео карту.

4.2. Вход в систему и завершение работы.

По умолчанию в системе запускается графический менеджер. Для входа в систему необходимо указать логин и пароль. Загрузится та графическая среда, которую вы выбрали, можно изменить выбранный менеджер окон, в пункте сеанс. Если вы предпочитаете работу в консоле, то можно переключится на один из виртуальных терминалов использую сочетания клавиш <*Ctrl*>+<*Alt*>+<*Fn*> где <*Fn*> одна из функциональных клавиш*, F1,F2,F3…,n* – соответствует номеру виртуальной консоли.

Переключаться между консолями выйдя из графического режима можно сочетанием клавиш <*Alt*>+<*Fn*>. Клавиша <*Ctrl*> нажимается, только если вы находитесь в графической оболочке, дальше ее при работе с виртуальными консолями использовать нет необходимости.

Вернуться в графический режим можно нажав <*Alt*>+<*F7*> , на седьмой терминале по умолчанию запускается *X-сервер*.

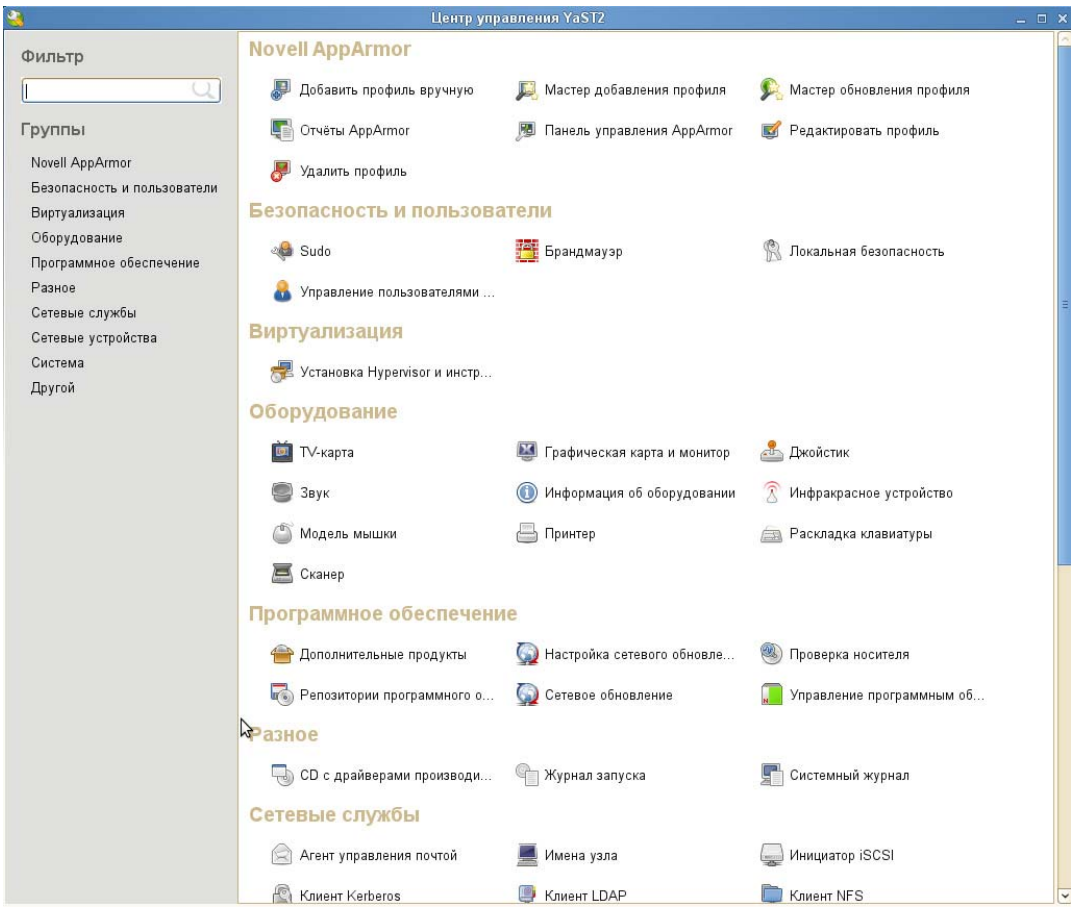
Для выхода из консоли можно воспользоваться командой *logout*, либо ее псевдонимом(*alias*) *exit*. Команда logout не выключает компьютер, а прекращает терминальный сеанс работы с ним. Для перезагрузки необходимо использовать команду *reboot*, для выключения команду *poweroff*, чтобы выполнить две эти команды необходимо обладать правами суперпользователя. Помимо этого есть еще несколько команд управляющие завершением работы с компьютером. Это команда *halt*, которая сбрасывает систему, но не выключает питания компьютера, и команда *shutdown*, позволяющая управлять процессом выключения компьютера, задать время, выбрать режим выключение или перезагрузка.

Завершение работы компьютера в графическом режиме обычно не вызывает затруднений. Выберите в панели пункт **Компьютер** и выберите команду **Выключить**. Далее в появившемся окошке выберите один из вариантов выключения компьютера.

4.3. Базовая настройка.

YAST – конфигуратор системы *OpenSUSE*. Во многих *Linux* системах есть подобные конфигураторы. **YAST** является аналогом панели управления в системе Windows. Однако нужно понимать, что, как и в *Windows* вы не сможете полностью настроить систему с помощью панели инструментов, для более тонкой настройки, возможно, понадобиться редактирования файла реестра, так же и **YAST** не сможет полностью настроить, только основные и фундаментальные вещи, и для более точной или не стандартной настройки придется редактировать конфигурационные файлы в каталоге */etc*.

В *openSUSE* есть два приложения для настройки первое на уровни пользователя, настраивает внешний вид интерфейса **Центр управления**, задача второго настройка системы центр управления **YAST2** или просто **YAST**. **YAST2** это графическое приложение, есть еще **YAST** терминальная приложение для настройки системы из окна терминала.



Конфигураторы в центре управления **YAST2**, разбиты на несколько групп.

- Программное обеспечение – позволяет удалять, добавлять программы, настраивать сетевое программное обеспечение.
- Оборудование – настройки устройств компьютера.
- Система – обще системное конфигурирование.
- Сетевые устройства – конфигурация соединений по интернету или локальной сети.
- Службы – настройка различных служб на компьютере.
- Novell AppArmor - установки программы защиты для преложений.
- Безопасность и пользователи – различные параметры безопасности и пользователей.
- Виртуализация - параметры виртуальных сред
- Разное – конфигураторы не относящиеся к не одной из выше перечисленной категории.

4.3.1.Оборудование

Конфигурирует следующие устройства.

- Tv карта – настройка ТВ тюнера.
- Джойстик – настройка джостиков и других игровых устройств.
- Информация об оборудовании – выводит подробную информацию об устройствах в компьютере.
- Модели мышки – настройка параметров мышки(например позволяет настроить мышку для работы левой рукой)
- Раскладка клавиатуры здесь все понятно.
- Графическая карта и монитор – конфигурация параметров монитора и видеоадаптера.
- Звук – настройка звуковой платы.
- Инфракрасное устройство - настройка ИК-порта, если есть на компьютере.
- Принтер – настройка принтера. (Правда, я рекомендую для настройки принтеров воспользоваться пакет cups)
- Сканер – изменение параметров сканера.

  Настройки DSL модемов перенесены в группу Сетевое оборудование.

4.3.2.Система

- LVM запускает менеджер логических дисков.
- Дата и время установка системных часов.
- Ядро позволяет выбрать планировщик ядра.
- Редактор */etc/sysconfig* – в каталоге */etc/sysconfig* находятся различные конфигурационные файлы.
- Системные службы уровни запуска – позволяет редактировать уровни запуска системы.
- Язык можно изменить основной язык системы.
- Загрузчик редактирование конфигурации загрузчика.
- Разделы работа с разделами жесткого диска.

4.3.3.Загрузка операционной системы.

Рассмотрим процесс загрузки операционной системы. После включения питания компьютера и завершения работы БИОС (*BIOS – Basic Input Output System*), *BIOS* передает управление программе, которая записана в первых 512 байтах жесткого диска(*MBR- MASTER BOOT RECORD*). Обычно это часть программы загрузчика, еще в *MBR* записана таблица

размещения разделов диска. Задача программы расположенной в *MBR* загрузить ядро операционной системы в оперативную память.

Наиболее популярным загрузчиком на сегодняшний день является GRUB. В старых дистрибутивах использовался *LILO*. *LILO* и сейчас часто встречается в различных дистрибутивах. К тому же в любом дистрибутиве есть возможность отказаться от *GRUB* и воспользоваться *LILO*. Но все-таки *GRUB* предпочтительнее.

В *openSUSE* конфигурационный файл *GRUB* находится в */boot/grub/menu.lst*.

Рассмотрим файл конфигурационный файл подробно.

 # Modified by YaST2. Last modification on Сбт Дек 12 14:07:46 MSK 2009

default 0

timeout 8

gfxmenu (hd0,0)/boot/message

##YaST - activate

###Don't change this comment - YaST2 identifier: Original name: linux###

title openSUSE 11.1 - 2.6.27.39-0.2 (default)

root (hd0,0)

kernel /boot/vmlinuz-2.6.27.39-0.2-default root=/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part1 resume=/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part2 splash=silent showopts vga=0x345

initrd /boot/initrd-2.6.27.39-0.2-default

###Don't change this comment - YaST2 identifier: Original name: failsafe###

title Failsafe -- openSUSE 11.1 - 2.6.27.39-0.2

root (hd0,0)

kernel /boot/vmlinuz-2.6.27.39-0.2-default root=/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part1 showopts ide=nodma apm=off noresume edd=off powersaved=off nohz=off highres=off processor.max_cstate=1 x11failsafe vga=0x345

initrd /boot/initrd-2.6.27.39-0.2-default

###Don't change this comment - YaST2 identifier: Original name: linux###

title Debug -- openSUSE 11.1 - 2.6.27.39-0.2

root (hd0,0)

kernel /boot/vmlinuz-2.6.27.39-0.2-debug root=/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part1 resume=/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part2 splash=silent showopts vga=0x345

initrd /boot/initrd-2.6.27.39-0.2-debug

Параметр **title** задает названия пункта меню при выборе загрузки системы. Это обычная строка символов. В данном случае *OpenSUSE* и *Windows* именно эти надписи, будут отображены на экране при выборе какую систему загружать.

Все параметры, набранные после ключевого слова *title*(до следующего *title*), относятся к данному пункту. Параметры, описанные до первого title, называются глобальными и влияют на процедуру загрузки.

default определяет какая система будет загружена по умолчанию если в течении времени заданное timeout пользователь не выбрал систему.

gfxmenu определяет графическое меню GRUB.

root определяет корневую файловую систему для Linux, для Windows этот параметр без полезен.

kernel задает имя файл образа ядра и параметры, переданные ядру при загрузке.

initrd задает имя файла RAM диска. Файл RAM диска содержит минимальный необходимый набор модулей ядра, для приведения системы в рабочее состояние.

rootnoverify параметр указывается для всех не UNIX систем.

chainloader параметр для всех систем поддерживающих цепочечную загрузку.

После загрузки ядра, задача которого обеспечить работоспособность устройств компьютера, монтируется корневая файловая система и порождается процесс *init*. Программа init читает конфигурационный файл */etc/inittab*, и в соответствии с этим файлом запускает различные процессы.

Основной строкой файла */etc/inittab* является строка

id:<число>:initdefault

Эта запись задает уровень запуска системы во время загрузки. Уровень запуска определяет, какие сценарии выполнит процесс init. Всего по умолчанию в системе шесть уровней запуска.

0	останов системы, выполнив в консоли команду init 0 вы сбросите систему

1	однопользовательский режим.
2	многопользовательский режим без поддержки сети
3	многопользовательский режим с поддержкой сети
4	не используется
5	многопользовательский режим с поддержкой сети и запуском графической оболочки
6	режим перезагрузки, набрав в консоли команду init 6 запустите перезагрузку системы.

Система Linux чаще всего загружается либо на 3 либо на 5 уровне. Перейти с уровня на уровень можно выполнив команду

init <Номер уровня>

Определив номер уровня загрузки система начинает выполнять сценарии загрузки из каталога */etc/rc.d/rc<Номер уровня>.d*, если посмотреть с помощью команды ls содержимое этого каталога, то можно увидеть, что там находятся ссылки со специальными именами.

S<номер><имя>

K<номер><имя>

<Номер> определяет порядковый номер выполнения сценария при загрузке. Ссылка на сценарий с номером 15 выполнится раньше, чем ссылка с номером 20.

<имя> задает имя сценария, который должен выполнится. Сами сценарии находятся в каталоге */etc/init.d*

Буква **S** и **K** определяют с какими параметрами будут выполнятся сценарии из каталога */etc/init.d*. У сценариев расположенных в этом каталоге есть много параметров, но два основных это **start** и **stop**. Ссылки начинающиеся на **S** будут запускать скрипты с опцией **start**, а начинающиеся с **K** будут выполняться с опцией **stop**.

Сценарии запуска служб можно выполнить и в любой момент работы системы.

Например, чтобы запустить *web* сервер *apache*, достаточно выполнить команду

/etc/init.d/apache2 start

Если web сервер уже запущен вам выдаст сообщение, что сервис уже запущен.

Задать, чтобы сервер автоматически запускался при загрузке системы двумя способами. Выполнив команду **chkconfig** либо воспользовавшись **YAST**ом.

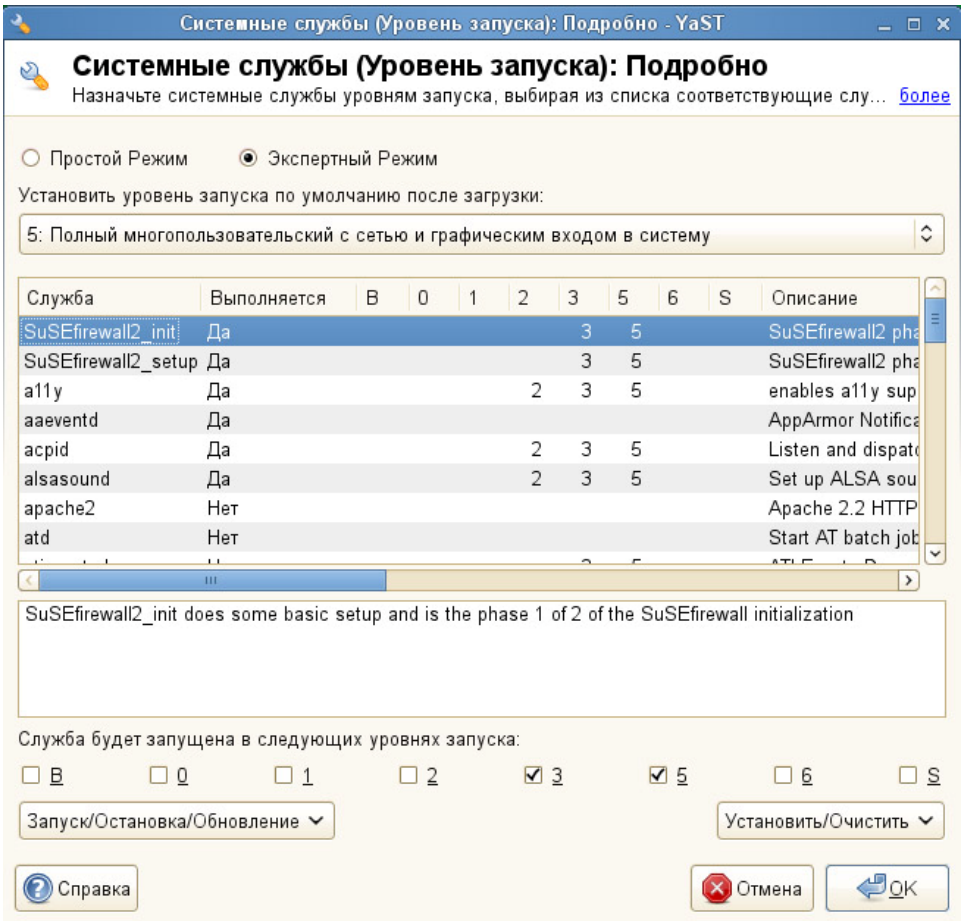
Консольная команда

chkconfig apache2 on

По умолчанию *apache2* будет стартовать на 3 и 5 уровне загрузки системы. Команда **chkconfig** позволяет, указать конкретно на каком уровне загрузки системы, должна стартовать служба.

Через **YAST**.

Запустите **YAST>Система>Системные службы**. Сразу переключите в экспертный режим он намного удобнее и нагляднее для работы.



4.4.Работа с пользователями.

Linux как и любая *UNIX*, является многопользовательской ОС. Что значит, что в системе одновременно могут работать несколько человек. Посмотреть, кто в данный момент работает в системе можно с помощью команды *who*.

Информация о пользователях, зарегистрированных в системе, находится в файле */etc/passwd*. Формат записей в этом файле следующий Семь полей разделенных двоеточием :.

имя_пользователя(login):полепароля:UID:GID:Полное имя:домашний каталог:оболочка

Рассмотрим значение полей в файле */etc/passwd* более подробно.

первое поле (имя пользователя, логин)	строка символов, обозначающая пользователя в системе.
второе поле (поле пароля)	в настоящее время в нем обычно стоит знак x, что означает, что для пользователя задан пароль. Пароли хранятся отдельно в файле */etc/shadow*. Иногда файл хранящий пароли может иметь другое время.
третье поле (UID)	Это номер пользователя в системе. Именно с **UID** работает система.
четвертое поле (GID)	номер группы, которой принадлежит пользователь. Имена и номера групп задаются в файле */etc/groups*
пятое поле (Полное имя)	Обычно поле содержит информацию о настоящем имени пользователя, и информацию о нем.
шестое поле (домашний каталог)	содержит имя домашнего каталога. Например */home/user1*
седьмое поле (оболочка)	командный интерпретатор, запускающийся при входе пользователя в систему.

Пароли в системах *UNIX* хранятся в зашифрованном виде.

В системах *UNIX* есть пользователь, который называется *root* и имеет **UID,GID** равный 0. Пользователь *root* имеет полномочия администратора из под его логина можно выполнять любые команды и получить доступ к любым файлам в системе. Даже если вы случайно запустите команду на удаление корневой файловой системы, команда будет беззаговорочно выполнена. Работать под пользователем *root* надо очень аккуратно. Лучше всего работать под обычным пользователем и только для выполнения ряда команд, которые требуют прав администратора, становится пользователем *root*.

Для того чтобы создать нового пользователя в системе существует команда **useradd**. Для ее выполнения требуются права *root*.

#useradd имя пользователя.

#passwd имя пользователя.

При добавлении нового пользователя автоматически создается его домашняя папка, в которую переписываются файлы из каталога */etc/skel*. В каталоге присутствуют специально созданные администратором служебные конфигурационные файлы для пользователей.

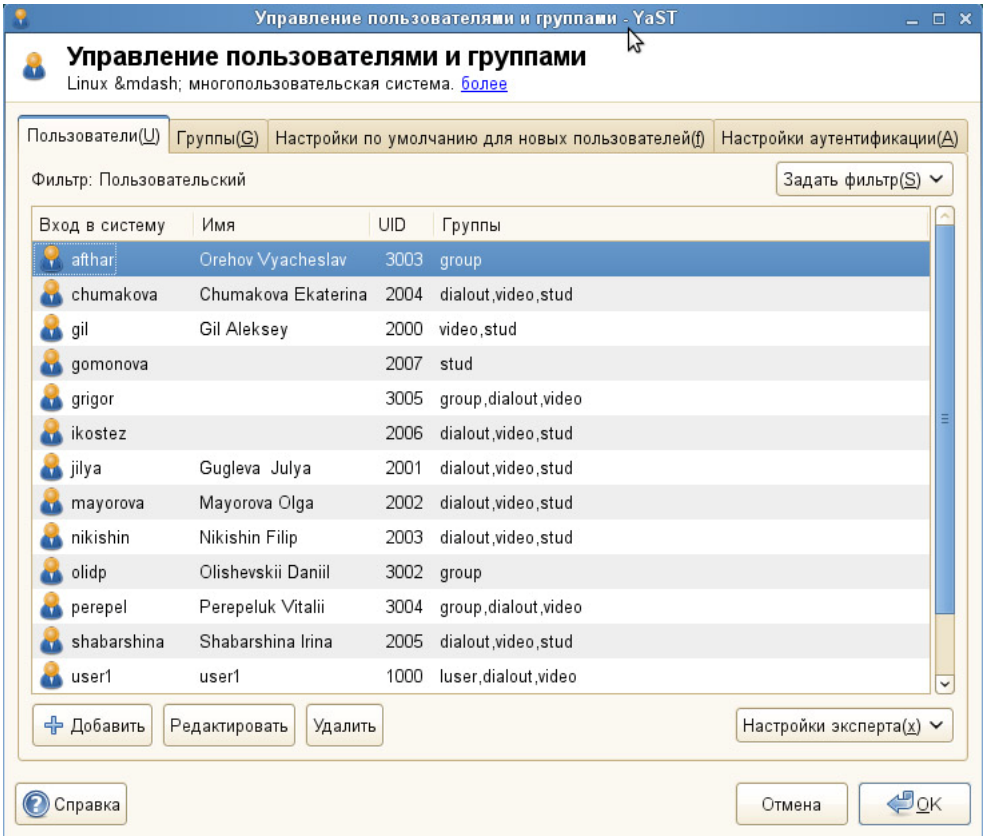
Команда **passwd** устанавливает пароль для пользователя. Не допускайте ситуаций, когда в системе существуют пользователи, для которых не задан пароль. Командой **passwd** может пользоваться не только root, но и обычный пользователь для того чтобы поменять свой пароль.

Команда **usermod** позволяет модифицировать запись о пользователе в файле **/etc/passwd**.

Команда **userdel** удаляет пользователя из файла */etc/passwd* и из системы.

Пользователей в *UNIX* системах объединяют в группы. Группы позволяют более эффективно управлять правами доступа в системе. Чтобы создать новую группу можно отредактировать файл */etc/groups*.

Но есть более удобный способ работы с пользователями и группами. Достаточно воспользоваться конфигураторами **YAST**. Запустите **YAST** и выберите пункт **безопасность управление пользователями и группами**.



4.5. Файлы устройств.

В *Linux* существуют специальные файлы устройств, позволяющие работать с устройствами как с обычными файлами. Находятся файлы устройств в директории */dev*(сокращенно от *devices*) . Для *Linux* нет разницы между файлом и устройством.

Примеры файлов устройств.

Файлы */dev/sd<x>* - файлы устройств жесткого диска (*SATA* или *SCSI*), *<x>* буква, обозначающая порядковый номер подключения к шине.

Например, *SATA* диск имеющий самый меньший номер подключения к шине будет иметь имя файла устройства */dev/sda*

Файлы */dev/sd<x><Номер>* - файл устройств, для разделов жесткого диска. Первый раздел диска, обычно содержащий корневую файловую систему, будет иметь имя */dev/sda1*.

Файлы */dev/ttyS<Номер>* - файлу устройств для последовательных портов.

Файлы устройств бывают двух типов: блочные и символьные. Основное отличие между блочными и символьными устройствами заключается в способе обмене информацией. Обмен информацией с блочными устройствами осуществляется блоками, с символьными отдельными символами(байтами).

4.6. Монтирование файловых систем.

В UNIX-подобных системах для того чтобы иметь возможность работать с файловой системой необходимо ее примонтировать к корневой файловой системе. Например, подключив к компьютеру флешку, необходимо сначала смонтировать файловую систему флешки, и только после этого она станет доступна для работы. Во всех дистрибутивах *Linux* это делается автоматически. При удалении ее из компьютера файловую систему обязательно необходимо размонтировать, иначе есть вероятность ее испортить, что грозит возможной потерей данных.

Все разделы диска монтируются автоматически при загрузке системы, и при выключении также автоматически отмонтируются.

Команда монтировки выглядит следующим образом

mount [опции] <устройство> <точка монтирования>

<точка монтирования> - пустой каталог, через который будет осуществляться доступ к файловой системе устройства.

<устройство> - это файл устройств, через который будет осуществляться доступ к подключаемому устройству. Если подключается сетевая файловая система, то вместо файла устройства указывается следующая конструкция *сервер:/каталог*, где сервер – это ip адрес либо имя машины, каталог имя ресурса который экспортирует сервер.

Например, команда монтирования *cdrom*. Файлу устройств для CD и DVD имеют название */dev/sr<Номер>*. Если устройство чтения компакт дисков единственное в системе, то его файл устройств будет иметь название */dev/sr0* и команда монтирования будет выглядеть следующим образом.

mount /dev/sr0 /media/cdrom

Рассмотрим еще один часто встречающийся случай. На компьютере установлено две системы, *Windows* и Linux и необходимо получить доступ к разделу диска *Windows* системы.

mkdir /mnt/win_d

Создаем папку, куда будем монтировать раздел *windows*.

Если неизвестно номер раздела windows на диске, то это можно посмотреть с помощью команды

fdisk –l /dev/sda

В нашем случае раздел *Windows* имеет второй номер и ему соответствует файл устройства */dev/sda2*. Тогда команда монтирования будет следующей

mount /dev/sda2 /mnt/win_d

У команды **mount** достаточно много опций, но чаще всего используются следующие

-t <тип файловой системы> позволяет явно указать какая файловая система создана в монтируемом разделе. Наиболее популярные системы *ext2,ext3* файловые системы *Linux*, *iso9660* – файловая система CD-ROM, *vfat* файловая система *FAT* и *FAT32*, *ntfs* – основная файловая система *Windows* начиная *Windows XP*.

-r монтирует файловую систему в режиме чтения

-w в режиме записи.

-a монтирует все файловые системы, перечисленные в файле */etc/fstab*

4.7.Монтирование файловых систем при загрузке.

При загрузке сценарий подключение разделов жесткого диска считывает данные из файла **/etc/fstab** и подключает перечисленные там жесткие диски или разделы. Сменные носители, такие как флеш, или CD-ROM подключаются автоматически.

/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part2 swap swap defaults 0 0

/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part1 / ext3 acl,user_xattr 1 1

/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part4 /space ext3 acl,user_xattr 1 2

/dev/disk/by-id/ata-Hitachi_HDT725050VLA360_VFK411R4DP10PL-part3 /usr ext3 acl,user_xattr 1 2

proc /proc proc defaults 0 0

sysfs /sys sysfs noauto 0 0

debugfs /sys/kernel/debug debugfs noauto 0 0

usbfs /proc/bus/usb usbfs noauto 0 0

devpts /dev/pts devpts mode=0620,gid=5 0 0

host:/export /export nfs defaults,noauto 0 0

Формат записи в */etc/fstab* следующий.

<устройство> <точка монтирования> <тип_ФС> <опции> <флаг РК> <флаг проверки>

<устройство> <точка монтирования> имеют тот же смысл что и в команде **mount**.

<тип_ФС> тип файловой системы.

<флаг РК> флаг резервного копирования, если стоит 1 то команда dump при создании резервной копии файловых систем, будет создавать копию этого раздела. 0 соответственно резервная копия файловой системы создаваться не будет.

<флаг проверки> определяет будет ли файловая система проверяться командой **fsck**.

<опции> опции передаваемые команде монтировании. Наиболее часто использующиеся опции приведены в таблице.

Опция	Значение
auto	Файловая система должна монтироваться автоматически при загрузке. Режим автоматической монтировки для файловых систем перечисленных в /etc/fstab задан по умолчанию, указывать опцию нет необходимости.
noauto	Файловая система не монтируется автоматически.
defaults	Используется стандартный набор опций.
exec	Разрешается запуск исполнимых файлов с подключаемой файловой системы. Задана по умолчанию.
noexec	Запрещает запуск исполнимых файлов.
ro	ФС монтируется в режиме только чтения.
rw	Монтируется в режиме чтения записи. Для файловых систем поддерживающих запись устанавливается по умолчанию.
user	Файловую систему разрешает монтировать от монтировать обычному пользователю.
nouser	Подключать отключать файловую систему имеет право только root.

Можно заметить, что в файле */etc/fstab* имена файлов устройств не */dev/sda<Номер>*. В */etc/fstab* имена файлов устройств используется так называемые длинные имена дисков. Длинные имена дисков основаны на так называемых *Universal Unique Indeficator(UUID)*. Использование коротких имен более намного проще, но использование длинных имен надежнее.

Рассмотрим ситуацию, в */etc/fstab* используются короткие имена дисков, вы добавили новый жесткий диск в компьютер и воткнули его в контролер с меньшем порядковым номером, чем диск с установленной системой. В результате система перестанет загружаться, потому что теперь файлы устройств */dev/sda1* будут соответствовать новому диску. Если в */etc/fstab* используются длинные имена, то подобного не произойдет. Поэтому длинные имена еще называются постоянными именами.

Посмотреть соответствие коротких и длинных имен можно командой

ls –l /dev/disk/by-uuid.

4.8.Установка программного обеспечения.

Программы в ОС *Windows* устанавливаются с помощью мастера установки. Для каждой программы используется свой собственный мастер установки, поставляющийся с дистрибутивом программы в виде файла *Setup.exe* или *Install.exe*.

В *Linux* работа с программным обеспечением сделана иначе. Существует два способа установки программ, из пакетов и из исходных кодов.

Пакет это файл, содержащий в себе все необходимое для установки и работы программы. В пакет обычно состоит из программы, конфигурационных файлов, файлов справки, необходимых библиотек. Пакет это не просто архив содержащий программу, но программы установки и удаления пакета, зависимости, конфликты. Зависимости определяют, какие еще программные продукты должны быть установлены в системе, чтобы программа из устанавливаемого пакета нормально работала. Конфликты определяют, с какими программными продуктами данный пакет будет конфликтовать. В названии пакетов содержится и некоторая полезная информация.

<имяпрограммы>-<версияпрограммы>-<выпуск пакета>.<архитектура>.rpm

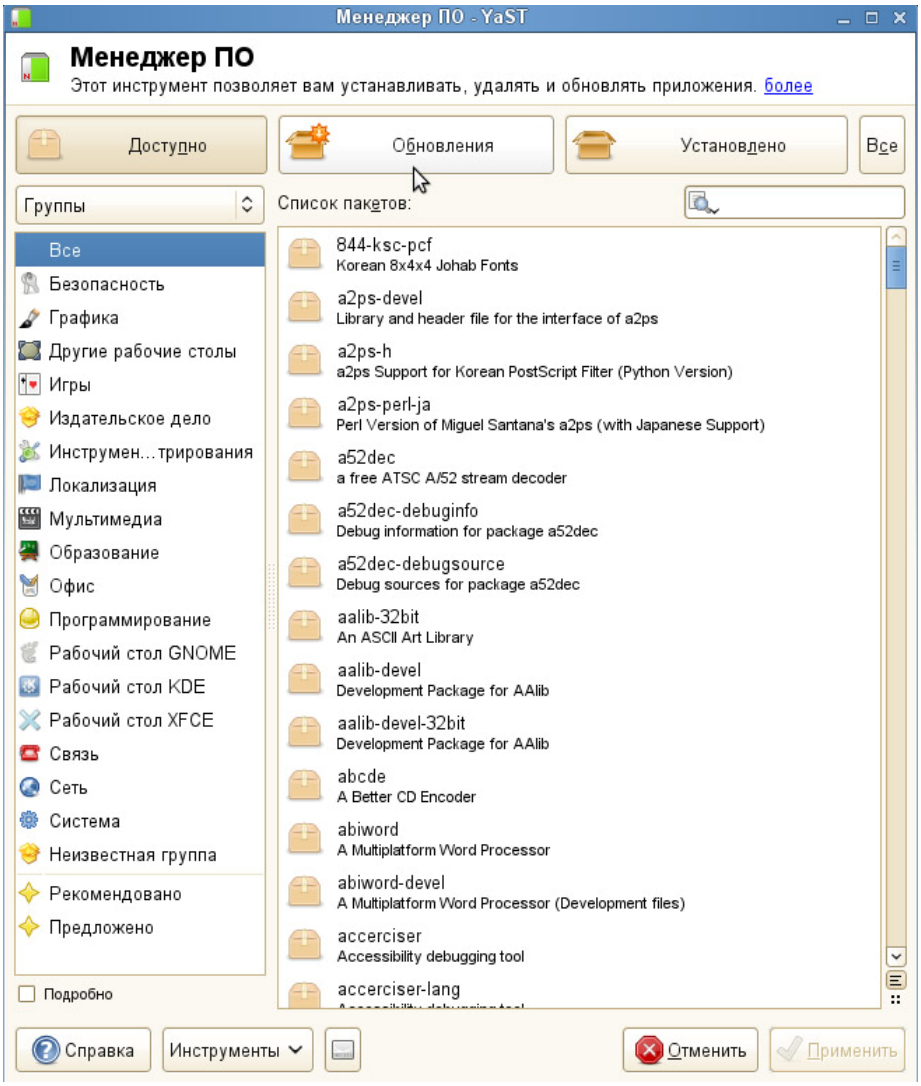
В *openSUSE*, как и во многих дистрибутивах используются пакеты формата *rpm*.

Для дистрибутивов *Linux* пакеты собираются в специальных хранилищах называемых репозиториями. Репозитории могут быть локакльными, CDROM, жесткий диск, а могут быть сетевыми. Хранить пакеты в одном месте, очень удобно с точки зрения управления. Легко можно увидеть какие программы обновились.

Файлы, хранящие данные о репозиториях находятся в */etc/zypp/repos.d*

Добавлять новые репозитории можно в ручную, с помощью команды **zypper**, или **YAST**.

YAST предоставляет наглядный графический интерфейс для управления пакетами программного обеспечения установленными на компьютере. **YAST | Программное обеспечение | Управление программным обеспечением** запустит менеджер ПО.



В верхней части окна имеются четыре кнопки, управляющие отображением пакетов ПО.

- **Доступно** – отображает пакеты, доступные для установки.
- **Обновления** – показывает список пакетов для которых доступны обновления.
- **Установлен** – список установленных пакетов.
- **Все** – показывает все пакеты.

Есть возможность отфильтровать пакеты по группам, можно ввести в окно поиска и останутся и будут отображены только те пакеты в названии или описании встречается искомая строка. Чтобы установить необходимый пакет выберите его в списке и нажмите кнопку установить. Пакет отобразиться в правой части окна. Если для работы пакету требуются дополнительное программное обеспечение оно тоже отобразиться в правой

части под именем пакета. Пиктограммка устанавливаемого пакета будут заменена на изображение жесткого диска с синей стрелкой. Но пока этот пакет еще не установлен.

Только когда вы нажмете **Применить**, пакет установится. До нажатия кнопки **Применить** вы можете удалять пакеты из списка или добавлять новые.

Рассмотрим консольную команду **zypper**. Формат

zypper <команда> пакеты.

Наиболее часто используемая команда это "**zypper update**", которая загружает из сети и устанавливает обновления тех компонентов и сервисов, которые были установлены ранее в вашей системе. Другие команды **zypper**.

zypper lr	Отображает перечень репозитариев (источников установки).
zypper ar	Добавление репозитария
zypper refresh	Обновление репозитариев
zypper rr	Удаление репозитария

zypper mr	Изменение репозитариев
zypper nr	Переименование репозитария
zypper install	Установка пакетов
zypper remove	Удаление пакета
zypper up	Обновление пакетов
zypper se	Поиск пакетов
zypper info	Получение информации о пакете
zypper lp	Список необходимых патчей
zypper patch	Применение патчей
zypper patches	Просмотр всех патчей
zypper lu	Получить список доступных обновлений командой
zypper up	Обновления установленных пакетов с новыми доступными пакетами

А как установить пакет если он не входит в репозиторий и был скачен из интернета в виде отдельного rpm файла. Для этих целей в *openSUSE* и других подобных дистрибутивах основанных на *Red Hat* менеджере пакетов используется команда *rpm*.

rpm -КлючРежима [дополнительные ключи] [параметры]

КлючРежима, указываемый первым, определяет режим работы. Самые часто используемые режимы перечислены в таблице.

Основные варианты вызова rpm

Команда	Назначение
rpm -i *файл-пакета.rpm*	Установка пакета (**i**nstall)
rpm -U *файл-пакета.rpm*	Обновление пакета (**U**pgrage)
rpm -e *пакет*	Удаление пакета (**e**rase)
rpm -q *пакет*	Получение информации (**q**uery)
rpm -y *пакет*	Проверка пакета (verif**y**)
rpm –b	Создание пакета .rpm из .src.rpm (**b**uild); здесь не рассматривается

Примеры.

Команда "**rpm -qi**" (info) выдает сводку информации о пакете - название, версия, объем и т.д., плюс краткую аннотацию.

rpm –qi <имя пакета>

Список файлов пакета - **rpm –ql <имя пакета>**

Для получения списка файлов используется ключ **"-l"** (list):

Часто возникает необходимость узнать, какому пакету принадлежит какой-то файл (например, чтобы знать, где искать к нему документацию). Для этого можно воспользоваться ключом "**-f**" (**f**ile).

4.9.Журналы системы.

UNIX системы и сервисные программы, очень “болтливы”. Во время своей работы они выдают множество сообщений, о своем состоянии и своих действий. Часть сервисов, например, таких как *http* демон *apache* записывает информацию в свои собственные лог файлы. Другие программы используют для вывода специальный демон протоколирования *syslog*. В *openSUSE* используется демон протоколирования *syslog-ng(syslog Next Generation*).

Все журналы в системе записываются в специальный каталог */var*. Например, **apache** хранит свои лог файлы в папке */var/log/apache2*. Служба *syslog-ng* записывает всю информацию, получаемую от системы и служебных программ в файл */var/log/messages*. Именно в */var/log/messages* необходимо искать сообщения о работе той или иной службы, если она не ведет свой собственный лог файл.

4.10. Безопасность.

UNIX-подобные системы изначально проектировались, как многопользовательские поэтому система безопасности в них достаточно надежная. Но любую систему можно обойти и *UNIX* системы не являются исключением. Все атаки в UNIX системах направлены на получения прав суперпользователя *root*. Став *root* и получив его права, злоумышленник может сделать с компьютером все что угодно. Но для этого нужно получить доступ на компьютер, атаковать удобнее изнутри, чем снаружи. Поэтому не давайте доступ к вашему компьютеру случайным людям. Проводите беседы с пользователями, регистрирующимися на компьютере, чтобы они не передавали свой логин и пароль третьим лицам. Закройте сетевые протоколы удаленной работы с компьютером, где используются открытая(не зашифрованная) передача пароля. Но чаще всего атака на компьютер начинается из сети. Злоумышленник с помощью программ сканеров пытается определить те сервисы, которые запущены на компьютере и воспользовавшись известными уязвимостями в сервисах, попытаться получить доступ к вашему компьютеру. Поэтому необходимо постоянно следить за обновлениями, появляющимися для установленных у вас программных продуктов. И конечно чтобы предотвратить атаки из сети необходимо запустить брандмауэр.

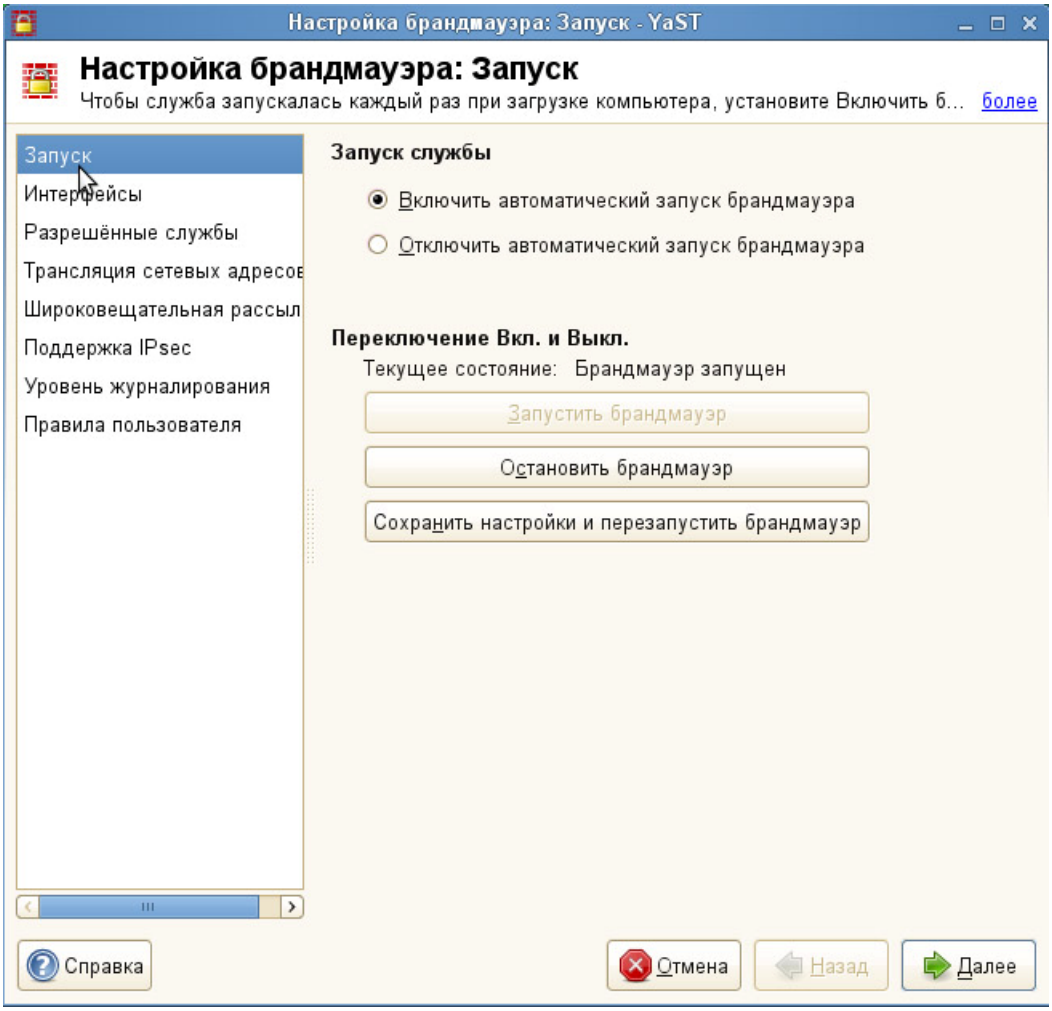
Брандмауэр он же *firewall*, межсетевой экран позволяет защитить ваш компьютер и вашу сеть от вторжения из вне. Принцип работы межсетевого

экрана фильтрации пакетов в соответствии с заданными правилами. Если пакет удовлетворяет правилам то проходит внутрь, если нет, то отбрасывается. По умолчанию *firewall* отбрасывает все входящие сетевые соединения, кроме тех, что вы лично разрешили.

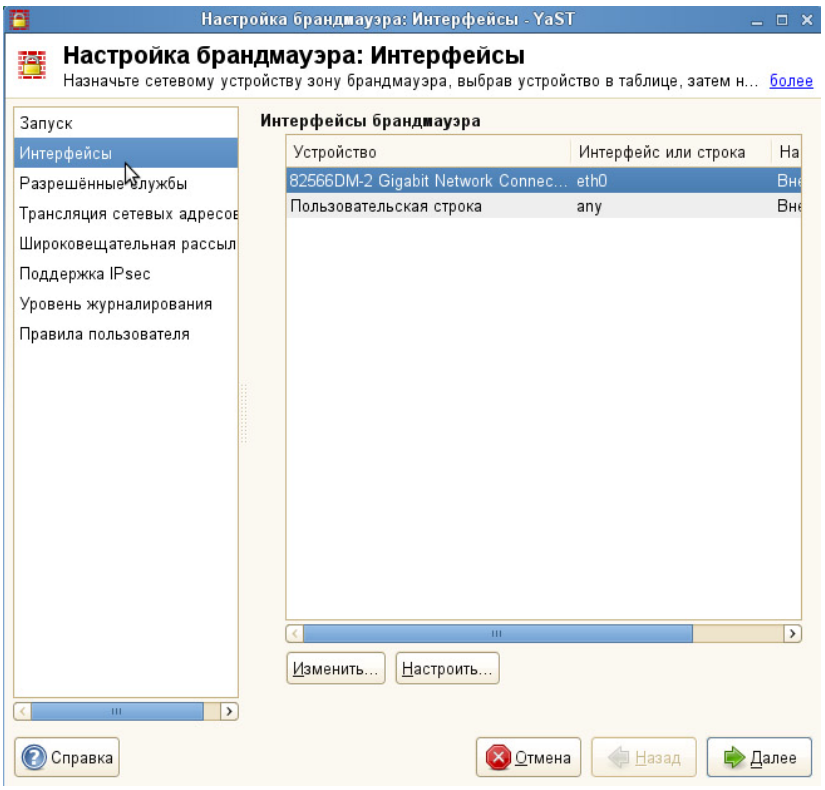
В состав дистрибутивов *Linux* входит пакет **iptables**. **Iptables** межсетвой фильтр с огромными возможностями. Правильно задав правила(фильтры) можно полностью оградить компьютер и сеть от вторжения извне.

Внутри **YAST** есть конфигуратор для настройки брандмауэра **iptables**. Все на что способен **iptables** можно оценить только при работе с командной строкой, но конфигуратор дает возможность быстро решить большинство задач, с которыми пользователь сталкивается, при администрировании *firewall*.

В разделе **Запуск**. Можно разрешить или запретить автоматический запуск брандмауэра, остановить его или перезапустить.



В разделе **Интерфейсы** перечислены все имеющиеся сетевые интерфейсы.



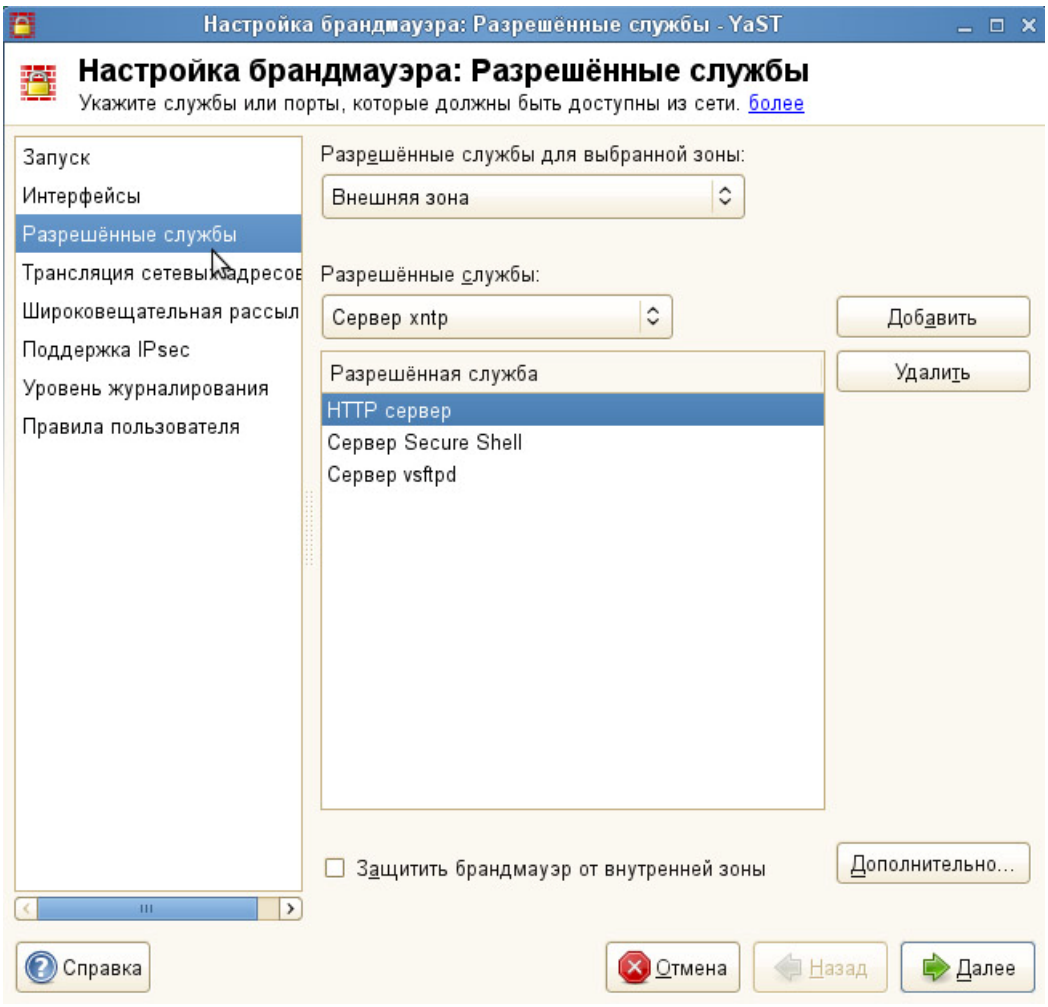
Если на компьютере есть несколько сетевых интерфейсов, то для каждого интерфейса можно присвоить какой зоне он относится.

- **Внутренняя зона** – интерфейс связывает компьютер с внутренней сетью.
- **Внешняя зона** – интерфейс связывает компьютер с Интернетом.
- **Демилитаризованная зона** – сеть не связанная с интернетом ни как.

Для каждой зоны можно настроить свои правила и фильтры.

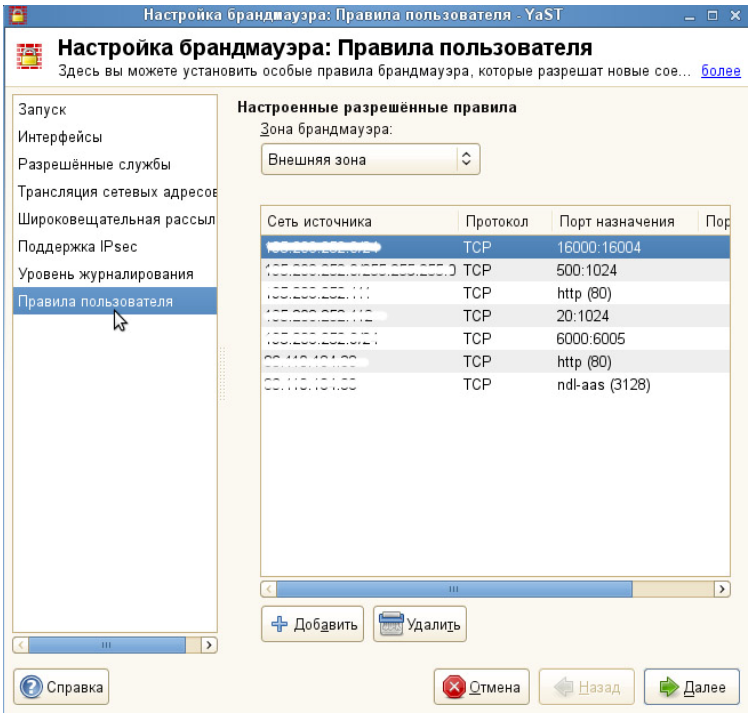
Зоны необходимо задавать, когда вы настраиваете *firewall* на шлюзе.

Разрешенные службы раздел где можно открыть доступ к отдельным сервисам запущенным у вас на компьютере. Например, если у вас на компьютере запущен *web-сервер*, то необходимо разрешить к нему доступ иначе информацию находящуюся на сервере никто не увидит.



Раздел **трансляция сетевых адресов**. Позволяет компьютерам из вашей локальной сети получить доступ к интернету.

Раздел **Подержка IPsec**. **IPsec** – это шифрованное соединение между доверенными узлами или сетями, через не доверенные.



Раздел **Правила пользователя** здесь можно задать свои собственные правила. Например, разрешить полный доступ к компьютеру с заданного IP адреса.

Контрольные вопросы и задание.

1. Какие способы установки openSUSE существуют?
2. Где взять дистрибутив системы?
3. Сколько основных разделов может быть на одном жестком диске?
4. Как определить размер раздела swap в системе?
5. Какой максимальный размер файла и файловой системы поддерживает ext3?
6. Кто такой суперпользователь?
7. Как из графического режима переключиться на консоль и вернуться назад?
8. Какой командой можно выключить компьютер?
9. Какой командой можно сбросить систему?
10. В какой папке в системе хранятся основные конфигурационные файлы?
11. Как называется конфигуратор в openSUSE?
12. Какая роль загрузчика?
13. Где располагается конфигурационный файл загрузчика GRUB?
14. Что задает параметр default в конфигурационном файле GRUB?
15. Что задает параметр initrd в конфигурационном файле GRUB?
16. Какой процесс порождается первым после загрузки ядра?
17. Где находится конфигурационный файл для загрузки системы?
18. Что такое уровень загрузки системы?
19. Что означает буквы S и K в ссылках на сценарии запуска системных служб?
20. Как в ручную запустить службу?
21. Где хранится информация о пользователях в системе?
22. Какая информация о пользователях хранится в файле паролей?
23. Как можно добавить пользователя в систему?
24. Что такое файлы устройств?
25. Каких двух типов бывают файлы устройств?

26. Команда mount зачем она нужна?
27. Перечислите необходимые параметры команды mount?
28. Как система при загрузке определяет, какие разделы диска, куда надо монтировать?
29. Что такое длинные и короткие имена в файлах устройств разделов диска?
30. Чем отличается установка программ под Windows и Linux?
31. Что такое репозиторий?
32. Какие способы работы с репозиториями предусмотрены в системе?
33. Как установить пакет не входящий в репозеторий?
34. Где в системе находятся журналы служб?
35. Какие задачи решает межсетевой фильтр?

5. Заключение.

Данное учебное пособие охватывает только небольшой объем знаний по ОС *UNIX*. Материал подобран с такой целью, чтобы сделать упор на особенности системы и чем *UNIX* подобная система отличается от ОС *Windows*. Но любая теория не является самодостаточной сама по себе, необходима еще и практика, пока вы сами не начнете работать с системой, решать конкретные задачи и набивать шишки вряд ли сможете полноценно овладеть *UNIX подобными* системами. Не бойтесь в интернет достаточно много статей и рецептов по решению проблем возникающих при работе на *UNIX подобных* системах. В этом плане проблем с документацией у вас не будет. Но если вы хотите стать действительно хорошим администратор *UNIX*, вам необходимо будет еще освоить и программирование на Си. Хотя сейчас политика *Unix-подобных* систем направлена на то чтобы сделать систему, удобную настолько, чтобы любой пользователь с ней мог работать и выполнять задачи администратора, но *UNIX* системы разрабатывались программистами для программистов, и только человек, хорошо, разбирающийся в программировании чувствует себя в системе комфортно.

Но не думайте, что *UNIX* подобные системы являются какой-то экзотикой, нет если в плане работы с конечным пользователем, *UNIX* уступает ОС *Windows*, то при выполнении серверных функций на порядок превосходит и альтернативы им нет. Недаром все ведущие компьютерные компании мира имеют свои *UNIX* системы, которые поставляются вместе с серверами. А сейчас почти все компьютерные компании имеют свои собственные *Linux* проекты. Очень часто можно услышать, в споре о надежность Linux и Windows, такой аргумент ядро Linux пишут студенты в свободное от учебы время и соответственно оно содержит кучу ошибок. На

самом деле, если взять статистику по внесению правок в ядро, окажется что 90% всех исправлений вносятся такими компаниями как *Intel*, *IBM*, *AMD*, *Novell* и т.д. Поэтому вам не придется жалеть о потраченном времени на освоения *UNIX*, если вы серьезно намерены работать в области IT, рано или поздно вы столкнетесь с Unix-подобными системой.