**КОМАНДНЫЙ ИНТЕРФЕЙС ОС Linux**

**ВВЕДЕНИЕ**

В настоящее время взоры пользователей персональных компьютеров, профессиональных программистов, системных администраторов и руководителей IT-служб все чаще обращаются к свободному программному обеспечению (СПО). Стоимость СПО значительно меньше стоимости коммерческого программного обеспечения. Распространение СПО может также происходить бесплатно на основе открытых лицензий.

Рынок современных Linux-подобных операционных систем (ОС) достаточно насыщен. Широкая маркетинговая кампания по популяризации Linux-систем порождает тенденцию к переходу на использование их в качестве базовой ОС. Поэтому все более актуальным становится изучение свободно распространяемой операционной системы Linux. Эта система является многозадачной и многопользовательской ОС с богатым пользовательским интерфейсом (графическим и командным).

Данное пособие может облегчить первоначальное освоение командного интерфейса ОС Linux пользователям, переходящим на эту систему с ОС Windows. Командный интерфейс Linux намного богаче возможностями, имеется большое количество дистрибутивов, источников информации о системе также много, поэтому сложно сориентироваться начинающему пользователю. В пособии изложены основные понятия файловой системы, возможности использования команд в командной строке и командных файлах (файлах-скриптах).

Контрольные вопросы, задания и приведенные тесты помогут в самостоятельной подготовке. Пособие предполагает наличие у студентов элементарного знания персонального компьютера и навыка работы в среде Windows на уровне пользователя в графическом и командном интерфейсе, что облегчит восприятие информации об ОС Linux по аналогии с ОС Windows.

**1 ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА LINUX**

Отдельной, фактически самодостаточной и ни от чего не зависящей, ветвью Unix-подобных ОС является Linux. Именно поэтому иногда говорят, что Linux – это не совсем Unix. История развития Unix началась в 1969 г. в Bell Telephone Laboratories, являвшейся подразделением компании AT&T. В работе со стороны Bell Telephone Laboratories приняли участие Кен Томпсон и Деннис Ритчи. ОС Unix стоит на трех китах: языке Си, файловой системе, командном языке. В дальнейшем к ним добавились система X Window и протоколы TCP/IP. Язык Си, на котором написана сама операционная система, с одной стороны, сочетает в себе свойства языка высокого уровня: описание типов, программные структуры if, for, while и т.п., а с другой - содержит средства, присущие обычно языкам уровня ассемблера: регистровые переменные, адресную (ссылочную) арифметику, возможности работы с полями бит и отдельными битами, и т.п. В августе 1991 г. финский студент Линус Торвальдс разместил в Internet скромное сообщение о том, что он разработал собственную ОС, которая получила впоследствии название Linux. Линус Бенедикт Торвальдс родился в Хельсинки в 1970 г. Еще в возрасте 10 лет он начал увлекаться программированием, активно работая на своем домашнем компьютере Commodore VIC-20. В 1989 г. на конференции ассоциации Usenix в Торонто представители корпорации AT&T объявили о новой системе цен на UNIX System V: около 40 тыс. долл. в расчете на один процессор (7,5 тыс. долл. для учебных заведений). Профессор Амстердамского университета Эндрю Таненбаум в поисках доступной по цене ОС занялся написанием Minix, усеченной версии UNIX, способной работать на ПК. Весной 1991 г., уже будучи студентом факультета вычислительной техники университета в Хельсинки, Линус Торвальдс решил на основе Minix разработать эффективную версию UNIX для платформы IBM PC (процессор i386), разобраться в том, как работает ОС, просто переписав ее ядро. Самого Торвальдса несколько смущало созвучие названия ОС “Linux” с его именем, поэтому он пытался назвать свою разработку FREAX. Но Ари Лемке, который предоставил место для системы на FTP сайте, назвал каталог pub/OS/Linux. Это название и закрепилось за новой ОС. В 1991 г. в Интернете была опубликована версия 0.1, этот факт стал решающим в дальнейшей судьбе Linux. Уже с самого начала Торвальдс получил несколько заинтересованных откликов. Первоначально Линус Торвальдс не хотел продавать свою разработку, что было установлено в уведомлении об авторских правах, распространяемом вместе с Linux. Причем требование Линуса налагало значительно более жесткие ограничения на распространение Linux, чем те, которые провозглашались в лицензии GNU: не разрешалось взимать никаких денег за передачу или использование Linux. Но уже в феврале 1992 г. к нему стали обращаться за разрешением брать плату за распространение дискет с Linux, чтобы покрыть временные затраты и стоимость дискет. Кроме того, необходимо было считаться и с тем, что при создании Linux использовалось множество свободно распространяемых по Интернету инструментов, самым важным из которых был компилятор GCC. Авторские права на него оговорены в Универсальной общественной лицензии (GPL), которую изобрел Ричард Столман. Торвальдсу пришлось пересмотреть свое заявление об авторских правах, начиная с версии 0.12, он перешел на использование лицензии GPL. Немаловажно отметить, что разрабатываемая многочисленными энтузиастами по всему миру система Linux не является однородной завершенной системой. Однородным в Linux является только ядро, поскольку поддерживается до сих пор единственным человеком, ее родоначальником Линусом Торвальдсом. Поэтому версия самой ОС определяется только по версии ее ядра. Первые версии Linux, появившиеся в 1991 г., размещались на двух дискетах. Первая дискета была загрузочной и содержала ядро, а вторая – корневую файловую систему и основные утилиты. Копии этих дискет можно было загрузить с сервера университета в Хельсинки. Конфигурирование и настройка системы производились вручную и были очень сложными. Поэтому до появления первых дистрибутивов установить Linux на свой компьютер мог только достаточно подготовленный специалист.

Вопросы к главе 1:

1. К какому семейству операционных систем относится Linux?

2. Кто и когда написал ядро Linux?

**2 КРАТКАЯ ХАРАКТЕРИСТИКА ДИСТРИБУТИВОВ LINUX**

С появлением дистрибутивов, включающих программу инсталляции, процесс установки Linux на персональный компьютер пользователя значительно упростился. Основным способом распространения этих дистрибутивов были комплекты дискет. Интернет только начинал распространяться, тем не менее все производители уже тогда размещали дистрибутивы на ftp-сайтах, а также на досках объявлений. Это был второй способ распространения Linux. Вскоре появился и третий, наиболее перспективный способ распространения Linux – на CD-ROM. Естественно, при этом появилась возможность добавить в дистрибутив массу дополнительных программ и документации, например систему X-Window, исходные коды программ, архивы документации с Интернет-сайтов, программное обеспечение от независимых производителей и многое другое. Стоит отметить, что нумерация дистрибутивов не была никак связана с нумерацией версий ядра. Например, версия дистрибутива могла иметь вид “the Fall 1993 release” или “the 2.0 release”, хотя еще не было выпущено даже ядро версии 1.0. Иногда это приводило к некоторой путанице. Сколько существует дистрибутивов Linux, столько существует и различных его вариантов, отличающихся способом установки, конфигурации, структурой организации дерева каталогов, размещением конфигурационных файлов, набором программных компонент и библиотек, коллекцией дополнительных программных пакетов, их названиями, комплектом сопроводительной документации и т.д. Наиболее известными на данном этапе являются следующие дистрибутивы: Debian, RedHat, Slackware, SUSE, Mandriva. Столь большое количество дистрибутивов системы является следствием ее большой популярности, которая положительно влияет на развитие самой ОС.

**2.1 Mandriva Linux**

Mandriva Linux (ранее известный как Mandrakelinux) – это французский дистрибутив, основанный Гаелем Дювалем (Gael Duval) [1]; завоевал большую популярность среди пользователей, привыкших к Windows или не желающих что-либо знать о внутренней структуре и принципах работы операционной системы, а также среди тех, кто хочет просто удобного и простого рабочего стола. Данный проект имеет генетические корни от дистрибутива RedHat, хотя сейчас Mandriva ничего общего с ним не имеет. Разработчики по умолчанию в качестве рабочего стола выбрали KDE и добавили простой графический инсталлятор, позволяющий установить операционную систему с минимальными знаниями о компьютере. Встроенные средства определения оборудования и работы с дисками считаются одними из лучших в промышленности. Mandriva Linux часто устанавливается и работает без проблем там, где другие операционные системы на запускаются. Кроме этого, обеспечивается высокий уровень удобства работы. Разработка дистрибутива является полностью открытой. Каждая версия проходит глубокое тестирование, что позволяет найти и исправить большое количество ошибок.

**2.2 Fedora Core**

Fedora Core – это самый используемый дистрибутив в мире. Разрабатывается при поддержке компании RedHat, пожалуй, наиболее крупной в мире Linux. Используется как тестовый полигон для внедрения новых технологий в RedHat Enterprise Linux [2]. Операционная система представляет смесь из консервативных и самых современных пакетов, построена на базе многих наукоёмких утилит, разработанных в компании. Пакеты не самые последние; после того, как была объявлена новая бета-версия, версия пакета замораживается за исключением обновлений, связанных с безопасностью. Результатом является хорошо протестированный и стабильный дистрибутив, программа бета-тестирования и средства составления отчётов об ошибках являются общедоступными, также существует несколько списков рассылок. RedHat Linux стал преобладающим дистрибутивом Linux на серверах по всему миру. Компания RedHat предоставляет комплекс услуг в области технической поддержки и сопровождения системы, а также проводит сертификационные курсы RHCE. Все это привело к тому, что на данный момент торговая марка компании - признанный бренд в мире ИТ-индустрии.

**2.3 Suse Linux**

Основан одноименной немецкой компанией, купленной впоследствии фирмой Novell. Отличительной особенностью является наличие собственного программного комплекса YaST2, позволяющего провести установку и настройку любых компонентов системы. До недавнего времени YaST2 весьма болезненно реагировал на попытки ручного конфигурирования системы, что могло привести в некоторых случаях к полной недееспособности системы. Однако в текущих версиях ручной и автоматический способ настройки функционируют приемлемо [3]. Дистрибутив Suse Linux включает также очень большой набор драйверов, в том числе и проприетарных (Проприетарные - содержащие программный код, лицензия которого запрещает копирование, модификацию и распространение. И они могут (и как правило) быть бесплатными. Ещё бывают "свободные", с лицензией, разрешающей копирование, модификацию и распространение). Сказывается направленность разработчиков на ноутбуки – практически стопроцентная гарантия установки и работы SuSe с любыми ноут-специфичными устройствами, включая WiFi, Winmodem&apsbi, видеокарты и т.д. Не случайно именно этот дистрибутив избирают в качестве основы производители, рискующие предустанавливать Linux на ноутбуки. Русификация в некоторых приложениях отсутствует или выполнена с ошибками.

**2.4 Debian GNU/Linux**

Debian GNU/Linux основан Айаном Мардоком (Ian Murdock) в 1993 г. Полностью некоммерческий проект, представляющий собой самую чистую форму идеалов, разработанных движением за свободное программное обеспечение. Интернациональная армия разработчиков принимает участие в разработке грамотно и строго организованного проекта, гарантирующего качество и высокую стабильность [4].

Имеются три ветки разработки: “stable” – официально выпущенный дистрибутив. Содержит не самые свежие, но хорошо протестированные и проверенные версии пакетов. Любые обновления, выходящие для этой ветки, – “secure-updates”, закрывающие критические проблемы безопасности;

“testing” – ветка, готовящаяся стать релизом. Не обеспечивается столь высокий уровень безопасности ветки “stable”, но имеет более свежие версии софта; “unstable” – это тестируемая ветка, более “сырая”, нежели “testing”, но имеющая самые свежие версии пакетов. Разработчики не придерживаются строгого графика выпуска новых версий. Новые дистрибутивы выходят только тогда, когда они готовы и хорошо протестированы.

Недостатком Debian для конечного пользователя можно считать отсутствие удобного графического инсталлятора, а также необходимость знания информации об аппаратуре компьютера. Поэтому этот дистрибутив имеет репутацию сложного в установке и использовании. Многие пользователи Debian шутят, что их инсталлятор потому такой плохой, что нужен он только один раз; после того, как Debian GNU/Linux установлен и работает, все последующие обновления любого масштаба могут быть выполнены с помощью утилиты apt-get. Русификация присутствует.

**2.5 Ubuntu/Kubuntu**

Ubuntu означает “гуманность по отношению к другим” в переводе с одного из африканских языков. И это действительно так – дистрибутив в первую очередь ориентируется на домашнего пользователя. Ориентирован исключительно на работу в графическом режиме, отсутствуют некоторые консольные утилиты, однако имеется возможность установки из сетевых депозитариев. Комплектуется ограниченным набором приложений по принципу «одна задача – один инструмент» [5]. Дистрибутив поддерживается компанией Canonical Ltd. Разработчики соблюдают высокую скорость выхода новых релизов – релиз-цикл составляет примерно полгода. Считается самым популярным Linux-дистрибутивом в мире. На это немало влияет высокая степень локализации дистрибутива. С технической точки зрения Ubuntu построен на Debian “unstable”, но с более свежими пакетами. Сохраняется возможность обновления программных пакетов напрямую из депозитариев Debian. Интересной особенностью является отключение по умолчанию учетной записи “root”, все действия по настройке и администрированию необходимо проводить через утилиту sudo, что на первый взгляд кажется неудобным, но позволяет избежать некоторых неприятностей.

Kubuntu отличается от своего собрата тем, что в качестве рабочего стола предоставляется KDE, а не GNOME. Соответственно набор программного обеспечения составляется на основе библиотеки QT, a не Gtk+.

**2.6 Gentoo Linux**

Проект основан Даниэлем Роббинсом (Daniel Robbins), основателем Stampede Linux и разработчиком FreeBSD. Им была привнесена система портов, похожая на систему автоматической сборки под названием “ports” в FreeBSD. Первый стабильный релиз был анонсирован в марте 2002 г. [6]. Gentoo является типичным представителем source-based дистрибутивов, то есть все программное обеспечение представлено в виде исходных кодов для установки. Этот дистрибутив не подойдет начинающему пользователю, однако в последнее время он приобрел довольно широкую популярность.

**2.7 Slackware Linux**

Проект, созданный Патриком Фолькердингом (Patrick Volkerding) в 1992 г., является старейшим в мире Linux [7]. В дистрибутиве отсутствуют графический инсталлятор и утилиты настройки системы. Любая настройка проводится посредством правки конфигурационных файлов. Из-за этого Slackware Linux рекомендуют людям, которые хотят глубоко изучить основы функционирования GNU/Linux.

**2.8Knoppix**

Разработка начата Клаусом Кноппером (Klaus Knopper) в Германии на базе Debian [8]. Этот дистрибутив однозначно поднял планку стандартов среди уже устоявшихся дистрибутивов. Его система автоматического определения оборудования, автоматическая загрузка, возможность установки системы на жесткий диск, саморазворачивающееся в памяти ПО, – все это сделало этот дистрибутив надежным помощником при восстановлении установленных ОС, тестировании оборудования компьютера или просто демонстрации возможностей Linux тем, кто его ещё не видел. Новые версии выпускаются приблизительно раз в несколько месяцев. Свежие пакеты берутся из нестабильной ветки Debian.

**2.9 CentOS**

CentOS – это дистрибутив, который полностью собран из исходных текстов RedHat Enterprise Linux [9]. Так как проект некоммерческий, не предоставляется никакой технической поддержки, но доступны любые обновления прародителя. Таким образом, этот дистрибутив подходит тем, кто хочет иметь мощь и стабильность RedHat Enterprise Linux без каких-либо затрат. Русификация аналогична Fedora Core.

**2.10 ASPLinux**

ASPLinux – один из самых известных и распространенных дистрибутивов на постсоветском пространстве [10]. Одноименная российская компания собирает Linux на основе Fedora Core (ранее RedHat Linux) и адаптирует его под региональные особенности. Это выражается в переводе контекстной справки к большому числу графических и консольных утилит. Отдельно стоит упомянуть систему справки (man) – превосходные переводы, которые оценят пользователи любых категорий. В качестве рабочего стола по умолчанию пользователю предлагается Gnome, конечно, не исключена возможность установки KDE. Широкий выбор приложений делает эту систему универсальной для любых категорий пользователей.

**2.11 ALTLinux**

Российский дистрибутив, разрабатываемый компанией ALTLinux [11]. Один из самых распространенных на территории стран СНГ. Переводная документация и русифицированные приложения делают этот дистрибутив превосходным для использования. По умолчанию в качестве рабочего стола предлагается KDE, возможна установка Gnome.

Вопросы к главе 2:

1. Перечислите наиболее распространенные дистрибутивы Linux.

2. Охарактеризуйте коротко несколько дистрибутивов Linux.

**3 ОБЛАСТИ ПРИМЕНЕНИЯ LINUX - ПОДОБНЫХ СИСТЕМ**

Несмотря на прогнозы экспертов о том, что Linux постигнет та же участь, что и другие превосходные технологии, преданные преждевременному забвению (AmigaDOS, Next и OS/2), Linux продолжает развиваться, проникая в сферу корпоративных сетей. Популярность Linux в научных и академических учреждениях не вызывает удивления. Но этой операционной системе доверяют и web-серверы. Она встраивается в микропрограммное обеспечение устройств контроля над промышленными процессами. Она часто применяется и для обслуживания таких корпоративных сервисов, как FTP, электронная почта, брандмауэры и proxy-серверы. Многие разработчики и администраторы Unix обратили свое внимание на Linux вследствие таких его преимуществ, как наличие инструментария разработки, многочисленные хорошо написанные книги и полный исходный код. Oracle и Informix объявили о планах перенести менеджеры баз данных на Linux. Netscape Communications, WordPerfect и Sybase уже предлагают готовые решения для Linux. Тенденцию использования Unix-систем поддерживают все больше производителей различных портативных цифровых устройств (карманных компьютеров и пр.). С 2001 г. компания Sony адаптировала Linux под платформу игровой приставки PlayStation 2. Первоначально Linux для PS2 была выпущена в японском варианте, а осенью 2001 г. компания анонсировала систему, превращающую игровую приставку в персональный компьютер: в дистрибутиве есть электронные таблицы, текстовый редактор, MP3-плейер – и все это реализовано под управлением графической системы X-Window. Действия Sony еще сильнее подчеркивают тенденцию к распространению Linux. Что касается рынка дистрибутивов, то, по мнению аналитиков, на рынке будут доминировать два дистрибутива – RedHat и SUSE, а их доли на рынке будут 65 и 35 % соответственно. Аналитики компании Gartner полагают, что ни претензии со стороны SCO, ни усилия корпорации Microsoft не смогут замедлить распространение Linux в ближайшие несколько лет [12]. Однако для Linux не все так безоблачно – по мнению Gartner, главные угрозы для Linux исходят из среды, которая породила открытую операционную систему. Экспертами были сформулированы 5 факторов, представляющих большую угрозу для ОС:

1) открытый код может получить и модифицировать любой желающий, в результате чего может возникнуть нежелательное ветвление программы;

2) большие затраты на техническую поддержку увеличивают общую стоимость владения;

3) увеличение числа открытых лицензий на программное обеспечение, находящихся в юрисдикции OSS, и, как следствие, сложность работы с ПО;

4) слишком частые обновления программ, в результате чего программы приобретают зависимости от других пакетов;

5) возможность патентования тех или иных лицензий или программ, в результате чего возрастает риск использования ПО.

Однако, несмотря на этот список, Gartner отмечает постоянно возрастающий интерес предприятий к открытой ОС. Сейчас можно наблюдать тенденцию в изменении использования Linux: если несколько лет назад открытая ОС использовалась на небольших серверах масштаба рабочих групп, то теперь Linux начинают использовать на серверах масштаба предприятия, доверяя этой ОС критически важные вычисления. “Скорее всего c мейнфреймов в различных дата-центрах Unix (дата-центр – центр обработки данных) будет вытеснена и на ее место придет Linux”, – говорит один из аналитиков Gartner, делая прогноз на ближайшие пять лет. – “Для той части ИТ-рынка, где применяются приложения среднего масштаба, развернется жесткая борьба между Windows и открытой ОС по очень многих параметрам”. Также существует большая вероятность того, что Linux изменит существующие сегодня системы оплаты за программное обеспечение. В будущем будет доминировать система подписки на то или иное программное обеспечение. По оценкам Gartner, в шести из десяти дата-центров будет применяться именно открытая ОС. Однако, предупреждают аналитики, не следует ждать массовой миграции с Windows. Отмечается также тот факт, что прогнозы делались исходя из той предпосылки, что с миром Linux не будет происходить каких-либо значимых неприятностей. Также аналитики отметили факт выхода открытой версии Solaris, но, по их мнению, это не окажет существенного влияния на предпочтения пользователей. В Garnter полагают, что если с открытой версией Solaris что-то и будет происходить, то это будет заметно через год-два. Рассмотрим теперь соотношение операционных систем Linux и Windows. По данным аналитического исследования, проведенного специалистами компании IDC [13], в настоящее время доля серверов с ОС от Microsoft составляет 36,8 % от общего объема продаж в денежном выражении, тогда как на долю Unix-серверов приходится 31,7 % систем. Впрочем, лидерство Microsoft может оказаться временным. Доля Linux растет значительно быстрее доли Windows. В денежном выражении доля серверов с этой ОС за год выросла на 34,5 %, а в штучном – на 20,5 %. Для серверов на базе Windows рост продаж составил 17,7 % в денежном и 15,3 % в штучном выражении. При сохранении такого соотношения темпов роста Linux имеет все шансы занять лидирующую позицию на рынке серверных ОС Linux.

Вопросы к главе 3:

1. Перечислите основные области применения ОС Linux.

**4 ОРГАНИЗАЦИЯ ФАЙЛОВОЙ СИСТЕМЫ ОС LINUX**

*Файловая система* (ФС) – это часть операционной системы, включающая:

совокупность всех файлов на диске;

служебные информационные структуры, включая каталоги;

 системные программные средства ОС, управляющие работой ФС.

**4.1 Структура логического раздела**

Unix-подобные ОС (в том числе Linux) используют несколько типов файловых систем (filesystems), отличающихся производительностью, функциональностью и надежностью хранения данных: s5, ufs, nfs, vxfs, ext2fs, ext4fs и др.

На примере одной Unix-подобной системы изучим основные понятия файловых систем, для этого рассмотрим структуру раздела, содержащего простейшую файловую систему s5 операционной системы Unix System V (рис. 1).

|  |
| --- |
| Загрузочный блок |
| Суперблок |
| Массив индексных дескрипторов |
| Область данных |

**Рисунок 1 - Структура раздела Linux**

**Загрузочный блок** содержит небольшую программу-загрузчик, обеспечивающую загрузку операционной системы.

В **суперблоке** находится общая информация о файловой системе, такая, как тип файловой системы; размер файловой системы в логических блоках, включая сам суперблок, массив дескрипторов и блоки данных; размер массива индексных дескрипторов; число свободных блоков, доступных для размещения; число свободных блоков для размещения дескрипторов; размер логического блока; список номеров свободных дескрипторов; список адресов свободных блоков и т.п.

В **массиве индексных дескрипторов** содержится пронумерованный список индексных дескрипторов (inode). Индексный дескриптор - это метаданные файла, содержащие информацию о файле. Каждый файл связан с одним inode, хотя может иметь несколько имен в файловой системе, каждое из которых будет указывать на один и тот же inode. Каждому индексному дескриптору соответствует номер в списке. Поля индексного дескриптора содержат следующую информацию:

- тип файла (обычный, каталог, специальный, ссылка и др.);

- права доступа к файлу;

- число ссылок на индексный дескриптор, т.е. количество имен, которые имеет файл в файловой системе;

- идентификаторы владельца файла и группы владельца;

- размер файла в байтах;

- временные характеристики (время последнего доступа к файлу, время последней модификации файла, время последней модификации inode);

- адресную информацию файла; и др.

**Адрес файла** – это информация о расположении на внешнем носителе данных файла. Поскольку дисковые блоки хранения данных файла могут располагаться непоследовательно, индексный дескриптор должен хранить номера кластеров, принадлежащих данному файлу. Так, например, в индексном дескрипторе файловой системы ufs адресная информация хранится в виде массива, который имеет фиксированный размер и состоит из 15 полей (по 4 байта). Первые 12 полей адресуют непосредственно номера кластеров файла. Тринадцатое поле адресует дополнительный кластер, в свою очередь содержащий адреса (номера) кластеров данного файла (это называется косвенной адресацией). Четырнадцатое поле указывает на дополнительный кластер, хранящий адреса других дополнительных кластеров, каждый из которых адресует кластеры данного файла (двойная косвенная адресация). Последний пятнадцатый элемент используется для тройной косвенной адресации, когда для нахождения адреса кластера файла используется три дополнительных кластера.

Такой подход позволяет при относительно небольшом фиксированном размере индексного дескриптора поддерживать работу с файлами, размер которых может изменяться от нескольких байт до десятка мегабайт. При размере кластера 1024 байта для относительно небольших файлов до 12 Кбайт используется прямая индексация, обеспечивающая максимальную производительность. Для файлов, размер которых не превышает 268 Кбайт (12Кбайт + 256\*1Кбайт), достаточно простой косвенной адресации. Наконец, при использовании тройной косвенной адресации можно обеспечить доступ к (12+256+2562 +2563) кластерам.

В **области данных** хранится информация, содержащаяся в файлах и каталогах, включая корневой каталог.

**4.2 Типы файлов**

Файловые системы поддерживают несколько функционально различных типов файлов:

- обычные файлы (ОС не контролирует содержимое этих файлов, к этим файлам относятся текстовые, исполняемые, графические файлы и т.д.);

- каталоги (это особые файлы, содержащие системную информацию о наборе файлов, зарегистрированных в них);

- специальные файлы (фиктивные файлы, соответствующие физическим устройствам ввода-вывода; эти файлы могут быть символьными и блочными);

- ссылки (мягкие, иначе символические, и жесткие); - и др.

***4.2.1 Файлы физических устройств***

С точки зрения ОС Linux все подключаемые к компьютеру устройства (жесткие и съемные диски, терминал, принтер, модем и т. д.) представляются файлами. Это позволяет пользователю и программам обращаться к внешним устройствам, не вдаваясь в подробности того, как они реализованы. Если, например, надо вывести на экран какую-то информацию, то система как бы производит запись в файл /dev/tty0l.

Физически внешние устройства бывают двух типов: *символьными* (или байт-ориентированными) и *блочными* (или блок-ориентированными). Различие между ними состоит в том, как происходит считывание и запись информации в эти устройства (обмен с оперативной памятью). Взаимодействие с символьными устройствами производится посимвольно, в режиме потока байтов. К таким устройствам относятся, например, терминалы. На блок-ориентированных устройствах информация записывается (и соответственно считывается) блоками. Примером устройств этого типа являются жесткие диски. На диск невозможно записать один байт или считать с него один байт, обмен с диском производится только блоками (по 512 байт). Взаимодействием с физическими устройствами в Linux управляют драйверы устройств, которые либо встроены в ядро, либо подключаются к нему как отдельные модули. Для взаимодействия с остальными частями операционной системы каждый драйвер образует коммуникационный интерфейс, который выглядит как файл. Большинство таких файлов для различных устройств как бы "заготовлены заранее" и располагаются в каталоге /dev. В табл. 1 приведена небольшая справка по именам наиболее часто используемых специальных файлов каталога /dev. Устройства USB воспринимаются операционной системой Linux как устройства SCSI. Подключив к системе USB-ключ или флэш-карту, его (или ее) можно добавить к файловой системе как устройство /dev/sda1. Стандартное обозначение жесткого диска состоит из трех букв. Обозначение дисков типа IDE начинается с /dev/hd. Первое устройство типа IDE обозначается hda, второе – hdb и т.д. Точно так же третий жесткий диск типа IDE (при наличии такового) будет помечен как /dev/hdc. Обозначение дисков типа SCSI начинается с /dev/sd. Первое устройство типа SCSI обозначается sda, второе – sdb. Третий жесткий диск типа SCSI будет помечен как /dev/sdc и т.д. Цифра, которая стоит после обозначения жесткого диска, соответствует номеру раздела на этом диске. Каждому типу устройств в системе может соответствовать несколько файлов устройств. Поэтому файлы устройств характеризуются двумя номерами: старшим и младшим. Старший номер устройства говорит ядру о том, к какому драйверу относится данный файл, а младший номер показывает, к какому именно устройству данного типа следует обращаться. Для файлов устройств команда ls -l вместо размера файла выдает старший и младший номера данного устройства.

Таблица 1 Основные специальные файлы

|  |  |
| --- | --- |
| Имя | Значение |
| /dev/console | Системная консоль, т. е. монитор и клавиатура, физически подключенные к компьютеру |
| /dev/hd | Жесткие диски с IDE-интерфейсом. Устройство /dev/hda1  соответствует первому разделу на первом жестком диске (/dev/hda), т. е. на диске, подключенном как Primary Mastеr |
| /dev/sd | Жесткие диски с SCSI-интерфейсом |
| /dev/fd | Файлы дисководов для гибких дисков. Первому дисководу соответствует /dev/fd0, второму – /dev/fd1 |
| /dev/tty | Файлы поддержки пользовательских консолей. Название  сохранилось с тех пор, когда к системе Unix подключались телетайпы в качестве терминалов. В Linux эти файлы устройств обеспечивают работу виртуальных консолей (переключаться между которыми можно с помощью комбинаций клавиш  <Alt>+<F1> – <Alt>+<F6>) |
| /dev/pty | Файлы поддержки псевдотерминалов. Применяются для  удаленных рабочих сессий с использованием tеlnеt |
| /dev/ttS | Файлы, обеспечивающие работу с последовательными портами.  /dev/ttS0 соответствует СОМ1 в MS-DOS, /dev/ttS1 – COM2. Если мышь подключается через последовательный порт, то /dev/mouse является символической ссылкой на соответствующий /dev/ttSN |
| /dev/cua | Специальные устройства для работы с модемами |
| /dev/null | Это устройство – просто черная дыра. Все, что записывается в /dev/null , навсегда потеряно. На это устройство можно перенаправить вывод ненужных сообщений. Если /dev/null используется как устройство ввода, то оно ведет себя как файл нулевой длины |

***4.2.2 Ссылки***

Среди типов файлов, поддерживаемых Linux, ОС различает мягкие (символические) и жесткие ссылки. Файл в Linux может иметь несколько имен, создаются так называемые *жесткие ссылки*, когда несколько имен файлов ссылаются на один и тот же индексный дескриптор. Такие имена могут указывать только на файлы на том же самом носителе, на котором находится каталог, содержащий это имя. В каталоге хранится имя файла и номер индексного дескриптора, через который определяется вся остальная информация о файле. После создания жесткой ссылки (команда ls) невозможно различить, где исходное имя файла, а где ссылка. Если удаляется одно из имен файлов, то файл еще сохраняется на диске. Файл считается удаленным, когда на него нет ни одной ссылки (ни одного имени). Очень трудно различить первоначальное имя файла и позже созданные жесткие ссылки на него. Поэтому они применяются там, где отслеживать различия и не требуется. Одно из применений жестких ссылок состоит в том, чтобы предотвратить возможность случайного удаления файла.

В Linux имеется и другой тип ссылок, так называемые *символические (мягкие) ссылки*, это отдельные файлы, в которых хранится ссылка (путь) на данный объект (файл). Аналогами этих файлов в ОС семейства Windows являются ярлыки. Для того чтобы создать символическую ссылку, используется уже упоминавшаяся команда ls с дополнительной опцией -s:

ls -s имя\_файла\_или\_каталога имя\_ссылки

В отличие от жестких ссылок символические ссылки могут указывать на файлы, расположенные в другой файловой системе, например на монтируемом носителе или даже на другом компьютере. Если исходный файл удален, символическая ссылка не удаляется, но становится бесполезной.

**4.3 Именование файлов и каталогов**

Как и во многих современных операционных системах, в ОС Linux файлы организованы в виде иерархической структуры, называемой файловой структурой. Иерархическая файловая структура может быть древовидной и сетевой. В древовидной структуре действует принцип один файл – одно полное имя, в сетевой - один файл – много полных имен (за счет наличия уникального цифрового имени). Основной главный каталог в иерархии файлов называется корневым каталогом (root directory), имеющим имя "/". В качестве синонима слова “каталог” используют часто термин “директорий”. Каждый файл имеет имя, определяющее его расположение в иерархической структуре. *Файл (file)* – это поименованная совокупность однородной информации, хранящейся на внешнем носителе. *Имя файла (filename)* – строка символов, которая создается в каталоге для идентификации файла. Имена файлов в Linux могут иметь длину до 255 символов и состоять из любых символов, кроме символа с кодом 0 и символа / (слэша). Однако имеется еще ряд символов, которые имеют специальное значение и которые поэтому не рекомендуется включать в имена, чтобы не было проблем при переносе файлов в другие ОС. Это следующие символы:

!@#$ & ~ % \* ( )[ ]{ }' " \ : ; > < ` пробел.

Если все же требуется, чтобы имя файла содержало один из этих символов, то необходимо перед этим символом поставить символ обратного слэша \. Например:

mkdir \\my\&his

Можно также заключить имя файла или каталога с такими символами в двойные кавычки. Например:

mkdir "My files",

так как без кавычек команда mkdir My files

создаст каталог с именем My.

Linux не поддерживает концепцию расширений файлов (как, например, MS-DOS, Windows). Имена файлов могут быть произвольными. Но последние части имени, отделенные точками, используют для обозначения файлов каких-то особых типов (например, .tar, .gz используется для обозначения сжатых архивов). У файла может быть несколько расширений, например, Prog.c.z говорит о том, что файл Prog.c сжат с помощью алгоритма Зива-Лемпеля, а компилятор языка С (без расширения .с) может отказаться компилировать файл. Чаще расширение в имени – это напоминание пользователю о типе файла. Исполняемые и неисполняемые файлы в Linux распознаются не по расширениям имен файлов. Имена файлов, начинающиеся с символа точка (.), относятся к скрытым (hidden) файлам, которые не выводятся по умолчанию командой ls. Для того чтобы вывести имена скрытых файлов, необходимо использовать ключ -a в команде ls, что позволяет просмотреть все файлы, содержащиеся в рабочей директории. Имена в Linux, как и в других Unix-системах, чувствительны к регистру. Это значит, что файлы с именами “file1” и “File1” будут разными. Путь (pathname) или маршрут – это строка из одного или более имен каталогов, отделенных символом “/”. Путь специфицирует место расположения файла внутри иерархической файловой структуры. Маршрут может быть абсолютным и относительным.

**Абсолютный** (absolute) **маршрут**, также называемый полным (full), содержит перечень всех имен каталогов: от корневого до искомого каталога, где зарегистрирован файл. Например, абсолютный путь к каталогу (директории) выглядит так: /home/study/usr1. Это означает, что в корневой директории содержится директория home, в которой содержится директория study, где и находится искомая директория usr1.

**Относительным** (relative) **маршрутом** называется путь к файлу от текущего каталога. Текущий каталог всегда отображается в приглашении к работе, о котором будет рассказано в главе 5. Имена файлов могут быть также представлены в двух формах: абсолютной и относительной. **Абсолютное имя**, иногда называемое полным (full) именем, содержит полный путь и простое имя, идентифицирующее файл в пределах одного каталога. Формальным признаком полного имени является то, что оно начинается со слэша (“/”). **Относительное имя** содержит относительный маршрут и простое имя. Например, если текущий каталог /user, то относительное имя для файла /user/an/main.exe можно записать как имя an/main.exe. Есть несколько специальных имен:

1) .. – имя родительского каталога, т.е. каталога, находящегося на ступеньку выше данного на пути к корневому каталогу;

2) . – имя текущего каталога; 3) ~ – имя домашнего каталога пользователя, в котором пользователь имеет все права, эти каталоги обычно размещаются в каталоге /home и имеют имена, совпадающие с именем пользователя;

4) / – имя корневого каталога, как указывалось ранее.

При вызове любой команды, принимающей в качестве аргументов имена файлов, могут быть использованы так называемые **шаблоны (маски)**, в которых используются специальные символы для определения групп файлов. Это следующие символы:

1) \* – любое число любых символов;

2) ? – один произвольный символ;

3) […] – любой из набора символов, указанных в квадратных скобках перечислением и/или с указанием диапазона;

4) [!...] – любой символ, кроме символов из набора.

Используя шаблоны, можно создавать сложные критерии для выбора файлов. Примеры имен, подходящих под шаблон:

1) а\* – все файлы, имена которых начинаются с буквы "a";

2) b\*.txt – все файлы, имена которых начинаются с буквы "b" и расширение которых – txt;

3) [abc]\* – все файлы, имена которых начинаются с "a", "b" или "c";

4) [A-Z]\* – все файлы, имена которых начинаются с любой из заглавных букв латинского алфавита (пример диапазона значений);

5) [!a-z]\* – все файлы, кроме тех, имена которых начинаются со строчной латинской буквы.

Так, команда ln [a-d]\* показывает все файлы текущего каталога, которые начинаются с букв “a”, “b”, “c”, “d”, аналогичный эффект дадут команды ln [abcd]\* и ln [bdac]\*.

**4.4 Структура корневого каталога**

Как ранее указывалось, на верхнем уровне файловой системы находится *корневой каталог (root directory)*, являющийся основой файловой структуры и началом пути ко всем ее файлам. В Linux корневой каталог содержит стандартный набор подкаталогов, который может немного изменяться от дистрибутива к дистрибутиву.

Стандартные и наиболее распространенные дополнительные подкаталоги корневого каталога Linux перечислены в табл. 2.

Рассмотрим некоторые из этих каталогов подробнее. Таблица 2 Стандартное содержимое корневого каталога

|  |  |
| --- | --- |
| Каталог | Содержимое |
| 1 | 2 |
| /bin | Этот каталог содержит в основном готовые к исполнению  программы, большинство из которых необходимо во время старта  системы (или в однопользовательском системном режиме,  используемом для отладки). Здесь хранится значительное количество общеупотребительных команд Linux |
| /boot | Содержит основные постоянные файлы для загрузки системы, в частности загружаемое ядро. Файлы из этого каталога нужны только во время загрузки системы |
| /dev | Каталог специальных файлов или файлов устройств (описан ранее) |
| /etc | Файлы настроек, используемые системой и многими приложениями. Этот каталог и его подкаталоги содержат большинство данных,  необходимых для начальной загрузки системы, и основные конфигурационные файлы. В /etc находятся, например, файл inittab,  определяющий загружаемую конфигурацию и файл паролей пользователей passwd. Часть конфигурационных файлов может находиться и в /usr/etc. Каталог /etc не должен содержать двоичных файлов (их следует перенести в /bin или /sbin). Ниже приводится назначение основных (но далеко не всех) подкаталогов каталога /etc |
| /home | Хранятся домашние каталоги всех пользователей системы |
| /lib | Этот каталог содержит разделяемые библиотеки функций,  необходимых компилятору языка С, и модули (драйверы устройств).  Даже если в системе не установлен компилятор языка С, разделяемые библиотеки необходимы, поскольку они используются многими прикладными программами. Они загружаются в память по мере необходимости выполнения каких-то функций, что позволяет уменьшить объем кода программ – в противном случае один и тот же код многократно повторялся бы в различных программах |
| /mnt | Это точка монтирования для временно монтируемых файловых систем. Если необходимо монтировать несколько дополнительных носителей, например, дискеты, CD-ROM, дополнительный жесткий диск и т. д., то можно создать в нем соответственно дополнительные подкаталоги для каждого носителя |
| /opt | Каталог, в который часто устанавливают прикладные программы, например текстовые процессоры и офисные пакеты |
| /rооt | Домашний каталог суперпользователя (пользователя rооt). Следует  Обратить внимание на то, что он расположен не там, где располагаются личные каталоги остальных пользователей (в /home) |
| bin | Подобно каталогу /bin, содержит в основном исполняемые файлы – программы и утилиты ОС, используемые в процессе загрузки и запускаемые системным администратором. В стандарте ЕНS говорится, что в этот каталог надо помещать те исполняемые файлы, которые используются после успешного подключения файловой системы /usr. Минимальное содержимое этого каталога включает программы clоck, getty, init, update, mkswap, swapоn, swapоff, halt, rebооt, shutdоwn, fdisk, fsck.\*, mkfs.\*, lilо, arp, ifcоnfig, rоute |
| /tmp | Каталог для временных файлов. В любой момент суперпользователь может удалить файлы из этого каталога без большого ущерба для остальных пользователей. Однако не стоит удалять файлы из этого каталога, если вам не стало ясно, что конкретный файл или группа файлов мешают продолжению продуктивной работы на машине. Система сама периодически очищает этот каталог, поэтому не следует хранить тут файлы, которые вам могут понадобиться в дальнейшем |
| /usr | Этот каталог огромен и его структура в основном повторяет структуру корневого каталога. В его подкаталогах находятся все основные приложения. В соответствии со стандартом ЕНS рекомендуется выделять для этого каталога отдельный раздел диска или вообще располагать его на сетевом диске, общем для всех компьютеров в сети. Такой раздел или диск монтируют только для чтения и располагают в нем общие конфигурационные и исполняемые файлы, документацию, системные утилиты и библиотеки, а также включаемые файлы (файлы типа include) |
| /var | Этот каталог содержит файлы, в которых сохраняются различные переменные данные, определяющие конфигурацию некоторых программ при следующем запуске или временно сохраняемую информацию, которая будет использоваться позже в ходе текущего сеанса. Объем данных в этом каталоге может сильно изменяться, поскольку он содержит, например, файлы протоколов (логи), файлы спулинга и блокировки (lоcking), временные файлы и т. д. |

Конкретное содержимое каталога /etc меняется от дистрибутива к дистрибутиву, но практически во всех дистрибутивах в каталоге /etc имеются следующие подкаталоги:

- /etc/Xll – содержит файлы настроек системы X Window, которая обеспечивает работу графического интерфейса;

- /etc/opt – содержит файлы настроек программ, находящихся в каталоге /opt.

По умолчанию каталог /mnt может быть пустым, а может содержать какие-нибудь подкаталоги. Обычно в нем имеются как минимум два подкаталога: /mnt/floppy (применяется для подключения дискеты) и /mnt/cdrom (применяется для подключения компакт-диска). О монтировании файловых систем будет рассказано ниже.

Каталог /usr содержит много важных подкаталогов, большая часть которых перечислена в табл. 3.

Таблица 3 Стандартные подкаталоги каталога /usr

|  |  |
| --- | --- |
| Подкаталог | Содержимое |
| /usr/Х11R6 | Файлы, применяющиеся для запуска системы Х Window |
| /usr/bin | Не самые важные, но полезные команды |
| /usr/games | Игры (кроме тех, которые были установлены в каталоге /opt) |
| /usг/include | Файлы, необходимые для компиляции программ на языке С |
| /usr/lib | Библиотеки, используемые большинством программ, установленных в каталоге /usr |
| /usr/local | Программы и другие файлы, которые могут  использоваться только на локальном компьютере (даже если остальное содержимое каталога /usr открыто для совместного использования) |
| /usr/bin | Полезные команды, предназначенные для системных администраторов |
| /usr/share | Информация, которая может использоваться любым компьютером, работающим под управлением Linux, независимо от его аппаратной конфигурации |
| /usr/src | Исходные коды ядра и других программ |

Следует помнить, что в подкаталоге /usr обычно размещаются файлы, предназначенные для совместного использования несколькими компьютерами.

**4.5 Монтирование файловых систем**

В ОС Linux используется монтирование. Монтирование – это объединение файловых систем разных устройств в одну систему. Для этого выбирается одна корневая (системная) файловая система и в ней указывается каталог, к которому будет присоединяться корневой каталог другой файловой системы. Для Linux файловая система является единым целым, независимо от того, на каких дисках или разделах расположены ее файлы. После монтирования для пользователя нет разницы между корневой и смонтированной системами.

Если в ОС Microsoft Windows каждому логическому диску присваивается своя буква (С:, D: и т. д.), то в ОС семейства UNIX (Linux) нет понятия логических дисков (рис. 2).



**Рисунок 2 - Разделы жестких дисков в ОС семейства Windows и UNIX**

В данном примере пользователь создал три раздела Linux. Первый жесткий диск IDE состоит из единственного раздела, выделенного под корневой каталог. Второй жесткий диск IDE разбит на два раздела. Один из них отведен под каталог /usr, а второй – под каталог /var. Перемещаясь по файловой системе, нельзя определить, на каком жестком диске или в каком разделе находится тот или иной каталог, это никак не отображается в приглашении командной строки и не влияет на использование команд. Следует заметить, что для разделов жесткого диска точка подключения (mount point) находится не в каталоге /mnt.

Все, что находится в каталоге /mnt (например, файлы /mnt/floppy и /mnt/cdrom), называется временными или съемными носителями – это диски, которые добавляются к системе, когда нужна находящаяся на них информация, и удаляются из системы по окончании работы. В некоторых дистрибутивах есть графические средства, которые автоматически подключают к файловой системе компакт-диск после закрытия его накопителя. Тем не менее в большинстве случаев носители приходится подключать вручную.

Съемные носители (дискеты, компакт-диски и др.) требуется время от времени менять, поэтому пользователь должен “сообщить” Linux, что в соответствующий накопитель был вставлен новый носитель. Это делается с использованием режима командной строки или окна виртуального терминала, запущенного из графического режима, с помощью команды mount.

В некоторых дистрибутивах для подключения компакт-диска достаточно выполнить команду mount /mnt/cdrom. Точно так же для подключения дисковода гибких дисков иногда можно воспользоваться командой mount /mnt /floppy или mount /dev/fd0 /mnt /floppy.

Чтобы удалить дискету или компакт-диск из файловой системы Linux, необходимо выполнить следующую команду: umount /mnt/тип\_носителя. Не вынимайте дискету из дисковода до тех пор, пока не отсоедините ее от файловой системы – это может привести к потере данных. То же самое касается и компакт-дисков. Конечно же, с содержимым компакт-диска ничего не случится, однако в большинстве случаев вы не сможете открыть накопитель, не отключив предварительно компакт-диск. Кроме того, вам не удастся отключить носитель, если вы находитесь в каталоге, содержащемся на этом носителе. Прежде чем отключать съемный носитель, убедитесь в том, что текущий каталог находится не на этом диске.

Если на компьютере установлены и Windows, и Linux с возможностью двойной загрузки (dual-boot setup), разделы Windows можно подключить к файловой системе Linux. Это значительно упрощает перенос файлов из одной операционной системы в другую. О том, как подключить раздел Windows к файловой системе Linux, можно прочитать в документации к соответствующему дистрибутиву.

Вопросы к главе 4:

1.Охарактеризуйте структуру логического раздела файловой системы s5.

2. Какие типы файлов используются в ОС Linux?

3. Как даются имена файлам и каталогам?

4. Как могут использоваться шаблоны имен файлов?

5. Что такое монтирование?

6. Какие подкаталоги содержатся в корневом каталоге?

**5 КОМАНДЫ ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX**

**5.1 Оболочка, виртуальная консоль и терминал**

Оболочка shell (шелл, оболочка, командный интерпретатор) – это программа, которая принимает команды, введенные пользователем с клавиатуры, и передает их на выполнение операционной системе. Поэтому оболочку можно назвать еще командным процессором или интерпретатором (более привычный термин для пользователей DOS, Windows). Приглашение к работе выводит именно оболочка, ожидая ввода пользователем очередной команды. Каждый раз, когда очередной пользователь входит в систему, команда login запускает для него командный процессор – оболочку. Оболочка – это не графический интерфейс, а совокупность команд и синтаксических правил, которые позволяют выполнять необходимые задачи. Оболочка является не только командным процессором, но и мощным языком программирования. В ней имеется целый ряд встроенных (внутренних) команд и операторов, кроме того, в качестве команды может использоваться любая программа, хранящаяся в виде файла на диске.

В 70 – 80-е командная оболочка была единственным интерфейсом пользователя Unix. Сейчас пользователь свободен в выборе интерфейса – графического интерфейса (GUI) или интерфейса командной строки (CLI, shell). В состав большинства дистрибутивов Linux входит оболочка bash [Bourne Again Shell, расширенная версия Bourne shell, написанная Стивом Борном (Steve Bourne)]. Кроме bash доступны и другие интерпретаторы: ksh, tcsh, zsh, csh. Запустить оболочку можно, просто введя команду bash (если именно эта программа-оболочка используется в системе Linux), и тем самым запустить новый экземпляр оболочки. Выходя из него (по команде exit или по комбинации клавиш <Ctrl>+<D>), можно вернуться к предыдущему экземпляру оболочки.

На рис. 3 приведен пример запуска оболочки из графического режима в ОС SuseLinux через выбор в главном меню команды Terminal Program Konsole. В случае успешного запуска на экране появится окно, изображенное на рис. 4, в котором высветится командная строка. 

**Рисунок 3 - Запуск оболочки shell**



**Рисунок 4 - Рабочий стол с открытым экземпляром оболочки shell**

Во многих ОС Linux приглашение командной строки заканчивается символом $ и содержит некоторую полезную информацию, например имя пользователя и имя текущего каталога. У пользователя root приглашение заканчивается символом #.

Когда пользователь входит в систему, текущим каталогом становится домашний каталог данного пользователя. Из приглашения, приведенного на рис. 4, можно увидеть, что приглашение заканчивается знаком «>», а также что пользователь вошел в систему под именем user и текущим каталогом является его домашний каталог.

Терминал (terminal) – это устройство, которое предназначено для взаимодействия пользователя с компьютером и состоит из монитора и клавиатуры. К вашему персональному компьютеру наверняка не подключены удаленные терминалы, но есть клавиатура и монитор, которые и играют роль терминала пользователя. У мейнфреймов имелся особый терминал, который предназначался для системного администратора и назывался консолью (console).

Linux позволяет подключать к компьютеру удаленные терминалы и обеспечивает возможность работы с несколькими виртуальными терминалами с одной консоли. В любом виртуальном терминале можно зарегистрироваться под другим именем. По умолчанию Linux открывает при запуске 6 параллельных сеансов работы (виртуальных терминалов). Для переключения между виртуальными терминалами используются комбинации клавиш <Ctrl>+<Alt>+<Fl>—<Ctrl>+<Alt>+<F6>.

Каждый раз при входе в оболочку запускается экземпляр оболочки — отдельная оболочка со своими параметрами. Это значит, что, войдя в оболочку из двух разных виртуальных терминалов, на самом деле запускаются две программы. Если изменить параметры одной оболочки, то они не будут изменены в другой. Впрочем, если внесенные изменения постоянны, они будут применяться ко всем новым экземплярам оболочки, открытым после внесения изменений.

Команда tty позволяет посмотреть, в каком терминале находится пользователь в данный момент. Имя терминала выводится в следующем виде: /dev/tty2.

Завершение сеанса работы с системой в одном из терминалов можно сделать нажатием комбинации клавиш <Ctrl>+<D>. Это не приведет ни к остановке работы компьютера, ни к перезагрузке системы, так как Linux – многозадачная и многопользовательская система. Завершение работы одного пользователя не означает, что надо выключать компьютер. Можно завершить сеанс работы, введя одну из команд: logout или exit. По умолчанию графический интерфейс расположен в виртуальном терминале номер 7, для перехода в него достаточно нажать <Ctrl+Alt+F7>.

Программы, предоставляющие окно для взаимодействия с командной строкой, называются эмуляторами терминала. Пользователь может использовать любой из доступных в большинстве дистрибутивов Linux эмуляторов: xterm, rxvt, konsole, kvt, gnome-terminal, nxterm и eterm. Несмотря на большое число эмуляторов, все они выполняют одну и ту же функцию – предоставляют доступ к командной строке.

**5.2 Структура команд оболочки shell**

Команды в оболочке shell обычно имеют следующий формат:

<имя команды> [ключи] [аргумент(ы)].

Например:

ls – lа /usr/bin,

где ls – имя команды выдачи содержимого каталога;

lа – ключи ( “-“ – признак ключей; l – длинный формат; а – показать все файлы, включая скрытые);

/usr/bin – каталог, для которого выполняется команда.

Эта команда отобразит на экране в длинном формате содержимое каталога /usr/bin, включая скрытые файлы.

В описанном формате команд квадратные скобки [ ] говорят о том, что ключи и аргументы являются необязательными. Поэтому можно задать команду ls, которая по умолчанию выдаст имена файлов и каталогов, зарегистрированных в текущем каталоге.

Такая структура команды выдерживается далеко не всегда. Не всегда перед ключами ставится минус, не всегда ключи идут одним словом. Есть разнообразие и в представлении аргументов. К числу команд, имеющих нестандартные форматы, относятся и такие команды, как сс, tar, dd, find и др.

**5.3 Запуск программ из командной оболочки**

Команды бывают внутренние (встроенные) и внешние (загружаемые). Встроенные – это команды, код которых включен в код самой оболочки, внешние – код которых расположен в отдельном файле на диске. Встроенную команду оболочка всегда найдет, а для поиска внешней команды пользователь в принципе должен указать оболочке полный путь до соответствующего файла. Однако для облегчения жизни пользователей оболочка умеет искать внешние команды в каталогах, которые перечислены в специально заданных "путях поиска". Только если она не находит нужных файлов в таких каталогах, она решает, что пользователь ошибся при вводе имени команды.

Запуск команд осуществляется в командной строке, которая начинается с **приглашения** к работе. Когда оно появляется на экране терминала, это означает, что система Linux ожидает инструкции от пользователя. Стандартное приглашение по умолчанию для обычного пользователя зависит от дистрибутива ОС, может меняться с помощью соответствующей команды. Ранее указывалось, что может содержать приглашение к работе (подразд. 5.1).

Как правило, в командной строке записывается одна команда, но в Linux имеется возможность задать в одной строке несколько команд. При этом могут использоваться следующие символы:

1) ; – определяет последовательное выполнение команд;

2) & – определяет асинхронное (фоновое) выполнение предшествующей команды;

3) && – определяет выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорирование последующей команды;

4) || – определяет выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорирование последующей команды.

Например, при последовательности команд

command1; command2

оболочка вначале запустит на выполнение команду command1, дождется, пока ее выполнение завершится, после чего запустит command2, дождется ее завершения, после чего снова выведет приглашение командной строки, ожидая следующих действий пользователя.

Если после команды стоит один амперсанд &, то выполнение команды происходит в асинхронном режиме, на экран выводится номер процесса, соответствующий выполняемой команде, и система вновь выходит на диалог с пользователем.

Например,

find / -name conf -print & pwd.

Первая команда find запускается в фоновом режиме, осуществляет поиск файла с именем conf, начиная от корневого каталога “/”, затем запускается команда pwd в обычном режиме.

Иногда необходимо, чтобы все фоновые процессы завершились, прежде чем будет выполняться какой-то расчет. Для этого служит специальная команда wait [PID]. Эта команда ждет завершения указанного идентификатором (числом) фонового процесса. Если команда без параметра, то она ждет завершения всех фоновых процессов, дочерних для данного "sh".

Для группировки команд также могут использоваться фигурные “{}” и круглые “()” скобки. Рассмотрим примеры, сочетающие различные способы группировки. Если введена командная строка command1&& command2; command3,

где command1, command2 и command3 – какие-то команды, то command2 будет выполнена только при успешном завершении command1; после любого из исходов обработки command2 (т.е. command2 будет выполнена либо пропущена) будет выполнена command3.

Если же введена командная строка

command1&& { command2; command3},

то обе команды (command2 и command3) будут выполнены только при успешном завершении command1.

Если { command1; command2} &,

то в фоновом режиме будет выполняться последовательность команд

command1 и command2.

Для уничтожения фонового процесса надо знать его номер, т.к. команда CTL/C прерывает только процессы переднего плана. При запуске фонового процесса на экран выдается число, соответствующее номеру (идентификатору) этого процесса (PID). Если этот номер забыт или надо убедиться, что этот процесс не закончен, с помощью команды

ps -aux

можно получить перечень идентификаторов процессов (PID), имена пользователей, текущее время, затраченное процессами, и т.д. В выведенной таблице можно найти номера процессов, подлежащих уничтожению, например это “849” и “866”. Тогда командой

kill -9 866 849

можно уничтожить эти процессы. При уничтожении процессов пользователь должен иметь то же имя, какое было приписано уничтожаемым процессам (или иметь имя привилегированного пользователя).

Круглые скобки “()”, кроме выполнения функции группировки, выполняют и функцию вызова нового экземпляра интерпретатора shell. Если текущий каталог /mnt/lab/asu, то в последовательности команд cd ..; ls; ls две команды ls выдадут дважды содержимое каталога /mnt/lab, а последовательность (cd ..; ls) ls выдаст сначала содержимое каталога /mnt/lab, а затем содержимое /mnt/lab/asu, т.к. при входе в скобки вызывается новый экземпляр оболочки shell, в рамках которого и осуществляется переход. При выходе из круглых скобок происходит возврат в старую оболочку и в старый каталог

/mnt/lab/asu.

**5.4 Редактирование командной строки. История команд**

Для редактирования командной строки можно использовать команды, приведенные в табл. 4.

Заметим, что оболочка shell имеет встроенную подпрограмму, предназначенную для облегчения ввода команд в командной строке. Эта подпрограмма вызывается при нажатии клавиши <Tab> после того, как уже введено некоторое число символов.

Если эти символы являются началом названия одной из стандартных команд, которые известны оболочке, то возможны два варианта реакции оболочки на нажатие клавиши <Tab>. Если по введенным символам команда определяется однозначно, оболочка просто добавляет окончание команды в командную строку.

Таблица 4 Клавиши редактирования командной строки

|  |  |
| --- | --- |
| Клавиша | Описание реакции системы |
| 1 | 2 |
| <→ > или <Сtгl>+<F> | Переместить вправо по командной строке в пределах уже набранной цепочки символов плюс  один символ справа (место для ввода следующего символа) |
| <→ > или <Сtгl>+<B> | Переместить на один символ влево |
| <Esc+<F> | Переместить на одно слово вправо |
| <Esc+<B> | Переместить на одно слово влево |
| <Нome> или <Сtгl>+<A> | Переместить в начало набранной цепочки  символов |
| <End> или <Сtгl>+<E> | Переместить в начало/конец набранной цепочки символов |
| <Del> или <Сtгl>+<D> | Удалить символ, на который показывает курсор |
| <Backspase> | Удалить символ в позиции, предшествующей курсору |
| <Сtгl>+<K> | Удалить правую часть строки, начиная с  символа, на который указывает курсор |
| <Сtгl>+<U> | Удалить левую часть строки, включая символ, который находится слева от курсора |
| <Enter> или <Сtгl>+<М> | Запуск на выполнение команды, определяемой набранной цепочкой символов |
| <Сtrl>+<L> | Очистить экран и поместить текущую команду в верхней строке экрана |
| <Сtгl>+<T> | Поменять местами два символа: символ, на который показывает курсор, и символ слева от курсора, затем курсор переместить на один символ вправо |
| <Esc+<T> | Поменять местами два слова: слово, на которое указывает курсор, и слово слева от первого |
| <Сtгl>+<K> | Вырезать часть строки от текущей позиции курсора до конца строки (вырезанная часть строки сохраняется в буфере, ее можно вставить в другое место строки) |
| <Esc+<D> | Вырезать часть строки от текущей позиции курсора до конца текущего слова (если курсор  указывает на пробел между словами, то вырезается все слово справа от курсора) |
| <Esc+<Del> | Вырезать часть строки от текущей позиции курсора до начала текущего слова (если курсор  указывает на пробел между словами, то вырезается все слово слева от курсора) |
| <Ctrl>+<W> | Вырезать часть строки от текущей позиции курсора до предыдущего пробела |
| <Ctrl>+<Y> | Вставить последний вырезанный текст в  позицию курсора |
| <Esc+<C> | Символ, на который указывает курсор, заменить на тот же, но заглавный, а курсор переместить на первый пробел справа от текущего слова |
| <Esc+<U> | Сделать символы данного слова заглавными, начиная с символа, на который указывает курсор, а курсор установить на пробел справа от слова |
| <Esc+<L> | Превратить символы, начиная с символа, на который указывает курсор, до конца данного слова в прописные (маленькие) буквы, а курсор установить на пробел справа от слова |
| <Shift>+<PgUp> <Shift>+ <PgDown> | Эти команды позволяют просмотреть несколько страниц экранного вывода (количество зависит от размера видеопамяти); полезны в тех случаях, когда та или иная команда выводит на экран очень много информации, быстро пробегающей  по экрану и как бы исчезающей для пользователя; как видите, эта информация не пропадает |
| <Ctrl>+<C> | Прервать выполнение запущенной команды |
| <Ctrl>+<D> | Выход из программы - оболочки bash |

Если однозначно восстановить имя команды по введенным символам невозможно, то выдается список возможных вариантов продолжения для того, чтобы пользователь мог ввести еще несколько

символов, позволяющих однозначно завершить ввод команды нажатием клавиши <Tab>.

Если ввести символ табуляции после того, как введены одна из команд и пробел, оболочка предположит, что пользователь ищет имя файла, который должен вводиться как параметр команды, и выдаст в качестве подсказки список файлов текущего каталога. Если же достаточная часть имени файла введена, то заканчивается ввод этого имени в командную строку.

Для практической работы с оболочкой также полезно знать, что оболочка запоминает некоторое число введенных команд (по умолчанию 1000, это значение задается в переменной HISTSIZE;) и позволяет вызывать их путем выбора из списка так называемой истории команд.

Историю команд можно просмотреть, введя в командной строке history (здесь вы сможете воспользоваться комбинациями клавиш <Shift>+<PgUp> и <Shift>+<PgDown>, чтобы просмотреть то, что выдаст эта команда). История команд сохраняется в файле, определяемом переменной HISTFILE. Для работы с историей команд в оболочке shell используются следующие комбинации клавиш (табл. 5).

Таблица 5 Клавиши для управления историей команд

|  |  |
| --- | --- |
| Клавиша | Описание реакции системы |
| <→> или <Ctrl>+<P> | Переход к предыдущей команде в списке (движение назад по списку) |
| <→ > или <Ctrl>+<N> | Переход к следующей команде в списке (движение вперед по списку) |
| <PgUp> | Переход (вызов в командную строку) к самой первой команде, сохраненной в истории команд |
| <!>, <N> | Выполняется (без нажатия клавиши <Enter>) *n*-я команда из списка истории команд |
| <!>, <->, <N> | Выполняется *n*-я от конца списка команда |
| <!>, строка символов | Выполняется команда, имя которой начинается на  строку символов (поиск нужной команды осуществляется движением в обратном порядке от конца файла истории и выполняется первая по­павшаяся команда, которая начинается на строку символов) |
| <Ctrl>+<0> | To же, что нажатие клавиши <Enter>, затем  отображается очередная команда из файла истории |

**5.5 Источники справочной информации в ОС Linux**

Все команды в ОС Linux хорошо документированы. Для того чтобы узнать, какие параметры и аргументы есть у конкретной команды, можно воспользоваться несколькими источниками. Во-первых, это страницы справочного руководства man. Команда man – это система встроенной помощи Linux, формат команды вызова экранной документации:

man <имя\_команды>.

В ответ на это будет загружено соответствующее экранное руководство, будет выдано описание соответствующей команды или информация по теме, обозначенной ключевым словом. Поскольку информация обычно не помещается на одном экране, при просмотре можно пользоваться клавишами <PageUp> и <PageDown>, а также клавишей пробела. Нажатие клавиши <Q> в любой момент приводит к выходу из режима просмотра и возврату в режим ввода команд. Точно так же можно получить информацию по самой команде man:

man man.

Во-вторых, команда info является некоторой альтернативой команде man. Для получения информации по отдельной команде надо задать в командной строке info с параметром, являющимся именем интересующей вас команды, например,

info man.

Информация в большинстве случаев несколько отличается от той, которую дает команда man, самое существенное отличие заключается в том, что выдаваемая info информация представлена в гипертекстовом формате. Работая в текстовом режиме, можно запустить info в одной из альтернативных консолей

(<Ctrl>+<Alt>+<F2>, <Ctrl>+<Alt>+<F3> и т. д.) и переключаться за помощью в случае необходимости. В тех случаях, когда неизвестно, где именно найти нужную информацию, можно воспользоваться гипертекстовыми ссылками, предоставляемыми командой info. Эти ссылки обозначены символом звездочки (\*), что несколько отличается от способа обозначения гипертекстовых ссылок в широко распространенных браузерах (типа Internet Explorer). Перемещаться по ссылкам можно также с помощью клавиши <Таb>; достигнув названия нужной темы, нажмите клавишу <Enter>. Нажатие клавиши <Р> возвращает вас к предыдущей странице, <N> вызывает переход на следующую страницу, a <U> переводит на один уровень вверх по иерархической структуре страниц документации.

Кроме того, можно вызвать переход по ссылке другим способом, аналогичным системе меню. Для этого надо нажать клавишу <М> и набрать в появившейся внизу экрана строке ввода некоторое число начальных символов названия нужного раздела помощи. Число символов должно быть достаточным для однозначного определения раздела помощи. Выход из программы – по клавише <Q>.

В-третьих, практически у каждой команды есть параметр -help, который выводит на экран список всех ее параметров. Последний способ хорош тогда, когда уже известно, какие параметры есть у команды. Например, чтобы просмотреть список параметров команды ls, выполните следующее:

ls –help.

Если ввести в командной строке help без параметров, то можно получить список всех встроенных команд оболочки. Если ввести команду help name, где name – имя одной из этих команд, то можно получить очень краткую справку о применении этой команды.

5.6. Команды, обеспечивающие навигацию по файловой системе

Здесь будут рассмотрены три команды, необходимые для

навигации по файловой системе:

1) pwd – показать рабочий каталог;

2) cd – сменить директорию;

3) ls – показать файлы и папки.

Поскольку интерфейс командной строки не имеет возможности графически представить содержимое файловой системы, необходимы альтернативные средства ее представления. В каждый момент времени пользователь может находиться в определенном каталоге. Как отмечалось ранее, сеанс работы в терминале начинается с домашнего каталога пользователя.

Если пользователь введет команду pwd (рис. 5), то увидит полный путь к текущему каталогу, например /home/user.

Для перехода из текущего каталога в другой каталог используется команда cd, в качестве аргумента которой указывается имя каталога (абсолютное или относительное), в который необходимо перейти. Например, если текущим каталогом является /home/dk401, то для перехода в каталог /home/dk401/dk40101 надо выполнить команду

cd dk40101 .

Чтобы перейти в родительский каталог, можно в команде использовать имя родительского каталога “..”, т.е. c d . . .

Чтобы перейти в каталог, который находится на два каталога выше текущего, можно использовать команду

c d . . / . .

Для возвращения в домашний каталог из любого текущего каталога файлового дерева достаточно выполнить команду без аргументов:

cd

или

$ cd ~ .

Если в команде указано имя обычного файла, то выводится сообщение об ошибке:

cd file1 .

Переход, например, в директорию bin, находящуюся в корневом каталоге, может быть осуществлен следующим образом:

cd /bin .

Вывод содержимого каталога можно получить с помощью команды ls, в качестве аргумента которой должно указываться имя файла (каталога):

ls [опция] [имя\_файла].

Данная команда имеет большое количество опций (ключей), рассмотрим некоторые из них на примерах использования команды:

1) ls – просмотр содержимого рабочей (текущей) директории;

2) ls /bin – вывод содержимого директории /bin;

3) ls -la .. – просмотр содержимого родительской директории в расширенном виде (отображение владельцев файлов, прав доступа, даты создания и другой информации), отображаются все файлы, включая скрытые; скрытые файлы начинаются с точки;

4) ls -l /etc /home – вывод содержимого двух директорий etc и home в расширенном виде.

Ниже приведен пример описания строки полного (расширенного) вида команды ls -l:

-rwxrw-r-- 1 bshotts bshotts 276480 Feb 11 20:41 web\_site.txt. 

**Рисунок 5 - Пример использования команды pwd**

Рассмотрим подробнее результат вывода:

- первый столбик символов первого слова содержит информацию о типе файла. Знак тире “-“ обозначает обычный файл, буква “d” обозначает директорию, буква “l” обозначает мягкую ссылку, буквы “c” и “b” обозначают специальные файлы (символьные и блочные соответственно);

- остальные девять символов первого слова (rwx-rw-r--) определяют права доступа к файлу. Набор из трех первых символов представляет права владельца на чтение (r), запись (w) и исполнение (x), следующие три символа представляют права группы владельца, а последние три – для всех остальных, прочерк (-) означает отсутствие соответствующих прав (подробнее в подразд. 5.8);

- второе слово (1) указывает число ссылок к файлу;

- третье слово (bshotts) определяет имя пользователя - владельца файла;

- четвертое слово (bshotts) определяет имя группы, к которой относится владелец файла; - пятое слово (276480) указывает на размер файла в байтах;

- далее (Feb 11 20:41) показываются дата и время создания файла (20:41) в часах и минутах или время последнего изменения файла. Если последние изменения в файл вносились более шести месяцев назад, то будут отображены только год и месяц;

- в конце строки указывается имя файла (dat) или каталога.

**5.7 Команды работы с файлами и каталогами**

***5.7.1 Просмотр текстовых файлов***

Текстовые файлы содержат символы, которые могут воспроизводиться устройствами ввода-вывода (печатные символы). Для файлов такого типа в ОС используются команды, позволяющие просмотреть текстовые файлы: cat, more, pg, head, tail. Эти команды не используются для просмотра каталогов и бинарных файлов.

Формат команды cat следующий:

cat имя\_файла [имя\_файла . . . ]

Содержимое каждого из указанных файлов выводится в канал стандартного вывода (обычно на дисплей). Команда не имеет возможности остановки вывода по заполнению экрана, если содержимое файла превышает число строк экрана дисплея.

Для прекращения вывода следует использовать комбинацию

клавиш: <Ctrl>s или <Ctrl>q.

Нажатие клавиши <Del> завершает команду и возвращает к текущему процессу оболочки.

Команды head и tail позволяют вывести только часть файла.

Форматы команд следующие:

head [-число] имя\_файла [ имя\_файла . . . ]

tail [-число] имя\_файла [ имя\_файла . . . ]

Команда head выводит первые строки файлов в количестве, определенном опцией [-число] (по умолчанию 10 строк).

Команда tail выводит последние строки файлов в количестве, определенном опцией [-число] (по умолчанию 10 строк).

Команда more осуществляет постраничный вывод текста файла. Формат команды следующий:

more имя\_файла [ имя\_файла . . . ]

Подсказка, которая выводится в нижней строке экрана,

указывает на часть просмотренного текста, выраженную в процентах: more - (xx %).

На подсказку можно ответить следующим образом:

- нажатием клавиши <Spaсe>, при этом осуществляется вывод очередной страницы;

- нажатием клавиши <Enter>, осуществляется вывод очередной строки;

- нажатием клавиши h, осуществляется вывод окна подсказок;

- нажатием клавиши b, осуществляется вывод предыдущей страницы;

- нажатием клавиши q, осуществляется завершение команды.

Команда pg осуществляет постраничный вывод текста файла. Она с одинаковым успехом позволяет просмотр как предыдущей, так и последующей страницы текста. Формат команды следующий:

pg имя\_файла [ имя\_файла . . . ]

На подсказку (:) в нижней части экрана можно ответить:

- нажатием клавиши <Enter>, при этом осуществляется вывод очередной страницы;

- нажатием клавиши h, осуществляется вывод окна подсказок;

- нажатием клавиши +n, осуществляется переход вперед на n страниц;

- нажатием клавиши –n, осуществляется переход назад на n страниц.

Команды hd и od используются в тех случаях, когда файл содержит информацию, которую нельзя представить в виде печатных символов, такой информацией может быть набор двоичных данных или данных, содержащих управляющие символы. Команда hd выводит данные в шестнадцатеричном формате. Команда od выводит данные в восьмеричном формате. Обе команды позволяют в качестве аргументов указывать несколько имен файлов.

Команда less позволяет просматривать содержимое текстовых файлов, в качестве аргумента задается имя файла. В процессе просмотра можно пользоваться клавишами PageUp и PageDown для перемещения по файлу. Выход из режима просмотра осуществляется с помощью клавиши q.

Команда file проверяет файл и выводит информацию о типе файла, вызывается аналогично команде less с аргументом в виде имени файла и распознает многие типы содержимого файлов (программы-скрипты, простые текстовые файлы, архивы, исполняемые файлы, HTML-файлы, изображения JPEG, файлы PostScript и др.).

***5.7.2 Команды создания файлов и каталогов***

В ОС Linux файлы создаются текстовыми редакторами, которые позволяют поместить данные в сформированный файл.

Существуют и специальные команды, например, для создания пустых (не содержащих данные) файлов используется команда touch:

touch имя\_файла [имя\_файл ... ].

Если имя\_файла является новым для каталога, то создается пустой файл. Если имя файла уже существует в каталоге, то его данные (по умолчанию) будут потеряны.

Для создания каталогов используется команда mkdir, для этого необходимо задать имя каталога, например, следующая команда

mkdir directory

создает в текущем каталоге пустой с именем directory.

***5.7.3 Команды для манипуляций с файлами и каталогами***

Рассмотрим некоторые базовые команды для манипуляций с

файлами и каталогами:

1) cp – копирование файлов и директорий;

2) mv – перемещение/переименование файлов или директорий;

3) rm – удаление файлов или директорий;

4) grep – поиск текста внутри файлов;

5) diff – сравнение файлов;

6) cmp – сравнение файлов;

7) nl – нумерация строк, подсчет символов, слов и строк файла;

8) wc – подсчет символов, слов и строк файла.

Команда cp копирует файлы и директории, например,

cp file1 dir1

производит копирование файла file1 из текущего каталога в подкаталог dir1.

Можно скопировать несколько файлов в директорию:

cp file1 file2 file3 dir1. Для копирования из каталога – источника в каталог – приемник файлов формата html, причем только тех которые отсутствуют в каталоге – приемнике, используется каманда:

cp -u \*.html destination.

Другие примеры использования команды приведены в табл. 6.

Команда mv выполняет различные функции в зависимости от ситуации, в которой используется. Она переносит файлы в указанную директорию или переименует файл / директорию.

Переименование файла:

mv file1 file2.

Перенос файлов в каталог directory:

mv file1 file2 file3 directory.

Таблица 6 Примеры использования команд

|  |  |
| --- | --- |
| Команда | Результат |
| cp filе1 filе2 | Копирует содержимое filе1 в filе2. Если filе2 не существует, он создается, иначе перезаписывается |
| cp -i filе1 filе2 | То же самое, что и в первом примере, только в случае существования файла требуется подтверждение пользователя о перезаписи (интерактивный режим) |
| cp -R diг1 diг2 | Копирует содержимое diг1 в diг2. Если diг2 не существует, создается, иначе в diг2 создается поддиректория с именем diг1 |

Примеры использования команд приведены в табл. 7.

Таблица 7 Примеры использования команд

|  |  |
| --- | --- |
| Команда | Результат |
| mv filе1 filе2 | Если filе2 не существует, filе1 получает имя filе2. Иначе содержимое filе2 заменяется содержимым filе1 |
| mv -i filе1 filе2 | Интерактивный режим - подтверждение перезаписи |
| mv filе1 filе2  filеЗ dir1 | Файлы перемещаются в dir1, которая должна существовать, иначе команда сообщит об ошибке |
| mv dir1 dir2 | Если dir2 не существует, dir1получает имя dir2, иначе становится поддиректорией dir2 |

Команда rm удаляет указанные файлы и директории, например,

удаление файла:

rm file.

Удаление директории:

rm -r directory.

Примеры применения команды приведены в табл. 8.

Таблица 8 Примеры использования команд

|  |  |
| --- | --- |
| Команда | Результат |
| гm filе1 filе2 | Удаляет filе1 и filе2 |
| гm -i filе1 filе2 | Подтверждение перед удалением |
| гm -г diг1 diг | Директории удаляются вместе с содержимым |

В Linux нет команды восстановления файлов undelete, как в MS-DOS, т.е. файлы не восстанавливаются штатными средствами. Поэтому полезно перед удалением файлов, имена которых заданы шаблоном, проверить эти шаблоны командой ls, чтобы увидеть результат. Для поиска строк информации файлов, соответствующих шаблону (набору символов), используется команда grep:

grep [опции] шаблон имя\_файла [имя\_файла ... ].

Здесь шаблон в общем виде представляет собой регулярное выражение, которое может содержать метасимволы. Однако шаблон может быть набором из алфавитно-цифровых символов. Например, grep dp40102 /etc/passwd

выводит строки файла /etc/passwd, содержащие набор символов

dp40102.

Для сравнения двух текстовых файлов и определения различия между ними используется команда diff:

diff [опции] имя\_файла1 имя\_файла2.

Для различающихся строк файлов выводится информация, строкам из файла с именем имя\_файла1 предшествует символ <, а строкам из файла имя\_файла2 – символ >.

Менее информативная команда cmp сравнивает данные двух файлов:

cmp [опции] имя\_файла1 имя\_файла2.

При этом файлы могут быть как текстовыми, так и не текстовыми. Команда cmp file1 file2

сообщает о первом отличии между файлами, например:

file1 file2 differ: char 5, line 1.

Для нумерации строк текстового файла используется команда nl:

nl [опции] имя\_файла [имя\_файла ... ].

Опциями файла можно задать шаг нумерации, начальное значение и т.д.

Для подсчета символов, слов и строк файла используется команда wc:

wc [опция] имя\_файла.

По умолчанию команда выводит имя файла и разделенные пробелами числа строк, слов и символов.

**5.8 Управление правами доступа к файлам**

Установка прав доступа к файлам в ОС Linux позволяет эффективно защищать файлы от несанкционированного доступа к нему с целью изменения его содержимого, считывания из него информации или его исполнения. В Linux для каждого файла в системе назначены права доступа для владельца (user), пользователей группы владельца (group) и для всех остальных (other). Разрешения устанавливаются на следующие операции:

- разрешение на чтение (read) файла (r):

- разрешение на запись (write) (w);

- разрешение на исполнение (execute) или поиск (search) (x).

Символ минус (-) говорит о том, что операция запрещена.

Чтобы просмотреть права доступа для файла, можно воспользоваться командой ls, например:

ls -l some\_file,

при этом может быть выдана информация подобного содержания:

-rw-rw-r-- 1 me me 1097374 Sep 26 18:48 some\_file.

Из этой строчки можно узнать о файле следующее:

- владельцем файла является пользователь "me";

- владелец имеет право на чтение и запись файла;

- пользователи группы владельца имеют те же права, что и владелец;

- все остальные пользователи могут читать файл.

А вот другой пример: -rwxr-x--x 1 root root 316848 Feb 27 2000 /bin/bash.

Это означает, что:

- владельцем обычного файла /bin/shell является

суперпользователь (root);

- владелец имеет все права на этот файл;

- группа пользователей владельца – это root;

- пользователи группы владельца могут читать и исполнять файл;

- остальные пользователи могут только исполнять файл.

Следует различать назначение прав доступа к файлам и каталогам.

Более подробно права доступа к файлу можно трактовать следующим образом:

- read (r) дает возможность пользователю открыть файл и просмотреть его содержимое. Команды, требующие доступ к содержимому файла, требуют права доступа на чтение, например, команды cat, more, cp;

- write (w) позволяет изменять содержимое файла. Команды, требующие доступ к изменению файла, требуют права доступа на запись, например, команды vi, mail, cp;

- eXecute (x) дает возможность пользователю выполнить

двоичную программу или командный файл подобно любой команде ОС Unix.

Права доступа к каталогу:

- read (r) дает возможность пользователю вывести список имен файлов и каталогов (прочитать содержимое каталога);

- write (w) позволяет изменять содержимое каталога, т.е. добавить или удалить имя (файла или каталога) из каталога.

Обычно только владелец каталога имеет разрешение на запись в

каталог;

- eXecute (x) дает возможность осуществить поиск по каталогу.

Без этого права доступа невозможно использовать имя каталога как компоненту полного имени файла. Отсутствие права на исполнение не позволит сменить каталог при выполнении команды cd.

Владельцем файла является пользователь, процесс которого создал файл. Этот пользователь имеет привилегии в работе с созданным файлом. В частности, он может изменить права доступа к файлу, имя владельца и группы. Изменить имя владельца может только администратор (root) или текущий владелец файла. Это он может сделать с помощью команды

chown:

chown имя\_владельца имя\_файла [имя\_файла ... ].

Имя\_владельца – это имя нового владельца файла (файлов).

После изменения имени старый владелец теряет свои привилегии. Изменить имя группы владельца может администратор (root) или владелец файла. Это он может сделать с помощью команды chgrp:

chgrp имя\_группы имя\_файла [имя\_файла ... ].

Например:

ls -l dat

-rw-r--r-- 1 dk30101 dk301 24 22 Sep 11:30 dat

chgrp sys dat

chown root dat

ls -l dat

-rw-r--r-- 1 root sys 24 22 Sep 11:30 dat.

Права доступа могут модифицироваться владельцем файла или пользователем root с помощью команды chmod:

chmod режим [режим …] имя\_файла [имя\_файла …].

Полномочия доступа можно задавать абсолютно (путем спецификации режима в восьмеричном представлении) или символически (задавая отдельные спецификации). Последний способ более популярен и позволяет добавлять и удалять права доступа выборочно.

Символический режим изменения прав доступа состоит из следующих компонентов:

[ugoa] [ + | - | = ] [ [ r ] [w] [x]].

Первая компонента может состоять из одного или более символов: u (user), g (group), o (other) или a (all), указывающих на категорию пользователей, для которой устанавливаются права доступа, владелец (u), группа (g), прочие (o) или все (a). По умолчанию используется a.

Вторая компонента должна быть одним из трех символов: +, - или =, означающих соответственно добавление полномочия для заданного класса пользователей, удаление полномочия для заданного класса пользователей или установку полномочия для заданного класса пользователей и отмену всех других неспецифицированных полномочий.

Третья компонента может быть любой комбинацией символов r, w или x, соответствующих разрешениям на чтение, запись или выполнение/поиск объекта файловой системы. Пробелы между компонентами выражения не допускаются.

Допустим, команда ls -l script выдает для файла script следующие права доступа: rwxrwxr--, владелец и группа имеют права доступа на чтение, запись и выполнение, а прочие – только разрешение на чтение. Тогда команда

chmod g-w, o+x-r script

изменит права доступа следующим образом: rwxr-x--x. В результате владелец сохранил права доступа на чтение, запись и выполнение, а прочие – только разрешение на чтение. Тогда команда

chmod g-w, o+x-r script

изменит права доступа следующим образом: rwxr-x--x. В результате владелец сохранил права доступа на чтение, запись и выполнение, группа утратила право доступа на чтение, а прочие утратили разрешение на чтение, но получили право на исполнение файла.

В абсолютном режиме слово “режим” команды chmod представляет собой три цифры, каждая из которых отражает права доступа для соответствующей категории пользователей (владелец, группа, прочие).

Группу символов, описывающих права доступа к файлам, можно описать группой битов (в двоичной системе счисления). Если бит равен 1, то это означает разрешение соответствующего права, если 0, – запрещение, например:

 rwx rwx rwx = 111 111 111;

 rw- rw- rw- = 110 110 110;

 rwx --- --- = 111 000 000;

 и так далее.

Следует заметить, что

 111 (rwx) в двоичной системе счисления при переводе в десятичную систему счисления равно 7;

 110 (rw-) равно 6;

 101 (r-x) равно 5;

 100 (r--) равно 4;

 и так далее.

Теперь, представляя трехсимвольный набор разрешений для владельца, группы пользователей владельца и остальных пользователей в виде десятичного числа, можно удобно использовать команду chmod: chmod 600 some\_file

Выполнение данной команды сделает доступными права чтения и записи для владельца, а пользователи группы владельца и все остальные не будут иметь никаких прав на файл.

В табл. 9 приведены числа для наиболее часто используемых настроек.

Команда chmod может быть использована и для смены прав доступа к каталогам. В большинстве случаев права устанавливаются так же, как и для файлов, исключение составляет право на исполнение, оно регулирует доступность отображения списка файлов.

Таблица 9 Наиболее часто используемые настройки

|  |  |
| --- | --- |
| Число | Значение |
| 777 | (rwхrwхrwх) все права для всех пользователей |
| 755 | (rwxr-хr-х) все права для владельца, права на чтение и исполнение для пользователей группы владельца и всех остальных |
| 700 | (rwх -) все права для владельца, никаких прав для остальных |
| 666 | (rw-rw-rw-) для всех права на чтение и запись |
| 644 | (rw-r--r -) права на чтение и запись для владельца, для всех остальных – только на чтение |
| 600 | (rw' -) права на чтение и запись для владельца, никаких прав для  остальных |

Значение

777 (rwxrwxrwx) все права для всех пользователей

755 (rwxr-xr-x) все права для владельца, права на чтение и исполнение для

пользователей группы владельца и всех остальных

700 (rwx------) все права для владельца, никаких прав для остальных

666 (rw-rw-rw-) для всех права на чтение и запись

644 (rw-r--r--) права на чтение и запись для владельца, для всех остальных –

только на чтение

600 (rw-------) права на чтение и запись для владельца, никаких прав для остальных Набор основных настроек для каталогов приведен в табл.10.

Таблица 10 Набор основных настроек для каталогов

|  |  |
| --- | --- |
| Число | Значение |
| 777 | (rwхrwхrwх) нет ограничений. Все пользователи могут создавать файлы, просматривать список файлов, удалять файлы. Не самый безопасный режим |
| 755 | (rwхr-хr-х) все права для владельца. Остальные пользователи могут просматривать список файлов, но не могут создавать или удалять файлы. Обычно применяется для директорий с общим доступом |
| 700 | (rwх ) все права для владельца, никаких прав для остальных.  Применяется для директорий, предназначенных для личных данных владельца |

Абсолютная форма не допускает выборочной установки прав доступа и требует указания о соответствующих правах явно сразу для всех категорий пользователей. Так, например, команда

chmod 755 script

изменит права доступа следующим образом: rwxr-xr-x.

Пользователь может управлять установкой прав доступа к файлам и каталогам, когда они создаются, с помощью команды umask, которая позволяет вывести или изменить текущую маску создания файла, эта маска определяет заданные по умолчанию права доступа, устанавливаемые при создании нового файла или каталога:

umask [маска ].

Маска представляет собой восьмеричное число из трех цифр, где каждая цифра определяет права доступа владельца, группы и всех прочих пользователей. Действие этой маски для каждой категории пользователей следующее: если рассматривать восьмеричное значение маски как двоичное число, то биты маски, равные 0, будут соответствовать разрешению доступа, а 1– запрещению доступа. Однако действие маски различно по отношению к файлам и каталогам. Табл. 11 иллюстрирует действие маски.

Обычно по умолчанию (это определяет администратор системы) вновь создаваемым файлам устанавливаются права доступа 666 (rw-rw-rw-), а каталогам - 755 (rwxr-xr-x). Чтобы вывести на экран установленную маску, можно выполнить команду без аргументов:

umask 022 $

Чтобы у вновь создаваемых файлов для категории “Прочие” было отобрано право на запись, команда должна иметь следующий вид:

umask 002

Иногда бывает необходимо иметь права суперпользователя для внесения изменений в систему. Для этого существует команда su, которая создает отдельную сессию суперпользователя.

Для выхода из сессии суперпользователя (и других пользователей тоже) используется команда exit.

Иногда бывает необходимо иметь права суперпользователя для внесения изменений в систему. Для этого существует команда su, которая создает отдельную сессию суперпользователя.

Для выхода из сессии суперпользователя (и других пользователей тоже) используется команда exit.

Таблица 11 Связь маски команды с правами доступа

|  |  |  |  |
| --- | --- | --- | --- |
| Число | Права доступа к файлу | Права доступа к каталогу | Описание |
| 0 | rw- | rwx | Предоставляет права чтения и записи файлам, записи и поиска - каталогам |
| 1 | rw- | rw- | Предоставляет права чтения и записи файлам и каталогам |
| 2 | r - - | r-x | Предоставляет права чтения файлам, чтения и поиска - каталогам |
| 3 | r - - | r - - | Предоставляет права только чтения файлам и каталогам |
| 4 | -w- | -wx | Предоставляет права записи файлам, записи и поиска - каталогам |
| 5 | -w- | -w- | Предоставляет права записи файлам и каталогам |
| 6 |  | --x | Не предоставляет никаких прав доступа файлам, предоставляет право поиска каталогам |
| 7 | \_\_\_\_\_\_ | \_\_\_\_\_\_ | Отменяет все права доступа к файлам и каталогам |

**5.9 Перенаправление ввода и вывода, конвейеры**

Большинство команд посылают результаты своей работы в *стандартный вывод* (standard output). По умолчанию стандартный вывод связан с дисплеем. Для перенаправления стандартного вывода на другое устройство или в файл используется символ “>”:

ls > file\_list.txt.

В примере результат работы команды ls будет сохранен в файле file\_list.txt, причем, если такого файла не существует, он будет создан, иначе – перезаписан. На дисплее результаты не отображаются.

Каждый раз, когда используется одиночный символ “>”, содержимое файла перезаписывается. Для добавления перенаправленного вывода в файл можно использовать двойной символ “>>”:

ls >> file\_list.txt. Если файла с таким именем нет, он создается, иначе результаты работы программы будут дописаны в конец файла.

Многие программы принимают входные данные из *стандартного ввода*. По умолчанию стандартный ввод ассоциируется с клавиатурой, но, как и стандартный вывод, он может быть перенаправлен. Чтобы, например, перенаправить стандартный ввод из файла, нужно использовать символ “<”:

sort < file\_list.txt.

Здесь команде sort (сортировка строк) передается содержимое файла file\_list.txt, и результат выводится на экран, поскольку перенаправление вывода не было использовано. А можно написать так:

sort < file\_list.txt > sorted\_file\_list.txt

Теперь отсортированное содержимое файла file\_list.txt будет записано в файл sorted\_file\_list.txt.

Таким образом, как ввод, так и вывод команды могут быть перенаправлены. Порядок перенаправления не важен. Главное требование – символы перенаправления должны находиться после всех ключей и аргументов команды.

Одной из мощных возможностей CLI является возможность объединения нескольких команд в *конвейер* (символ “|”). С их помощью вывод одной команды передается как ввод для следующей, и так далее, например:

ls -l | less

В примере вывод команды ls -l передается команде less, в результате вывод можно просматривать, перемещаясь произвольно вверх и вниз.

Использование конвейера позволяет решать многие задачи. Вот несколько примеров:

1) ls -lt | head (отображает 10 самых новых файлов в директории);

2) du | sort –nr (отображает список директорий с указанием занимаемого места на диске, сортировка по убыванию размера);

3) find -type f -print | wc –l (отображает количество файлов в директории и ее поддиректориях).

Один из классов программ, которые используются в конвейерах, – *фильтры*. Они принимают стандартный ввод, обрабатывают его соответствующим образом и направляют на стандартный вывод (табл.12).

В Linux есть команда lpr, принимающая стандартный ввод и передающая его на принтер. Часто эта команда используется вместе с конвейерами и фильтрами. Пример:

cat poorly\_formatted\_report.txt | fmt | pr | lpr.

При этом командой cat в стандартный вывод передается содержимое файла "poorly\_formatted\_report.txt", с помощью конвейера оно передается команде форматирования fmt, затем результаты ее работы обрабатываются командой pr для разделения на страницы, и, наконец, команда lpr передает на печать форматированный текст.

Следующий пример:

cat unsorted\_list\_with\_dupes.txt | sort | uniq | pr | lpr.

Здесь содержимое файла "unsorted\_list\_with\_dupes.txt" (с несколькими повторяющимися строками) сортируется, затем проверяется на уникальность строк и очищается от дубликатов. После этого отсортированный набор уникальных строк разбивается на страницы и передается на принтер.

Таблица 12 Основные команды, работающие в качестве фильтров

|  |  |
| --- | --- |
| Команда | Результат |
| sогt | Сортирует данные из стандартного ввода и направляет результаты в стандартный вывод |
| uniq | Удаляет повторяющиеся строки из стандартного ввода и выводит список уникальных строк |
| greр | Проверяет данные стандартного ввода на соответствие регулярному выражению и выводит подходящие фрагменты |
| fmt | Форматирует стандартный ввод |
| pr | Принимает стандартный ввод в виде текста, разбивает его на страницы |
| head | Выводит 10 самых новых файлов. Может использоваться для отображения начала файла |
| tail | Отображает последние строки. Может использоваться, например, для просмотра лог-файлов |
| tr | Обрабатывает символы. С помощью команды можно преобразовывать символы из нижнего регистра в верхний, и наоборот. Или заменять символы конца строки (DOS -> UNIХ) |
| sed | Редактор потока. Осуществляет более сложные операции, чем tг |
| awk | Язык программирования для конструирования фильтров |

Задание для самостоятельной работы:

Пользуясь справочником и встроенным руководством, ознакомьтесь со следующими командами ОС UNIX: clear, date, echo, exit, man, who.

Вопросы к главе 5:

1. Какой формат имеет командная строка? Что такое опция и аргументы команды?

2. Какие возможности имеются у пользователя для корректировки или отмены выполнения команды при её вводе?

3. Где хранится учетная информация о пользователях системы?

**6 ПРОГРАММИРОВАНИЕ В ОБОЛОЧКЕ SHELL**

**6.1 Создание и запуск командного файла**

Командный файл (файл-скрипт, файл-сценарий или shell-файл) в Linux представляет собой обычный файл с ASCII-текстом. Для создания скрипта можно использовать простой текстовый редактор.

Ниже в табл. 13 перечислены основные текстовые редакторы, поддерживаемые Linux.

Таблица 13 Текстовые редакторы в Linux-подобных системах

|  |  |
| --- | --- |
| Название | Описание |
| vi | Один из первых текстовых редакторов UNIX, имеющийся во всех разновидностях этой ОС. Он характеризуется не совсем удобным и не интуитивным интерфейсом, но, с другой стороны, vi быстрый и функциональный |
| emacs | Редактор содержит все возможности, которые когда-либо были  придуманы для текстовых редакторов, написан Ричардом Столлманом (Richard Stallman) |
| pico | Очень простой редактор с ограниченными, но достаточными для начинающего пользователя возможностями |
| gedit | Редактор, поставляемый со средой Gnome |
| kedit | Редактор, входящий в состав KDE |
| kwrite | Редактор в составе KDE, поддерживающий подсветку синтаксиса, – полезную возможность для программистов |

Воспользовавшись одним из текстовых редакторов, можно написать классический пример первого файла - скрипта:

#!/bin/bash

# Мой первый скрипт

echo "Hello World!"

Первая строка содержит информацию о программе, которая будет интерпретировать скрипт (в данном случае это /bin/bash).

Вторая строка – обычный комментарий, который игнорируется командным интерпретатором. Все, что находится в строке в командном файле левее символа "#", воспринимается интерпретатором как комментарий. Величина пользы от комментариев растет прямо пропорционально размеру скрипта. Спустя время будет гораздо проще разобраться в скрипте с комментариями к выполняемым действиям, чем просто в наборе команд и функций.

Последняя строка содержит команду echo, которая выводит на дисплей строку “Hello World!”.

Скрипт готов, осталось только дать имя файлу, например

my\_script.

После написания скрипта нужно установить разрешение оболочке на исполнение этого файла. Для этого можно воспользоваться командой chmod:

chmod 755 my\_script

Выполнение команды даст владельцу полные права на файл, а всем остальным пользователям – права на чтение и исполнение.

Для запуска такого скрипта имеется несколько возможностей, зависящих от дистрибутива Linux, можно поместить его в каталог с исполняемыми файлами и просто набрать в командной строке (my\_script), или запустить в новом экземпляре оболочки shell (sh my\_script или sh < my\_script), или следующим образом:

./my\_script .

Результатом выполнения будет строка “Hello World!”, отображенная на дисплее.

Следует заметить, что когда пользователь набирает определенную команду, система не ищет ее по всем дискам, такой процесс занимал бы много времени. Вместо этого оболочка просматривает специальный список каталогов, где хранятся исполняемые файлы. Если среди этих файлов команда не обнаруживается, на дисплее появляется сообщение "command not found". Этот список каталогов называется путем (path) поиска исполняемых файлов, который можно посмотреть следующим образом:

echo $PATH.

Команда отобразит список каталогов, разделенных двоеточием, в котором ОС ищет исполняемые файлы, если не находит в текущем каталоге. Для того чтобы скрипт можно было вызывать из каталога без указания пути, нужно добавить в список путь к файлу-скрипту (например, /home/directory):

export PATH=$PATH:directory

Такое решение будет действовать до следующего входа в систему.

**6.2 Переменные и параметры командного файла**

В командном файле, как правило, не обойтись без так называемых shell - переменных. Переменная с точки зрения оболочки – это параметр, обозначаемый именем. Значения переменным присваиваются с помощью оператора следующего вида:

name=value,

где name – имя переменной, a value – присваиваемое ей значение (может быть пустой строкой). Имя переменой – это начинающаяся с буквы последовательность букв, цифр и подчеркиваний, присваиваемое значение – строка символов (любой текст). Если значение содержит специальные символы, то его надо взять в кавычки. Присвоенное значение этих кавычек, естественно, не содержит. Если переменная задана, то ее можно удалить, используя встроенную команду оболочки unset.

Примеры присваивания:

1) var\_1=13 (“13” – это не число, а строка из двух цифр);

2) var\_2="ОС LINUX" (здесь двойные кавычки " " необходимы, так как в строке есть пробел).

Возможны и иные способы присваивания значений shell-переменным. Так, например, запись DAT=`date` приводит к тому, что сначала выполняется команда date (обратные кавычки говорят о том, что сначала должна быть выполнена заключенная в них команда), а результат ее выполнения вместо выдачи на стандартный выход приписывается в качестве значения переменной.

Можно присвоить значение переменной и с помощью команды read, которая обеспечивает прием значения переменной с клавиатуры в диалоговом режиме. Обычно этой команде в скрипте предшествует команда echo, которая позволяет предварительно выдать какое-то сообщение на экран. Например:

echo -n "Введите трехзначное число:"

read x.

После вывода на экран сообщения интерпретатор остановится и будет ждать ввода значения с клавиатуры; то, что будет введено, то и станет значением переменной x.

Одна команда read может прочитать (присвоить) значения сразу для нескольких переменных. Если переменных в команде больше, чем их введено (через пробелы), оставшимся присваивается пустая строка. Если передаваемых значений больше, чем переменных в команде, то лишние игнорируются. При обращении к shell-переменной необходимо перед именем ставить символ “$”. Так, в приведенном выше примере команда echo $var\_2 выдаст на экран ОС Linux.

Для того чтобы имя переменной не сливалось со строкой, следующей за именем переменной, используются фигурные скобки. Пусть переменная a=/mnt/lab/asu/, тогда команда cat ${a}prim выдаст на экран содержимое одного файла /mnt/lab/asu/prim. Если также предположить, что в системе есть переменные prim и prim=dir, то команда echo ${a}$prim выдаст на экран /mnt/lab/asu/dir.

В ОС Linux в качестве средств **экранирования** используются двойные кавычки (" "), одинарные кавычки (' ') и бэк-слэш (\).

Из примеров очевидно действие этих символов. Можно в одной строке записывать несколько присваиваний:

x=22 y=33 z=$x

A="$x" B='$x' C=\$x

D="$x + $y + $z" E='$x + $y + $z' F=$x\ +\ $y\ +\ $z.

Тогда

echo A = $A B = $B C = $C

echo D = $D E = $E F = $F.

Выдадут на экран

A = 22 B = $x C = $x

D = 22 + 33 + 22 E = $x + $y + $z F = 22 + 33 + 22.

Приведем еще примеры, связанные с экранированием перевода строки. Пусть переменной string присвоено значение массива размером 2x3:

abc

def.

Обратим внимание, что для избежания присваивания лишних пробелов вторая строка массива начата с первой позиции следующей строки:

string="abc

def".

Рассмотрим три варианта записи переменной в команде echo: команда echo $string выдаст на экран строку abc def; команда echo '$string' выдаст $string; команда echo "$string" выдаст

abc

def. Напомним, что бэк-слэш (\) не только экранирует следующий за ним символ, что позволяет использовать специальные символы просто как символы, представляющие сами себя, он может экранировать и сам себя – \\. В командном же файле бэк-слэш позволяет объединять строки в одну, экранировать конец строки. Кстати, эффект продолжения командной строки обеспечивает и символ конвейера.

Когда оболочка запускает на выполнение какую-то программу или команду, она передает им часть переменных окружения. Для того чтобы переменная окружения передавалась запускаемому из оболочки процессу, ее нужно задавать с помощью специальной команды export, то есть вместо name=value надо записать

export name=value.

В таком случае все запускаемые из оболочки программы, в том числе вторичные экземпляры самой оболочки, будут иметь доступ к заданным таким образом переменным, то есть могут вызывать их значения по именам.

В командный файл при запуске могут быть переданы параметры, при этом важна очередность их передачи. При обращении к параметрам в файле перед цифрой ставится символ доллара “$” (как и при обращении к переменным): $1 – первый по порядку параметр; $2 – второй параметр и т.д. до 9. В (стандартной) переменной с именем "0" будет храниться имя вызванного командного файла ($0). Так, если командный файл primer вызывается с параметрами file1 и file2, то эти параметры попадают в новую среду под стандартными именами $1 и $2.

Поскольку число переменных, в которые могут передаваться параметры, ограничено цифрой 9, то для передачи большего числа параметров используется специальная команда shift. Пусть командный файл many вызывается с 13-ю параметрами:

many 10 20 30 40 50 60 70 80 90 100 110 120 130

и имеет вид

###

# many: Передача большого числа параметров.

echo "$0: Много параметров"

echo " Общее число параметров = $#

Исходное состояние: $1 $5 $9 "

shift echo "1 сдвиг: первый=$1 пятый=$5 девятый=$9"

shift 2

echo "1 + 2 = 3 сдвига: первый=$1 пятый=$5 девятый=$9"

perem=`expr $1 + $2 + $3`

echo $perem

В результате первого применения команды shift" второй параметр вызывается как $1, третий параметр вызывается как $2, десятый параметр, который был исходно недоступен, вызывается как $9. Но стал недоступным первый параметр. После выполнения скрипта на экране будет:

many: Много параметров

Общее число параметров = 13

Исходное состояние: 10 50 90

1 сдвиг: первый=20 пятый=60 девятый=100

1 + 2 = 3 сдвиг: первый=40 пятый=80 девятый=120

150

Команда set устанавливает значения параметров, это бывает очень удобно. Например, команда date выдает на экран текущую дату, скажем, "Mon May 01 12:15:10 2000", состоящую из пяти слов, тогда последовательность команд

set `date`

echo $1 $3 $5

выдаст на экран

Mon 01 2000.

Каждый процесс имеет среду, в которой он выполняется. Оболочка использует ряд глобальных переменных этой среды. Если набрать команду set без параметров, то на экран будет выдана информация о ряде стандартных переменных, созданных при входе в систему (и передаваемых далее всем новым процессам "по наследству"), а также переменных, созданных и экспортируемых вашими процессами (табл. 14).

Конкретный вид и содержание выдаваемой информации в немалой степени зависят от того, какая версия Linux используется и как инсталлирована система.

Исходная среда устанавливается автоматически при входе в систему с использованием файлов типа /etc/rc и /etc/.profile.

Таблица 14 Стандартные переменные окружения

|  |  |
| --- | --- |
| Название | Описание |
| HOME | Это имя домашнего директория, в котором пользователь оказывается после входа в систему |
| PATH | Эта переменная задает последовательность файлов, которые просматривает shell в поисках команды. Имена файлов разделяются здесь двоеточиями. Последовательность просмотра соответствует очередности следования имен в списке, но первоначально поиск происходит среди так называемых встроенных команд. В число встроенных команд входят наиболее часто используемые команды, например echo, cd, pwd, date. После этого система просматривает директорий /bin, в котором могут находиться команды sh, cp, mv, ls и т.п. Затем директорий /usr/bin с командами cat, сс, expr, nroff, man и многими другими. Далее поиск происходит в текущем директории, где, скорее всего, находятся написанные вами командные файлы.  После набора командной строки и нажатия <Enter> shell (после выполнения необходимых подстановок) распознает имя, соответствующее команде и осуществляет ее поиск в директориях, перечисленных в списке. Если команда размещена вне этих директориев, она не будет найдена. Если присутствует несколько команд с одинаковым именем, то вызвана будет та, которая расположена в директории, просматриваемом первым |
| IFS= | (Внутренний Разделитель Полей) перечисляет символы, которые служат для разделения слов (полей). Таковыми являются "пробел", "табуляция" и "перевод строки", поэтому здесь слева от присваивания ничего не видно и занято две строки |
| LOGNAME | Имя входа (имя пользователя) |
| MAIL | Имя файла, в который поступает (электронная) почта |
| PWD | Имя текущего директория |
| PS1=${PWD}:" " | Вид приглашения к работе. В данном случае будут выдаваться имя текущего директория, двоеточие и пробел |
| SHELL=/bin/bash | Эта переменная указывает оболочку, которую использует пользователь. В данном случае используется не (стандартный) shell (sh), а версия bash |
| UID | Идентификатор пользователя |

Один из способов изменить среду – это разместить эту информацию в своем домашнем каталоге в специализированном файле ".profile" (${HOME}/.profile), присвоив нужные значения переменным среды. Тогда при каждом входе в систему этот файл будет автоматически выполняться и устанавливать новую среду.

Этот файл должен обязательно размещаться в домашнем каталоге пользователя. Если внесены изменения в .profile, то для переноса этих изменений в среду необходимо выполнить этот файл. Для этого можно выйти и заново войти в систему, а можно выполнить:

.profile.

При использовании глобальных переменных в командном файле следует впереди перед ними ставить знак “$”.

Рассмотрим пример shell - файла, иллюстрирующий большие возможности специальных переменных:

# specific: Специальные параметры (переменные)

echo $0 – выводит имя shell - файла;

echo $? – выводит код завершения команды;

echo $$ – выводит идентификатор последнего процесса;

echo $! – выводит идентификатор последнего фонового процесса;

echo – выводит пустую строку;

echo $\* – выводит значения параметров, как строки;

echo $@ – выводит значения параметров, как слова;

echo

set -au

echo $- – выводит режимы работы интерпретатора.

При запуске этого файла на выполнение в виде specific par1 par2 par3 получим следующую информацию на экране:

specific – имя shell - файла;

0 – код завершения;

499 – идентификатор последнего процесса;

98 – идентификатор последнего фонового процесса;

par1 par2 par3 – значения параметров как строки;

par1 par2 par3 – значения параметров как слова;

au – режимы работы интерпретатора.

Код “0” соответствует нормальному завершению процесса. Значение специальной переменной “$$” соответствует номеру процесса, выполняющего данный shell - файл. Каждый такой файл, выполняемый компьютером, инициирует один или несколько процессов, автоматически получающих номера по порядку. Поэтому, используя номер процесса в качестве имени файла, можно быть уверенным, что каждый новый файл будет иметь новое имя и не запишется на место уже существующего. Недостатком такого способа именования файлов является то, что неизвестно, какие имена будут присвоены файлам, следовательно, такие файлы можно легко потерять. Это создает дополнительные проблемы при отладке программ. Различие между $\* и $@ состоит в том, что первая переменная может быть представлена как строка

"par1 par2 par3",

а вторая – как совокупность слов

"par1" "par2" "par3".

Для иллюстрации командой set установлены следующие режимы интерпретатора: a – все последующие переменные экспортируются;

u – отсутствие параметра считать ошибкой, что и отразилось в специальной переменной "$-".

**6.3 Условный оператор if**

В общем случае оператор if имеет структуру:

if условие

then список

[elif условие

then список]

[else список]

fi

В данном случае квадратные скобки использованы для указания на необязательность конструкций, elif – сокращенный вариант от else if, может быть использован наряду с полным, т.е. допускается вложение произвольного числа операторов if. Разумеется, “список” в каждом случае должен быть осмысленным и допустимым в данном контексте. Самая усеченная структура этого оператора:

if условие

then список

fi.

Это означает, что если выполнено условие, то выполняется "список", иначе он пропускается. Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка shell.

Два возможных формата команды:

test условие

или

[ условие ].

Для программистов более привычен второй вариант (условие в квадратных скобках), оболочка распознает эту команду по открывающей скобке “[“ как слово, соответствующее команде test. Следует заметить, что между скобками и содержащимся в них условием обязательно должны быть пробелы. Пробелы должны быть и между значениями и символом сравнения или операции. Не путать с противоположным требованием для присваивания значений переменным.

В shell используются условия различных типов (табл.15).

Например, если имя существующего файла abc, то условие [ -f abc ] будет истинно, команда echo $? выдаст код завершения 0; условие [ -d abc ] – ложно, код завершения 1.

Таблица 15 Условия проверки файлов

|  |  |
| --- | --- |
| Условие | Значение |
| -f filе | Файл filе является обычным файлом |
| -d filе | Файл filе - каталог |
| -с filе | Файл filе - специальный файл |
| -r filе | Имеется разрешение на чтение файла filе |
| -w filе | Имеется разрешение на запись в файл filе |
| -s filе | Файл filе не пустой |

Кроме этого, имеются условия проверки строк, которые приведены в табл.16.

Когда команда test дает значение “истина”, то вырабатывается код завершения 0, в противном случае – 1. Рассмотрим несколько примеров, в которых первая строка – набор команд, вторая – код завершения команды:.

1) x="who is who"; export x; [ "who is who" = "$x" ]; echo $?

0;

2) x=abc ; export x ; [ abc = "$x" ] ; echo $?

0;

x=abc ; export x ; [ -n "$x" ] ; echo $?

0;

3) x="" ; export x ; [ -n "$x" ] ; echo $?

Таблица 16 Условия проверки строк

|  |  |
| --- | --- |
| Условие | Значение |
| str1 = str2 | Строки str1и st1совпадают |
| str1 != str2 | Строки str1и st1не совпадают |
| -n str1 | Строка str1существует (непустая) |
| -z str1 | Строка str1не существует (пустая) |

Условия сравнения целых чисел приведены в табл. 17.

Таблица 17 Условия сравнения целых чисел

|  |  |
| --- | --- |
| Условие | Значение |
| х -eq y | х равно y |
| х -nе y | х не равно y |
| х -gt y | х больше y |
| х -gе y | х больше или равно y |
| х -lt y | х меньше y |
| х -lе y | х меньше или равно y |

То есть в данном случае команда test воспринимает строки символов как целые числа. Поэтому во всех остальных случаях “нулевому” значению соответствует пустая строка. В данном же случае, если надо обнулить переменную, скажем, x, то это достигается присваиванием x=0.

Примеры сравнения целых чисел:

1) x=abc ; export x ; [ abc -eq "$x" ] ; echo $?

"[": integer expression expected before -eq

x=321 ; export x ; [ 321 -eq "$x" ] ; echo $?

0

2) x=3.21 ; export x ; [ 3.21 -eq "$x" ] ; echo $?

"[": integer expression expected before -eq

x=321 ; export x ; [ 123 -lt "$x" ] ; echo $?

0

Сложные условия реализуются с помощью типовых логических операций, приведенных в табл. 18.

Таблица 18 Сложные условия

|  |  |
| --- | --- |
| Условие | Значение |
| ! | (nоt) инвертирует значение кода завершения |
| -о | (оr) соответствует логическому ИЛИ |
| -a | (and) соответствует логическому И |

Примеры сложных условий:

[ ! privet ] ; echo $?

1

x=privet; export x; [ "$x" -a -f specific ] ; echo $?

0

x="";export x; [ "$x" -a -f specific ] ; echo $?

1

x="";export x; [ "$x" -a -f specific -o privet ] ; echo $?

0

x="";export x; [ "$x" -a -f specific -o ! privet ] ; echo $?

1

Если командный файл prim содержит следующие команды

if [ $1 -gt $2 ]

then pwd

else echo $0 : Hello!

fi, то вызов командного файла prim 12 11 выдаст, например, /home/user, а запуск prim 12 13 даст prim : Hello!

В условии можно использовать то свойство оболочки shell, что команды могут выдавать различный код завершения. Пусть shell - файл prim1 содержит команды

if a=`expr "$1" : "$2"`

then echo then a=$a code=$?

else echo else a=$a code=$?

fi,

тогда вызов файла prim1 by by выдаст then a=2 code=0, а prim1 by be выдаст else a=0 code=1.

Еще пример на вложенность:

# Оценка достижений

echo -n " А какую оценку получил на экзамене?: "

read z

if [ $z = 5 ]

then echo Молодец !

elif [ $z = 4 ]

then echo Все равно молодец !

elif [ $z = 3 ]

then echo Все равно !

elif [ $z = 2 ]

then echo Все !

else echo !

fi.

Можно обратить внимание на то, что желательно использовать сдвиги при записи программ, чтобы лучше выделить вложенность структур.

**6.4 Оператор вызова case**

Оператор выбора case имеет структуру:

case строка in

шаблон) список команд;;

шаблон) список команд;;

...

esac.

Здесь case in и esac – служебные слова, “строка” (это может быть и один символ) сравнивается с “шаблоном”. Затем выполняется “список команд” выбранной строки, служебное слово esac необходимо для завершения структуры.

Ранее рассматривавшийся пример со структурой if проще и нагляднее можно реализовать с помощью структуры case:

# Структура case.

echo -n " А какую оценку получил на экзамене?: "

read z

case $z in

5) echo Молодец ! ;;

4) echo Все равно молодец ! ;;

3) echo Все равно ! ;;

2) echo Все ! ;;

\*) echo ! ;;

esac.

Непривычно выглядят в конце строк выбора “;;”, но написать здесь “;” было бы ошибкой. Для каждой альтернативы может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то символ “;” будет использоваться как разделитель команд. Обычно последняя строка выбора имеет шаблон \*, что в структуре case означает “любое значение”. Эта строка выбирается, если не произошло совпадение значения переменной ни с одним из ранее записанных шаблонов, ограниченных скобкой “)”. Значения просматриваются в порядке записи.

Данный пример показывает, что команда case может использоваться для реализации простейших меню.

Другим примером использования структуры case может быть файл - сценарий prim2 типа справочник:

# Для различных фирм по имени выдается

# название холдинга, в который она входит

case $1 in

ONE|TWO|THREE) echo Холдинг: ZERO ;;

MMM|WWW) echo Холдинг: Not-Net ;;

Hi|Hello|Howdoing) echo Холдинг: Привет! ;;

\*) echo Нет такой фирмы ;;

esac.

При запуске файла prim2 Hello на экран будет выведено:

Холдинг: Привет!

А при вызове prim2 HELLO" на экран будет выведено:

Нет такой фирмы. Разумеется, на месте отдельных команд могут быть последовательности команд с внешними параметрами или вызовы более сложных командных файлов.

Рассмотрим командный файл prim3, который добавляет информацию к файлу, указанному первым параметром (если параметр один), со стандартного входа либо (если 2 параметра) из файла, указанного в качестве первого параметра, например:

# Добавление в файл.

# Использование стандартной переменной.

# "$#" – число параметров при вводе расчета

case $# in

1) cat >> $1 ;;

2) cat >> $2 < $1 ;;

\*) echo "Формат: prim3 [откуда] куда" ;;

esac.

Параметр $1 (при $#=1) – это имя файла, в который происходит добавление со стандартного входа. Параметры $1 и $2 (при $#=2) – это имена файлов, из которого $1 и в который $2 добавлять. Во всех других случаях (\*) выдается сообщение о том, каким должен быть правильный формат команды.

**6.5 Оператор цикла с перечислением for**

Оператор цикла for имеет структуру:

for имя [in список значений]

do

список команд

done,

где for – служебное слово, определяющее тип цикла; do и done – служебные слова, выделяющие тело цикла. Фрагмент [in список значений] может отсутствовать. Например,

for i in f1 f2 f3

do

procsort $i

done.

В этом примере имя i играет роль параметра цикла, это имя можно рассматривать как shell-переменную, которой последовательно присваиваются перечисленные значения (i=f1, i=f2, i=f3), и выполняется в цикле команда procsort. Часто используется форма for i in \*, означающая “для всех файлов текущего каталога”.

Пусть procsort в свою очередь представляется командным файлом

cat $1 | sort | tee /dev/lp > ${1}\_sorted,

т.е. последовательно сортируются указанные файлы, результаты сортировки выводятся на печать (/dev/lp) и направляются в файлы

f1\_sorted, f2\_sorted и f3\_sorted.

Можно сделать более универсальной команду lsort, если не фиксировать перечень файлов в команде, а передавать произвольное их число параметрами.

Тогда головная программа будет следующей:

for i

do

procsort $i

done

Здесь отсутствие после i служебного слова in с перечислением имен говорит о том, что список поступает через параметры команды. Результат предыдущего примера можно получить, набрав

lsort f1 f2 f3.

Усложним ранее рассмотренную задачу определения холдинга фирмы. Теперь можно при вызове указывать произвольное количество фирм. При отсутствии в структуре оператора for фрагмента [in список значений] значения берутся из параметров вызывающей команды.

# Справочник.

# Для различных фирм по имени выдается

# название холдинга, в который она входит

for i

do

case $i in

ONE|TWO|THREE) echo Холдинг: ZERO ;;

MMM|WWW) echo Холдинг: Not-Net ;;

Hi|Hello|Howdoing) echo Холдинг: Привет! ;;

\*) echo Нет такой фирмы ;;

esac

done

При вызове "holding Hello HELLO ONE" на экране будет:

Холдинг: Привет!

Нет такой фирмы Холдинг: Not-Net

**6.6 Оператор цикла с истинным условием while**

Структура while предпочтительнее тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла while имеет следующую структуру:

while условие

do

список команд

done,

где while – служебное слово, определяющее тип цикла с истинным условием. Список команд в теле цикла (между do и done) повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен 0) или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие должно выполняться.

# Структура while

# Расчет позволяет напечатать 50

# экземпляров файла file22

n=0

while [ $n -lt 50 ] # пока n < 50

do

n=`expr $n + 1`

cat file22 > /dev/lp

done

Обратим внимание на то, что переменной n вначале присваивается значение 0, а не пустая строка, так как команда expr работает с shell-переменными как с целыми числами, а не как со строками.

n=`expr $n + 1`,

т.е. при каждом выполнении значение n увеличивается на 1.

То же самое можно реализовать и сложнее. Файл рr-br приведен для иллюстрации бесконечного цикла и использования команды break, которая обеспечивает прекращение цикла.

###

# рr-br: Структура "while" c "break"

# Расчет позволяет напечатать 50 # экземпляров файла "file-22"

n=0

while true

do

if [ $n -lt 50 ] # если n < 50

then n=`expr $n + 1`

else break

fi

cat file-22 > /dev/lp

done

Команда break [n] позволяет выходить из цикла. Если n отсутствует, то это эквивалентно break 1: n указывает число вложенных циклов, из которых надо выйти, например, break 3 – выход из трех вложенных циклов.

В отличие от команды break команда continue [n] лишь прекращает выполнение текущего цикла и возвращает на начало цикла. Она также может быть с параметром. Например, continue 2 означает выход на начало второго (если считать из глубины) вложенного цикла.

Команда exit [n] позволяет выйти вообще из процедуры с кодом возврата 0 или n (если параметр n указан). Эта команда может использоваться не только в циклах. Даже в линейной последовательности команд она может быть полезна при отладке, чтобы прекратить выполнение (текущего) командного файла в заданной точке.

**6.7 Оператор цикла с ложным условием until**

Оператор цикла until имеет структуру:

until условие

do

список команд

done,

где until – служебное слово, определяющее тип цикла с ложным условием. Список команд в теле цикла (между do и done) повторяется до тех пор, пока сохраняется ложность условия или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие не должно выполняться. Отличие от оператора while состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) после каждого (в том числе и первого!) выполнения команд тела цикла.

Программистов, знакомых с операторами until в других языках, может вначале сбивать такая семантика этого оператора.

Пример:

until false

do

read x

if [ $x = 5 ]

then echo enough ; break

else echo some more

fi

done.

Здесь программа с бесконечным циклом ждет ввода слов (повторяя на экране фразу some more), пока не будет введено 5. После этого выдается enough, и команда break прекращает выполнение цикла.

Другой пример иллюстрирует возможность использовать в условии вычисления:

until date | grep 12:00:

do

sleep 30

done.

Здесь каждые 30 с выполняется командная строка условия. Команда date выдает текущую дату и время. Команда grep получает эту информацию через конвейер и пытается совместить заданный шаблон 12:00: со временем, выдаваемым командой date. При несовпадении grep выдает код возврата 1, что соответствует значению «ложь», и цикл выполняет ожидание в течение 30 с, после чего повторяется выполнение условия. В полдень (возможно с несколькими секундами) произойдет сравнение, условие станет истинным, команда grep выдаст на экран соответствующую строку, и работа цикла закончится.

**6.8 Функции в shell**

Функция позволяет подготовить список команд shell для последующего выполнения.

Описание функции имеет вид:

имя()

{

список команд

}.

После чего обращение к функции происходит по имени. При выполнении функции не создается нового процесса. Она выполняется в среде соответствующего процесса. Аргументы функции становятся ее позиционными параметрами; имя функции – ее нулевой параметр. Прервать выполнение функции можно оператором return [n], где n – код возврата (необязательный параметр).

Рассмотрим пример командного файла fun, содержащего описание и вызов функции с именем fn:

echo $$

fn() # описание функции

{

echo xx=$xx

echo $#

echo $0: $$ $1 $2

xx=yy ; echo xx=$xx

return 5

}

xx=xx ; echo xx=$xx

fn a b # вызов функции fn с параметрами

echo $?

echo xx=$xx

Запуск данного файла выдаст на экран:

749

xx=xx

xx=xx

2

fun: 749 a b

xx=yy

5

xx=yy *Задание для самостоятельной работы:*

Реализовать файл-скрипт (командный файл) для OC Linux, который создает указанный файл в одном из созданных пользовательских каталогов. Если файл создан, то посмотреть номер индексного дескриптора этого файла. Вывести список файлов нового подкаталога. Вывести на экран созданный файл. Скопировать созданный файл в другой каталог. В противном случае вывести соответствующее сообщение. Имя файла можно указать через переменные или через параметры при запуске командного файла.

*Вопросы к главе 6:*

1. Назовите способы создания и запуска командного файла.

2. Как могут быть переданы параметры в командный файл?

3. Опишите назначение операторов if, case, for, while, until.

**7 ТЕСТЫ ДЛЯ САМОПРОВЕРКИ**

В приведенных тестах выберете один правильный ответ.

**7.1 Тест к главе “ Команды ОС Linux ”**

1. Что делает команда cat file1 >> file2?

a) записывает содержимое файла file1 в файл file2;

b) меняет местами содержимое файлов file1 и file2;

c) добавляет содержимое файла file1 в конец файла file2;

d) выводит одно за другим содержимое файлов file1 и file.

2. В каком каталоге находятся домашние каталоги

пользователей?

/bin; /tmp; /usr; /home; /etc.

3. В какой директории программам желательно сохранять файлы,

которые регулярно изменяются, но не являются временными?

a) /bin; b) /tmp; c) /var; d) /root; e) /etc.

4. На экран терминала данную информацию

325201 drwx------ 3 user1 users 4096 May 13 21:16 Desktop/

228983 -r--r--r-- 1 user1 users 40 Apr 23 18:59 err

228881 -rw-r--r-- 1 user1 users 52 Apr 24 19:44 file1

228984 lrwxrwxrwx 1 user1 users 5 Apr 23 19:32 soft -> file1

356714 drwxr-xr-x 2 user1 users 4096 Apr 23 18:06 z/

356715 drwxr-xr-x 2 user1 users 4096 Apr 23 18:06 zz/

356716 drwxr-xr-x 2 user1 users 4096 Apr 23 18:06 zzz/

может выводить команда ls –la, или ls –l, или ls, или ls –li, или ls –lu.

5. В текущем каталоге находятся файлы file1 file2 и каталог dir1.

Какое действие будет выполнять команда mv file1 file2 dir1?

a) file1 и file2 будут скопированы в директорию dir1;

b) file1 и file2 будут перенесены в директорию dir1;

c) file1 будет переименован в file2 и будет выдано сообщение об

ошибке;

d) file1 будет переименован в file2, параметр dir1 будет

проигнорирован. 6. Какой конвейер команд позволяет посчитать количество

файлов в текущем каталоге?

ls | wc –l, или ls -l | wc –l, или ls | fmt, или ls | wc –w.

7. В текущей директории находятся следующие файлы: file12,

file14, file8A, file1F, file2F. Какая команда удаляет файлы file12, file14,

file1F?

a) rm file1?;

b) rm file??;

c) rm file\*[24F];

d) rm file[0-9]?;

e) rm file[0-9]\* .

8. В текущем каталоге находятся подкаталоги dir, dir/dirs,

dir/dirs/dir2. Все указанные каталоги не содержат файлов и

подкаталогов, кроме перечисленных. Какая программа удалит все эти

каталоги?

a) rm dir/dirs/dir2;

b) rm –p dir/dirs/dir2;

c) rm –r dir;

d) rmdir dir/dirs/dir2.

9. С помощью какой команды создается пустой файл?

touch, или rm, или mk, или mkdir.

10. Какое определение является более точным для символической

ссылки?

a) это специальный файл, который содержит в качестве данных

символическую ссылку на другой файл;

b) это ссылка на системный файл;

c) это ссылка на файл;

d) это обычный файл.

11. Что указано во второй колонке строки

-rw-r- -rwx 3 user users 25 Now 10 15:30 file1?

a) число жестких ссылок; b) размер файла;

c) количество файлов;

d) содержимое файла.

12. С помощью какой команды можно изменить имя владельца файла?

chmod, или cat, или chown, или chawn.

13. Если у файла Script установлены следующие права

-rwxrwxr-- 1 user users 23 Okt 16 14:25 Script,

как изменятся права после выполнения команды

chmod g-w, o+x-r Script?

a) -rwxr-x--x 1 user users 23 Okt 16 14:25 Script;

b) drwxrwx--x 1 user users 23 Okt 16 14:25 Script;

c) -r-xrwx--x 1 user users 23 Okt 16 14:25 Script.

**7.2 Тест к главе “Программирование в оболочке Shell”**

1. Что хранит переменная $0 во время выполнения файла-скрипта?

a) первый параметр;

b) количество параметров, переданных скрипту;

c) имя исполняемого скрипта;

d) все параметры, переданные скрипту, заключенные в кавычки.

2. Какой должна быть первая строка скрипта, чтобы он выполнялся с помощью Bourne shell?

#!/bin/csh, или #/bin/sh, или #!/bin/sh, или $/bin/sh.

3. Какое выражение правильно добавляет в переменную окружения PATH каталог /usr/bin?

a) PATH+=/usr/bin;

b) PATH=/usr/bin;

c) $PATH:/usr/bin;

d) PATH=$PATH:/usr/bin.

4. Что делает следующий скрипт? #!/bin/sh

n=$1

f=$1

while [ $n -gt 1 ]

do

n=`expr $n – 1`

f=`expr $f \* $n`

done

echo "f($1) = $f."

a) считает факториал числа;

b) считает сумму 1 + 2 + 3 + .. + n;

c) определяет, является ли введенное число факториалом некоторого целого числа;

d) определяет, представимо ли введенное число в виде 1 + 2 + 3 + ... + n;

e) перемножает все цифры введенного числа.

5. Если значение переменной x равно 10, чему будет равно $x$x?

x10$, или 1010, или 10$x, или x$10, или $x$x.

6. Каким символом обозначаются комментарии в шелл?

//, или /\*, или {, или #.

7. В каком случае следующий командный файл с приведенным фрагментом кода выведет YES?

if [ -f $A ]

then

echo YES

fi

a) когда переменная A имеет значение;

b) когда файл, имя которого записано в переменную A, существует;

c) никогда не выведет;

d) в коде допущена синтаксическая ошибка.

8. Какой оператор применяется для завершения оператора for в bash? }, или end, или esac, или ), или done, или fi.

9. Выберите вариант, в котором на стандартный вывод будет выведено содержимое переменной var:

a) echo var;

b) echo "var";

c) echo "$var";

d) echo '$var'.

10. Каково назначение команды unset?

a) переопределение констант;

b) переназначение констант;

c) запрет переменной на изменение (unable to set) ;

d) удаление переменной (переменная перестает иметь значение).

11. Для чего используется переменная окружения PATH?

a) для настройки приглашения командной строки;

b) для определения директорий, в которых шелл ищет команды;

c) для сохранения выполненных команд и создания макросов;

d) для определения рабочей директории.

**Список использованных источников**

1. Сайт Mandriva Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.mandriva.ru/

2 Сайт Fedora Core Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://fedoraproject.org/

3. Сайт Suse Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.opensuse.org/

4. Сайт Debian GNU/Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.debian.org/

5. Сайт Ubuntu Linux[Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.ubuntu.ru/

6. Сайт Gentoo Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.gentoo.org/

7. Сайт Slackware Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.slackware.com/

8. Сайт Knoppix Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.knoppix.net/

9. Сайт CentOS [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.centos.org/

10. Сайт ASP Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.asplinux.ru/

11. Сайт ALT Linux [Электрон. ресурс]: [официальный сайт]. – Режим доступа: http://www.altlinux.ru/

12. Gartner делает прогнозы относительно развития Linux // Электронный журнал об информационной безопасности [Электронный ресурс]. – Режим доступа: http://www.security.uz/

13. Windows лидирует на рынке серверов // Журнал Xakep-online

[Электронный ресурс]. – Режим доступа: http://www.xakep/ru]

14. Таненбаум Э. Современные операционные системы / Э.Таненбаум. – СПб.: Питер, 2002. –1040 с.

15. Костромин В.А. Самоучитель Linux для пользователя / В.А. Костромин. – СПб.: БХВ-Петербург, 2004. – 672 с.

16. Сивер Э., Спейнауэр С., Фиггинс С., Хекман Д. Linux: Справочник / Э.Сивер, С.Спейнауэр и т.д. пер. с англ. – СПб.: Символ-Плюс, 2001. – 912 с.

17. Федорчук А.В. Доступный UNIX: Linux, FreeBSD, DragonFlyBSD, NetBSD, OpenBSD / А.В. Федорчук – СПб.: БХВ-Петербург, 2006. – 672 с.

18. Маслаков В.Г. Видеосамоучитель. Linux / В.Г. Маслаков – СПб.: Питер, 2008. – 304 с.

19. Колисниченко Н.А. Linux на ноутбуке / Н.А. Колисниченко. – СПб.: БХВ-Петербург, 2009. – 704 с.

**ОГЛАВЛЕНИЕ**

Введение……………………………………………………..…………...3

1. История развития операционных систем семейства Linux…….…4

Вопросы к главе 1……………………………………………………..…..5

2. Краткая характеристика дистрибутивов Linux………………………6

2.1.Mandriva Linux ……………………………………………………….6

2.2. Fedora Core……………………………………………………………7

2.3. Suse Linux ……………………………………………………….……8

2.4. Debian GNU/Linux…………………………………………………….8

2.5. Ubuntu/Kubuntu ……………………………………………………..9

2.6. Gentoo Linux…………………………………………………………10

2.7. Slackware Linux……………………………………………………..10

2.8.Knoppix………………………………………………………………10

2.9. CentOS…………………………………………………………….…11

2.10. ASPLinux…………………………………………………………...11

2.11. ALTLinux…………………………………………………………...11

Вопросы к главе 2…………………………………………………….…11

3. Области применения Linux -подобных систем…………………….12

Вопросы к главе 3…………………………………………………….…14

4. Организация файловой системы ОС LINUX………………………..15

4.1. Структура логического раздела…………………………………...15

4.2. Типы файлов………………………………………………………….17

4.2.1. Файлы физических устройств…………………………………....17

4.2.2. Ссылки……………………………………………………………..20

4.3. Именование файлов и каталогов…………………………………..20

4.4. Структура корневого каталога……………………………………..23

4.5. Монтирование файловых систем………………………………….27

Вопросы к главе 4…………………………………………………….…29

5.Команды операционной системы LINUX ………………………..….30

5.1. Оболочка, виртуальная консоль и терминал……………………..30

5.2. Структура команд оболочки shell………………………………....33

5.3. Запуск программ из командной оболочки…………………….…34

5.4. Редактирование командной строки. История команд………..…36 5.5. Источники справочной информации в ОС Linux………….....................40

5.6.Команды, обеспечивающие навигацию по файловой системе………………………………………………………………………………..41

5.7. Команды работы с файлами и каталогами……………………….44

5.7.1. Просмотр текстовых файлов………………………………….…44

5.7.2. Команды создания файлов и каталогов…………………………46

5.7.3. Команды для манипуляций с файлами и каталогами………….46

5.8. Управление правами доступа к файлам…………………….…….49

5.9. Перенаправление ввода и вывода, конвейеры……………………55

Задание для самостоятельной работы………………………...………..58

Вопросы к главе 5…………………………………………………….…58

6. Программирование в оболочке Shell….…………………………....59

6.1. Создание и запуск командного файла……………………………59

6.2. Переменные и параметры командного файла ……………………61

6.3. Условный оператор if………………………………………………67

6.4. Оператор вызова case………………………………………………71

6.5. Оператор цикла с перечислением for ……...……………………..73

6.6. Оператор цикла с истинным условием while…………...………...75

6.7. Оператор цикла с ложным условием until……………...…………76

6.8. Функции в shell ……………………………………………...............78

Задание для самостоятельной работы……………………………...….79

Вопросы к главе 6……………………………………………………….79

7. Тесты для самопроверки……………………….…………………….80

7.1. Тест к главе “ Команды ОС Linux ”…………………………..…...80

7.2. Тест к главе “ Программирование в оболочке Shell ”……..……..82

Список использованных источников..………………………………....85