Учебный курс

Основы администрирования Linux
L-adm-1

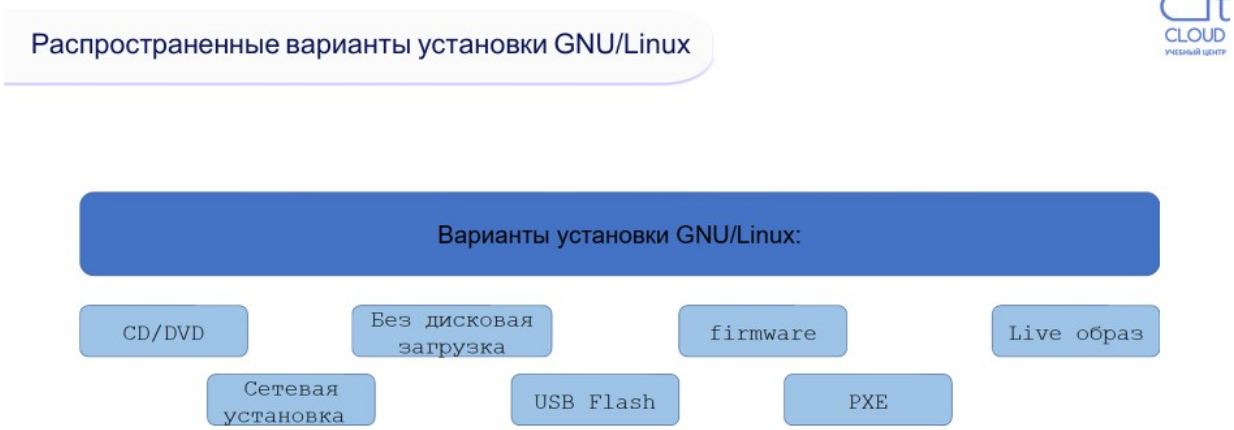
Автор: Лесковец В.В.

г.Екатеринбург
2026

Оглавление

Глава 1. Установка GNU/Linux.

1.1. Варианты установки GNU/Linux.



Операционные системы GNU/Linux могут работать на самых разнообразных аппаратных платформах: от устройств типа Интернета вещей до суперкомпьютеров.

Кроме обычных вычислительных систем Linux активно применяется и в виртуальных средах: виртуальные машины и контейнеры.

Для основных ныне использующихся дистрибутивов GNU/Linux можно выделить следующие стандартные варианты установки на компьютеры:

1. С CD/DVD или USB Flash диска.
2. С FTP/HTTP/NFS/SMB сервера.
3. Запуск сетевой установки с PXE

Загрузочный USB диск создается из образа CD/DVD.

Для установки из сети с FTP, HTTP или NFS сервера обычно требуется наличие либо загрузочного диска, либо специального загрузочного компакт-диска.

Другая альтернатива подготовить серверы для централизованной сетевой установки с использованием PXE (Preboot eXecution Environment).

В процессе установки может потребоваться наличие диска, содержащего драйверы для блочных устройств и/или для SCSI хост адаптеров.

Большинство производителей дистрибутивов предлагает оригинальные программы для установки GNU/Linux, обычно не совместимые между собой.

Примечание: Многие из этих программ предоставляют возможности настройки и автоматизации процедуры установки, как, например, установочная программа anaconda от Red Hat Linux. Использование возможности автоматизации установки приобретает особое значение при необходимости массовой установки GNU/Linux на множестве однотипных серверах или, например, узлах вычислительного кластера.

Возможности автоматизации установки обычно заключаются в способности установочной программы выбирать программное обеспечение для установки из заранее подготовленного списка.

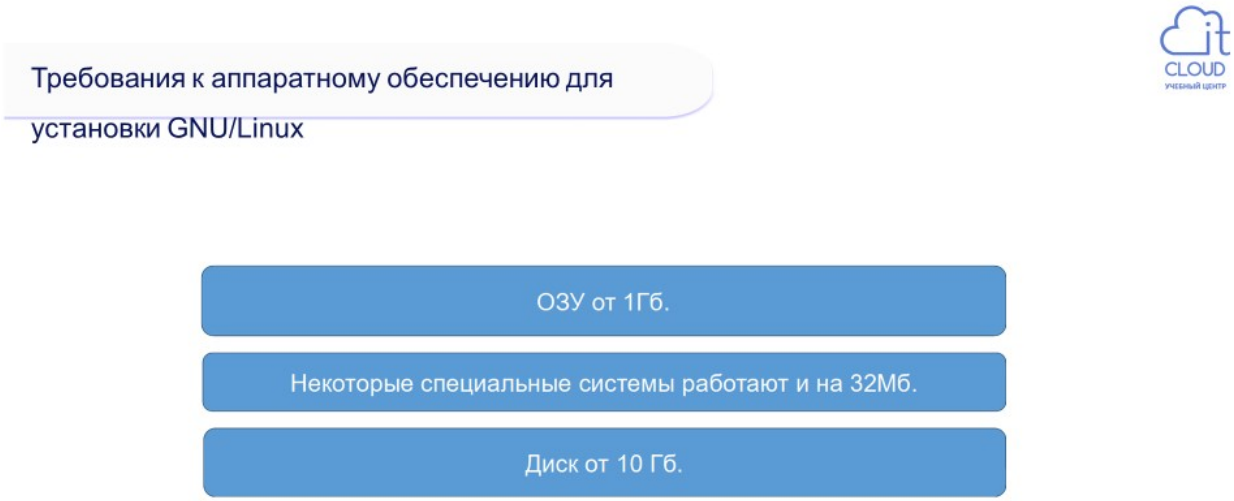
Примечание: Для компьютеров с одинаковой или близкой конфигурацией важно также, чтобы программа установки была способна автоматически настраивать устанавливаемое программное обеспечение с учетом заранее занесенных в файл ответов на вопросы, связанные с созданием дисковых разделов, файловых систем и конфигурации оборудования.

Возможна ли установка конкретного дистрибутива GNU/Linux с некоторого требуемого вида носителя зависит от производителя этого дистрибутива.

Информация о возможных видах установки данного дистрибутива может быть найдена либо на дисках установочного комплекта, либо на сайте производителя этого дистрибутива.

GNU/Linux может быть установлен на жесткий магнитный диск совместно с другими операционными системами, причем для нормальной установки необходимо минимум два раздела (первичных или логических). Один из них используется для корневой файловой системы, а другой – для раздела подкачки.

1.2. Требования к аппаратному обеспечению для установки GNU/Linux.



GNU/Linux может быть установлен практически на любой компьютер с архитектурой x86_64 (а также на множество других аппаратных платформ).

Примечание: Поддерживаются процессоры от i386 до самых современных. Современные дистрибутивы Linux как правило ориентированны на 64-разрядные процессоры.

Требования к объему оперативной памяти зависит, во-первых, от выбранного дистрибутива, а во-вторых от предполагаемых задач, которые планируется запускать на данном компьютере.

Примечание: Старые ядра способны работать даже при 2 Мб. Однако для нормальной работы рекомендуется не менее 32 Мб. При необходимости работы в X Window это значение увеличивается до 64 Мб минимум. Многие дистрибутивы требуют еще больших объемов ОЗУ. Например, полнофункциональная работа RH Enterprise Linux 3.0 возможна лишь при минимум 256 Мб ОЗУ.

1. Обычно требуется не менее 1ГБ оперативной памяти.
2. Рекомендуемый объем ОЗУ для x86_64 обычно 2ГБ.
3. Объем диска так же зависит от выбранного дистрибутива и профиля системы.
4. Обычно для сервера требуется не менее 2ГБ дискового пространства.
5. Для рабочей станции требуется не менее 10Гб.

6. Для установки GNU/Linux нужен стандартный SATA контроллер для жестких дисков и, по желанию, HDD/SSD диск. Также могут быть использованы SCSI диски.

Примечание: При этом следует убедиться в том, что SCSI хост адаптер поддерживается в данном ядре Linux (см. Hardware-HOWTO, SCSI-HOWTO, документацию на ядро и информацию от поставщика дистрибутива). Обычно поддержка распространенного SCSI оборудования в GNU/Linux проблем не вызывает. Кроме IDE и SCSI дисков поддерживаются устаревшие RLL и MFM диски. Ядра 2.4 и 2.6 поддерживают USB диски, хотя корневую файловую систему на них обычно не устанавливают. Поддерживаются AT диски, SCSI и ATA RAID контроллеры. Для проверки поддержки конкретного оборудования следует обратиться к документации производителя и, возможно, в Hardware-HOWTO.

GNU/Linux работает с любыми существующими видеоадаптерами в текстовом режиме.

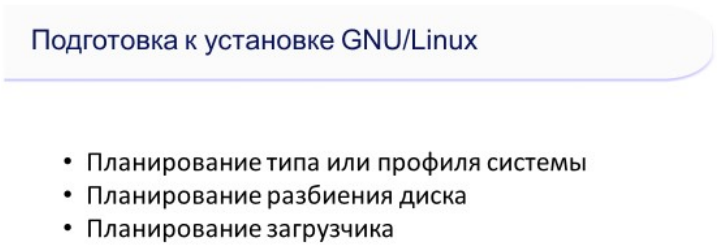
Работа в графическом режиме зависит от поддержки данного графического оборудования реализацией X Window.

Примечание: При установке GNU/Linux следует заранее уточнить версию реализации X Window, использованную в дистрибутиве. Возможно, будет необходимо проверить поддерживает ли данная реализация установленный видеоадаптер.

Установка проприетарного видео драйвера возможна после установки ОС.

GNU/Linux обеспечивает поддержку ATAPI и SCSI CD/DVD дисков. Наличие их не является обязательным в случаях, когда с них не будет производиться установка.

1.3. Подготовка к установке GNU/Linux.



Для успешной установки GNU/Linux важно подготовить список комплектации оборудования (installation hardware checklist). В этом документе должны быть отражены основные аспекты конфигурации аппаратного обеспечения.

Другим важным моментом при подготовке к установке GNU/Linux является планирование типа или профиля системы, определяемого основным направлением использования системы. От этого профиля зависит набор пакетов программного обеспечения, который требуется установить.

Распространенные программы установки GNU/Linux от основных производителей дистрибутивов часто предлагают следующие профили установки:

1. Рабочая станция;
1. Сервер;
2. Сервер для облака и др.

Набор программных пакетов, который должен быть установлен, может быть полностью определен пользователем, либо же выбран и дополнен на основе предопределенного набора. Поэтому до начала установки следует составить список требуемого программного обеспечения для установки.

Перед установкой следует решить, какая схема аутентификации будет использоваться.

Особое внимание следует уделить вопросу планирования разбиения дисков на разделы и определению точек монтирования. Неудачный выбор схемы разбиения жесткого диска на разделы может привести к недостатку дискового пространства в одной файловой системе при явном избытке свободного пространства в другой.

Наиболее часто на отдельных файловых системах монтируются каталоги:

/boot – для него редко требуется более 1 Гб, тип файловой системы желательно установить ext3 или ext4;

/boot/efi — требуется в обязательном порядке создавать на компьютерах с загрузкой посредством UEFI. Нужно 600 Мб. Больше обычно не требуется.

/home – его размер следует выбирать исходя из планируемого количества пользователей, отводя для каждого из них некоторый средний размер занимаемого пространства (например, 100 Мб x 10 пользователей = 1 Гб);

/usr – в этот каталог устанавливается основная масса программного обеспечения, поэтому в зависимости от профиля установки для массовых дистрибутивов рекомендуется выбирать от 1 Гб для Internet серверов до 20-30 Гб для рабочих станций;

/opt – каталог для установки дополнительного программного обеспечения, использование которого целиком зависит от воли сборщиков дистрибутива, поэтому он может совсем не использоваться, но он может потребовать и значительного дискового пространства.

/var – в большинстве случаев для этого каталога требуется не более 10 Гб, однако на серверах баз данных это пространство может быть больше на порядки;

/tmp – обычно достаточно 200-300 Мб;

В случае, если указанные выше каталоги смонтированы на отдельных разделах, то для корневого раздела не требуется более 1 Гб дискового пространства.

Обычно программы установки современных дистрибутивов GNU/Linux предоставляют возможность создать на жестких дисках требуемые разделы и определить файловые системы, которые будут размещены на них.

Примечание: Разделы могут быть созданы также и заранее, например, с помощью команды fdisk. Если же уже существующая структура разделов на жестком диске не оставляет свободного пространства для GNU/Linux, то можно уменьшить размеры FAT разделов с помощью утилиты fips.exe. Она обычно находится на первом диске установочного комплекта. Существуют также программы, позволяющие уменьшать размеры разделов с файловой системой NTFS.

Следует заранее решить каким образом будет загружаться устанавливаемый GNU/Linux. В подавляющем большинстве случаев для этого используется загрузчик GRUB или GRUB2, размещаемый либо в MBR, либо в активном разделе.

Примечание: Практикуется также и способ загрузки с дискеты. Гораздо реже на обычных (обладающих собственными дисками) рабочих станциях и серверах встречается загрузка по сети.

После проведенной подготовки можно переходить к установке GNU/Linux.

1.4. Установка GNU/Linux.

Различные программы для установки GNU/Linux, поставляемые производителями дистрибутивов, значительно отличаются между собой, однако среди них можно выделить следующие основные стадии:

1. Подготовительная. На этой стадии обычно определяются:
    1. Определение типа клавиатуры и терминала. Настройка мыши.
    2. Локализация системы и установка раскладки клавиатуры.
    3. Определение источника установка (CD, HTTP, FTP, NFS и т.д).
    4. В случае загрузки по сети – настройка сетевых интерфейсов.
2. Загрузка требуемых драйверов, например, для блочных устройств.
3. Подготовка файловых систем:
    1. Разбиение диска на разделы.
    2. Создание файловых систем.
    3. Монтирование их к временной корневой файловой системе, используемой на стадии установки.
4. Установка базовой системы.

Примечание: Базовая система представляет собой минимально функциональный GNU/Linux, пригодный для продолжения дальнейшей установки. Этот этап характерен для Debian. В RH установка базовой системы производится в начале установки программных пакетов.

5. Определение профиля устанавливаемой системы и выбор программного обеспечения для установки.
6. Установка программного обеспечения.

Примечание: Обычно эта фаза не интерактивна, но, например, в Debian многие устанавливаемые пакеты сразу же конфигурируются, поэтому в этом дистрибутиве на этой стадии программа установки требует от пользователя ответов на вопросы относительно конфигурации устанавливаемого программного обеспечения.

7. Установка и настройка загрузчика.
    1. Изготовление загрузочной дискеты.
    2. Установка доменного имени системы.
    3. Установка пароля root.
8. Действия после установки:
    1. Определение типа аутентификации: с помощью базы данных паролей, NIS/NIS+, LDAP, Kerberos и т. д.

2. Заведение обычных пользователей.
3. Настройка профилей пользователей.
4. Настройка сети и, возможно, несложная настройка фильтра IP пакетов.
5. Конфигурирование дополнительного оборудования, например, звуковой подсистемы.
6. Установка последних обновлений.
7. Установка дополнительного ПО, не вошедшего в основной дистрибутив.

В зависимости от используемого дистрибутива могут различаться стадии установки базового ПО.

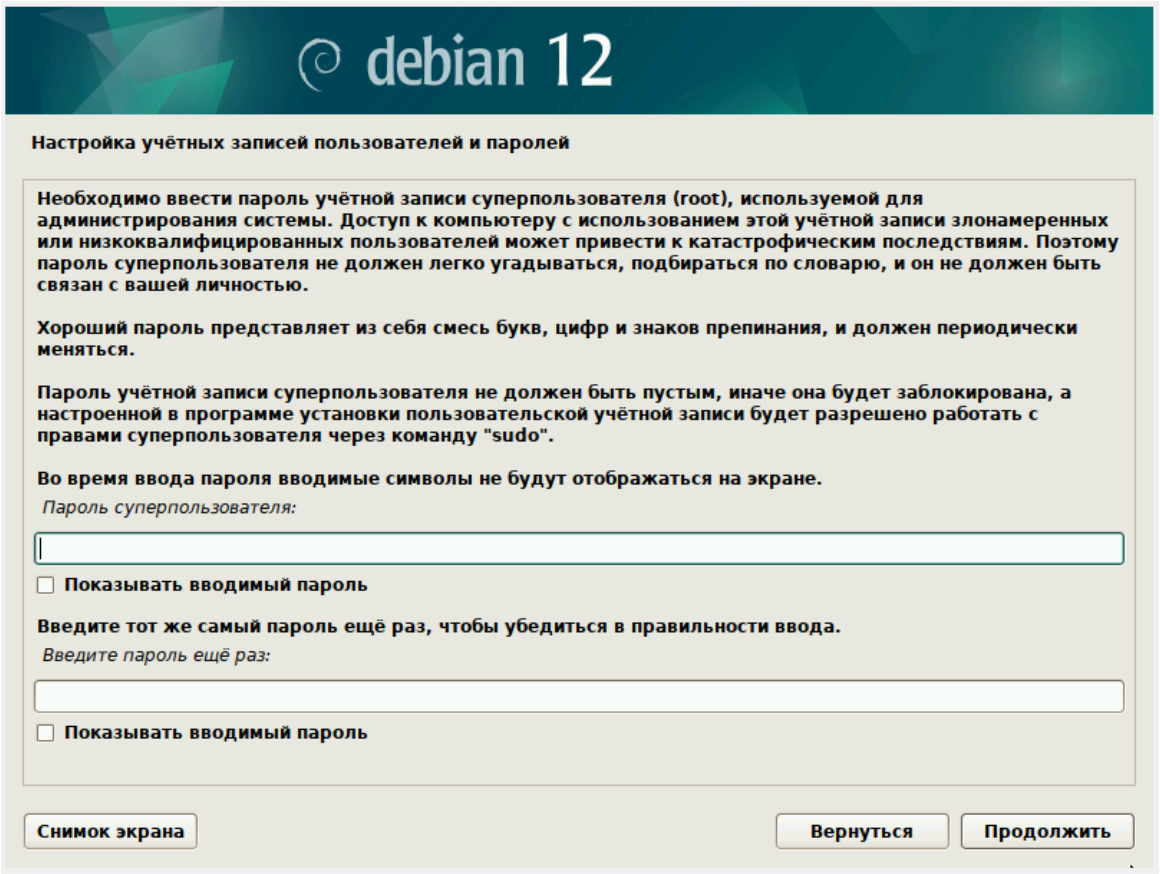
Примечание: Например, RH подобные дистрибутивы не выделяют установку базовой подсистемы в отдельную фазу, так как дистрибутив рассчитан на использование конкретной версии ядра. А Debian, напротив, рассчитан на возможность использования различных ядер даже не Linux, а, например, HURD. Поэтому программа установки Debian сначала устанавливает базовый комплект и выбранное ядро, а затем устанавливает загрузчик и пытается загрузиться. После успешной перезагрузки программа установки продолжает работу, устанавливая дополнительное ПО.

В некоторых дистрибутивах система после установки базового комплекта и установки загрузчика система полностью работоспособна и дальнейшая установка ПО и настройка оборудования выполняется после перезагрузки.

1.5. Пример установки ОС Debian.

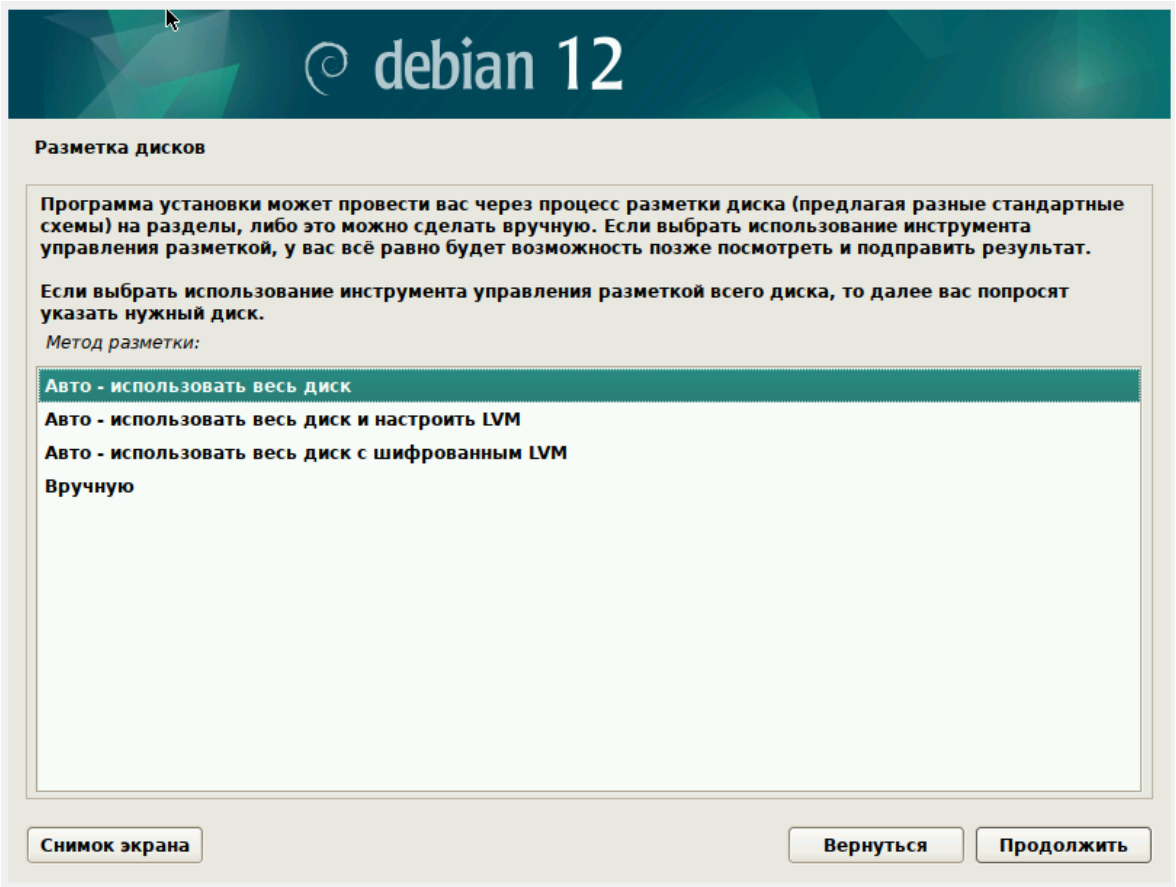
Выбор языка установки и работы системы.

Настройка имени ПК – данное имя будет именем ПК в сети.



Настройка пароля суперпользователя – учетной записи с неограниченными привилегиями. В некоторых ОС данную учётную запись предлагают заблокировать на этапе установки, и создать вместо неё учётную запись с правами администратора.

Настройка имени первой пользовательской учетной записи, также аналогично записи суперпользователя далее происходит настройка пароля. Под данной учетной записью далее происходит вход в систему.



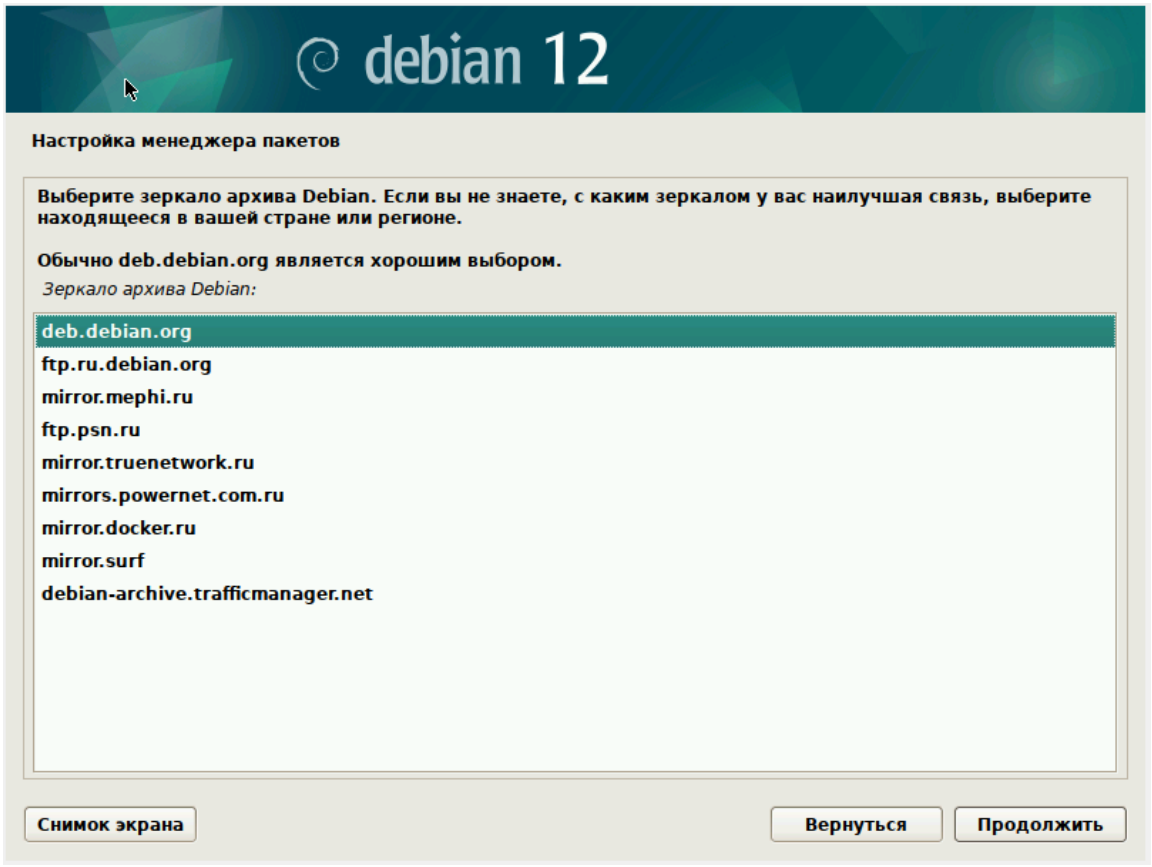
Выбор типа разметки диска – для новичков рекомендуется автоматический вариант.

Помимо простой автоматической разметки диска также предлагаются варианты с использованием системы LVM в вариантах с шифрованием данных и без.

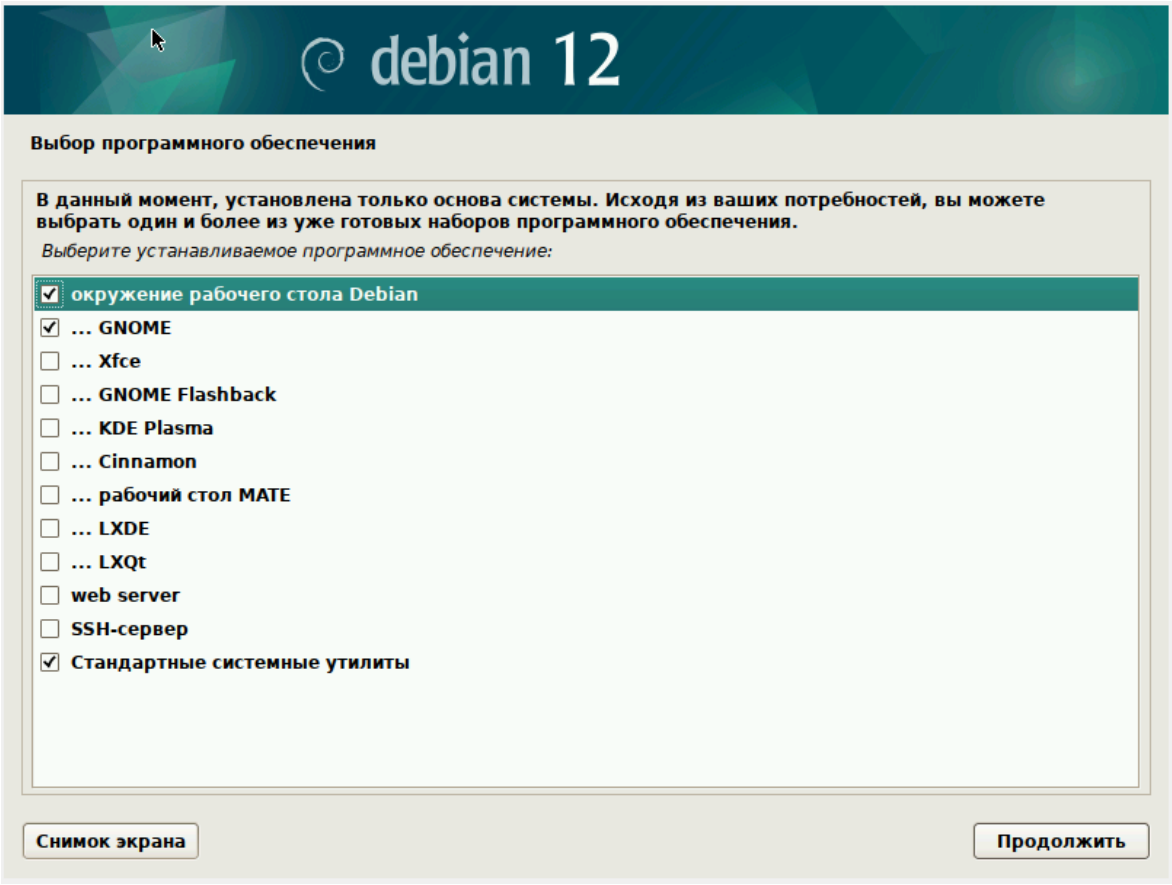
Примечание: LVM (Logical Volume Manager) — это подсистема Linux, обеспечивающая гибкое управление дисковым пространством, позволяя объединять физические диски в пулы, динамически изменять размер логических разделов «на лету» и создавать снимки (snapshots) данных. LVM абстрагирует файловые системы от физических носителей, позволяя томам занимать место на нескольких дисках одновременно. Данная технология рассматривается в курсах продвинутого администрирования Linux систем.

Также вы можете выбрать вариант "Вручную" - для создания разделов с нуля.

Выбор схемы разметки диска при автоматическом типе разметки. Отдельные разделы для каталогов /home, /var и /tmp рекомендуются в случае либо ограниченного размера диска, либо больших объемов пользовательских данных.

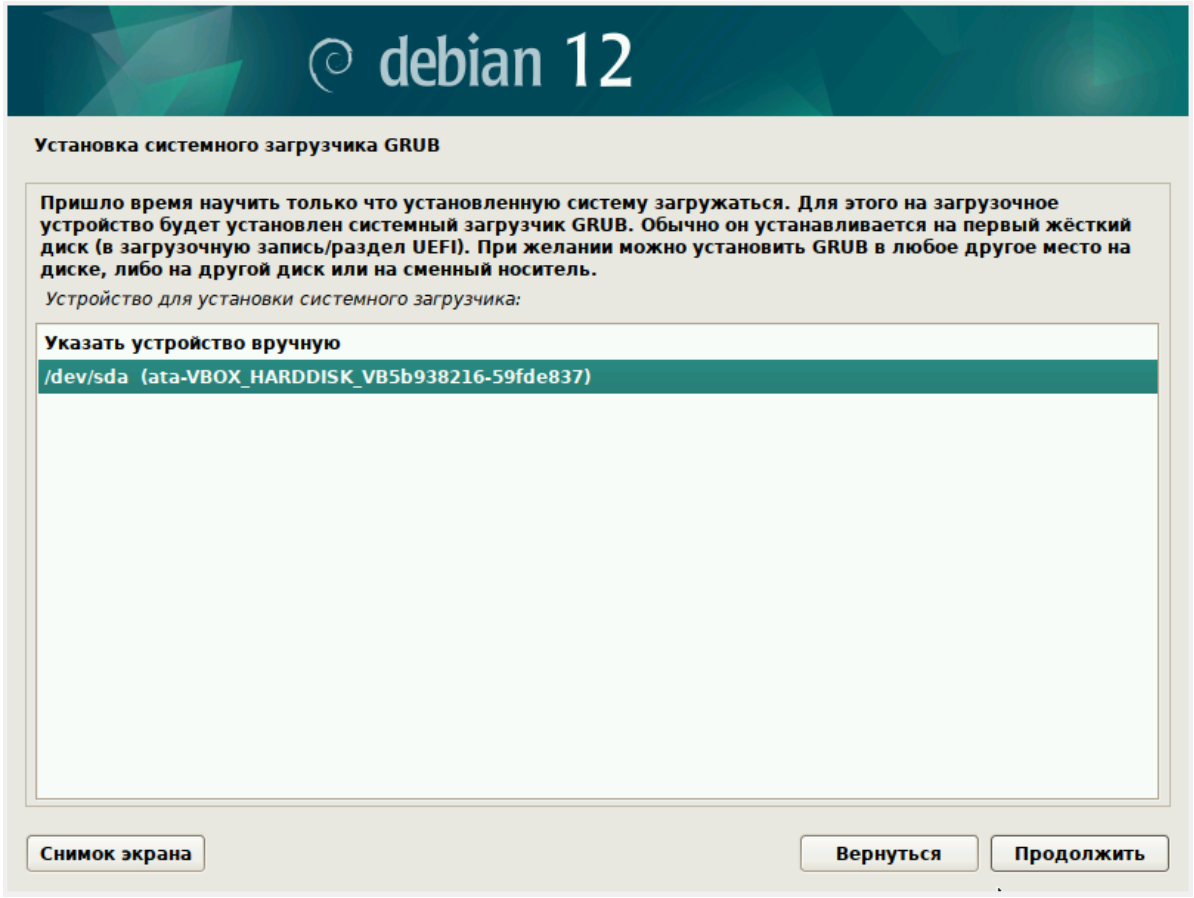


Выбор зеркала для установки пакетов – влияет на скорость скачивания устанавливаемой системы.

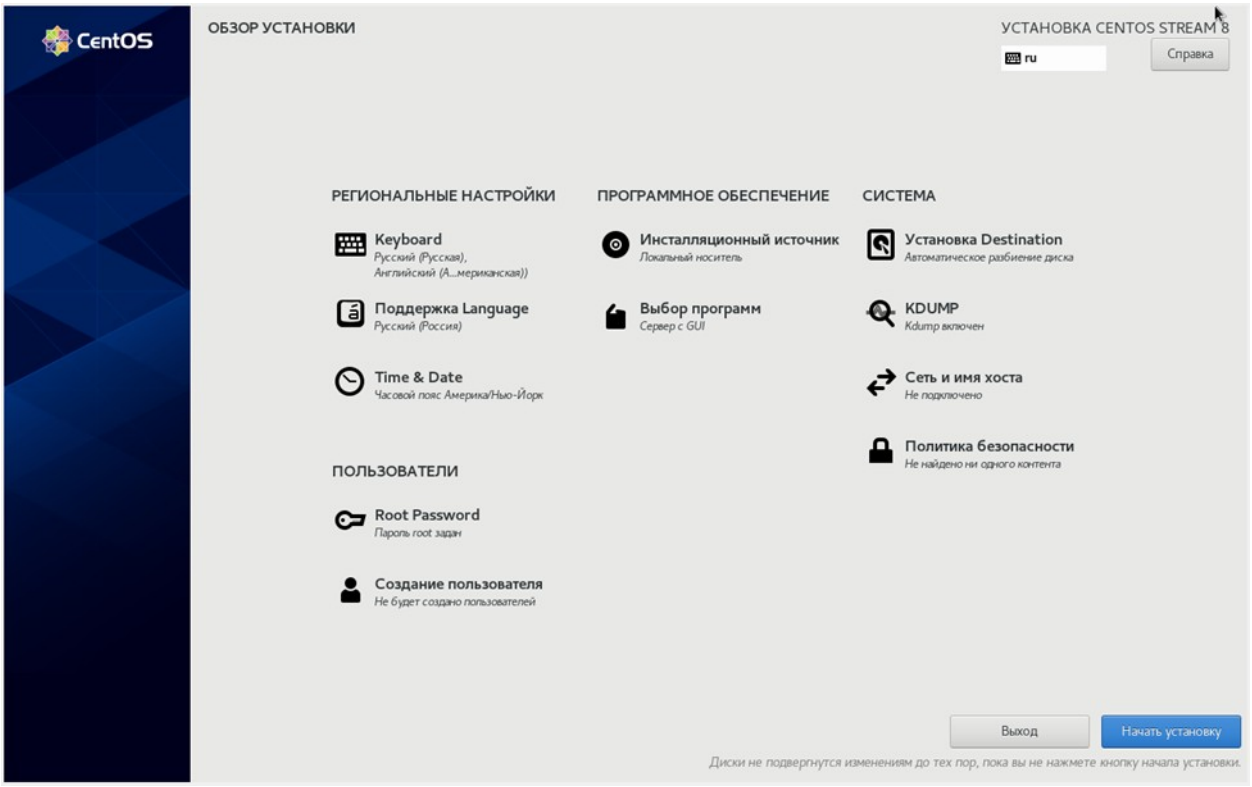


Выбор графического окружения, от него зависит внешний вид системы после установки. Можно выбрать современные графические окружения – GNOME или KDE требующие больше системных ресурсов, либо более простые. Данный выбор следует делать исходя из ресурсов ПК и задач, выполняемых на нем. Иногда разумным решением является отсутствие графического окружения.

Также предлагается сетевое ПО для веб-серверов и удаленного управления.



Выбор места установки загрузчика.



У разных дистрибутивов используются разные программы установки ОС. На рисунке представлена программа установки anaconda, используемая в RedHat системах. Ее главная особенность – все опции находятся в одном меню вместо последовательного их выбора.

Глава 2. Начало работы в GNU/Linux

2.1. Учётная запись и вход в сеанс

GNU/Linux является многопользовательской операционной системой, в которой одновременно может работать несколько различных пользователей. Для каждого пользователя, имеющего право на работу с данной системой, системным администратором создается так называемая учетная запись (account).

Учетная запись содержит в себе важнейшую для входа в систему информацию о пользователе

1. имя пользователя и идентификатор пользователя
2. пароль и его атрибуты
3. домашний каталог
4. оболочку (login shell), запускаемую при входе в сеанс
5. первичная группа

Имя пользователя идентифицирует пользователя в системе, а пароль предназначен для исключения несанкционированного входа в сеанс под именем этого пользователя.

Одна из важнейших составляющих системного администрирования операционной системы GNU/Linux – разработка и неукоснительное выполнение так называемой системной политики. Системная политика –

документ, определяющий правила работы системы и процедуры обеспечения ее работоспособности. В системной политике должны быть установлены правила именования пользователей.

Примечание: Так, например, может быть положено имя пользователя составлять из первой буквы его имени и не более восьми букв из его фамилии (например, для пользователя Натальи Симоновой имя пользователя может быть nsimonova).

В системе не может быть зарегистрировано два пользователя, имеющих одинаковые имена. Имя пользователя в GNU/Linux может содержать в себе разнообразные символы.

Примечание: Во многих системах допускается регистрация пользователей с именами, составленными из символов кириллицы.

В целях исключения возможных конфликтных ситуаций с программным обеспечением и из соображений совместимости с другими системами настоятельно рекомендуется придерживаться следующих правил:

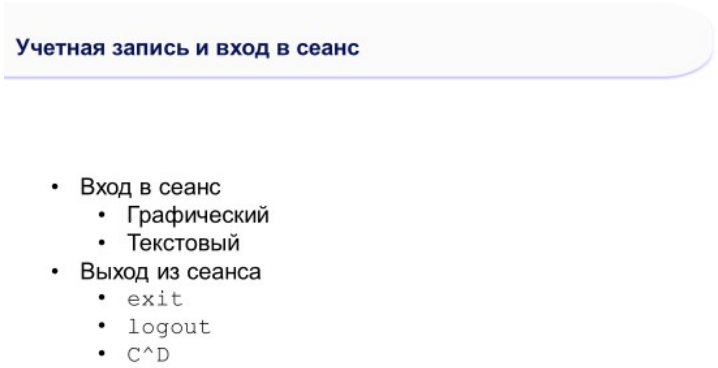
1. Используйте буквы английского алфавита, цифры и символ подчеркивания.
2. Избегайте использования символа пробела.
3. Запрещено использование символа двоеточия.
4. Следует избегать использования метасимволов (@,#,$ и прочих).
5. Не используйте специальные символы ASCII, например, символ табуляции и прочие.

В современных версиях Linux обычно не налагается жестких ограничений на длину имени пользователя, как, например, это было до недавнего времени во многих UNIX системах. Обычная схема аутентификации (то есть проверки подлинности) пользователя, производимая при попытке его входа в систему, требует ввода пользователем его пароля. Пароль пользователя является важнейшим инструментом защиты системы от несанкционированного доступа, поэтому в системной политике должны быть с особой тщательностью сформулированы правила формирования паролей пользователей. Нельзя быть уверенным, что пароль выбран достаточно хороший для надежного исключения несанкционированного проникновения в систему. Однако, имеются стандартные рекомендации правил выбора паролей и работы с ними:

1. Пароль не должен быть слишком коротким. Обычно рекомендуется иметь пароли не менее восьми символов длиной при условии наличии в

них букв в верхнем и нижнем регистрах, цифр и метасимволов.

2. Не следует выбирать пароли, основанные на имени пользователя или основанные на его характерных чертах (антропометрических, эмоциональных и прочих).
3. Пароль не должен быть словарным словом, а если он основан на модификации словарного слова, то эта модификация должна изменить не менее трети символов слова при условии добавления в слово дополнительных символов и перемешивании символов в нем.
4. Пароли, основанные на русских словах, набранных транслитом или подобным способом, нельзя признать надежными.
5. Опасно использовать пароли с неограниченным сроком действия.
6. Нельзя позволять пользователям менять пароль сразу же после того, как он был сменен после системного предупреждения о его устаревании, так как пользователь с большой долей вероятности попытается вернуть себе старый пароль.
7. При смене устаревшего пароля новый пароль должен значительно отличаться от старого, в противном случае ценность смены пароля крайне незначительна.
8. Пароль должен быть (при условии выполнения предыдущих требований) таким, чтобы его можно было запомнить наизусть. Если пользователь не способен на это, то либо следует обучить пользователей специальным приемам образного запоминания, либо позволить пользователю хранить пароль в надежно защищенном месте в виде твердой копии.
9. Нельзя произносить вслух пароли и передавать их по незащищенным каналам связи.
10. Следует выявлять попытки пользователей обменяться паролями и объяснять им порочность подобной практики.
11. При использовании программ автоматической генерации паролей следует регулярно проверять степень энтропии результатов генерации паролей этими программами.



Характерное приглашение на ввод имени пользователя для входа в сеанс при использовании текстовой оболочки выглядит так:

```
GNU/Linux on TTY2
login:
```

После приглашения `login:` следует ввести имя пользователя, зарегистрированного в системе. Ввод следует завершить нажатием на клавишу Enter.

После этого на экран будет выведено приглашение ввести пароль:

```
password:
```

Ввод пароля не сопровождается отображением вводимых символов из соображений безопасности.

Если при вводе пароля была допущена ошибка, то можно воспользоваться клавишей Back Space (забой) для исправления неверно введенного символа.

Если при вводе пароля все-же будет допущена ошибка, то на экране будет отображено сообщение `Login incorrect`

Пример:

```
GNU/Linux on TTY2
```

```
login:student
password:
login incorrect
```

Примечание: В этом примере пользователь student попытался войти в сеанс, однако ввел неверный пароль. Поэтому система выдала сообщение login incorrect, и вход в сеанс осуществлен не был.

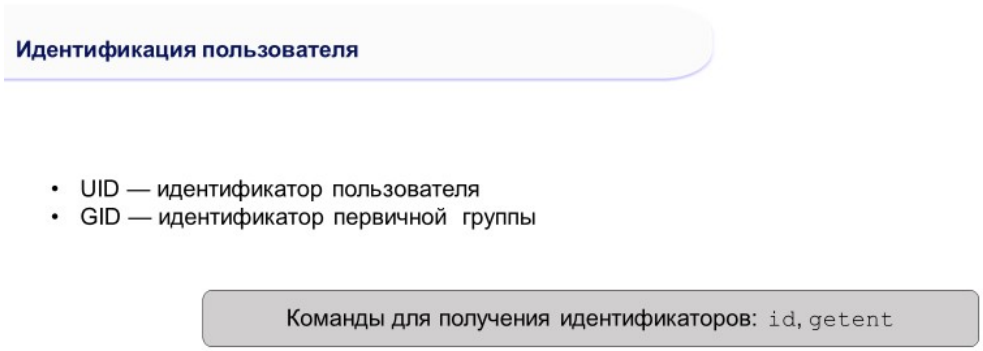
Система Linux может быть настроена так, что после нескольких неудачных попыток входа терминал может быть заблокирован.

Для выхода из сеанса необходимо набрать команду `exit` . Можно также использовать команду `logout` .

Удобно также использовать сочетание клавиш C^D, однако оно может быть заблокировано с помощью специальной настройки оболочки bash (блокировка достигается командой `set -o ignoreeof`).

После выхода из сеанса на экране вновь появится приглашение ввести имя пользователя `login:` .

2.2. Идентификация пользователя.



Каждая пользовательская учетная записи содержит два числа – UID и GID.

Эти числа идентифицируют пользователя в системе и предназначены для определения его прав доступа к файлам.

UID – идентификатор пользователя (User ID)

GID – идентификатор первичной группы пользователя (Group ID).

Каждый пользователь в Linux может быть одновременно членом многих групп, но обязательно должен состоять в одной (первичной) группе.

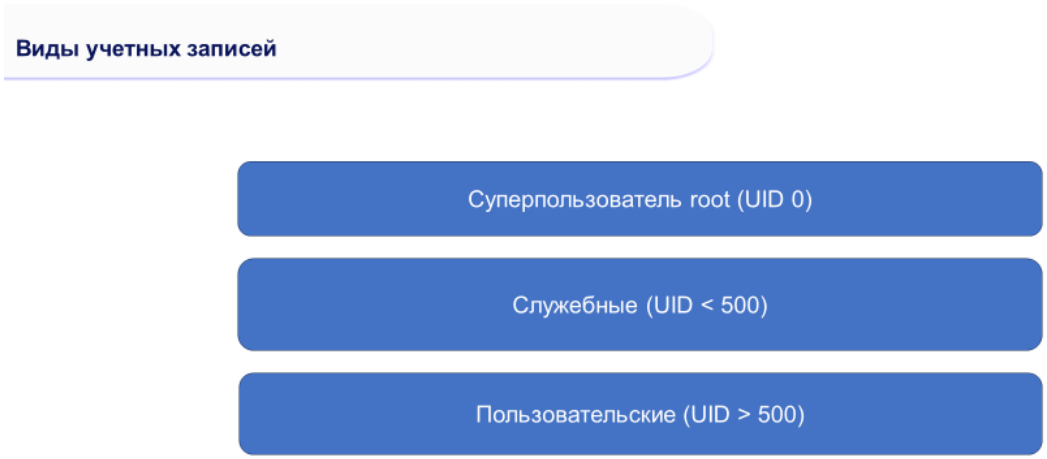
Для определения идентификаторов пользователя, а также всех групп, в которых пользователь участвует, можно использовать команду id

Пример:

```
$ id
uid=503(user2) gid=503(user2) группы=503(user2),22(cdrom)
```

Примечание: Из информации, выведенной командой id легко увидеть, что UID пользователя user2 – 503, первичной группой для этого пользователя является группа user2 с GID 503. Кроме этой группы пользователь входит в группу cdrom, идентификатор которой 22. Группа user2 является приватной группой для пользователя user2. Пользователь, имеющий приватную группу, является ее единственным членом. Такие группы создаются автоматически в Red Hat Linux (и подобных дистрибутивах) специально для регистрируемых пользователей.

2.3. Виды учетных записей



Суперпользователь — имя `root`, домашний каталог `/root`, UID=0. В некоторых дистрибутивах по умолчанию, пароль для `root` не задан, поэтому войти в систему с использованием учетной записи root нельзя

Обычные пользователи — учетные записи пользователей, которым разрешен терминальный вход в систему. Домашние каталоги, по умолчанию, размещаются в `/home`, UID с 500 (в некоторых дистрибутивах с 1000) до 59999

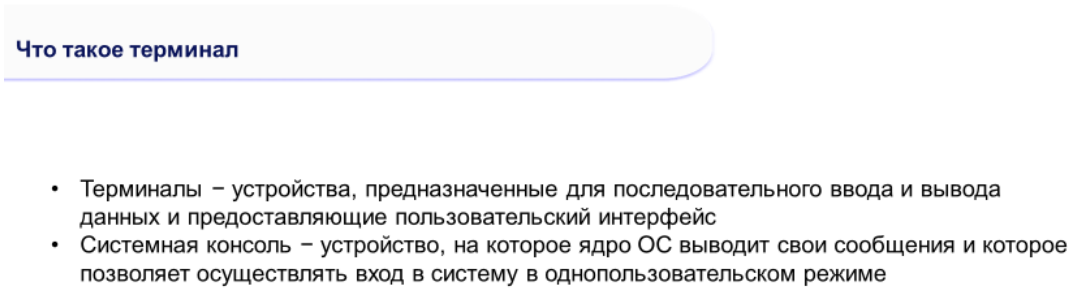
Системные пользователи — учетные записи, которые нужны для функционирования операционной системы и служб. Учетные записи с UID/GID от 0 до 99 используются самой ОС и одинаковы для всех систем. Учетные записи с UID/GID от 100 до 499 (999) используются службами, создаются динамически и могут отличаться

Пример: Так, например, для пользователя mail команда id может выдать следующий результат:

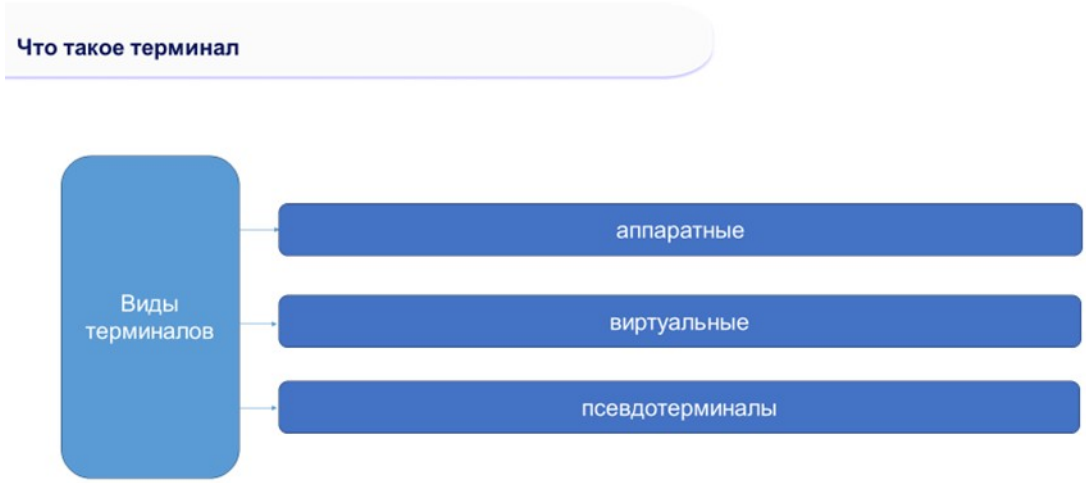
```
$id mail
uid=8(mail) gid=12(mail) groups=12(mail)
```

Примечание: Пользователь mail здесь имеет UID 8 и первичную группу mail с GID 12. Обратите внимание, что для получения идентификаторов любого пользователя достаточно указать имя этого пользователя в качестве аргумента команды id.

2.4. Что такое терминал



Работа человека с компьютером подразумевает использования специального устройства для этого предназначенного. Таким устройством является терминал. Посредством терминала человек может отдавать команды управления, а компьютер сообщать некоторые сведения человеку.Терминал, на который ядро ОС выводит свои сообщения называется системная консоль.

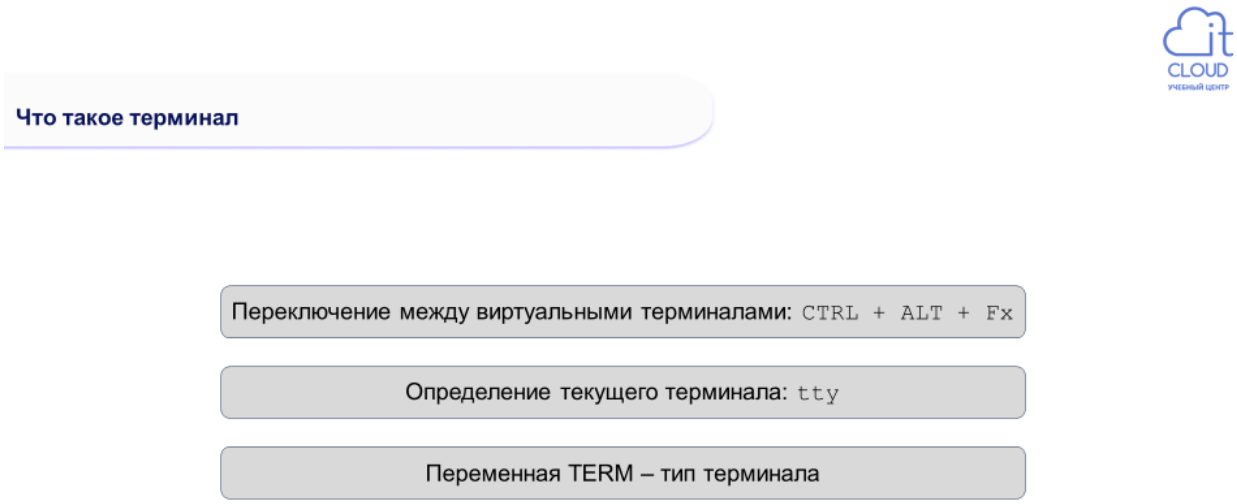


Аппаратный текстовый терминал представляет собой дисплей, клавиатуру, коммуникационный последовательный порт, встроенное программное обеспечение, поддерживающее обработку управляющих последовательностей символов (escape - последовательностей) и настройку коммуникационного порта.

Виртуальный терминал использует видеокарту, клавиатуру и мышь для эмуляции терминала.

Псевдотерминал — программа эмулирующая терминал, используется при удаленных подключениях или при запуске в графике эмуляторов терминала.

В наши дни традиционные аппаратные терминалы используются редко, но виртуальные терминалы и псевдотерминалы используются в Linux на постоянной основе.



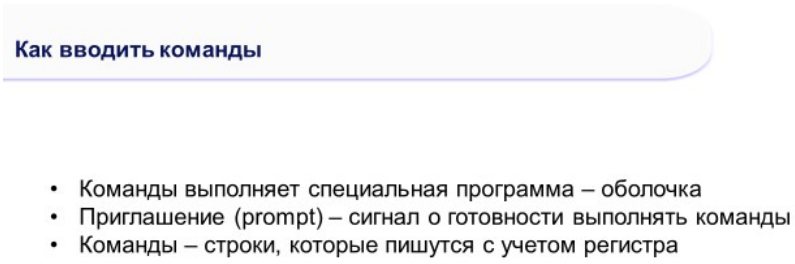
Для переключения между виртуальными терминалами в системе используется сочетание клавиш `CTRL + ALT + Fx`, где x — номер терминала.

Просмотреть на каком терминале вы сейчас работаете можно выполнив команду `tty`, а определить тип терминала можно выведя значение переменной `TERM`.

Часто используемые команды при работе с терминалами:

1. `stty` для вывода и управления параметрами терминала
2. `infocmp` для просмотра информации о терминале и управляющих последовательностях, а также их символьных обозначениях
3. `tput` и `echo -e` производят установку параметров терминала. `tput` использует символьные обозначения, а `echo -e` – сами последовательности
4. `setterm` также устанавливает параметры терминала, но с помощью собственных аргументов
5. `screen` – это простой оконный менеджер, позволяющий производить параллельную работу с несколькими окнами псевдо терминалов на одном физическом
6. `tmux` является мультиплексором терминала, то есть утилита позволяющая работать с псевдотерминалами представленными в виде окон в сессии `tmux`

2.5. Как вводить команды в оболочке?



Оболочка предоставляет интерфейс командной строки, в котором управление операционной системой и запуск программ осуществляется с помощью команд в текстовом виде.

Команды вводятся с учетом регистра символов.

Ввод команд осуществляется с клавиатуры. Команда запускается на исполнение нажатием на клавишу Enter. Однако вместо этого можно пользоваться сочетаниями клавиш `C^J` или `C^M`.

Для удобства пользователей на экран выводится приглашение для ввода команд. Оно называется приглашением командной строки.

Пример: Типичный его вид следующий:

```
[user1@host etc]$
```

Примечание: В этом примере в приглашении командной строки выводится имя пользователя user1, имя хоста host и имя текущего каталога.

Вид приглашения командной строки легко изменить с помощью переменной окружения `PS1`, однако следует придерживаться следующего общепринятого правила:

1. Приглашение должно заканчиваться символом доллара `$` или реже

знаком больше > если это сеанс простого пользователя.

2. Если в сеанс зашел суперпользователь, то приглашение командной строки заканчивается символом решетки #.

Пример: Ниже приведен пример типичного приглашения строки сеанса суперпользователя:

```
[root@host root]#
```

Если команда введена неверно, то система выводит соответствующее сообщение об ошибке.

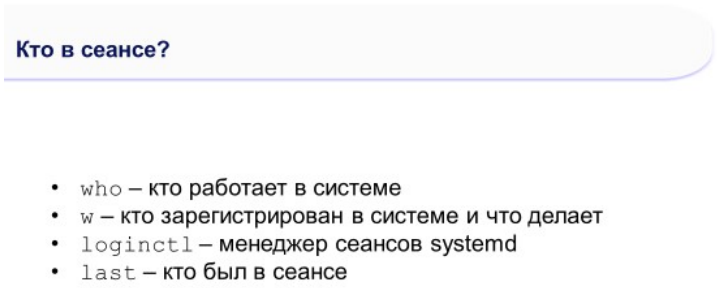
Пример: если команда who введена ошибочно, то на экране будет выведено:

```
$ hwo
bash: hwo: command not found
```

Примечание: Оболочка сообщает, что команда hwo не найдена. Это сообщение выведено вследствие ошибочного ввода команды who.

Очистить экран можно с помощью команды `clear`.

2.6. Кто в сеансе?



Поскольку Linux – многопользовательская система, важно знать сеансы каких пользователей работают параллельно с вашим.

Команда `who` выводит список пользователей, вошедших в сеанс и работающих в настоящее время.

Пример:

```
$ who
user1 tty3 Sep 21 19:10 (localhost)
admin seat0 Sep 21 18:48
user2 pts/1 Sep 21 20:40 (localhost)
```

Примечание: Команда who отобразила список из трех пользователей системы. Во втором столбце команда показывает терминал, с которого пользователь вошел в систему. Пользователь admin вошел в сеанс с помощью графического менеджера сеанса, поэтому во втором столбце указано seat0. Уже после входа в сеанс пользователя admin (время входа отображается в четвертом столбце списка), он решил открыть дополнительный сеанс в графической X сессии от имени пользователя user2, что видно по имени псевдотерминала pts/1. В то же время, на третьем виртуальном терминале tty3 начиная с 19:10 открыт сеанс от имени пользователя user1.

Информация о пользователях, работающих в настоящее время в системе, команда `who` извлекает из файла `/var/run/utmp` (файл бинарный).

Важной опцией команды `who` является `-b`. С ней команда показывает

время, когда была загружена система.

Пример:

```
$ who -b
system boot Sep 21 17:16
```

Примечание: Как видно из листинга, система была загружена 21 сентября в 17:16.

Альтернативно сколько времени прошло с момента загрузки Linux до настоящего момента можно определить, используя команду `uptime`.

Пример:

```
$ uptime
21:02:14 up 3:46, 2 users, load average: 0.06, 0.04, 0.03
```

Примечание: Выведенный листинг отображает текущее время, продолжительность периода, прошедшего после загрузки системы, количество пользователей системы в настоящее время и среднюю загрузку системы за прошедшие 1, 5 и 15 минут.

Кто вошел в систему и какие команды им запущены можно увидеть, выполнив команду `w` .

Пример:

```
$ w
22:29:33 up 5:13, 3 users, load average: 0.00, 0.00, 0.00
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
user1 tty2 localhost 10:29pm 15.00s 0.04s 0.03s -bash
root pts/1 localhost 6:51pm 34.00s 0.06s 0.05s -bash
```

Команда `loginctl` предоставляет информацию о сеансах пользователей и позволяет управлять ими.

Пример:

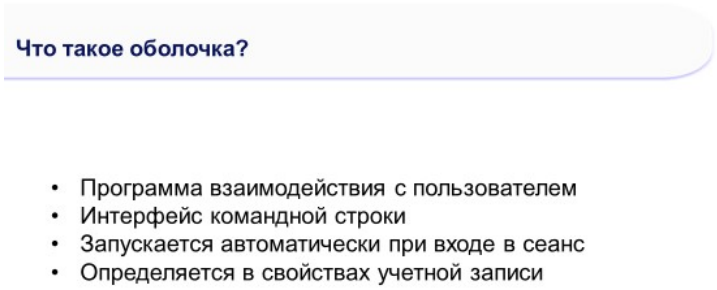
```
$ loginctl
SESSION UID USER SEAT TTY
5 500 admin pts/1
8 501 user1 seat0

2 sessions listed.
```

Информацию о том, какие пользователи и когда входили в сеанс ранее,

можно получить с помощью команды `last`, которая извлекает данные из бинарного файла `/var/log/wtmp`.

Глава 3. Основы командной строки

3.1. Что такое оболочка?



Командная оболочка (shell) – это программа, взаимодействующая с пользователем с помощью интерфейса командной строки и позволяющая пользователю запускать прикладные программы и выполнять различные команды операционной системы. Оболочка интерпретирует введенные пользователем команды, и преобразует их в инструкции операционной системы.

Показывая пользователю, что оболочка готова интерпретировать команды, она выводит специальное приглашение командной строки, заканчивающееся обычно символом доллара `$` в сеансе обычного пользователя. В сеансе суперпользователя оболочка обычно в качестве приглашения использует символ решетки `#`, предупреждая о возможности нарушения работоспособности всей системы вследствие ошибочных действий.

В GNU/Linux может быть использовано множество различных оболочек, однако стандартом де-факто является оболочка Bourne Again Shell – bash.

Оболочка запускается при входе пользователя в сеанс. Какая конкретно оболочка будет запущена определяется учетной записью пользователя. Определить, какая оболочка установлена в учетной записи пользователя, можно

путем вывода переменной SHELL:

Пример:

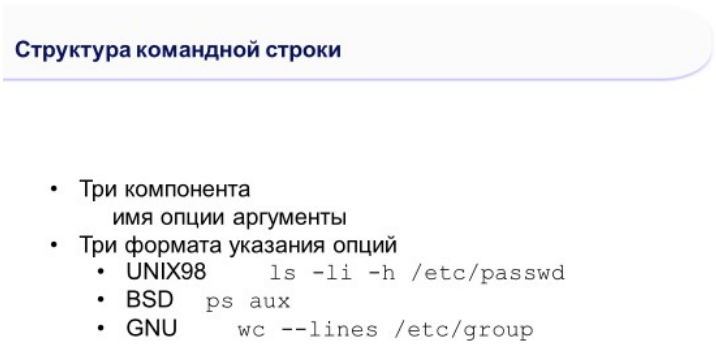
```
$ echo $SHELL
/bin/bash
```

Примечание: Переменная окружения SHELL содержит в себе полное имя исполняемого файла оболочки пользователя, используемой при входе в сеанс этого пользователя. В данном случае – это Bash. Символ доллара $ перед переменной окружения используется для извлечения ее значения.

Команды пользователя представляют собой строки, вводимые с клавиатуры.

После того, как команда введена и нажата клавиша Enter, команда интерпретируется оболочкой и, при удачной интерпретации, исполняется. Если команда введена синтаксически неверно, то выдается сообщение об ошибке. Помимо отдельных команд, вводимых последовательно с клавиатуры, в командной оболочке можно также использовать файлы сценариев. Сценарии позволяют выполнять достаточно сложные задачи с использованием условных переходов, циклов и подпрограмм.

3.2. Структура командной строки.



Пользователь вводит команды с клавиатуры, и они отображаются после приглашения командной строки. Обычно на экране оболочкой отображается курсор, показывающий позицию вывода следующего символа, вводимого с клавиатуры.

Для того, чтобы команда была правильно интерпретирована оболочкой и команда правильно обработала переданные ей аргументы, следует придерживаться соглашений о структуре командной строки. В общем виде командная строка состоит из следующих трех частей:

1. Имя команды – соответствует имени исполняемого файла системной команды или же встроенной команды оболочки.
2. Опции – дополнительные инструкции, сообщающие команде детали действий, которые она должна выполнить.
3. Аргументы – объекты, над которыми команда должна произвести заданные действия.

Примечание: То есть, образно говоря, команда сообщает операционной системе что надо сделать, опции уточняют как эти действия должны быть произведены, а аргументы – это то, над чем эти действия будут произведены.

Команды, вводимые в оболочке представляют собой строки, причем команда, опции и аргументы должны быть отделены друг от друга пробелами

или табуляцией.

Во многих командах в качестве аргументов используются имена файлов. Обобщающее название таких команд - “файловые команды”.

Существует три основных формата командной строки, поддерживаемых GNU/Linux. Их основное отличие – стиль указания опций.

В формате UNIX98 (иначе - POSIX формат) опции указывают в виде одиночных букв, перед которыми ставится символ – (тире):

```
команда -опции аргументы
```

Формат UNIX98 краток и удобен для команд с большим набором опций, так как опции чаще всего можно указывать друг за другом

Пример:

```
$ ls -dl /etc/default
```

Примечание: В этом примере команда ls, которая обычно выводит содержимое каталога, указанного в качестве аргумента, ведет себя иначе, так как используются опции -d и -l. Опция -d заставляет команду ls выводить информацию о самом каталоге, а не о файлах в нем. Опция -l сообщает команде, что вывод должен быть осуществлен в подробном формате.

В BSD формате тире перед опциями отсутствует

```
команда опции аргументы
```

В формате BSD также можно указывать несколько опций подряд

Пример:

```
$ ps aux
```

Примечание: Команда ps выводит список процессов в системе. Три используемые опции – a, u, x модифицируют поведение команды так, что она отображает список всех процессов в системе, указывая пользователей, от имени которых запущены эти процессы.

Третий используемый в Linux формат командной строки – длинная нотация GNU. В этом формате опция записывается целым словом, перед которым надо указать двойное тире -- :

```
команда --опция1 --опция2 аргументы
```

Удобство этого формата состоит в интуитивной ясности опций, поскольку они записываются целыми словами.

Команды GNU поддерживают специальную опцию --help, обеспечивающую возможность получения краткой справки по команде.

Пример:

```
$ gzip --help
```

Примечание: Команда gzip позволяет сжимать файлы. Однако, в данном случае она просто выводит информацию о себе, так как установлена опция --help.

Некоторые команды используют опции, не относящиеся ни к одному из перечисленных форматов.

Примечание: Так встречаются опции вида +5 и подобные. Опции, используемые в системе XFree86 (свободно распространяемая реализация графической системы X Window), указываются после знака тире - , но являются "длинными", хотя и не в стиле GNU.

Например:

```
 $ xterm –display :0.0
```

Эта команда запускает графический эмулятор командной строки – программу `xterm`. Легко заметить, что в качестве опции используется целое слово, однако опция отмечена лишь одним тире - .

Многие команды позволяют использовать различные форматы.

Пример: Команды, показанные ниже, делают одно и то же – выводят информацию о подкаталоге mydir текущего каталога:

```
$ls –d mydir
$ls --directory mydir
```

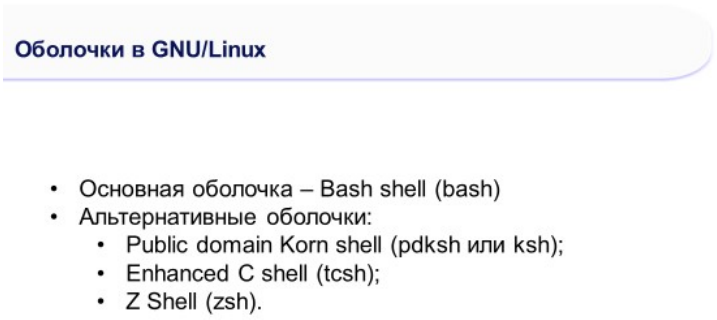
Примечание: Обе команды сделают одно и то же, поскольку у команды ls, опции -d и --directory эквивалентны и устанавливают вывод информации о самом каталоге, вместо вывода информации о его содержимом.

После ввода команды необходимо нажать клавишу Enter, после чего интерпретатор командной строки производит синтаксический анализ, разбирая командную строку на части: имя команды, список опций и список аргументов.

Если разборка строки закончилась удачей, производится попытка

исполнить команду.

3.3. Оболочки в GNU/Linux.



В GNU/Linux можно использовать множество различных оболочек, а также можно написать свою собственную оболочку, если существующие варианты не удовлетворяют имеющихся требований.

Наиболее распространены четыре вида оболочек:

1. Bash shell (bash) – используется по умолчанию;
2. Public domain Korn shell (pdksh или ksh);
3. Enhanced C shell (tcsh);
4. Z Shell (zsh).

Примечание: В скобках указаны команды для запуска этих оболочек (имена исполняемых файлов оболочек).

Различные оболочки обладают различным набором функций и даже различными встроенными командами.

Встроенные команды оболочки – это команды, которые реализованы внутри нее, а не в виде внешних программ.

При запуске скриптов необходимо убеждаться, что скрипт предназначен для выполнения в данной оболочке.

Bash shell является лидером по популярности и большинство пользователей GNU/Linux используют именно эту оболочку.

Bash устанавливается по умолчанию для пользователей GNU/Linux и

именно она обычно загружается после входа в сеанс.

Оболочка Bash позволяет настраивать вид приглашения, однако чаще всего, обычные пользователи видят приглашение, заканчивающееся символом `$`, а для суперпользователя (root) приглашение заканчивается символом `#`.

Примечание: Это сделано для лишнего напоминания суперпользователю о требуемой при его работе осторожности.

В системе может быть установлено множество оболочек. Для получения их списка можно воспользоваться командой `chsh -l` , если она установлена в системе и настройки безопасности позволяют ей воспользоваться.

Можно также просто просмотреть содержимое файла `/etc/shells`, содержащего в себе список установленных в системе оболочек.

Пример: Ниже приведено типичное содержимое этого файла:

```
# cat /etc/shells
/bin/sh
/bin/bash
/bin/csh
/bin/tcsh
/bin/ksh
/bin/zsh
```

Примечание: В этом примере с помощью команды cat получено содержимое файла /etc/shells, содержащего список оболочек в системе.

Примечание: Следует еще раз отметить, что в GNU/Linux, вовсе не обязательно, чтобы оболочка, указанная в файле /etc/shells была на самом деле установлена в системе. Этот файл более необходим для программ, обеспечивающих удаленный доступ к системе. Например, большинство программ – серверов FTP запрещают входить в сеанс пользователям, оболочка по умолчанию которых не указана в данном файле.

Для временной загрузки другой оболочки необходимо просто набрать имя соответствующего исполняемого файла.

Пример: для запуска оболочки Enhanced C shell выполняем команду:

```
$ tcsh
$ ps
PID TTY TIME CMD
2349 pts/0 00:00:00 bash
10295 pts/0 00:00:00 tcsh
10319 pts/0 00:00:00 ps
```

Примечание: Эта команда запустит оболочку Enhanced C shell. Далее с помощью команды ps демонстрируется, что оболочка tcsh была запущена из bash.

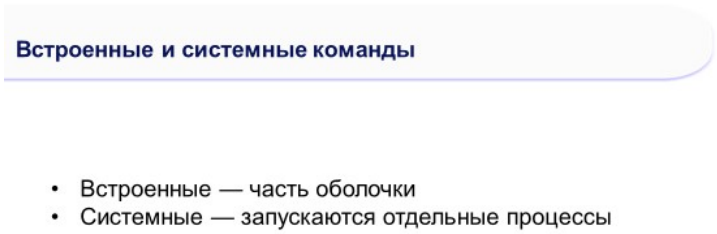
- Временно загружать оболочку иногда требуется для того, чтобы выполнить сценарий командной строки, предназначенный для этой оболочки.
- Для выхода из временно загруженной оболочки достаточно набрать команду `exit` .
- Если системная политика это позволяет, то пользователь может изменить для себя оболочку, которая будет запускаться при его входе в сеанс. Это можно сделать с помощью команды `chsh -s <shell>` , где `<shell>` - имя исполняемого файла оболочки

Пример:

```
$ chsh -s /bin/csh
```

Примечание: Здесь установлена оболочка, которая будет загружаться по умолчанию, C shell, исполняемый файл которой /bin/csh. При этом, всякий раз, когда пользователь будет входить в сеанс, будет загружена именно эта оболочка.

3.4. Встроенные и системные команды.



Все команды GNU/Linux делятся на два больших класса: встроенные и системные.

Встроенные команды интерпретируются и выполняются самой оболочкой.

Системные команды представляют собой исполняемые файлы, находящиеся в специальных каталогах.

Встроенные команды представляют собой процедуры оболочки и, следовательно, выполняются быстрее системных команд.

В силу того, что они привязаны к конкретной оболочке, необходимо понимать, что одинаковые команды в разных оболочках могут вести себя по-разному.

Примечание: например, команда set в Bash и C shell используется совершенно по-разному.

Основная масса встроенных команд, все – таки, одинакова в различных оболочках.

Командная оболочка `bash`, например, предоставляет пользователю такие встроенные команды, как `cd`, `alias`, `bg`, `kill`, `pwd` и `echo`.

Исполняемые файлы системных команд обычно находятся в одном из каталогов, указанных ниже:

```
/bin
/sbin
/usr/bin
/usr/sbin
/usr/local/bin
/usr/local/sbin
```

Пользователь может написать свои собственные системные команды и использовать их. Обычно для размещения таких команд используется домашний каталог пользователя или его подкаталог `bin`.

Если в системе имеется одновременно и встроенная и системная версия какой-либо команды, то если команда вызвана без указания пути к ней, то выполняется встроенная команда. Если при вызове команды указан путь, то выполняется системная команда.

3.5. Ввод, редактирование и исполнение команд.

При работе в командной строке можно использовать привычные клавиши управления курсором и клавиши редактирования. Однако, во-первых, не все виды клавиатур обеспечивают такие клавиши, как `Home`, а, во-вторых, во многих случаях привычные клавиши управления курсором не работают.

Ниже приведена таблица клавиатурных сочетаний, которые могут быть использованы при работе в командной строке.

Клавиши	Действие
Ctrl-B	Курсор влево
Ctrl-F	Курсор вправо
Alt-B	Курсор на слово влево
Alt-F	Курсор на слово вправо
Ctrl-A	Курсор в начало строки
Ctrl-E	Курсор в конец строки
Ctrl-H	Удаление символа перед курсором
Ctrl-D	Удаление символа в позиции курсора
Alt-D	Удаление слова
Ctrl-U	Удаление части строки слева от курсора
Ctrl-K	Удаление части строки справа от курсора
Ctrl-M или Ctrl-J	Ввод
Ctrl-V	Отмена специального значения символа

Ctrl-L	Очистка экрана
Alt-T	Перемена мест аргументов
Alt-L	Перевод слова в нижний регистр
Alt-U	Перевод слова в верхний регистр
Ctrl-C	Остановка выполнения задания
Ctrl-Z	Приостановка выполнения задания

Иногда бывает необходимо ввести в командную строку символический код клавиатурного сочетания вместо выполнения команды, связанной с этим сочетанием. В этом случае перед вводом клавиатурного сочетания, имеющего специальное значение, необходимо ввести `Ctrl-V`.

Пример:

```
$ echo "1 2 3"
1 2 3
```

Примечание: Обратите внимание на то, что в предыдущем примере в качестве аргумента команды echo использована строка, в которой имеются символы табуляции. Для ввода такой строки необходимо перед каждым символом табуляции нажать сочетание Ctrl-V, так как в противном случае Bash ошибочно воспринимает символ табуляции как специальный символ.

Нажатие `Ctrl-C` приводит к передаче процессу сигнала INT, что эквивалентно команде `kill -2` . Однако не все задания могут быть остановлены так.

Нажатие `Ctrl-Z` приводит к приостановке активного задания, которое затем может быть переведено в фоновый режим командой `bg`, либо это задание может быть уничтожено командой `kill`.

Оболочка Bash предоставляет специальную возможность исправить ошибку, допущенную при вводе команды. Исправление достигается изменением цепочки символов в выполненной ранее команде и выполнением новой измененной команды. Для этого надо набрать: `^заменяемое^замена` и нажать Enter при этом будет выполнена команда с замененной подстрокой

Пример:

```
$ ls /dmb
ls: /dmb: No such file or directory
$ ^dmb^tmp
```

```
ls /tmp
0103740143 0793455464 1645210352 AcromIO5pU
```

Примечание: Команде ls в этом примере был указан в качестве аргумента несуществующий каталог, поэтому было получено сообщение об ошибке. Далее, пользуясь механизмом замены символов, подстрока dmb была заменена на tmp, команда была исполнена автоматически и на экран было выведено содержимое каталога /tmp.

В случае если необходимо ввести длинную команду, которая не помещается в одну строку, необходимо воспользоваться символом обратной косой черты \ и продолжить ввод на следующей строке

Пример:

```
$ find . \
> -name "*prim*" \
> -user 501 \
> -type f \
> -ls
1352836   20   -rw-r--r--   1   user   user   16408   Окт   14   1999
./Documents/DocBook/html/primaryie.html
1353135   16   -rw-r--r--   1   user   user   16093   Окт   14   1999
./Documents/DocBook/html/primary.html
```

*Примечание: Команда find . -name "*prim*" -user 501 -type f -ls , выполненная здесь, довольно длинная. Она ищет все обычные файлы в текущем каталоге, в имени которых встречается строка prim, принадлежащие пользователю с UID=501, и печатает найденные имена файлов в формате, подобном ls -l. Так как команда длинная, то ее удобно ввести по строкам, разделяя каждую строку, входящую в команду, с помощью символа обратной косой черты. Знаки больше, показанные в листинге, являются вторичным приглашением командной строки. Они могут быть установлены с помощью переменной окружения PS2.*

Можно вводить несколько команд в одной строке, разделяя их символом точка с запятой ;

Пример:

```
$ cd /opt; ls -l; cd; pwd
итого 8
drwxr-xr-x 7 root root 4096 Июл 23 08:21 drweb
drwxr-xr-x 7 oracle oinstall 4096 Июл 23 07:20 oracle
/home/user
```

Примечание: В командной строке, приведенной выше, объединено сразу четыре команды, разделенные точкой с запятой.

Если команды отделены друг от друга с помощью двух амперсандов `&&`, то вторая команда будет выполнена только в случае успешного выполнения первой. То есть, в случае, когда первая команда вернула нулевой код возврата.

Пример:

```
$ ls nofile && cat nofile
ls: nofile: No such file or directory
$ ls /etc/motd && cat /etc/motd
/etc/motd
It's time to host!
```

Примечание: Первая команда ls nofile возвратила ненулевой код возврата, так как такого файла не оказалось, поэтому вторая команда не была выполнена. Во втором случае первая команда закончилась удачно, поэтому была выполнена вторая команда, отобразившая на экране содержимое файла /etc/motd.

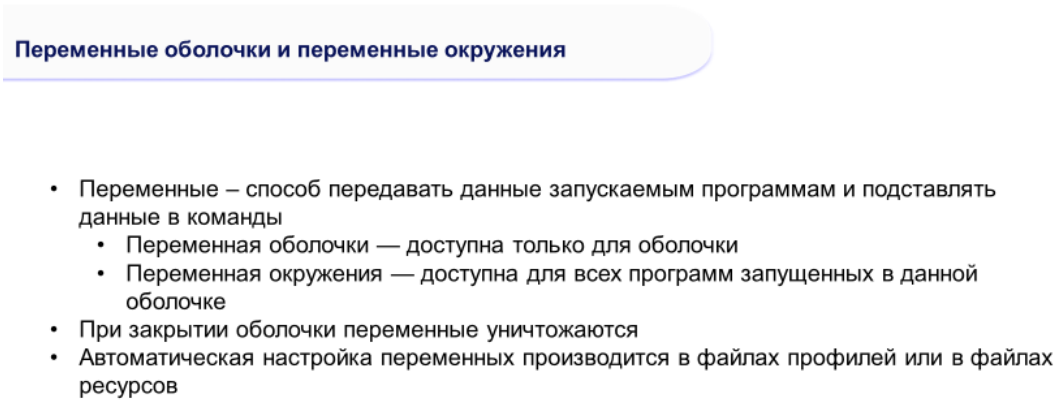
При необходимости выполнять вторую команду только в случае неудачи первой следует использовать две вертикальные черты ||

Пример:

```
$ ls nofile || echo "Net EGO"
ls: nofile: No such file or directory
Net EGO
```

Примечание: Данный пример демонстрирует, что вторая команда в цепочке echo "Net EGO" была выполнена, так как первая команда ls nofile такой файл не обнаружила и закончилась с ошибкой.

3.6. Переменные оболочки и переменные окружения.



Bash предоставляет возможность временного сохранения данных в переменных, называемых переменными оболочки. Переменные оболочки размещаются в памяти автоматически при присвоении им значения. Для создания переменной необходимо указать имя переменной, знак равно и строку – значение переменной (без пробелов). Для извлечения значения переменной в командную строку необходимо перед именем переменной поставить знак $.

Пример: Переменной VAR1 присваивается значение Privet!.

```
$ VAR1=Privet!
$ echo $VAR1
Privet!
```

Примечание: Команда echo здесь получает в качестве аргумента значение, сохраняемое в переменной VAR1, и выводит его на экран.

В оболочках переменные могут содержать лишь строковые данные.

Примечание: Даже если переменная содержит в себе цифру, то все равно переменные Bash хранятся в виде строк.

Имя переменной должно состоять только из букв и цифр или подчеркивания. В качестве первого символа в имени переменной должна

использоваться либо буква, либо символ подчеркивания. Желательно (но не обязательно) использовать в именах переменных только большие буквы для того, чтобы не путать имена команд и имена переменных.

Пример:

```
$ hostname
host.example.ru
$ echo $HOSTNAME
host.example.ru
```

Примечание: Обратите внимание на то, что в этом примере команда hostname выводит ту же информацию, которая содержится в переменной HOSTNAME. Тем не менее, это совершенно разные объекты.

Если переменная должна содержать, в качестве значения, строку с пробелами, то строку следует экранировать с помощью одиночных или двойных кавычек.

Пример:

```
$ VAR1='Bolshoy Privet!'
$ echo $VAR1
Bolshoy Privet!
```

Примечание: В этом примере переменная VAR1 содержит строку с пробелом.

Переменным можно присваивать значения других переменных или же их собственные значения, возможно, несколько изменяя их.

Пример:

```
$ VAR1='Vam vsem '$VAR1
$ echo $VAR1
Vam vsem Bolshoy Privet!
```

Примечание: Здесь переменной VAR1 было присвоено значение, составленное из строки Vam vsem и ее собственного старого значения. Следует отметить, что в данном случае пробела между строкой и символом доллара нет!

При необходимости добавления строки к значению переменной имя переменной следует взять в фигурные скобки для отделения имени переменной от последующей строки.

Пример:

```
$ VAR1=${VAR1}'?'
$ echo $VAR1
Vam vsem Bolshoy Privet!?
```

Примечание: Пример, приведенный выше демонстрирует, как к значению переменной можно добавить строку. Если бы имя переменной не было бы экранировано с помощью фигурных скобок, то конструкция $VAR1? Была бы интерпретирована Bash неверно.

Для получения списка всех переменных, определенных в текущей оболочке, следует использовать команду set.

Пример:

```
$ set | less
```

Примечание: Команда выведет список всех переменных, определенных в данной оболочке, и их значения, отобразив его с помощью программы постраничного просмотра less.

Если вы хотите уничтожить переменную, то для этого достаточно выполнить команду unset, указав в качестве ее аргумента имя переменной.

Пример:

```
$ VAR1="It's a variable"
$ echo $VAR1
It's a variable
$ unset VAR1
$ echo $VAR1
```

Примечание: После того, как переменная VAR1 была удалена командой unset , команда echo $VAR1 ничего не выводит на экран.

Переменные оболочки доступны только в том экземпляре запущенной оболочки, в которой они были описаны.

Примечание: Нельзя извлечь значение переменной из порожденной оболочки или из программы, запущенной из этой оболочки.

Однако существует способ превратить переменную оболочки в так называемую переменную окружения. Для этого необходимо произвести операцию экспортирования переменной, после чего она станет доступной для

любых программ, запущенных из оболочки. Экспортирование переменной осуществляется с помощью команды `export`.

Пример:

```
$ VAR1=01/01/1970
$ echo $VAR1
01/01/1970
$ bash
$ echo $VAR1

$ exit
exit
$ export VAR1
$ bash
$ echo $VAR1
01/01/1970
$ exit
exit
```

Примечание: Этот пример демонстрирует, что переменная VAR1, описанная в текущей оболочке, при запуске новой оболочки командой bash, не известна в порожденной оболочке. Однако, после экспортирования ее она превращается в переменную окружения и становится доступной в порожденной оболочке.

Весь набор переменных окружения содержится в массиве, называемом environ. Все переменные окружения и их значения могут быть получены с помощью команды `env`.

Окружение – это один из способов передачи информации процессам. Очень часто изменение значения какой-либо переменной окружения приводит к изменению поведения запускаемой программы. Ниже приведена таблица часто используемых переменных окружения.

Переменная	Сущность
HOME	Путь к домашнему каталогу пользователя.
LOGNAME	Имя пользователя.
MAIL	Путь к почтовому ящику пользователя.
PATH	Путь поиска исполняемых файлов.
PS1	Вид приглашения оболочки.
PWD	Имя текущего каталога.
SHELL	Текущая оболочка.
USER	Имя пользователя.

Переменная окружения `PS1`, определяющая вид приглашения оболочки, кодируется с помощью специальных символов.

Пример:

```
[user1@host tmp]$ echo $PS1
[\u@\h \W]\$
```

Примечание: Символ \u устанавливает вывод имени пользователя, затем идет двоеточие, затем символ \W устанавливает вывод имени текущего каталога, а символ \h заставляет отображать в строке приглашения Bash имя хоста.

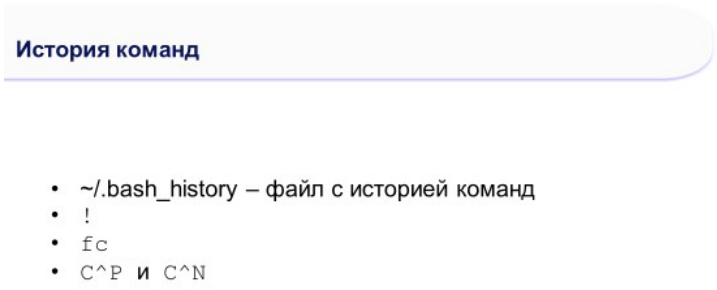
Выходя из сеанса, пользователь завершает работу оболочки, поэтому установленные им переменные уничтожаются. Для того чтобы необходимые значения переменных окружения автоматически устанавливались при входе в сеанс, их необходимо инициализировать в файлах профиля.

Переменные окружения, общие для всех пользователей, хранятся в файле `/etc/profile`.

Настройки специфичные для конкретных пользователей хранятся в одном из файлов домашнего каталога пользователя: либо в `~/.bash_profile`, либо `~/.bash_login`, либо `~/.profile`.

Эти файлы выполняются при каждом входе в сеанс. При каждом запуске оболочки bash выполняется файл профиля `~/.bashrc`.

3.7. История команд.



Оболочка Bash предоставляет пользователю возможность выполнять ранее введенные команды. Строки введенных пользователями команд сохраняются в файлах `~/.bash_history` (в переменной окружения `HISTFILE` можно указать другой файл).

Пример:

```
$ echo $HISTFILE
/home/user1/.bash_history
```

Количество команд, запоминаемых в файле истории, устанавливается с помощью переменной `HISTFILESIZE` . По умолчанию переменной `HISTFILESIZE`, присваивается значение 1000 .

Пример:

```
$ echo $HISTFILESIZE
9999
```

Содержимое файла истории можно вывести с помощью команды `history`.

Пример:

```
$ history

685 echo $HISTFILE
686 echo $HISTFILESIZE
687 history
```

Примечание: В этом примере показаны лишь последние команды из файла истории. Перед каждой командой, запомненной в этом файле, выводится ее номер, с помощью которого эту команду можно вызвать заново.

Наиболее простой способ для повтора команды по ее номеру ввести знак восклицания и номер команды.

Пример:

```
$ !685
echo $HISTFILE
/home/user1/.bash_history
```

Примечание: Пример, приведенный выше, демонстрирует, как из файла истории была вызвана и выполнена команда с номером 685.

Последнюю выполненную команду можно выполнить снова, если ввести в командной строке два знака восклицания !!. Исполнить заново, недавно исполненную команду, можно введя в командной строке после знака восклицания первые символы ее командной строки.

Пример:

Если необходимо вновь выполнить команду `ls /tmp`, выполненную недавно, достаточно ввести в командной строке `!l`

При этом история команд будет просмотрена с конца до тех пор, пока не будет найдена команда с подходящими первыми символами в ее строке. Как только такая команда будет найдена, она будет исполнена. В противном случае будет выдано сообщение об ошибке.

```
$ ls /opt
drweb oracle
$ pwd
```

```
/home/user1
$ cd /tmp
$ !l
ls /opt
drweb oracle
```

Примечание: Как видно из этого примера, пользователь ввел команду ls /opt, а затем еще несколько команд. После исполнения этих команд пользователю вновь понадобилось исполнить команду ls /opt, выполненную недавно. Пользователь добился этого, введя в командной строке !l.

Можно вызвать команду из истории, указав строку символов, содержащуюся в команде. Для этого следует ввести эту строку после знака восклицания и знака вопроса `!?`.

Пример:

```
$ !?cho
echo $HISTFILE
/home/user1/.bash_history
```

Примечание: Здесь из файла истории была вызвана последняя выполненная команда, содержащая подстроку cho.

Недостатком описанных выше способов вызова команд из истории является отсутствие возможности их интерактивно отредактировать. Команды, подходящие для заданных шаблонов, извлекаются из файла истории и выполняются немедленно.

Имеется способ вызова недавно выполненной команды из истории в текстовый редактор, последующего ее редактирования в этом редакторе и исполнения отредактированной команды с помощью команды `fc`.

Аргумент команды `fc` – строка, с которой начинается искомая в файле истории команда. Найденная команда отображается в редакторе, используемом в системе по умолчанию. После редактирования команда исполняется. Для того, чтобы отказаться от выполнения найденной команды следует просто удалить в редакторе всю ее командную строку.

Команда `fc -l` выводит список из последних выполненных команд, подобный списку, выводимому командой `history`. Отличие в том, что `fc -l` выводит не всю историю, а только последние команды.

С опцией `-s` команда fc работает в не интерактивном режиме, подобно тому, как работает команда знак восклицания.

Вызов команды `fc -s <строка>` приводит к поиску последней команды в истории, начинающейся с заданной строки, и исполнению ее.

Опция `s` обозначает режим замены.

Пример:

```
$ ls -ld /tmp
drwxrwxrwt 30 root root 4096 Окт 2 22:01 /tmp
$ fc -s tmp=opt ls
ls -ld /opt
drwxr-xr-x 4 root root 4096 Июл 23 08:21 /opt
```

Примечание: Обратите внимание на конструкцию tmp=opt, она позволяет команде fc до исполнения найденной по заданной строке ls команде сначала выполнить подстановку в командной строке opt вместо tmp.

Наиболее часто для работы с историей команд используют обычные клавиши управления курсором – вверх (или Ctrl-P) для получения из истории предыдущей команды и вниз – (или Ctrl-N) для следующей команды в истории.

Найденные команды не исполняются сразу, а позволяют отредактировать командную строку и затем выполнить команду.

3.8. Автоматическое дополнение в командной строке.

Bash предоставляет удобный механизм дополнения имен файлов и команд по первым символам их имен. Bash пытается продолжить введенные символы как имя команды или имя файла после нажатия на клавишу табуляции. Если оболочка не может продолжить имя файла, то выводится звуковой сигнал. Это может происходить по двум причинам:

1. Файла или команды с таким именем не существует.
2. Имеется несколько вариантов продолжения строки.

Во втором случае при повторном нажатии на клавишу табуляции Bash выводит список возможных подстановок, ориентируясь на который, пользователь может ввести еще несколько символов командной строки и снова нажать на клавишу табуляции. Механизм продолжения в Bash действует не только для имен файлов и команд. Если строка начинается с одного из символов $, или ~, или @, то Bash попытается дополнить строку как:

1. имя переменной оболочки ($);
2. имя пользователя (~);
3. имя хоста (@).

Имеется несколько удобных сочетаний клавиш, предназначенных для различных способов продолжения строки команды:

1. Alt-? - выводит список возможных продолжений строки.
2. Ctrl-x / - выводит список возможных имен файлов.

3. Ctrl-x ~ - выводит список возможных имен пользователей.
4. Ctrl-x @ - выводит список возможных имен хостов.
5. Alt-* - вставляет в командную строку сразу все возможные варианты продолжения.
6. Alt-/ - осуществляет продолжение строки, как имени файла.
7. Alt-~ - осуществляет продолжение строки, как имени пользователя.
8. Alt-$ - осуществляет продолжение строки, как имени переменной.
9. Alt-@ - осуществляет продолжение строки, как имени хоста.

При вводе имени команды механизм дополнения позволяет быстро находить программу, которая находится в каталогах, перечисленных в переменной `PATH`.

3.9. Псевдонимы команд (aliases).

- `alias` – задать псевдоним
- `unalias` – удалить псевдоним

Для ускорения набора часто повторяющихся сложных команд можно создать псевдоним командой `alias`.

Для удаления псевдонима достаточно указать его в качестве аргумента команде `unalias`.

Команда `unalias -a` удаляет все описанные в оболочке псевдонимы.

Примечание: Например, весьма распространенной практикой является назначение псевдонима r команде fc –s, позволяющей искать в файле истории и запускать команду, начинающуюся с заданных символов:

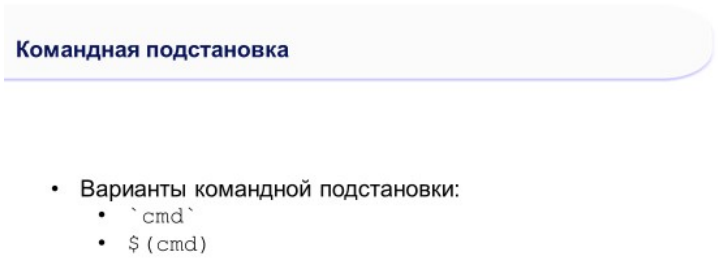
Пример:

```
$ alias r='fc -s'
$ alias
alias ls='ls --color=auto'
alias mv='mv -i'
alias r='fc -s'
alias rm='rm -i'
$ r ls
ls -d /bin
/bin
```

Примечание: В этом примере создан псевдоним r для команды fc -s. Для проверки

была вызвана команда alias без аргументов, которая вывела список всех псевдонимов, описанных в оболочке. Среди них заметно только что созданный псевдоним r. Далее этот псевдоним был использован с аргументом ls. В результате чего была найдена соответствующая команда в файле истории.

3.10. Командная подстановка.



Командная подстановка (command substitution) позволяет результат выполнения одной команды поместить в указанном месте командной строки другой команды.

Механизм подстановки команд позволяет передавать в командную строку результаты выполнения заключенной в скобки команды, перед которыми должен стоять символ `$`.

Вместо доллара и скобок можно заключить выражение в обратные кавычки ``.

Пример: Предположим, что необходимо вывести на экран такую строку с информацией о первичной группе текущего пользователя: “My UID is ...”, где вместо многоточия должен быть выведен UID пользователя. Одним из способов решения этой задачи является получение UID пользователя с помощью команды `id -u`, и подстановки этого значения вручную в качестве аргумента команды echo. Однако, ясно, что это весьма несовершенный способ. Можно объединить эти две команды в одну с помощью командной подстановки.

```
$ echo "My UID is `id -u`"
My UID is 501
```

Примечание: Взгляните, команда echo имеет единственный аргумент – строку,

заданную в двойных кавычках. Внутри кавычек находится команда id -u, заключенная в обратные кавычки, что интерпретируется Bash как командная подстановка.

Пример: Ниже приведен другой пример использования командной подстановки с использованием доллара и круглых скобок.

```
$ ls $(cat /etc/shells)
ls: /bin/ksh: No such file or directory
ls: /bin/zsh: No such file or directory
/bin/bash /bin/csh /bin/sh /bin/tcsh
```

Примечание: Приведенная выше команда позволила определить, имеются ли в системе все оболочки, зарегистрированные в файле /etc/shells. Для этого был использован механизм командной подстановки, где строки, выведенные командой cat из файла /etc/shells, содержащие пути к зарегистрированным оболочкам, были подставлены как аргументы команды ls. По результату работы команды видно, что оболочек ksh и zsh в системе нет, хотя в файле /etc/shells эти оболочки указаны.

Очень удобно использовать результат командной подстановки для назначения его в качестве значения переменной.

Пример:

```
$ SYSTEM_START=`who -b`
$ echo $SYSTEM_START
system boot Oct 3 20:06
```

Примечание: В этом случае была определена переменная SYSTEM_START, которая в результате подстановки результата работы команды who -b, получила в качестве значения строку с датой загрузки системы.

3.11. Вычисление арифметических выражений в командной строке.

В командной строке можно вычислять выражения, помещая выражения либо в квадратные скобки, либо в двойные круглые, перед которыми должен стоять символ $. Результаты выражений можно передавать как аргумент какой-либо команде или назначать переменной. Синтаксис с квадратными скобками считается устаревшим.

Пример:

```
$ echo $((192*512))
98304
```

*Примечание: В этом примере вычислено значение выражения 192*512. Команда echo выводит в стандартный поток вывода строку, указанную в качестве аргумента.*

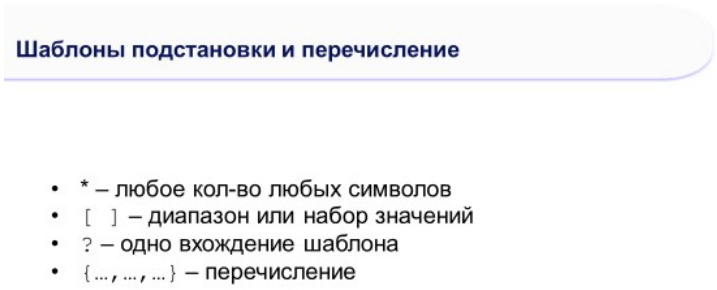
Пример: Ниже приведен пример назначения результата арифметического выражения переменной.

```
$ V1=5
$ V2=$(($V1/2))
$ echo $V2
2
```

Примечание: Переменной V1 присвоено значение 5. Далее производится присвоение

переменной V2 результата деления значения V1 на два. В силу того, что в Bash возможно вычисление только целочисленных выражений, то в результате значением стало V2 число 2.

3.12. Шаблоны подстановки и перечисление.



Для обработки одновременно нескольких файлов в качестве аргумента команды можно указать требуемую группу файлов, используя символы подстановки (шаблона). Символы подстановки позволяют оболочке произвести поиск файлов, имена которых удовлетворяют шаблону, и подставить их в качестве аргументов файловой команде. В качестве команды для испытания шаблонов подстановки можно использовать команду `echo`, которая просто выводит на экран то, что ей задано в качестве аргумента.

Символ звездочка `*` является шаблоном для любого количества любых символов в именах файлов и даже для их отсутствия. Единственный символ, который не удовлетворяет этому шаблону – лидирующая точка в именах скрытых файлов.

Примечание: Таким образом, подставив звездочку в качестве аргумента команде echo, мы увидим в результате либо саму звездочку, если в каталоге не скрытых файлов нет, либо оболочка подставит команде echo имена всех файлов в каталоге в командную строку в качестве аргументов и можно будет увидеть список этих файлов.

Пример:

```
$ echo *
f1 f2 f3 f4
```

```
$ rm *
$ echo *
*
```

*Примечание: В данном примере приводится некоторый каталог, в котором есть файлы, имена которых подставляются Bash в качестве аргументов команде echo, так как они удовлетворяют шаблону *. Затем, с помощью того же шаблона все эти файлы из каталога удаляются. После чего эксперимент с командой echo повторяется. На этот раз Bash не может подставить вместо звездочки имена файлов, поэтому команда echo выводит звездочку.*

Примечание: В пятой главе данного пособия будет рассказано о скрытых файлах, которые не отображаются командой ls (получение списка файлов в каталоге) без специальных опций этой команды. Имена этих файлов начинаются с точки. Шаблоном для имен таких файлов будет являться конструкция . .*

Звездочку можно использовать в любом месте строки – части имен файлов, для подстановки которых и строится шаблон.

Пример:

```
$ echo *
f1 f2 s s1 s2
$ echo s*
s s1 s2
```

Примечание: Обратите внимание, что в каталоге находится пять файлов с именами f1, f2, s, s1 и s2. Команда echo s выводит на экран имена только тех файлов, которые начинаются с буквы s и файл s.*

Символ ? заменяет один символ в имени файла, который должен находиться в той позиции, где находится знак вопроса.

Пример:

```
$ echo s?
s1 s2
```

Примечание: Легко заметить, что файл s не удовлетворяет заданному шаблону, поэтому Bash не подставляет его как аргумент команде echo.

Можно еще более сузить диапазон поиска имен файлов, используя набор символов, заключенных в квадратные скобки. Применение такого шаблона обозначает, что в указанном месте должен находиться один любой символ из заданного множества. Диапазон символов в наборе указывается через тире.

Примечание: [0-9] – шаблон подходит для любых цифр, а [a-zA-Z] – шаблон для букв английского алфавита в верхнем и нижнем регистрах.

Пример: Предположим, например, что требуется подставить в качестве аргументов имена всех файлов, которые начинаются либо с буквы f, либо с s, и имеют длину два символа. Для этого подойдет шаблон [fs]? .

```
$ echo [fs]?
f1 f2 s1 s2
```

Пример: Шаблону ?a[a-h]* удовлетворяют имена файлов, начинающихся с любого символа, второй символ которых должен быть a, а третий символ должен находиться в диапазоне от a до h. Далее могут следовать любые символы.

С помощью шаблона [!] можно указать множество символов (внутри квадратных скобок), которые не должны встречаться в именах файлов.

Пример:

```
$ echo *[!0-9]
s
```

Примечание: Такому шаблону здесь удовлетворяет единственный файл с именем s, так как шаблон требует, чтобы в конце имени файла не находилась цифра.

Очень удобен, хотя и не относится к шаблонам, механизм перечислений Bash. Он позволяет задать с помощью фигурных скобок множество вариантов, которое должна перебрать оболочка, составляя последовательно все возможные варианты строк с использованием подстрок, находящихся в фигурных скобках.

Перечисление может быть с помощью запятых, например {a,b,c,d,e} или диапазона значений, например: {a..e}. При использовании перечислений в командную строку вставляются все варианты перечисления, в то время как при использовании шаблонов вставляются все соответствующие шаблону имена файлов.

Пример:

```
$ echo s{1,2}
s1 s2
$ echo s1 s2
s1 s2
```

```
$ echo s{1..9}
s1 s2 s3 s4 s5 s6 s7 s8 s9
$ echo s{1..9..2}
s1 s3 s5 s7 s9
```

Примечание: Первые две команды совершенно эквивалентны. Bash подставляет все варианты, перечисленные в фигурных скобках, к строке, находящейся вне скобок и подставляет получившиеся строки, как аргументы командной строки. Следующие две команды показывают возможные переборы последовательных значений. В последней команде задан шаг перебора равный 2.

Можно использовать несколько перечислений одновременно.

Пример:

```
$ echo {s,f}{1,2}
s1 s2 f1 f2
```

Можно использовать перечисления в виде вложенных конструкций.

Пример:

```
$ echo {s,{f1,f2}}
s f1 f2
```

В фигурных скобках допускается использование шаблонов.

Пример:

```
$ echo {s,f?}
s f1 f2
```

3.13. Экранирование (quotation).

- \ – Экранирует следующий за ним единственный символ
- "" – Экранируют большинство спецсимволов, но позволяют интерпретировать подстановку переменных ($), команду ($()) и экранирование обратным слэшем (\).
- `` – Самый строгий метод. Всё, что находится внутри, воспринимается буквально, включая спецсимволы и переменные

Если метасимвол оболочки используется, “как он есть” (as is), то такой символ должен быть защищен от интерпретации оболочкой. Механизм защиты специальных символов оболочки от интерпретации называется экранированием (цитированием, quotation).

Пример: требуется назначить переменной STR1 значение Bolshoy Privet. Эта строка содержит пробел, который должен быть предотвращен от интерпретации оболочкой. Экранировать его можно, например, установив перед ним символ обратной косой черты:

```
$ STR1=Bolshoy\ Privet
$ echo $STR1
Bolshoy Privet
```

Для экранирования метасимволов оболочки используются:

1. одиночные кавычки '
2. двойные кавычки ""
3. символ обратной косой черты \

Одиночные кавычки устраняют интерпретацию специального значения всех метасимволов, заключенных в них, за исключением других одиночных

кавычек.

Пример:

```
$ STR1='Stroka soderzhit $TERM'
$ echo $STR1
Stroka soderzhit $TERM
```

Примечание: Заметно, что символ доллара, имеющий особое значение в оболочке, защищен от интерпретации одиночными кавычками.

Двойные кавычки устраняют интерпретацию метасимволов кроме:

1. других двойных кавычек " ;
2. символа доллара $;
3. обратной косой черты \ ;
4. обратной кавычки ` .

Пример:

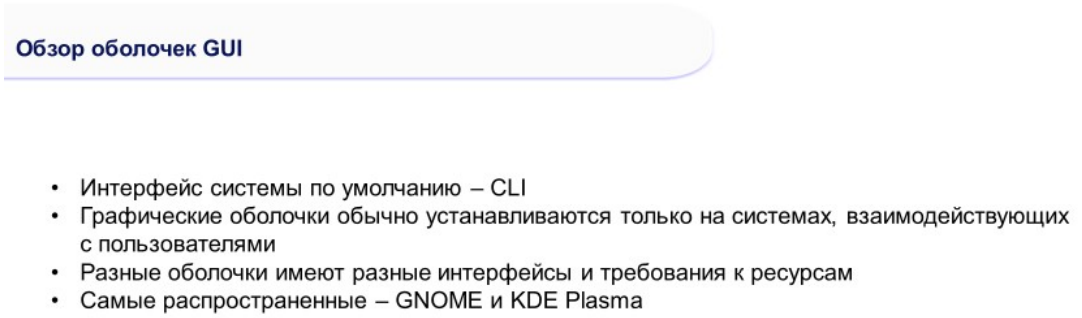
```
$ STR1="'Stroka soderzhit $TERM'"
$ echo $STR1
'Stroka soderzhit xterm'
```

Примечание: В этом примере показано, что использование двойных кавычек не предохраняет от интерпретации оболочкой символов доллара. В то же время, одиночные кавычки, заключенные в двойные, не оказали воздействия на интерпретацию символов доллара.

Символ обратной косой черты \ отменяет интерпретацию следующего за ним символа.

Глава 4. Использование графического интерфейса.

4.1. Обзор оболочек GUI



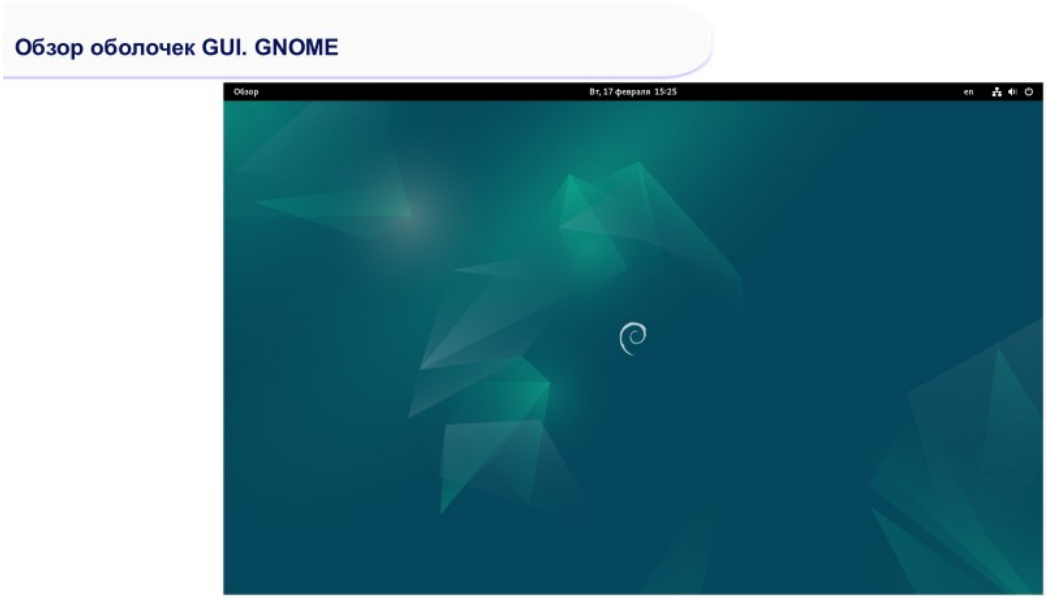
В ОС Linux пользователи применяют интерфейсы командной строки (CLI) и графический пользовательский (GUI). Если же речь идёт о встраиваемых системах, то работа ведётся посредством элементов управления самих аппаратных средств. В настольных системах чаще всего устанавливается графический пользовательский интерфейс. В этом случае командную строку можно открыть с помощью окна эмулятора терминала или отдельной виртуальной консоли.

Практически все низкоуровневые компоненты интерфейса Linux (в том числе и пользовательские компоненты GNU) применяют только командную строку. Она является оптимальным вариантом для автоматизации повторяющихся или отложенных задач. Кроме того, командная строка позволяет использовать крайне простой механизм межпроцессного взаимодействия.

Чтобы получить доступ к командной строке с рабочего стола Linux, нередко применяется программа графического эмулятора терминала. При этом те дистрибутивы, которые созданы специально для серверов, могут применять командную строку в качестве единственного интерфейса.

В случае с настольными системами самыми распространёнными

пользовательскими интерфейсами являются варианты, которые базируются на средах рабочего стола по типу GNOME и KDE Plasma Desktop.



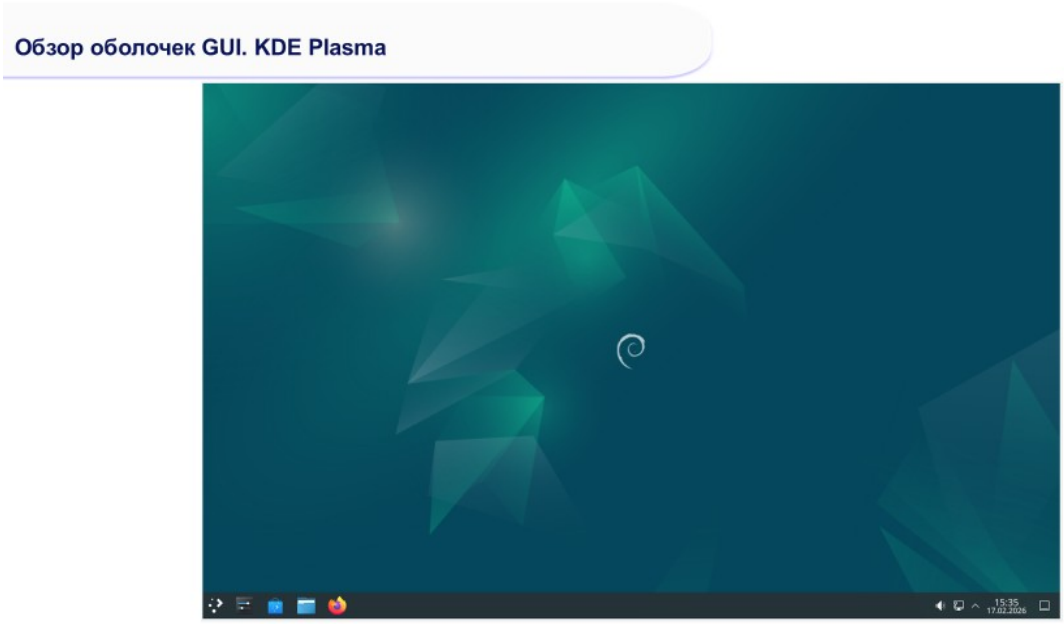
GNOME — свободная среда рабочего стола для UNIX-подобных операционных систем.

Разработчики GNOME ориентируются на создание полностью свободной среды, доступной всем пользователям вне зависимости от их уровня технических навыков, физических ограничений и языка, на котором они говорят. В рамках проекта GNOME разрабатываются как приложения для конечных пользователей, так и набор инструментов для создания новых приложений, тесно интегрируемых в рабочую среду.

GNOME — акроним от англ. GNU Network Object Model Environment («среда сетевой объектной модели GNU»). Под GNU в данном случае подразумевается не проект, а операционная система, официальной средой рабочего стола в которой и является GNOME.

Проект GNOME был основан в августе 1997 года Мигелем де Икасой и Федерико Меной Кинтеро как попытка создать полностью свободную рабочую среду для операционной системы GNU/Linux.

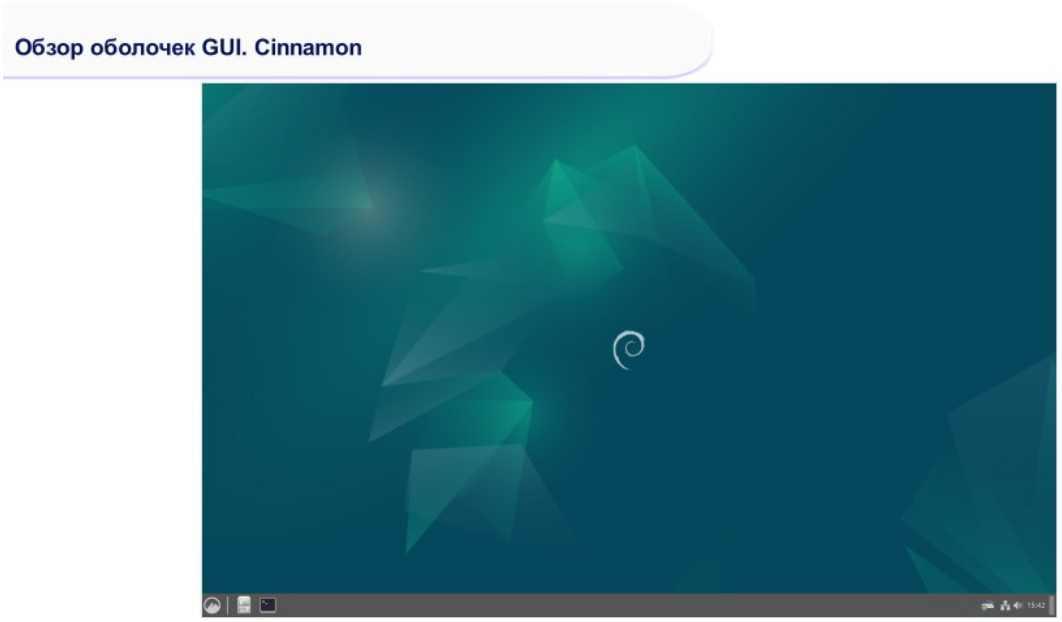
В то время популярность в среде Linux набирала KDE. Но KDE основана на инструментарии Qt фирмы Trolltech, который тогда был проприетарным продуктом. Чтобы не допустить ухудшения ситуации, была инициирована разработка GNOME — новой свободной рабочей среды на основе инструментария GTK+, созданного ранее для графического редактора GIMP и распространяемого на условиях GNU LGPL.



KDE Plasma — это технология пользовательского интерфейса, которую можно легко настроить для работы с различными форм-факторами, такими как настольные компьютеры, ноутбуки, планшеты, смартфоны или даже встроенные устройства.

Бренд Plasma для графических рабочих окружений был введен с KDE SC 4.4.

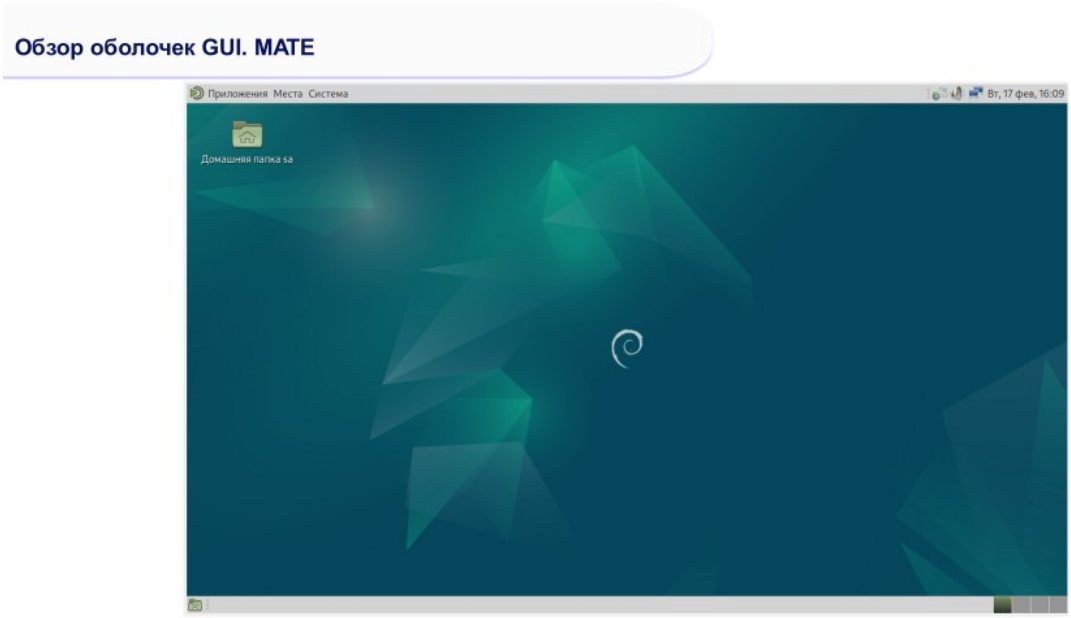
KDE Plasma 6 — шестая и текущая версия среды рабочего стола, созданная сообществом KDE для систем Linux. Ключевым изменением в KDE Plasma 6 является переход на Qt 6, изменение некоторых базовых настроек, проведение чистки устаревших возможностей и поставка обновлённого базового набора библиотек и компонентов KDE Frameworks 6. По умолчанию в KDE Plasma 6 предложен сеанс, использующий протокол Wayland, новый интерфейс переключения между задачами и плавающий режим показа панели.



Cinnamon— свободная оболочка для среды рабочего стола GNOME, являющаяся ответвлением от кодовой базы GNOME Shell. Основное направление разработки — предоставление пользователю более привычной, традиционной среды в стиле GNOME 2, удобной пользователям настольных ПК и ноутбуков, без недостатков GNOME Shell и Unity. Изначально разрабатывалась командой программистов Linux Mint.

После выхода GNOME 3 команда Linux Mint потеряла уверенность в будущем своего дистрибутива. Новая оболочка GNOME Shell совершенно не соответствовала задачам разработчиков и не вписывалась в идеи Linux Mint, при этом другие достойные альтернативы отсутствовали. Linux Mint 11 «Katya» был выпущен с окончательным выпуском GNOME 2, но для дальнейшего развития требовалось новое решение, так как ни составляющие GNOME 2, ни библиотеки GTK 2 более не имели поддержки со стороны разработчиков проекта GNOME. Тогда было решено доработать GNOME Shell до состояния, пригодного к использованию в дистрибутиве. Результатом этого стали «Mint GNOME Shell Extensions» (MGSE).

Поскольку GNOME Shell развивался в совершенно ином направлении, нежели ожидали разработчики Linux Mint, жизнеспособность MGSE была под сомнением. Ответом на эту проблему стало ответвление проекта GNOME Shell в Cinnamon, который стал подконтролен программистам Linux Mint с чётким предназначением для данного дистрибутива. Проект был представлен общественности 2 января 2012 года в блоге Linux Mint.

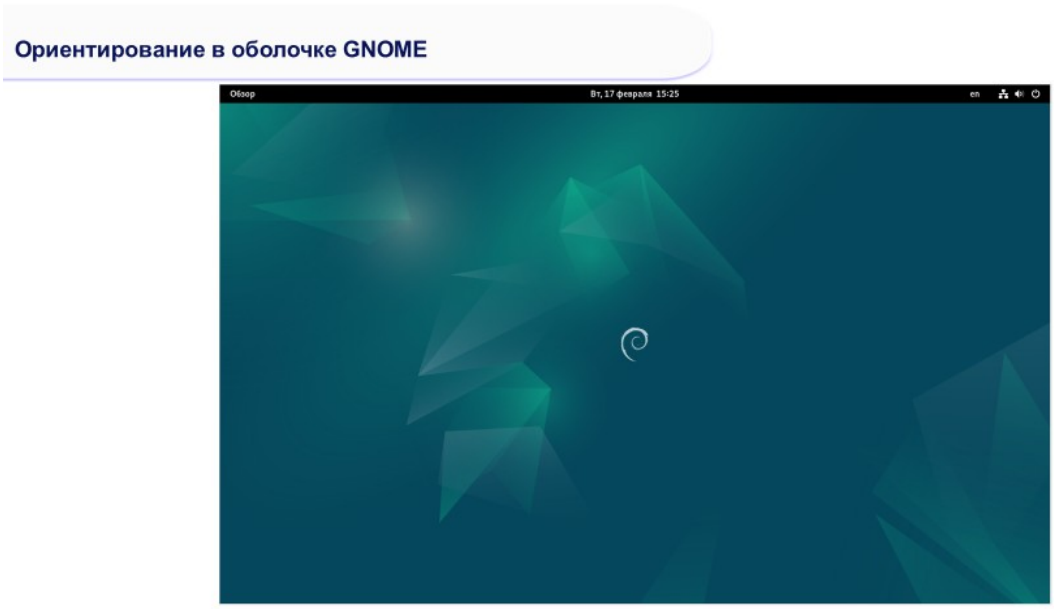


MATE — среда рабочего стола, являющаяся ответвлением от кодовой базы не поддерживаемой в настоящее время среды GNOME 2.

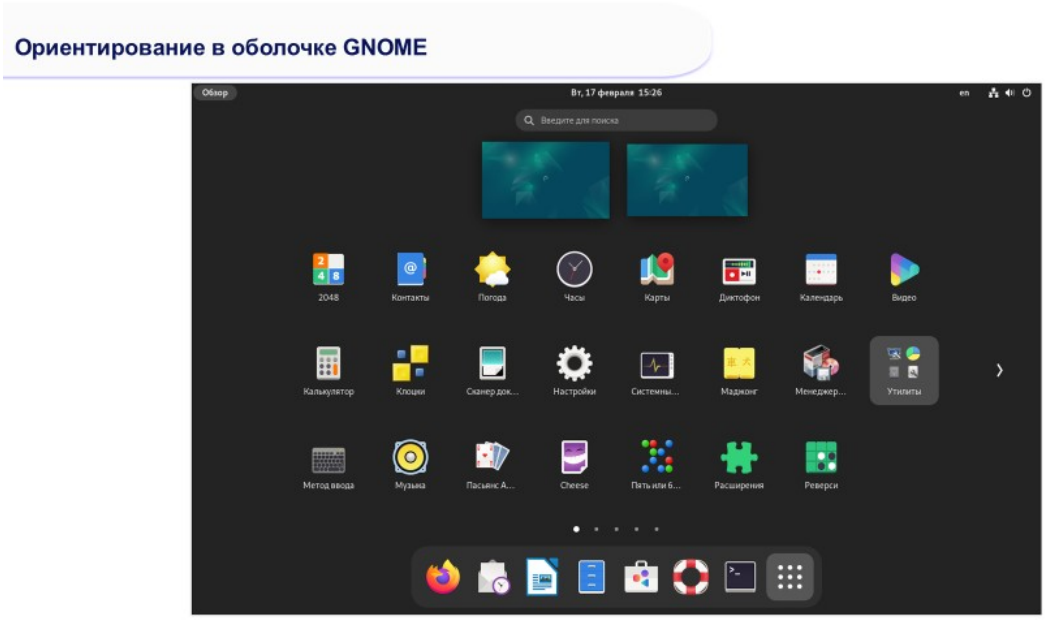
Выход GNOME 3, заменившего классический рабочий стол новым интерфейсом, построенным на основе GNOME Shell, привёл к появлению значительного количества критических замечаний. Многие пользователи отказались от использования нового GNOME, призывая кого-нибудь продолжить разработку GNOME 2. Проект MATE был запущен пользователями Arch Linux для того, чтобы выполнить эту миссию.

MATE находится в состоянии активной разработки. Разработчики внедряют новые технологии, сохраняя традиционный вид рабочего стола. Кроме того, одна из целей разработки — сохранить работоспособность MATE на старом оборудовании, не ориентируясь исключительно на современные конфигурации оборудования.

4.2. Ориентирование в оболочке GNOME

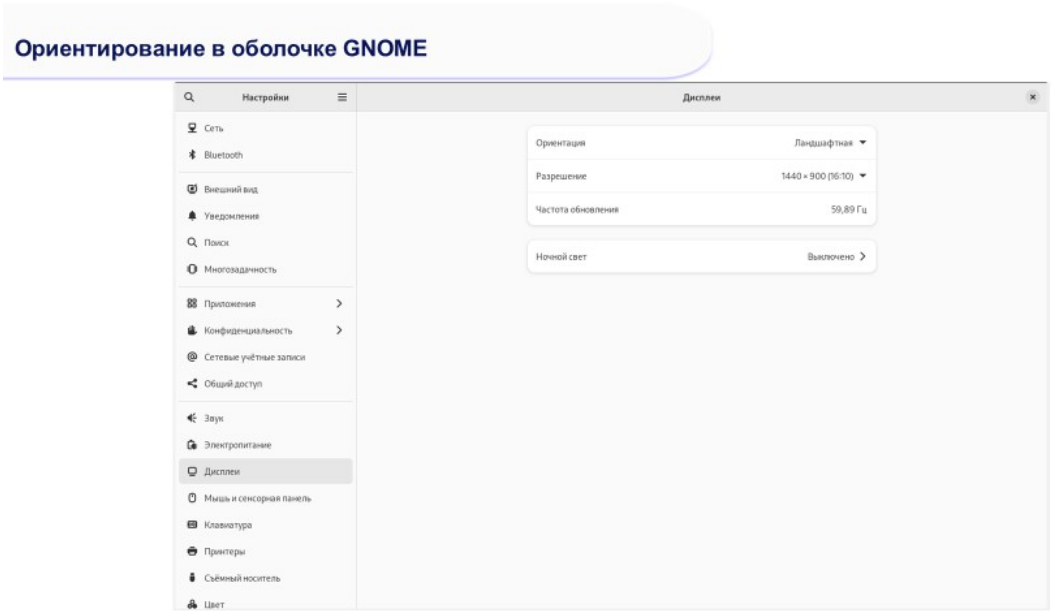


При входе в систему вы попадаете на рабочий стол и видите верхнюю панель. В центре панели доступен календарь, в правом углу – меню настроек и выхода из системы, а в левом – меню приложений и рабочих столов.

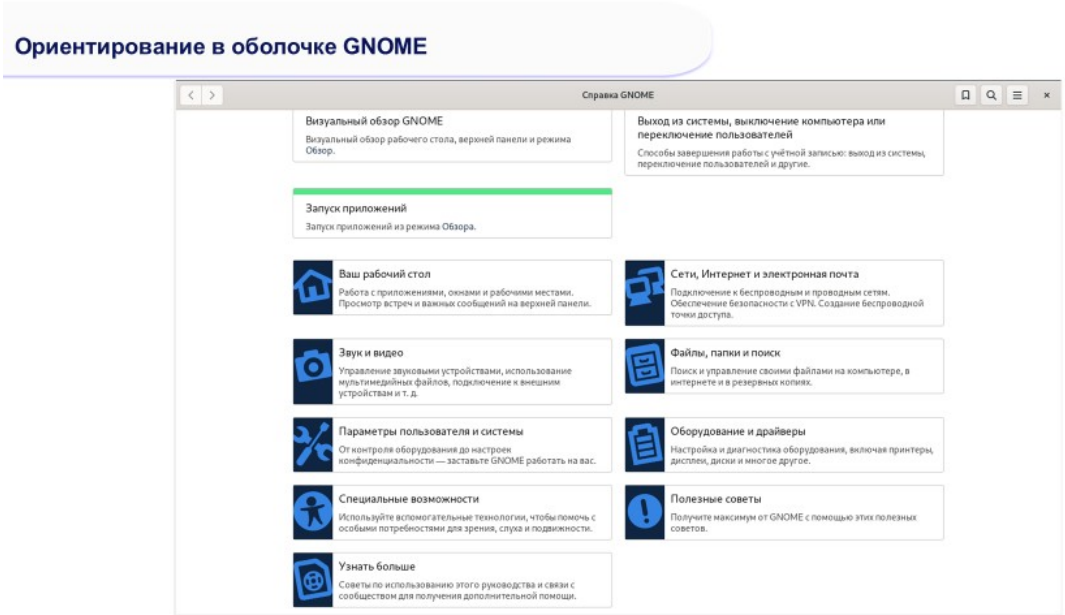


Из меню приложений и рабочих столов можно вызвать список всех приложений нажав на иконку из девяти точек. При этом рабочие столы

уменьшаются, а приложения показаны в виде постраничной сетки из подписанных иконок.

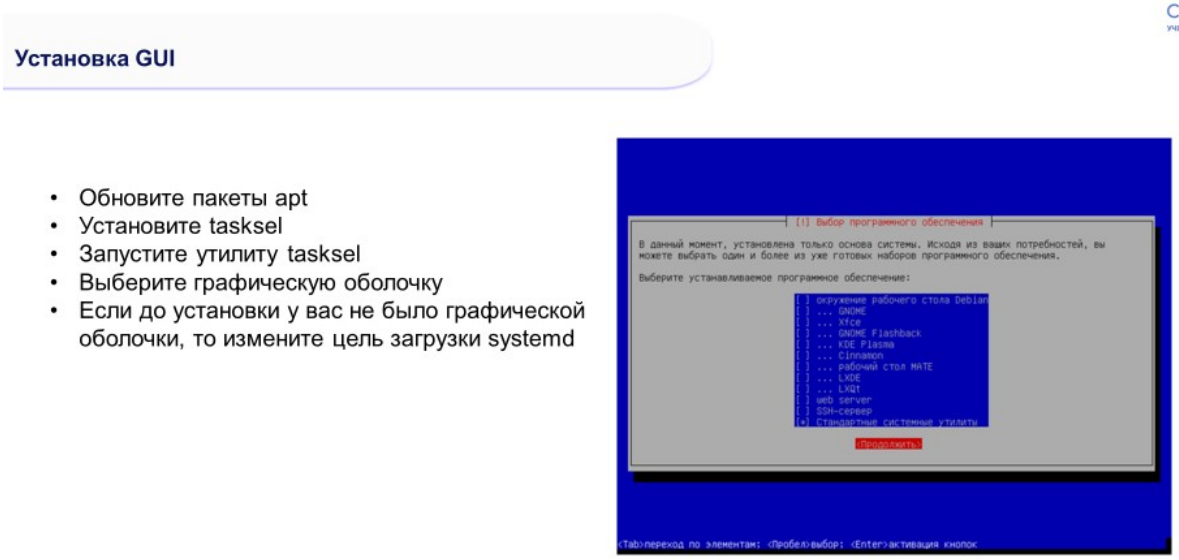


Меню настроек поделено на секции в соответствие с категориями настроек, по умолчанию открываются настройки дисплея



Через меню приложений можно вызвать справку по графической среде, которая содержит подробные инструкции для различных элементов интерфейса.

4.3. Установка GUI



После установки системы может потребоваться дополнительно установить графическое окружение, если оно отсутствует или вам неудобно работать с текущим. В Debian таком случае необходимо выполнить следующие действия:

Обновите пакеты apt:

```
sudo apt update
sudo apt upgrade
```

Установите tasksel:

```
sudo apt install tasksel
```

Запустите утилиту tasksel:

```
sudo tasksel
```

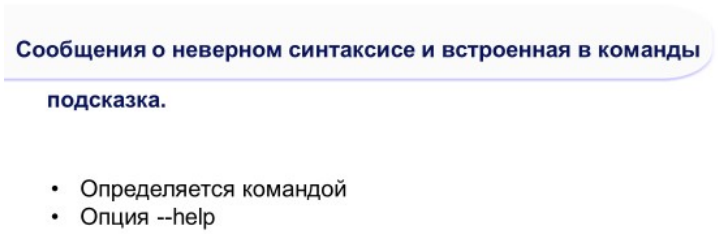
Выберите графическую оболочку с помощью стрелок и клавиши пробел, после чего нажмите Enter.

Если до установки у вас не было графической оболочки, то измените цель загрузки systemd командой:

```
sudo systemctl set-default graphical.target
```

Глава 5. Помощь и документация.

5.1. Сообщения о неверном синтаксисе и встроенная в команды подсказка.



Большинство команд выводит в ответ на попытки ввести их в неверном синтаксисе сообщение об ошибке и подсказку о том, как их следует использовать.

Пример:

```
$ pwd -h
bash: pwd: -h: invalid option
pwd: usage: pwd [-PL]
```

Примечание: Команда pwd, выводящая путь к текущему каталогу, была введена здесь с использованием опции, которая не соответствует ее синтаксису. Поэтому было получено сообщение об ошибке и на экран была выведена краткая подсказка об использовании команды.

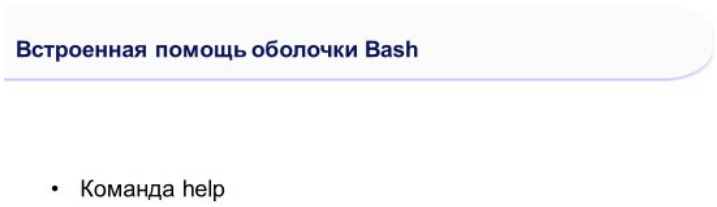
Команды GNU позволяют получить подсказку, пользуясь опцией `--help`.

Пример:

```
$wc --help
Usage: wc[OPTION]…[FILE]…
Print line, word, and byte counts for each FILE …
```

Примечание: В этом примере продемонстрировано, что использование опции --help приводит к выводу базовой помощи по команде. Во всяком случае, это верно для команд GNU, а их – подавляющее большинство.

5.2. Встроенная помощь оболочки Bash.



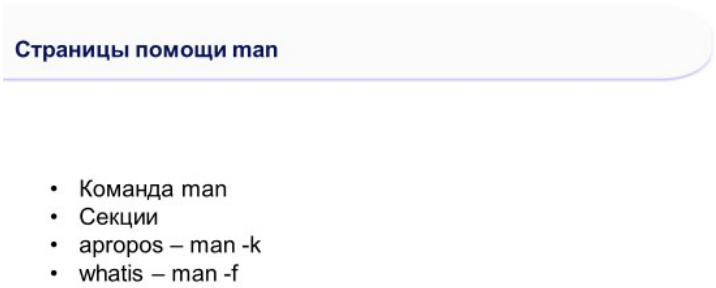
Оболочка Bash предоставляет по своим встроенным командам собственную помощь. Список всех встроенных команд Bash можно увидеть с помощью команды `help`.

Если этой команде передать в качестве аргумента любую встроенную команду Bash, то по использованию этой команды будет выведена подсказка.

Пример:

```
$ help pwd
pwd: pwd [-PL]
Print the current hosting directory. With the -P option, pwd
prints
the physical directory, without any symbolic links; the -L option
makes pwd follow symbolic links.
```

5.3. Страницы помощи man.



Система `man` (от слова manual – руководство) имеется в любой UNIX или UNIX-подобной системе. Это основное средство получения подробной информации о командах, структуре файлов конфигурации, системным вызовам и прочему.

Система `man` не рассчитана на обучение пользователей, но она предоставляет подробное описание команд.

Для получения подробной информации о команде, системном файле и т.п., необходимо вызвать команду `man` с аргументом – именем команды или иного требуемого объекта:

Пример:

```
$ man ls
```

Примечание: Эта команда выведет страницы помощи по системной команде ls.

Все страницы помощи `man` разделены на секции, приведенные в таблице ниже.

Секция	Информация
1	Описание команды пользователя.
2	Описание системных вызовов ядра.
3	Описание библиотек.
4	Информация о файлах устройств и иных специальных файлах.
5	Форматы конфигурационных файлов.
6	Помощь по играм.
7	Макросы, описание кодировок, различная информация для программиста.
8	Команды системного администрирования.
9	Процедуры и функции ядра.

Примечание: Часто используются секции с необычными именами, например, n или 1x, соответственно для команд языка TCL и для пользовательских команд с графическим интерфейсом.

Для указания команде `man` требуемой секции ее необходимо указать `man` в качестве первого аргумента.

Примечание: указание секции используется в тех случаях, если существует несколько man-страниц с одинаковыми именами.

Пример:

```
$ man 3 zlib
```

Примечание: Эта команда выведет информацию о библиотеке компрессии zlib. Информация находится в третьей секции страниц man.

Команда `man` находит среди всех страниц помощи нужную, форматирует ее и передает ее программе постраничного просмотра, используемому в системе по умолчанию.

Для постраничного просмотра обычно используется утилита `less`.

Ниже приведены некоторые внутренние команды утилиты `less`:

Команда	Действие
Ctrl-N стрелка вниз	Следующая строка.
Ctrl-P стрелка вверх	Предыдущая строка.
Ctrl-V PgDown	Страница вниз.
Alt-V PgUp	Страница вверх.
<Space>	Следующая страница.
/строка	Поиск подстроки вниз.
?строка	Поиск подстроки вверх.
n	Найти следующее вхождение.
q	Выход из man.

Если необходимо получить все возможные страницы по данной теме, то необходимо использовать опцию -a команды man.

Примечание: При этом man будет выводить все найденные страницы последовательно, то есть при выходе из страницы будет отображаться следующая, если таковая имеется.

Для получения помощи о команде или файле не зная его или ее точного названия используется опция `-k` команды `man` или же полностью эквивалентная команда `apropos`.

Каждая страница `man` начинается с раздела NAME, являющимся обязательным и содержащим описание объекта поиска.

Команда `man -k` производит поиск строки, заданной ей в качестве аргумента, по всем имеющимся страницам, просматривая нет ли в разделе NAME такой строки.

Пример:

```
$ man -k clock
CLOG_csync (4) - synchronize clocks for adjusting times in merge
adjtimex (2) - tune kernel clock
alarm (2) - set an alarm clock for delivery of a signal
clock (3) - Determine processor time
clock (n) - Obtain and manipulate time
clockdiff (8) - measure clock difference between hosts
hwclock (8) - query and set the hardware clock (RTC)
```

Тот же самый результат будет получен при выполнении команды `apropos clock`. Эта команда также просмотрит все страницы, отыскивая заданную подстроку.

Если необходимо в разделе NAME страниц помощи отыскивать не подстроку, а ее точное название, то следует использовать команду `man -f` или же `whatis`.

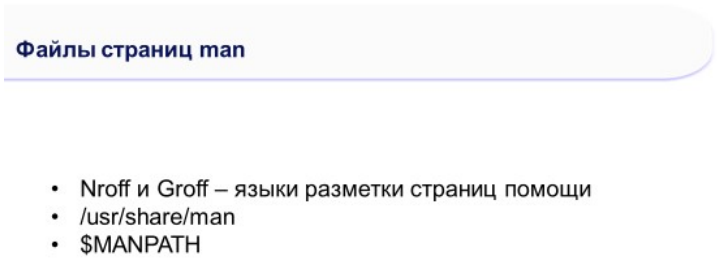
Пример:

```
$ man -f clock
clock (3) - Determine processor time
clock (n) - Obtain and manipulate time
```

Примечание: Легко заметить, что в данном случае были получены только те

страницы, имя которых точно совпадает с заданным критерием поиска. То есть, команда apropos отыскивает строку, а команда whatis – слово.

5.4. Файлы страниц man.



Страницы `man` в коммерческих UNIX системах размечены с помощью специального языка `nroff`. Утилита `nroff` не является публично доступной и свободной для использования. В силу этого в GNU была создана собственная утилита `groff`, реализующая функции `nroff`.

Пример: Ниже приведен пример фрагмента страницы `man` для команды `apropos`.

```
.TH apropos 1 "Jan 15, 1991"
.LO 1
.SH NAME
apropos \- search the whatis database for strings
.SH SYNOPSIS
.BI apropos
keyword ...
.SH DESCRIPTION
apropos searches a set of database files containing short
descriptions
of system commands for keywords and displays the result on the
standard output.
.SH "SEE ALSO"
whatis(1), man(1).
```

Примечание: Здесь легко заметить управляющие команды этого языка, начинающиеся с точки. Кстати, здесь с помощью конструкции .SH отмечаются

ненумерованные заголовки разделов man.

Страницы помощи `man` хранятся в отдельных файлах, имена которых состоят из имени раздела и суффикса – секции `man`.

Пример: Файл `rm.1` описывает команду `rm`, которая относится к первой секции системы `man`.

Чаще всего страницы `man` хранятся в сжатом с помощью утилит `gzip` или `bzip2` виде, поэтому к расширению имени файла добавляется, соответственно, `gz` или `bz2`. Стандартное место хранения страниц `man` – каталог `/usr/share/man`.

Примечание: Однако, местоположение, использовавшееся ранее – каталог `/usr/man` до сих пор используется достаточно часто, хотя это не отвечает стандарту файловой системы FHS (см. `man hier`).

В каталоге `/usr/share/man` имеются подкаталоги с именами `man1`, `man2` ... `man9`. Это – каталоги, в которых хранятся страницы соответствующих разделов.

Пример: В каталоге `/usr/share/man/man8/` хранятся страницы помощи по восьмой секции `man` – командам системного администрирования.

Если система локализована и в ней установлены национальные страницы помощи, то в каталоге `/usr/share/man` создается специальный подкаталог с локализованными страницами. Русские страницы `man` установлены в каталоге `/usr/share/man/ru`, в котором аналогично обычным страницам `man` находятся подкаталоги, соответствующие секциям локализованных страниц.

Пример:

```
$ ls /usr/share/man
man1 man2 man3 man4 man5 man6 man7 man8 man9 ru
$ ls /usr/share/man/ru
man1 man2 man3 man4 man5 man6 man7 man8 man9
```

Примечание: Каталог /usr/share/man/ru содержит подкаталоги с локализованными версиями страниц man.

Приложения и дополнительные команды, установленные в системе,

обычно размещают свои собственные страницы помощи в каталоге /usr/share/local/man. Во многих случаях стандартного списка секций помощи явно не достаточно, поэтому используют специальные секции, предназначенные для помощи по таким программам, как графическая подсистема X Window.

Примечание: Для этой подсистемы выделен специальная секция – 1x, хотя стандарт файловой системы предписывает использовать цифровые обозначения. Файлам страниц помощи для команд, относящихся к X Windows, принято устанавливать суффикс вида 1x, например, xterm.1x .

Документация в формате man может быть установлена и в нестандартных местах файловой системы. Переменная окружения MANPATH позволяет устанавливать пути доступа к различным каталогам, содержащим страницы man. Для установки и переменной MANPATH можно использовать специальную команду manpath.

Примечание: Однако в локализованных версиях GNU/Linux эта переменная может работать иначе и ее установка не приведет к изменению работы man, так как при поиске страниц на поведение man влияет переменная LANG.

Файл конфигурации /etc/man.config позволяет настраивать параметры системы man. Для разработки собственных страниц man можно использовать любой текстовый редактор, однако файл страницы помощи должен следовать формату groff. Для этого в файле должны быть выделены специальные поля, приведенные в таблице ниже, и он должен быть размечен с помощью макросов groff.

Поле	Назначение
NAME	Предназначено для указания информации, которая будет использована при поиске по ключевому слову.
SYNOPSIS	Указывает краткий список опций, заключенный в квадратные скобки.
DESCRIPTION	Детально описывает функции субъекта помощи (например, программы), здесь же приводится информация о формате команды, аргументах и так далее.
OPTIONS	Содержит список всех используемых опций и их действие.
FILE	Указывает, какие файлы используются программой.
AUTHOR	Имя автора с указанием адреса электронной почты.
SEE-ALSO	Предназначено для указания других файлов, команд и т.п., которые связаны с данной страницей помощи.
COPYRIGHT	Права собственности, политика распространения и т.п.

Файлы, относящиеся к программам, использующимся только в данной системе и не связанные с системой установки программного обеспечения (`rpm`, `dselect` и пр.), следует устанавливать в каталог `/usr/local/share/man` или `/usr/local/man`.

5.5. Система TexInfo.

Система `TexInfo` была разработана как свободная альтернатива `man` в рамках проекта GNU. Страницы `man` по многим командам и утилитам GNU предлагают обратиться к системе `TexInfo`, как к более полному источнику информации.

`TexInfo` представляет собой гипертекстовую документацию в специальном формате, организованную иерархически.

Точки разветвления называются узлами (node).

В системе `TexInfo` имеется возможность переходить между соседними узлами, родительскими и дочерними узлами, а также по специальным гипертекстовым ссылкам.

Для получения базовой помощи по системе `TexInfo` достаточно выполнить команду `info` без аргументов.

Примечание: Команда info, вызванная без аргументов, выводит наивысшую в иерархии страницу документации, отображающую верхний уровень разделов информационной системы.

Если указать конкретный объект, по которому требуется получить помощь, в качестве аргумента команды `info`, то будет выведена соответствующая страница `TexInfo`, в случае отсутствия таковой по этому

объекту будет выведена страница `man`.

Примечание: Поэтому, в любом случае, команда info обеспечивает не меньшие ресурсы помощи, чем система man.

Пример:

```
$ info ls
```

Примечание: Эта команда отобразит информацию, имеющуюся в системе TexInfo, которая относится к команде ls.

Файлы документации `info` обычно находятся в каталоге либо `/usr/share/info`, либо `/usr/info`.

Находясь в системе `info`, можно пользоваться следующими командами:

Команда	Действие
n	Следующий узел.
p	Предыдущий узел.
u	Переход к родительскому узлу.
l	Предыдущая страница.
Ctrl-N Курсор вниз	Следующая строка.
Ctrl-P Курсор вверх	Предыдущая строка.
Ctrl-F	Курсор вправо.
Ctrl-B	Курсор влево.
Ctrl-E End	Курсор в конец строки.
Ctrl-A Home	Курсор в начало строки.
Alt-F	Слово вправо.
Alt-B	Слово назад.
Ctrl-V PgDown	Страница вниз.
Alt-V PgUp	Страница вверх.
Alt-<	В начало страницы.
Alt->	В конец страницы.
sстрока	Поиск строки на странице.
q	Выход из info.

5.6. Документация, поставляющаяся с программными пакетами.

В каталоге `/usr/share/doc` находятся подкаталоги с именами вида приложение-версия.

Пример: anjuta-0.1.9 – каталог, соответствующий среде разработки anjuta, версии 0.1.9.

В этих каталогах размещается дополнительная документация от разработчиков программного обеспечения. Это могут быть подробные руководства по использованию программ или же просто файлы `README`.

Ранее эта документация размещалась в каталоге `/usr/doc`.

Примечание: Насколько подробная и исчерпывающая документация хранится в этих каталогах зависит от разработчика. Часто документация бывает очень подробной и по умолчанию в данном дистрибутиве может не устанавливаться. В таком случае обычно имеется отдельный пакет, содержащий документацию для данного программного продукта.

Информация, доступная в этих каталогах, может быть ограничена следующими файлами:

1. `README` – краткая информация о продукте, последние изменения, не вошедшие в документацию, предупреждения и, возможно, инструкции по установке.

2. `INSTALL` – инструкции по установке.
3. `ToDo` - что необходимо изменить или доделать в продукте в ближайшее время.
4. `Changelog` – история изменений и разработки продукта.
5. `License` – текст лицензионного соглашения.

В последнее время документация по продуктам часто поставляется в формате HTML.

Глава 6. Управление программным обеспечением.

6.1. В чем состоит управление программным обеспечением.

Одной из основных задач системного администрирования, возникающей сразу после установки системы и актуальной в течении всей ее эксплуатации, является управление программным обеспечением. Этот процесс имеет следующие составляющие:

- **Установка нового программного обеспечения.**

В течении эксплуатации системы неизбежно будет возникать необходимость установки нового программного обеспечения. Во-первых, по причине того, что маловероятно установить систему так, чтобы она вечно удовлетворяла постоянно изменяющимся требованиям эксплуатации. Во-вторых, в течении длительной эксплуатации системы могут проявиться концептуальные проблемы (например, с безопасностью) для каких-либо программных компонент. Для устранения этих проблем может потребоваться отказаться от данной программы вообще и перейти к другой аналогичной.

- **Обновление программного обеспечения.**

В коде программ постоянно обнаруживаются более или менее серьезные проблемы. Поэтому системный администратор должен отслеживать сообщения об ошибках и проблемах в программном обеспечении и обновлять его. Одна из наиболее распространенных причин взлома систем безопасности –

использование устаревшего программного обеспечения, имеющего известные проблемы с безопасностью.

• **Проверка подлинности ПО.**

Сопутствующая установке и обновлению программного обеспечения проблема состоит в проверке подлинности нового программного обеспечения. Если новые программы или пакеты обновления поступают из ненадежных источников, то при этом резко увеличивается вероятность попадания в систему программ-закладок (троянских коней), выполняющих в системе несанкционированные действия.

• **Удаление программного обеспечения.**

Необходимость удаления продиктована требованием наличия в системе только того программного обеспечения, которое действительно необходимо. В большинстве современных дистрибутивов GNU/Linux при установке системы на диск копируется заранее предопределенный поставщиком дистрибутива набор программ. Многие из них в работе не используются и просто занимают место на диске. Еще хуже, если в системе работает программа, открывающая сокеты для ненужных сетевых соединений. Такие программы становятся удобной мишенью для взломщиков. Также неиспользуемое программное обеспечение может требовать наличия в системе неиспользуемых учетных записей пользователей, которые также потенциально могут быть использованы взломщиками. Еще одна причина необходимости удаления программ заключается в возможных конфликтах нужных в системе программ с уже установленным программным обеспечением.

• Проверка целостности программного обеспечения.

Эта задача связана с защитой от возможной порчи программного обеспечения при сбоях в системе или в результате чьей-либо несанкционированной деятельности в системе. Проверка целостности заключается в проверке размеров файлов, прав доступа и владения, контрольных сумм, времени модификации и прочее.

• **Создание собственных пакетов или пересборка существующих.**

Сборка новых пакетов больше связана с деятельностью разработчиков программного обеспечения или создателей пакетов, отвечающих за их поддержку (maintainers). Однако пересборка существующего пакета требуется довольно часто и в работе обычного системного администратора. Это может быть связано, например, с желанием получить оптимизированное программное обеспечение для конкретной аппаратной платформы.

- **Регистрация и лицензирование.**

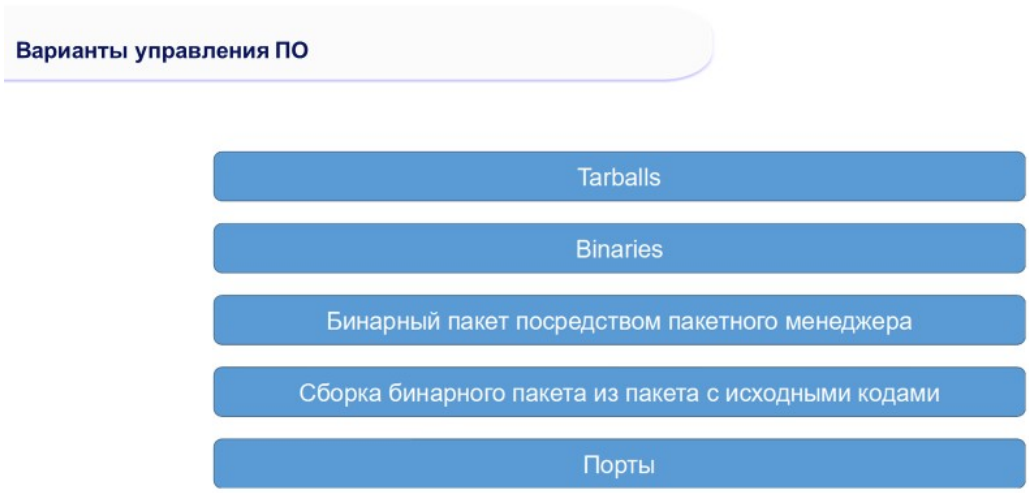
Менее распространенная задача в мире свободного программного обеспечения – регистрация и лицензирование программного обеспечения. В последнее время GNU/Linux часто используется для работы проприетарного (коммерческого несвободного) программного обеспечения, например, в качестве платформы для СУБД Oracle. Проприетарное программное обеспечение должно быть зарегистрировано и лицензировано. Имеется также обилие дистрибутивов GNU/Linux, таких, как Red Hat Enterprise Linux и SUSE Linux, требующих регистрации и лицензирования. Другой пример – большинство антивирусных пакетов для GNU/Linux являются проприетарными и требуют регистрации и лицензирования.

Существует несколько вариантов установки программного обеспечения:

1. Сборка и установка из архивов с исходным кодом (tarballs).
2. Установка из архивов с бинарным машинным кодом (binaries).
3. Установка из бинарных пакетов (package) с помощью систем управления пакетами (package manager).
4. Сборка бинарного пакета из пакета с исходным кодом (source package) с последующей установкой из получившегося бинарного пакета с помощью систем управления пакетами (package manager).
5. Сборка и установка программного обеспечения из архивов с исходным кодом под управлением специального скрипта автоматизации – порта (основной способ установки пакетов в Gentoo, где порты называются portage по аналогии с port во FreeBSD).

В подавляющем большинстве GNU/Linux дистрибутивов имеется система управления пакетами.

Такая система в значительной мере упрощает и стандартизирует процесс управления программным обеспечением.



Основываясь на информации, предоставляемой на сайте www.distrowatch.org можно утверждать, что наиболее распространены четыре системы управления пакетами:

1. RPM – Red Hat Package Manager. Применяется в RH и подобных ему системах, SUSE и многих других дистрибутивах. Предоставляет возможности как установки бинарных пакетов (файлы .rpm), так и с помощью предварительной пересборки из пакетов с исходным кодом (файлы .src.rpm или, реже, .srpm).
2. Система управления пакетами Debian. Кроме Debian используется в собранных на его основе дистрибутивах, например в Ubuntu. Предоставляет широкие возможности по управлению пакетами. Файлы пакетов имеют расширению .deb.
3. Система портов Gentoo. Этот дистрибутив ориентирован на сборку высоко оптимизированного программного обеспечения с помощью специальных скриптов для сборки программного обеспечения из архивов с исходным кодом. Позволяет также устанавливать заранее собранные пакеты (фактически, пакет – это заранее скомпилированное ПО из порта). Преимуществом этой системы является непревзойденные возможности постоянного обновления программного обеспечения.
4. Система управления пакетами, принятая в SlackWare и ему подобных дистрибутивах. Здесь применяется установка программного обеспечения из заранее собранных бинарных архивов в формате tar.

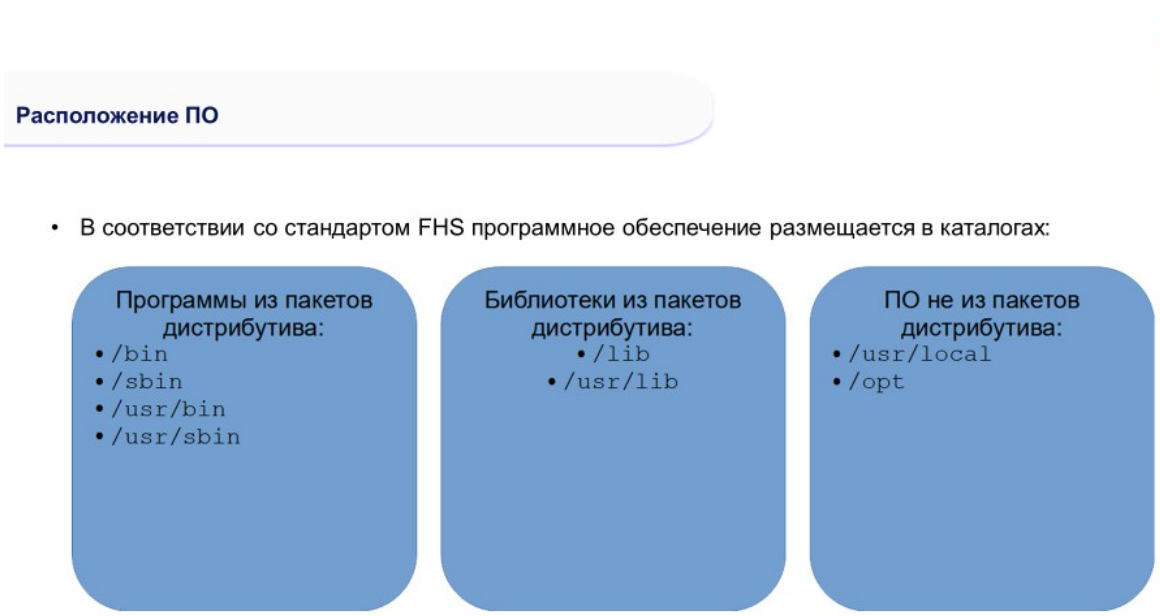
Первые две системы из приведенного выше списка имеют превалирующее распространение.

Система управления программным обеспечением предоставляет следующие преимущества:

1. Обеспечивается единообразное управление программным обеспечением.
2. Программы устанавливаются в стандартные места файловой системы.
3. Управление программным обеспечением значительно облегчено.
4. Во многих системах предоставляются возможности разграничения ролей пользователей, могущих выполнять те или иные функции в управлении программным обеспечением.
5. Легко обеспечивается возможность проверки целостности программного обеспечения.

Однако имеются и недостатки, связанные с использованием систем управления пакетами, ориентированных на бинарные пакеты:

1. Часто бывает затруднительно установить не все программное обеспечение из пакета, а только его часть.
2. Трудно устанавливать программы в нестандартные места файловой системы, например, в домашние каталоги пользователей.
3. Трудно, а иногда и невозможно устанавливать программы из других дистрибутивов или из предыдущей версии этого же дистрибутива.
4. Пакеты необходимо пересобирать для оптимизации под данную систему, а также для добавления или удаления некоторой функциональности.



Примечание: Каталог `/opt` обычно используется для программного обеспечения, не поставляемого в составе дистрибутива (опциональное ПО). В каталог `/usr/X11R6` помещаются файлы, относящиеся к системе X Window.

Файлы помощи для программного обеспечения, устанавливаемого с помощью систем управления пакетами, должны размещаться в `/usr/share/man`, а документация – в `/usr/share/doc`.

Программное обеспечение, устанавливаемое самостоятельно с помощью сборки из архивов с исходным кодом, размещается в подкаталогах каталога `/usr/local`.

При установке программного обеспечения очень часто возникает конфликт пакетов или зависимостей, который может иметь следующие причины:

1. Два пакета взаимно исключают совместную работу.

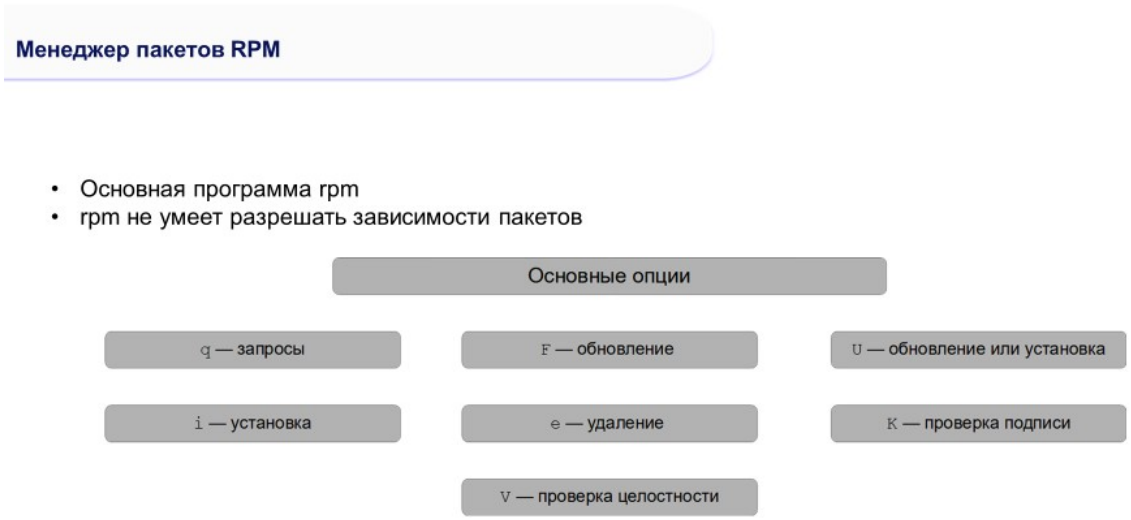
   Пример: нельзя использовать два сервера SMTP (Simple Mail Transfer Protocol). При попытке установить программу `postfix` в системе, где установлена почтовая программа `sendmail`, возникнет конфликт и менеджер пакетов выведет сообщение об ошибке.

2. Основная библиотека `libc` (`glibc`), используемая в системе, имеет версию, отличную от версии библиотеки, с которой собран устанавливаемый пакет. Этот вариант конфликта может быть также и для

других библиотек. Однако, конфликт с `libc` разрешить можно, фактически только перейдя на другой дистрибутив GNU/Linux.

3. Устанавливаемый пакет может требовать наличие других программ или библиотек, отсутствующих в настоящий момент в системе.
4. Может возникать также и конфликт версий конфигурационных файлов. Однако такой вид конфликта разрешается довольно легко – достаточно переименовать старые версии (или же просто одноименные конфигурационные файлы) файлов конфигурации, оставшиеся, например, от предыдущей версии программы.

6.2. Менеджер пакетов RPM.



В Red Hat Linux и подобных ему дистрибутивах используется менеджер пакетов RPM.

Менеджер пакетов RPM реализован с помощью группы программ, главная из которых `/bin/rpm`.

Ниже приведен список важнейших режимов работы RPM:

1. Запрос. Включается опцией `-q` или `--query`.
2. Проверка целостности файлов, установленных из пакета. Включается опцией `-V` или `--verify`.
3. Проверка электронной подписи пакета. Включается при использовании опции `-K` или `--checksig`.
4. Установка пакета. Требует использования опции `-i` или `--install`.
5. Обновление пакета. Работает с опцией `-U` или `--upgrade`.
6. Обновление версии пакета. Запускается при установленной опции `-F` или `--freshen`.
7. Удаление установленных пакетов. Этот режим включается при использовании опции `-e` или `--erase`.

Используя режим запроса (опция `-q`) можно, например, узнать точную версию установленного в системе пакета:

Пример:

```
$ rpm -q bash
bash-5.0.17-2.fc33.x86_64
```

Примечание: В этом примере был осуществлен запрос к базе данных RPM. В качестве ключа запроса использовалось краткое название установленного в системе пакета - `bash`. В результате исполнения запроса было получено полное имя пакета.

Для получения более подробной информации о пакете следует воспользоваться опциями `-qi` :

Пример:

```
$ rpm -qi bash
Name : bash
Version : 5.0.17
Release : 2.fc33
Architecture: x86_64
Install Date: Ср 28 окт 2020 19:37:44
Group : Unspecified
Size : 7709818
License : GPLv3+
Signature : RSA/SHA256, Вт 28 июл 2020 03:10:09, Key ID
49fd77499570ff31
Source RPM : bash-5.0.17-2.fc33.src.rpm
Build Date : Пн 27 июл 2020 18:17:35
Build Host : buildhw-x86-14.iad2.fedoraproject.org
Packager : Fedora Project
Vendor : Fedora Project
URL : https://www.gnu.org/software/bash
Bug URL : https://bugz.fedoraproject.org/bash
Summary : The GNU Bourne Again shell
Description :
The GNU Bourne Again shell (Bash) is a shell or command language
interpreter that is compatible with the Bourne shell (sh). Bash
incorporates useful features from the Korn shell (ksh) and the C
shell
(csh). Most sh scripts can be run by bash without modification.
```

Примечание: Использование опций -qi выводит подробную информацию о пакете, в том числе суммарный размер файлов, установленных из пакета (поле Size), и группу пакетов, к которой этот пакет принадлежит (поле Group).

Для получения списка файлов, установленных из пакета, необходимо установить опции `-ql` :

Пример:

```
$ rpm -ql bash
/etc/skel/.bash_logout
/etc/skel/.bash_profile
/etc/skel/.bashrc
/usr/bin/alias
/usr/bin/bash
/usr/bin/bashbug
/usr/bin/bashbug-64
/usr/bin/bg
...
```

Можно также ограничиться лишь выводом списка конфигурационных файлов, установленных из заданного в качестве аргумента пакета, используя для этого опции `-qc` :

Пример:

```
$ rpm -qc bash
/etc/skel/.bash_logout
/etc/skel/.bash_profile
/etc/skel/.bashrc
```

С помощью опций `-qd` можно получить информацию о файлах помощи и документации, установленных из этого пакета:

Пример:

```
$ rpm -qd bash
/usr/share/doc/bash/FAQ
/usr/share/doc/bash/INTRO
/usr/share/doc/bash/RBASH
/usr/share/doc/bash/README
/usr/share/doc/bash/bash.html
/usr/share/doc/bash/bashref.html
/usr/share/info/bash.info.gz
/usr/share/man/man1/..1.gz
/usr/share/man/man1/:.1.gz
/usr/share/man/man1/[.1.gz
/usr/share/man/man1/alias.1.gz
...
```

Сочетание опций `-qa`, позволяет получить список из всех пакетов,

установленных в системе. Этим удобно пользоваться, если название пакета не известно.

Пример: можно получить информацию о всех пакетах, в имени которых имеется строка ftp:

```
$ rpm -qa | grep ftp
ftplib-4.0-13.fc33.x86_64
ncftp-3.2.5-21.fc33.x86_64
ftp-0.17-84.fc33.x86_64
tftp-server-5.2-31.fc33.x86_64
lftp-4.9.2-1.fc33.x86_64
python3-requests-ftp-0.3.1-20.fc33.noarch
tftp-5.2-31.fc33.x86_64
```

Сочетание опций `-qf` позволяет в качестве ключа запроса указать имя файла и узнать из какого пакета установлен этот файл:

Пример:

```
$ rpm -qf /usr/bin/passwd
passwd-0.80-9.fc33.x86_64
```

Для получения информации о еще не установленном в системе пакете можно использовать опции `-qp`, позволяющие извлекать информацию из файла пакета (из .rpm файла):

Пример:

```
$ rpm -qip Загрузки/zoom_x86_64.rpm
warning: Загрузки/zoom_x86_64.rpm: Header V4 RSA/SHA1 Signature,
key ID 61a7c71d: NOKEY
Name : zoom
Version : 3.5.374815.0324
Release : 1
Architecture: x86_64
Install Date: (not installed)
Group : default
Size : 269036822
License : see https://www.zoom.us/
Signature : RSA/SHA1, Wed 25 Mar 2020 10:01:53 AM +05, Key ID
b903bf1861a7c71d
Source RPM : zoom-3.5.374815.0324-1.src.rpm
Build Date : Wed 25 Mar 2020 09:58:57 AM +05
Build Host : localhost
```

```
Relocations : /
Packager : Zoom Linux Team <linux-dev@zoom.us>
Vendor : Zoom Video Communications, Inc.
URL : https://www.zoom.us
Summary : Zoom, #1 Video Conferencing and Web Conferencing Service
Description :
Zoom, #1 Video Conferencing and Web Conferencing Service
Zoom, the cloud meeting company, unifies cloud video conferencing,
simple online meetings, and group messaging into one easy-to-use
platform. Our solution offers the best video, audio, and screen-
sharing experience across Zoom Rooms, Windows, Mac, Linux, iOS,
Android, and H.323/SIP room systems.
```

Опция `-V` переключает `rpm` в режим проверки целостности установленных файлов из пакетов.

Пример: проверки целостность пакета mozilla :

```
$ rpm -V bash
S.5....T. c /etc/skel/.bashrc
```

Примечание: Выводимая в первом столбце информация означает следующее: S - у файла не совпадает размер с указанным в базе данных, 5 - не совпадает сигнатура md5, T - время модификации изменено.

Установка или обновление пакета осуществляется с помощью опций:

1. `-i` - пакет будет установлен в случае отсутствия в системе его предыдущей версии;
2. `-U` - если в системе существовала предыдущая версия пакета, она будет заменена на устанавливаемую. В противном случае пакет будет просто установлен;
3. `-F` - возможно только обновление версии пакета. Если предыдущей версии в системе не обнаружено, то установка произведена не будет.

При установке или обновлении пакетов в качестве аргумента для rpm указывают имена файлов пакетов (.rpm файлы).

Имеется возможность проверить последствия установки без проведения реальной установки пакетов. Для этого предназначена опция `--test` :

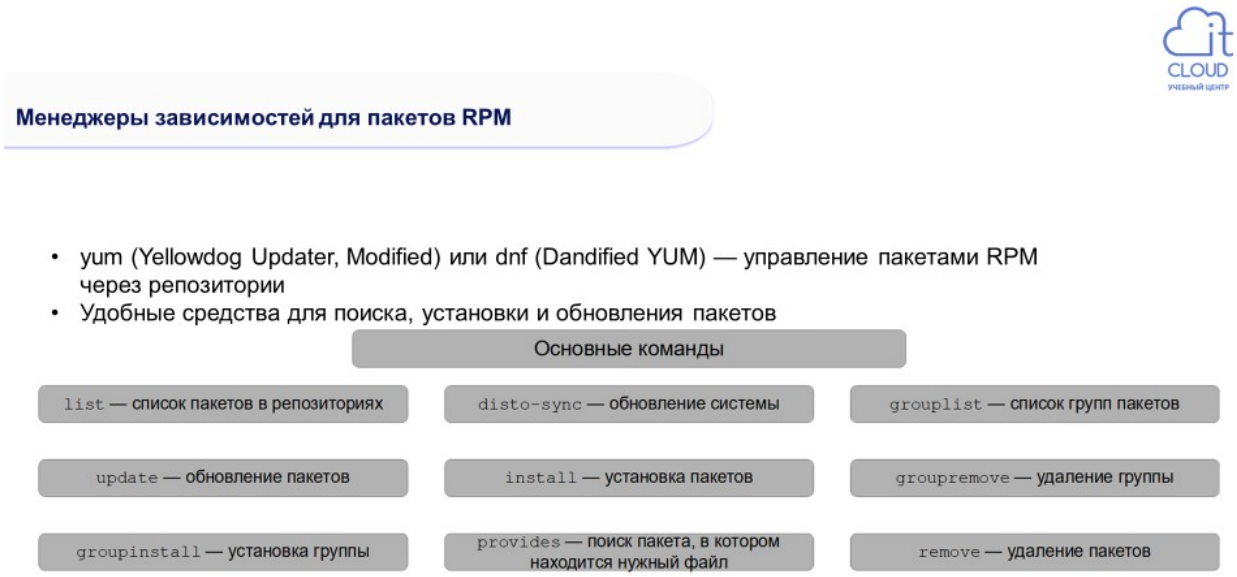
Эта же опция полезна для проверки последствий удаления пакетов из системы без реального удаления.

Пример: можно узнать результаты удаления пакета `postfix`:

```
$ rpm -e --test postfix
ошибка: удаление этих пакетов нарушит зависимости:
MTA нужен для fetchmail-6.2.1-alt1
MTA нужен для mutt-1.4i-alt3
MTA нужен для gnome2-media-cddbslave-2.2.1-alt1
MTA нужен для gnome2-media-gcdplayer-2.2.1-alt1
MTA нужен для pilot-mailsync-0.7.1-alt1
```

Примечание: Информация, выведенная этой командой, сообщает о зависимости нескольких пакетов в системе от пакета postfix.

Бывают, все же, случаи, когда установка или удаление пакета должно быть произведено не взирая на нарушение зависимостей. Для этого предназначена опция `--nodeps`.



Как правило команду `rpm` не применяют напрямую в RH подобных системах. Вместо этого используются специальные системы упрощающие получение, обновление, поиск, установку, удаление и т. д. пакетов.

В таких системах не рекомендуется манипулировать пакетами напрямую командой `rpm`.

Среди основных это менеджеры пакетов `yum` (устарел) и `dnf`.

Данные утилиты используют собственные базы данных для работы с пакетами, именно по этой причине не рекомендуется использовать команду rpm для прямой установки, удаления или обновления пакетов.

Данные менеджеры используют понятие репозитория — некого хранилища информации о пакетах и собственно пакетов.

Репозитории могут быть как сетевыми, так и локальными.

Добавление репозитория обычно осуществляется установкой соответствующего пакета.

Описание репозиториев находится в каталоге `/etc/yum.repos.d`.

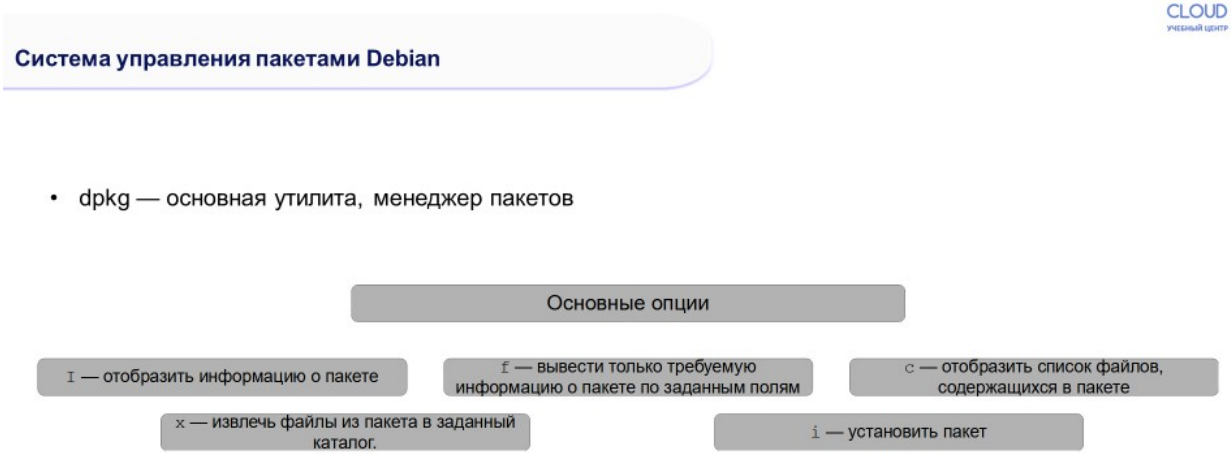
В файле, который описывает репозиторий, например, `/etc/yum.repos.d/rocky.repo`, необходимо убедиться, что репозиторий включен — опция `enabled=1`.

Основной файл конфигурации для `yum` `/etc/yum.conf`. Для `dnf` `/etc/dnf/dnf.conf`.

Основные команды и опции `yum` и `dnf` одинаковые:

1. `list` — список пакетов в репозиториях;
2. `install` — установка пакетов, которые указаны в команде;
3. `remove` — удаление пакетов;
4. `update` — обновление пакетов;
5. `disto-sync` — обновление системы;
6. `grouplist` — список групп пакетов;
7. `groupinstall` — установка группы;
8. `groupremove` — удаление группы;
9. `provides` — поиск пакета, в котором находится нужный файл.

6.3. Система управления пакетами Debian.



Хотя система управления пакетами Debian позволяет устанавливать программное обеспечение как с CD/DVD-ROM или архивов на жестких дисках, так и с сетевых источников, структура ее в первую очередь ориентирована на удобство сетевой установки непосредственно с зеркала архива Debian. Эта ориентация продиктована желанием обеспечить быструю и удобную установку нового программного обеспечения и обновление существующего с минимальными затратами времени без ожидания выхода пакетов обновлений на жестких носителях.

Структура размещения файлов на дисках установочного комплекта Debian GNU/Linux точно следует структуре архивов программного обеспечения на веб серверах Debian.

Обычно в репозитории представлены несколько типов дистрибутивов Debian:

1. Experimental(экспериментальная) - это группа пакетов Debian включающих программное обеспечение находящееся в настоящее время в процессе развития и не обязательно завершённое
2. Unstable(нестабильная) – работающие, но не прошедшие должного тестирования пакеты.
3. Testing(тестовая) – предстабильная версия, нужна для стабилизации и перепроверки..

4. Stable(стабильная) – стабильные.
5. Oldstable(старая стабильная) – предыдущий релиз.
6. Oldoldstable(старая старая стабильная) – прежний oldstable.

Репозиторий программного обеспечения Debian разделен на несколько секций. Критериями разделения являются:

- Соответствие программного обеспечения критериям свободы, принятым в Debian.
- Наличие или отсутствие экспортных ограничений на программное обеспечение, связанными с особенностями законодательства США.
- Находятся ли в пакете бинарные скомпилированные программы, или же исходный код.
- Относится ли пакет к стабильной ветви дистрибутива, либо к экспериментальной.

По критерию свободы программного обеспечения в архиве Debian выделены следующие секции:

- `main` - программное обеспечение, полностью соответствующее критерию свободы Debian. Все пакеты, принадлежащие этой секции никоим образом не зависят от каких-либо пакетов, принадлежащих другим секциям.
- `contrib` - свободное программное обеспечение (по критерию Debian).
- `non-free` - программное обеспечение, не соответствующее критерию свободы Debian.
- `non-free-firmware` - драйверы, не соответствующее критерию свободы Debian (начиная с Debian 12).

Кроме того все пакеты классифицированы по уровню приоритета:

- `required` – пакет должен быть установлен для правильной работы ОС.
- `important` - важные пакеты, присутствующие в большинстве UNIX подобных ОС.
- `standard` – утилиты командной строки.
- `optional` – дополнительные пакеты и пакеты X Window.
- `extra` – пакеты для специфического или частного применения, возможно, конфликтующие с пакетами из более приоритетных секций.

При работе с системой управления пакетами Debian могут использоваться следующие основные программы:

- `dselect` - основная программа управления пакетами, предоставляющая интерфейс меню и возможности индивидуального выбора пакетов. Эта утилита в основном предназначена для использования профессионалами.
- `aptitude` - исключительно удобная программа управления пакетами, ориентированная на текстовый интерфейс. Облегчает управление пакетами , так как визуализирует иерархическую структуру классификации пакетов, позволяя быстрее находить требуемые пакеты.
- `tasksel` - программа, позволяющая управлять целыми группами пакетов, относящимися к заранее определенным категориям (так называемые tasks - наборы пакетов). Эта программа идеальна при отсутствии необходимости индивидуального управления пакетами, так как позволяет разом установить целый набор пакетов, связанных с выполнением какой-либо задачи, например, программирования на языке Python.
- `dpkg` - утилита с интерфейсом командной строки для индивидуального управления пакетами. Позволяет установить программное обеспечение непосредственно из файла пакета (.deb файла).
- `apt` (`apt-get`)- утилита с интерфейсом командной строки для индивидуального управления пакетами и манипуляций с их источниками. Она не предназначена для работы с файлами пакетов, вместо этого она действует на основе информации об имени пакета, его статусе и местонахождении (не важно локальном или в сети).
- `synaptic` – графический менеджер пакетов.

Утилиты `dselect`, `aptitude` и `tasksel` предоставляют удобный пользовательский интерфейс, базируясь на функциональности программ `dpkg` и `apt`.

Программы `dpkg` и `apt` предполагают, что база данных системы управления пакетами Debian размещается в каталоге `/var/lib/dpkg`. Здесь находятся файлы с информацией о списке доступных и установленных пакетов, о статусе пакетов и прочем.

Команда `dpkg` удобна для непосредственного обращения к файлам пакетов Debian. Наиболее часто при этом используются следующие опции `dpkg`:

- `-I` - отобразить информацию о пакете.
- `-f` - вывести только требуемую информацию о пакете по заданным полям.
- `-c` - отобразить список файлов, содержащихся в пакете.
- `-x` - извлечь файлы из пакета в заданный каталог.
- `-i` - установить пакет.

Ниже приведен пример обращения к файлу пакета для получения информации о его зависимостях, версии и о пакетах, для которых наличие этого пакета рекомендовано.

Пример:

```
$ dpkg -f Download/rootstrap_0.3.17-1_i386.deb Depends Recommends
Version

Version: 0.3.17-1
Depends: python, user-mode-linux (>= 2.4.20-4um-1), debootstrap
(>= 0.1.16.4), dpkg (>= 1.9.19)
Recommends: uml-utilities
```

Для удаления пакетов с помощью dpkg используется опция `-r`. При таком удалении пакетов конфигурационные файлы не удаляются.

Если необходимо произвести полное удаление пакета, то следует использовать опцию `-P`.

Команда `dpkg -l` позволяет получить список всех установленных в системе пакетов. Если же при этом используется строка аргумента, то будет выведена информация об установленных пакетах, имена которых совпадают со строкой аргумента.

Пример:

```
$ dpkg -l mc

Desired=Unknown/Install/Remove/Purge/Hold
|    Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/   Err?=(none)/Hold/Reinst-required/X=both-problems   (Status,Err:
uppercase=bad)
||/ Name Version Описание
+++-==================-==================-
================================================
pn mc <none> (no description available)
```

```
$ dpkg -l bash

Desired=Unknown/Install/Remove/Purge/Hold
|    Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/   Err?=(none)/Hold/Reinst-required/X=both-problems   (Status,Err:
uppercase=bad)
||/ Name Version Описание
+++-==================-==================
================================================
ii bash 2.05b-14 The GNU Bourne Again SHell
```

Примечание: В информации, выведенной этими командами можно увидеть отличия для неустановленного пакета mc и установленного bash.

Информацию о доступных пакетах можно получить, используя опцию `-p` команды `dpkg`.

Пример:

```
$ dpkg -p mc
Package: mc
Priority: optional
Section: utils
Installed-Size: 5360
Maintainer: Adam Byrtek <alpha@debian.org>
Architecture: i386
Version: 1:4.6.0-4.6.1-pre1-1
Replaces: mc-common, manpages-pl (<= 20030210)
Depends: libc6 (>= 2.3.2.ds1-4), libglib2.0-0 (>= 2.2.3), libgpmg1
(>= 1.19.6-1)
Suggests: perl, mime-support
Conflicts: mc-common, suidmanager (<< 0.52)
Filename: pool/main/m/mc/mc_4.6.0-4.6.1-pre1-1_i386.deb
Size: 1970746
MD5sum: 578d4690f972aa6ae7851362fd331e5b
Description: Midnight Commander - a powerful file manager
GNU Midnight Commander is a text-mode full-screen file manager. It
uses a two panel interface and a subshell for command execution.
It
includes an internal editor with syntax highlighting and an
internal
viewer with support for binary files. Also included is Virtual
Filesystem (VFS), that allows files on remote systems (e.g. FTP
servers) and files inside archives to be manipulated like real
files.
```

При необходимости получить список файлов, установленных из пакета следует использовать опцию `-L` команды `dpkg`:

Пример:

```
$ dpkg -L info
/.
/usr
/usr/bin
/usr/bin/info
/usr/bin/infokey
/usr/share
/usr/share/info
/usr/share/info/info.info.gz
/usr/share/info/info-stnd.info.gz
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/info.1.gz
/usr/share/man/man1/infokey.1.gz
/usr/share/man/man5
/usr/share/man/man5/info.5.gz
/usr/share/doc
/usr/share/doc/info
/usr/share/doc/info/copyright
/usr/share/doc/info/changelog.gz
/usr/share/doc/info/changelog.Debian.gz
/usr/lib
/usr/lib/menu
/usr/lib/menu/info
```

Программа `dpkg` предоставляет возможность определить, из какого пакета был установлен некоторый файл. Для этого следует использовать опцию `-S`.

Пример:

```
$ dpkg -S /etc/bash.bashrc
bash: /etc/bash.bashrc
```

Как и `dpkg`, команда `apt` предназначена для установки пакетов из командной строки. Но, в отличие от `dpkg`, `apt` работает не напрямую с файлами пакетов, а с источниками пакетов (комплекты CD/DVD-ROM, FTP и т.д.).

`apt` – ориентирована на интерактивное использование. Помимо нее

имеются так же специализированные утилиты: `apt-get`, `apt-cdrom`, `apt-cache` и др.

Пакет, который требуется установить, при использовании `apt`, указывают не с помощью указания имени файла пакета, а с помощи имени самого пакета. При этом получение, установка и конфигурирование пакета будут произведены автоматически.

Перед установкой пакетов следует обновить локальный кеш репозиториев командой `apt update`.

Список источников пакетов находится в файле `/etc/apt/sources.list` источников.

Пример фрагмента этого файла:

```
deb cdrom:[Debian GNU/Linux testing _Sarge_ - Official Snapshot
i386 Binary-1 (20040410)]/ unstable contrib main
deb http://security.debian.org/ stable/updates main contrib
```

Примечание: В этом фрагменте первый источник - первый CD диск установочного комплекта, а второй источник пакетов - сайт обновлений, связанных с безопасностью.

Если среди источников имеются как локальные (например, CD-ROM), так и удаленные, то рекомендуется в начале файла `/etc/apt/sources.list` указывать локальные источники.

При необходимости добавления в список источников нового CD диска с пакетами Debian, следует использовать команду `apt-cdrom add`.

Для установки пакета с помощью `apt` достаточно указать его имя в качестве аргумента команды `apt install`.

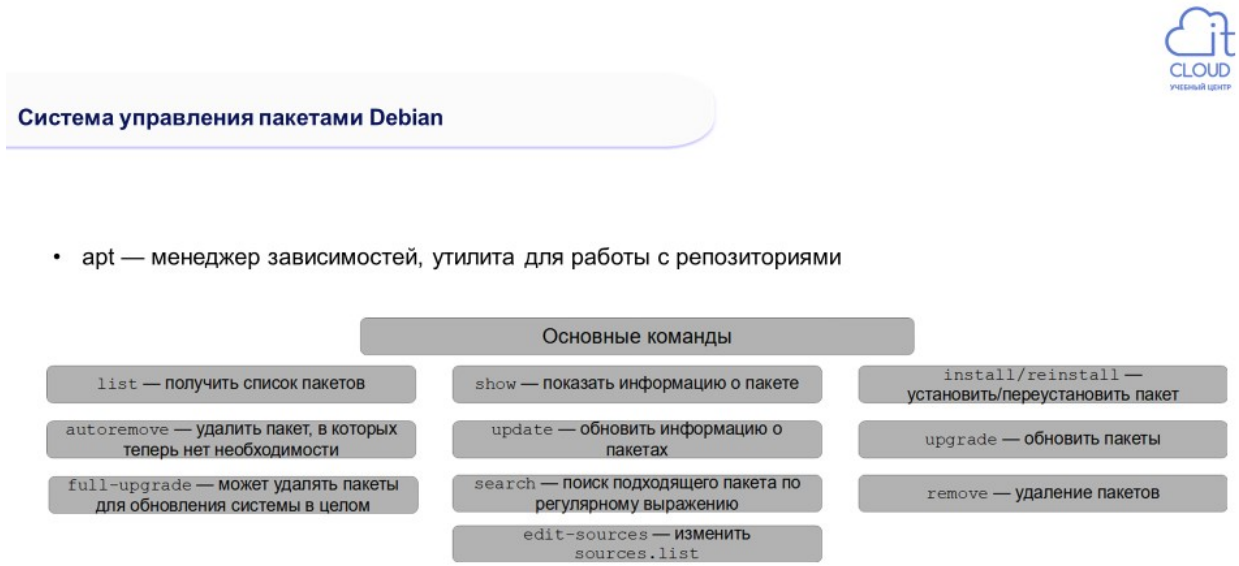
Пример:

```
# apt install lynx

Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
lynx
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0B/1855kB of archives.
After unpacking 4694kB of additional disk space will be used.
Media Change: Please insert the disc labeled
'Debian GNU/Linux testing _Sarge_ - Official Snapshot i386 Binary-
```

```
1 (20040410)'
in the drive '/cdrom/' and press enter

Selecting previously deselected package lynx.
(Reading database ... 101279 files and directories currently
installed.)
Unpacking lynx (from .../l/lynx/lynx_2.8.5-1_i386.deb) ...
Setting up lynx (2.8.5-1) ...
```

Примечание: Так как требуемый пакет находится на CD диске установочного комплекта, программа apt предложила поместить этот диск в дисковод и установила с него требуемое программное обеспечение.

Для поиска пакетов, которые предположительно содержат нужный файл (например программу) используется утилита `apt-file`.



Чаще всего производятся следующие действия с apt:

- `list` — получить список пакетов
- `search` — поиск подходящего пакета по регулярному выражению
- `show` — показать информацию о пакете
- `install/reinstall` — установить/переустановить пакет
- `remove` — удалить пакет
- `autoremove` — удалить пакет, в которых теперь нет необходимости. Пакет был установлен по зависимости, а зависимость удалена.
- `update` — обновить информацию о пакетах
- `upgrade` — обновить пакеты
- `full-upgrade` — может удалять пакеты для обновления системы в целом
- `edit-sources` — изменить `sources.list`

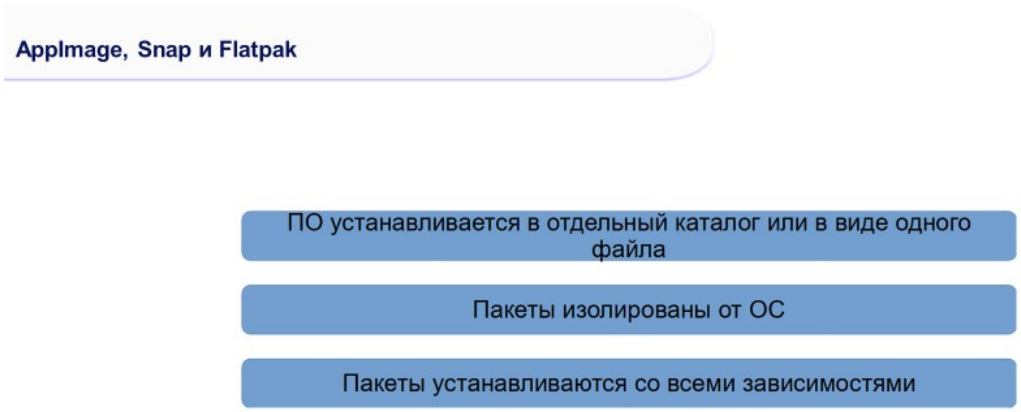
В специализированной команде `apt-get` больше возможностей по управлению пакетами, что удобно при использовании в сценариях

Наиболее важные опции команды `apt-get`:

- `-d` – команда `apt-get` только загрузит пакеты, но не будет устанавливать их.
- `-s` – команда не будет производить реальных действий, а только симулирует последствия планируемых действий, например, возникновение конфликтов.

- `-y` – отключение интерактивного режима работы программы, на все вопросы, которые обычно задает `apt-get` будет дан утвердительный ответ.
- `-u` – команда покажет обновленные пакеты и сведения об их состоянии.

6.4. AppImage, Snap и Flatpak



В традиционной системе управления программным обеспечением имеется «подводный камень». Проблема заключается в том, что бывает очень трудно или иногда не возможно установить программу не совместимую с данным конкретным дистрибутивом. Например вы хотите установить программу из исходных кодов, а библиотеки с нужной версией нет.

Проблему иногда можно решить с помощью виртуальных машин или контейнеров, но это не всегда правильный и удобный подход.

Чтобы преодолеть проблему не удовлетворенных зависимостей придумали несколько механизмов:

- `AppImage` — приложения получаем в виде единого файла. Каждое приложение самодостаточно: оно включает в себя все библиотеки, от которых зависит приложение. Стандарт AppImage 1.0 представлял собой iso-образ стандарта Rock Ridge (zisofs), включая в себя минимальный AppDir и небольшую библиотеку среды выполнения. Вторая версия может использовать другие файловые системы, такие как SquashFS.
- `Flatpak` — это утилита для развёртывания, управления пакетами и виртуализации для Linux. Предоставляет песочницу, в которой пользователи могут запускать приложения без влияния на основную систему. Приложения, использующие `Flatpak`, требуют

дополнительных разрешений на использование дискового пространства.

- `Snap` — проект с похожими идеями, что и в `Flatpak`. Разработан в Canonical. В отличие от `Flatpak` использует специальный демон — `snapd`.

Проекты основаны на схожих принципах: переносимость, простота использования, минимальное влияние на хостовую ОС.

В `Snap` и `Flatpak` используют принципы изоляции такие же как в контейнерах.

`Flatpak` поддерживает дополнительные репозитории пакетов.

`Flatpak` ориентирован в основном на графические пользовательские приложения.

В `Snap` доступно как прикладное, так и системное ПО.

Преимущества этих решений:

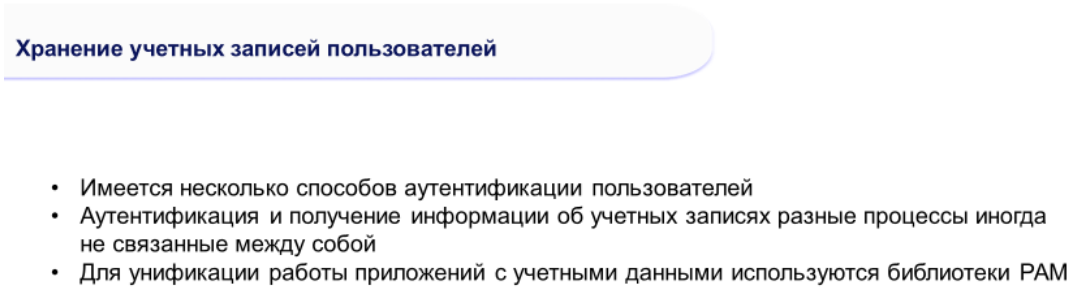
- Простота в использовании.
- Некоторая независимость от ОС.
- Можно устанавливать сразу несколько версий приложения.

Недостатки:

- Изоляция не полная.
- Пакеты занимают больше места на диске.
- Сборка пакета занимает больше времени и отнимает больше усилий.

Глава 7. Управление пользователями.

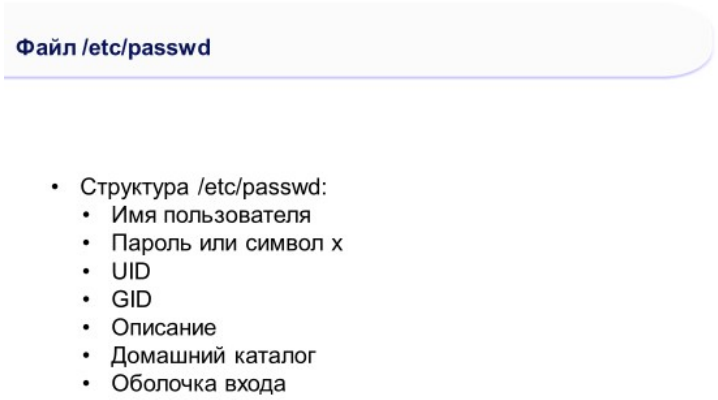
7.1. Хранение учетных записей пользователей.



В GNU/Linux процедура аутентификации пользователя при входе в сеанс может быть проведена разными способами. Вот некоторые из них:

- Традиционный способ, опирающийся на база данных учетных записей в файлах `/etc/passwd` и `/etc/shadow`.
- Аутентификация с помощью системы `Kerberos`.
- Аутентификация в `NIS/NIS+`.
- Аутентификация в `LDAP`.
- Использование специализированных систем аутентификации (например, TCB) и т.п.

Не смотря на наличие различных систем аутентификации наиболее простым, а следовательно, и наиболее распространенным способом является традиционный, использующий текстовые файлы учетных записей пользователей.



Файл `/etc/passwd` для аутентификации пользователей используется уже давно, а файл `/etc/shadow` стал использоваться с появлением системы теневых паролей.

Каждая запись в этих файлах соответствует конкретному пользователю системы.

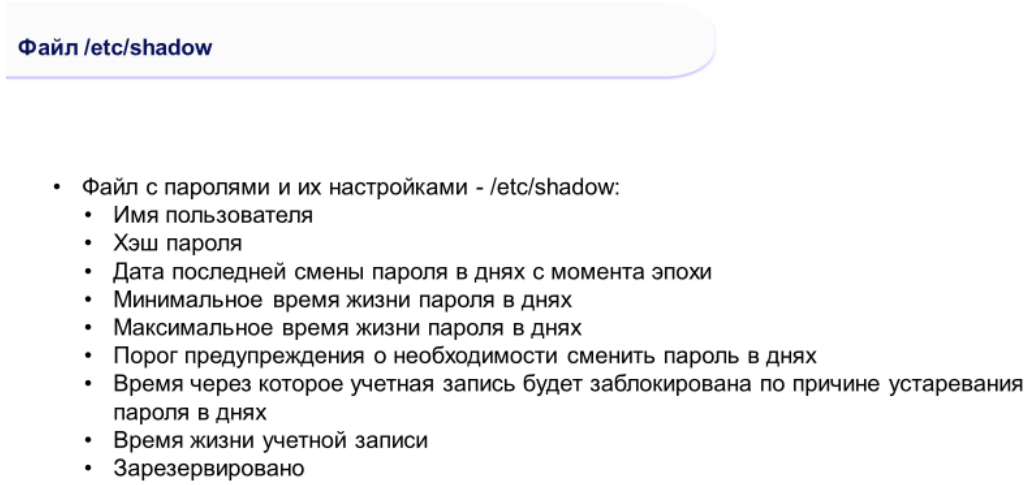
Поля записей разделены двоеточиями.

Структура записей в файле `/etc/passwd` следующая:

- Первое поле – имя пользователя.
- Второе поле содержит символ `x` если используется система теневых паролей `/etc/shadow`. Если эта система не используется, то во втором поле находится зашифрованный пароль пользователя.
- Третье поле – `UID` пользователя.
- Четвертое поле – `GID` пользователя.
- Пятое поле содержит необязательную справочную информацию о пользователе, например, его обычное (человеческое) имя.
- Шестое поле указывает домашний каталог пользователя.
- Седьмое поле соответствует имени исполняемого файла оболочки, запускаемой при входе в сеанс для этого пользователя.

Права доступа, устанавливаемые на файл `/etc/passwd` позволяют читать этот файл всем пользователям. Поэтому при хранении шифрованного пароля во втором поле этого файла представляет реальную угрозу

безопасности, так как любой злоумышленник, имеющий доступ к данной системе может воспользоваться программами подбора паролей для взлома системы.



Использование системы теневых паролей существенно снижает эту опасность, так как файл, где хранятся шифрованные пароли - `/etc/shadow` не позволяет его читать никому, кроме суперпользователя. Структура этого файла такова:

- Первое поле – имя пользователя.
- Второе поле – зашифрованный пароль.
- Третье поле – количество дней с 01 января 1970 г., прошедших с момента последней смены пароля.
- Четвертое поле – количество дней, которые должны пройти с момента последней смены пароля пользователя, прежде чем он сможет снова поменять пароль.
- Пятое поле – срок устаревания пароля в днях с момента его смены. До истечения этого срока пароль обязательно должен быть изменен.
- Шестое поле – время в днях до момента устаревания пароля, начиная с которого пользователь будет получать предупреждения о необходимости очередной смены пароля.
- Седьмое поле – период времени в днях с момента устаревания пароля, по прошествии которого учетная запись пользователя будет заблокирована по причине устаревания пароля.
- Восьмое поле устанавливает срок жизни учетной записи и предназначен для создания временных учетных записей. Оно содержит число дней с 01 января 1970 г., по прошествии которых учетная запись будет

заблокирована вне зависимости от состояния пароля пользователя.

- Девятое поле зарезервировано и в настоящее время не используется.

7.2. Регистрация, удаление и блокирование учетных записей пользователей.

Правами регистрации пользователей в системе обладает суперпользователь.

Для добавления учетной записи пользователя используется команда `useradd`. В качестве аргумента для этой команды должно быть указано имя пользователя.

Пример:

```
# useradd user1
# id user1
uid=504(user1) gid=504(user1) groups=504(user1)
```

Примечание: В этом примере в системе был зарегистрирован новый пользователь – user1. Для него была зарегистрирована его приватная группа пользователей – user1. Приватная группа пользователей состоит из единственного пользователя. Она является первичной группой для этого пользователя.

Регистрация пользователя приводит к появлению соответствующих записей в файлах `/etc/passwd` и `/etc/shadow`.

Пример:

```
# grep user1 /etc/passwd
user1:x:504:504::/home/user1:/bin/bash
# grep user1 /etc/shadow
user1:!!:12407::::::
```

Примечание: Также для этого пользователя был создан его домашний каталог:

```
# ls -a /home/user1
. .bash_logout .bashrc .i18n .mutt .rpmmacros Documents
.. .bash_profile .emacs .lpoptions .pinerc .xsession.d tmp
```

Создание приватной группы и домашнего каталога для пользователя характерно для Red Hat Linux и подобных дистрибутивов.

В других дистрибутивах может потребуется предварительно создать все группы, в которых должен участвовать пользователь, если они еще не зарегистрированы. Кроме этого создание домашнего каталога производится только при использовании опции `-m` команды `useradd`, либо при использовании опции `-d`, указывающей путь к домашнему каталогу.

В Red Hat создание приватной группы можно запретить, используя опцию `-n`. При этом для вновь зарегистрированных пользователей в системе будет установлена группа по умолчанию (см. ниже) в качестве первичной группы.

В Debian `useradd` создает приватную первичную группу для пользователя, но не создает домашний каталог для пользователя.

Отсутствие домашнего каталога может препятствовать входу пользователя в систему.

Настройки для команды `useradd` находятся в файле `/etc/default/useradd` и могут быть получены с помощью опции `-D` команды `useradd` .

Пример:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

Примечание: Выведенная информация командой useradd -D говорит о следующем:

1. `GROUP=100` - `GID` для вновь регистрируемых пользователей - 100 (для Red Hat эта настройка игнорируется при создании приватной группы пользователя);
2. `HOME=/home` - домашние каталоги для пользователей будут создаваться в каталоге `/home`;
3. `INACTIVE =-1` – блокирование учетной записи пользователя при устаревании его пароля не будет;
4. `SHELL=/bin/bash` – оболочка для вновь регистрируемых пользователей по умолчанию;
5. `SKEL=/etc/skel` – каталог “скелета”, из которого в домашние каталоги вновь регистрируемых пользователей копируются файлы, необходимые для каждого пользователя. Каталог `/etc/skel` обычно содержит файлы профиля для вновь регистрируемых пользователей и другие служебные файлы, которые копируются при регистрации пользователя в его домашний каталог.

Пример:

```
# ls -a /etc/skel
. .bash_logout .bashrc .mutt .xsession.d tmp
.. .bash_profile .lpoptions .rpmmacros Documents
```

Наиболее часто используемые опции команды `useradd`:

1. `-s` – указывает исполняемый файл оболочки по умолчанию;
2. `-d` – путь к домашнему каталогу пользователя;
3. `-m` – опция, указывающая на необходимость создать домашний каталог;
4. `-M` – не создавать домашний каталог;
5. `-k` – путь к альтернативному каталогу скелета;
6. `-u` – назначить `UID` пользователю;
7. `-g` – назначить `GID` (первичную группу) пользователю;
8. `-G` – список групп, в которых принимает участие пользователь (разделены запятыми);
9. `-e` – календарная дата, после которой учетная запись будет заблокирована (срок жизни учетной записи);
10. `-f` – количество дней, которое должно пройти после срока устаревания пароля до блокировки учетной записи.

Пример:

```
# useradd -M -g sinix -s /bin/nologin -e 01-01-2004 sinixuser

# id sinixuser
uid=505(sinixuser) gid=107(sinix) groups=107(sinix)

# grep sinixuser /etc/shadow
sinixuser:!!:12409:::::13326:
```

Примечание: Приведенная выше команда регистрирует пользователя без создания для него домашнего каталога (опция -M) с первичной группой sinix (опция -g). Для этого пользователя запрещен вход в сеанс, так как в качестве оболочки для этого пользователя указан файл /bin/nologin . Учетная запись пользователя будет заблокирована 1 января 2004 г. (опция -e).Последняя команда примера демонстрирует запись в файле теневых паролей для пользователя sinixuser. Восьмое поле записи содержит число дней с 1 января 1970 г. до дня, когда учетная запись пользователя будет заблокирована.

Если пользователь не имеет право входить в сеанс, то в качестве оболочки должен быть установлен один из следующих вариантов:

1. `/bin/false` – системная команда, всегда возвращающая код 1 (код ошибочного завершения);
2. `/dev/null` – специальный файл символьного устройства, при попытке запуска которого возникает ошибка;
3. `/sbin/nologin` – системная команда, возвращающая при запуске код ошибки и сообщение о невозможности входа в сеанс.

Вместо `useradd` можно использовать команду `adduser`. В RedHat разницы в командах нет, но в Debian это другая утилита, которая в интерактивном режиме запрашивает параметры создаваемого пользователя.

Если необходимо произвести некоторые изменения в учетной записи уже зарегистрированного пользователя, то для этого предназначена команда `usermod`.

Пример: смена оболочки по умолчанию для пользователя:

```
# usermod -s /bin/false sinixuser
```

Большая часть опций команд `useradd` и `usermod` совпадают.

Пример: для добавления новой группы, в которых участвует пользователь, можно использовать следующую команду:

```
# usermod -G "`id -G sinixuser | tr ' ' ','`,users" sinixuser

# id sinixuser
uid=505(sinixuser) gid=107(sinix) groups=107(sinix),100(users)
```

Примечание: Команда `id -G` выводит список групп, в которые входит пользователь, разделенных пробелами. Далее пробелы заменяются запятыми с помощью команды `tr`, так как список групп для команды `usermod` должен быть задан через запятую. К списку групп, в которых пользователь уже принимает участие, добавляется список новых групп, а далее полученный список подставляется в командную строку `usermod` с помощью командной подстановки.

Используя команду `usermod` можно также указать для пользователя его

новое имя с помощью опции `-l`.

Опции `-L` и `-U` позволяют, соответственно, блокировать и разблокировать возможность входа в сеанс для данного пользователя.

Для удаления учетной записи пользователя следует воспользоваться командой `userdel`.

Пример:

```
# userdel sinixuser
# id sinixuser
id: sinixuser: No such user
```

Перед удалением учетной записи пользователя необходимо решить, что делать с файлами пользователя, если таковые в системе имеются.

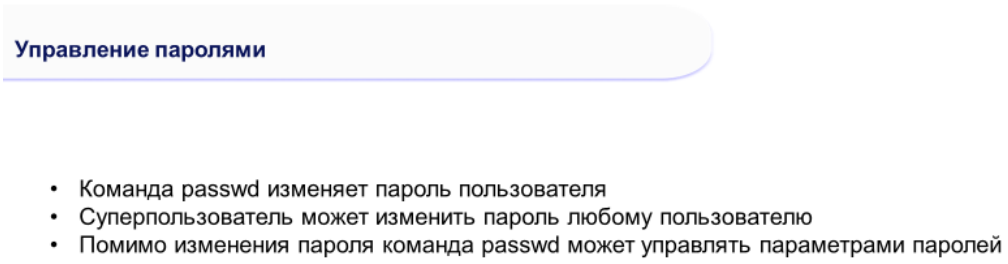
Сама команда `userdel` удаления файлов пользователя не производит. Поэтому все файлы пользователя, учетная запись которого подлежит удалению, должны быть найдены и либо удалены, либо сохранены в архив, либо переданы другому пользователю.

Пример:

```
#find / -user 1505 -exec rm -rf {} \;
```

Примечание: Здесь вместо имени пользователя использовался `UID`, поскольку пользователь с таким `UID` был уже удален.

7.3. Управление паролями.



Правила установки, использования и управления паролями являются важнейшей частью системной политики. Обычно они включают в себя, как минимум, следующее:

1. Определение категорий пользователей, которые имеют право самостоятельного выбора паролей с помощью команды `passwd`.
2. Правила выбора паролей и требования к их уровню сложности.
3. Сроки устаревания паролей.
4. Длительности периодов запрета на изменение паролей.

Четко сформулированная политика управления паролями значительно облегчает администрирование системы.

Для установки правила выбора паролей, их минимальной длины и требуемого уровня сложности, достаточно настроить модуль контроля паролей системы PAM для автоматической проверки соответствия выбираемого пользователем пароля системной политике.

Настройки PAM для команды `passwd` обычно находятся в файлах `/etc/pam.d/passwd` и, возможно, в `/etc/pam.d/system-auth`.

Команда `passwd` помимо изменения паролей предоставляет и другие возможности. Ниже приведен список наиболее часто применяемых опций команды `passwd` :

1. `-l` – блокирование учетной записи;
2. `-u` – разблокирование учетной записи;
3. `-S` – получение текущего состояния пароля;
4. `-d` – удалить пароль;
5. `-n` – установка периода запрета на смену пароля (минимальное время жизни пароля);
6. `-x` – установка максимального срока использования пароля;
7. `-w` – установка количества дней до момента устаревания пароля, начиная с которого пользователю будут выдаваться предупреждения о необходимости смены пароля;
8. `-i` – срок после устаревания пароля, по прошествии которого учетная запись блокируется.

Пример: блокирование учетной записи.

```
# id colobok
uid=503(colobok) gid=503(colobok) groups=503(colobok),22(cdrom)

# cat /etc/shadow
colobok:
$2a$08$Z4jZgPzM2GDVguFd4TRF3ubB.XWe4HHysgpYRwdogLj9WLlBXOJ:12316::
::::
# passwd -l colobok

# cat /etc/shadow
colobok:!
$2a$08$Z4jZgPzM2GDVguFd4TRF3ubB.XWe4HHysgpYRwdogLj9WLlBXOJ:12316::
::::
```

Примечание: После блокирования учетной записи в первой позиции второго поля файла /etc/shadow перед шифрованным паролем пользователя появляется знак восклицания. При разблокировании учетной записи он исчезает.

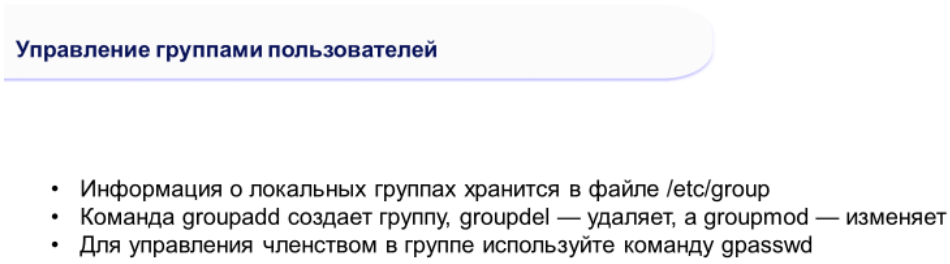
Обычно команда `chage`, которая управляет параметрами устаревания пароля, удобней чем `passwd`.

Пример: установка времени жизни учетной записи:

```
# chage -l test
Last password change : Feb 10, 2021
Password expires : never
Password inactive : never
```

```
Account expires : never
Minimum number of days between password change : 0
Maximum number of days between password change : 99999
Number of days of warning before password expires : 7

# chage -E "01 May 2021" test

# chage -l test
Last password change : Feb 10, 2021
Password expires : never
Password inactive : never
Account expires : May 01, 2021
Minimum number of days between password change : 0
Maximum number of days between password change : 99999
Number of days of warning before password expires : 7
```

7.4. Управление группами пользователей.



С помощью создания групп пользователей системный администратор может эффективно управлять деятельностью в системе целыми коллективами пользователей, предоставляя им разрешения на доступ к системным ресурсам.

Примечание: Каждый файл располагает в метаданных триадой бит, кодирующей права доступа для группы пользователей. Следовательно, изменяя членство пользователя в группах, администратор изменяет, таким образом, привилегии пользователя на доступ к различным файлам в системе, не меняя при этом права пользователя на принадлежащие ему файлы.

Информация о группах пользователей хранится в файле `/etc/group` в виде строк.

Формат записи: `name:password:GID:user(s)`, где:

1. Первое поле – имя группы.
2. Второе поле – пароль группы. Если он не используется, в этом поле ставится звездочка.
3. Третье поле - `GID` группы.
4. Четвертое поле содержит список пользователей, принадлежащих к данной группе, разделенных запятыми.

Для добавления новой группы необходимо воспользоваться командой `groupadd` , которая добавляет новую запись в файл `/etc/group` .

Пример:

```
# groupadd class
# grep class /etc/group
class:x:505:
```

Примечание: В этом примере добавлена новая группа class.

Пользователи для которых группа является первичной имеют информацию об этом в GID, который хранится в четвертом поле файла /etc/passwd.

Имена пользователей, входящих в группу, которая не является для них первичной, записываются через запятую в четвертом поле файла /etc/group.

Пример:

```
# grep sys /etc/group
sys:x:3:root,bin,adm
```

Примечание: В группу sys в данном примере входят пользователи root, bin и adm.

Для явного указания идентификатора группы необходимо воспользоваться опцией -g.

Пример:

```
# groupadd -g 512 project
```

Примечание: В этом примере создана новая группа пользователей project, имеющая GID 512.

Для удаления группы необходимо воспользоваться командой groupdel.

Пример:

```
# groupdel project
```

Примечание: В этом примере удалена группа пользователей project.

Группа пользователей может быть создана для работы над каким-либо проектом. В таком случае бывает удобно одного из пользователей сделать администратором группы и делегировать ему право добавлять уже

зарегистрированных в системе пользователей в эту группу и удалять их из группы при необходимости.

Для назначения администратора группы суперпользователю необходимо использовать команду `gpasswd -A`.

Пример:

```
# gpasswd -A ivanov developers
```

Примечание: Здесь администратором группы `developers` назначается пользователь `ivanov`.

Системный администратор или администратор группы может добавлять пользователей в группу с помощью команды `gpasswd -a`.

Пример:

```
# gpasswd -a marta developers
```

Примечание: Пользователь marta была добавлена в группу `developers` системным администратором.

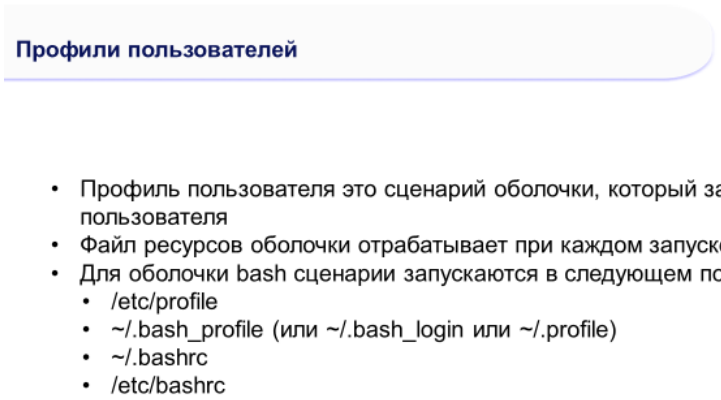
Удалить пользователя из группы можно командой `gpasswd -d`.

Пример:

```
$ gpasswd -a semko developers
```

Примечание: В этом примере администратор группы `developers` добавил в группу нового члена – пользователя `semko`.

7.5. Профили пользователей.



При входе пользователей в сеанс автоматически выполняются специальные файлы сценариев, называемые профилями пользователей.

Обычный подход к хранению настроек оболочки состоит в разделении настроек (профилей) на глобальный профиль (Master Profile) и пользовательские профили (Login Profiles).

Кроме профилей имеются еще и специальные файлы настроек оболочек (resource files), которые также являются сценариями оболочек.

Отличие профилей от файлов ресурсов состоит в том, что сценарии профилей исполняются единожды при входе пользователя в сеанс, а файлы ресурсов запускаются при каждом запуске оболочки.

Если оболочка Bash запущена интерактивно при входе пользователя в сеанс (то есть является оболочкой по умолчанию), то сначала исполняется общий для всех пользователей файл `/etc/profile` , а затем индивидуальный профиль пользователя, находящийся в его домашнем каталоге.

Для оболочки Bash индивидуальный профиль находится в файле, который может называться одним из следующих имен:

- `~/.bash_profile`
- `~/.bash_login`
- `~/.profile`

В файлах профилей чаще всего устанавливаются такие переменные

окружения, как:

- PATH - имена каталогов, в которых осуществляется поиск исполняемых файлов для запуска;
- TERM - тип терминала;
- USER - имя пользователя (устанавливается с помощью id -un);
- HOME - путь к домашнему каталогу пользователя;
- MAIL - путь к почтовому ящику пользователя;
- HOSTNAME - имя системы.

Переменные окружения, устанавливаемые в файлах профилей, должны быть экспортированы с помощью команды export.

Пример: к списку каталогов в переменной окружения PATH добавляется каталог bin, находящийся в домашнем каталоге пользователя:

```
PATH=$PATH:$HOME/bin
export PATH
```

Примечание: Имена каталогов, содержащихся в переменной PATH разделяются двоеточиями.

Помимо переменных окружения в файлах профиля часто устанавливается значение umask.

При необходимости исполнить файл профиля из командной строки следует использовать команду source .

Пример:

```
# source /etc/profile
```

Примечание: Эта команда является встроенной и выполняет в текущей оболочке команды из файла, указанного в качестве аргумента.

В противоположность профилям файл ресурсов оболочки ~/.bashrc выполняется только при интерактивном запуске оболочки Bash из командной строки, а не при входе в сеанс.

Для того, чтобы дополнительные настройки оболочки срабатывали не только при запуске оболочки из командной строки (то есть из уже запущенной оболочки), но и при запуске Bash по умолчанию при входе в сеанс, вызов инструкций в файле ~/.bashrc производится из пользовательского профиля.

Пример: Типичное содержимое файла пользовательского профиля таково:

```
$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi

# User specific environment and startup programs

BASH_ENV=$HOME/.bashrc
export BASH_ENV
```

Примечание: Здесь приведен пример содержимого файла пользовательского профиля, в котором проверяется наличие в домашнем каталоге пользователя файла ресурсов оболочки и, если он есть, содержимое его выполняется в контексте текущей оболочки. Это достигается с помощью так называемой inline подстановки - команды точка (.). Вызов . ~/.bashrc приводит к тому (обратите внимание на точку, с которой начинается команда), что переменные, псевдонимы и функции, определенные в файле ресурсов будут доступны в текущей оболочке. Inline подстановка всегда используется для передачи из одного файла сценария оболочки в другой скрипт переменных, псевдонимов и функций.

Переменная окружения `BASH_ENV` , определенная приведенном примере, предназначена для информирования оболочки, запускаемой не интерактивно (например, для выполнения сценария), что должны быть использованы ресурсы, определенные в файле, имя которого содержится в этой переменной.

Довольно часто в файле `/etc/profile` находится `inline` вызов общесистемного файла ресурсов `/etc/bashrc` (или `/etc/bash.bashrc`).

Примечание: Это не обязательно, но очень удобно, так как в этом файле можно определить, например, псевдонимы для команд, которыми часто пользуются различные пользователи системы, вместо определения этих псевдонимов в частных файлов ресурсов оболочки `~/.bashrc`.

Ниже приведен список действий, которые обычно выполняются автоматически при входе в сеанс Bash:

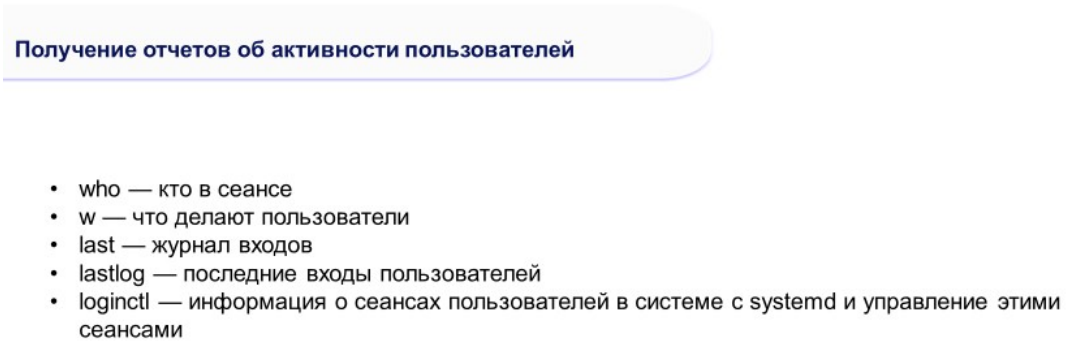
1. Исполняется общесистемный скрипт профиля `/etc/profile`.
2. Из `/etc/profile` проверяется наличие файла ресурсов оболочки `/etc/bashrc` и, при его наличии, он исполняется.
3. Выполняется пользовательский скрипт профиля в его домашнем каталоге

(например, `~/.bash_profile`).

4. В пользовательском профиле проверяется наличие в домашнем каталоге файла ресурсов оболочки `~/.bashrc` , и, при его наличии, он исполняется.

Примечание: При запуске оболочки из командной строки выполняется пункт 4 списка. В некоторых ОС из `~/.bashrc` производится запуск `/etc/bashrc`, если он еще не запускался.

7.6. Получение отчетов об активности пользователей.



Команда `who` позволяет получить список пользователей, находящихся в настоящее время в сеансе.

Информация об этом берется из специального двоичного файла `/var/run/utmp`.

Пример:

```
$ who
user1 :0 Jan 21 15:10
```

С помощью этой же команды, используя соответствующие опции, можно получать и другую информацию. Ниже приведены некоторые часто используемые опции команды `who` :

17. `-b` – время последней загрузки системы;
18. `-H` – печать заголовка;
19. `--login` – информация о системных процессах, контролирующих вход в сеанс;
20. `-q` – печатает имена всех пользователей в сеансе и их количество;
21. `-w` – текущий статус всех сеансов;
22. `-u` – подробная информация о сеансах;

23. `-a` – полная информация о статусе сеансов и процессов, контролирующих вход в сеанс.

Пример: приведенная ниже команда выведет информацию о сеансах пользователей:

```
$ who -uH
NAME LINE TIME IDLE PID COMMENT
user1 :0 Jan 21 15:10 ? 2014
root pts/1 Jan 21 15:21 . 2611 (localhost)
```

Просматривать и управлять сеансами пользователей в `systemd` можно с помощью команды `loginctl`.

Пример:

```
# loginctl list-sessions
SESSION UID USER SEAT TTY
3 1000 admuser seat0 tty1
4 0 root seat0 tty2
6 1000 admuser

3 sessions listed.

# loginctl session-status 6
6 - admuser (1000)
Since: Sun 2021-02-07 21:10:01 +05; 3 days ago
Leader: 1526 (sshd)
Remote: 192.168.101.11
Service: sshd; type tty; class user
State: active
Unit: session-6.scope
├─ 1526 sshd: admuser [priv]
├─ 1528 sshd: admuser@pts/0
├─ 1529 -bash
├─347639 sudo su -
├─347640 su -
├─347641 -bash
├─348247 loginctl session-status 6
└─348248 less

Feb 10 21:03:51 cent8-stream sudo[347637]: admuser : TTY=pts/0 ;
PWD=/home/adm>
Feb 10 21:03:51 cent8-stream sudo[347637]:
pam_systemd(sudo:session): Cannot cr>
Feb 10 21:03:51 cent8-stream sudo[347637]: pam_unix(sudo:session):
```

```
session open>
Feb 10 21:03:51 cent8-stream sudo[347637]: pam_unix(sudo:session):
session clos>
Feb 10 21:03:53 cent8-stream sudo[347639]: admuser : TTY=pts/0 ;
PWD=/home/adm>
Feb 10 21:03:53 cent8-stream sudo[347639]:
pam_systemd(sudo:session): Cannot cr>
Feb 10 21:03:53 cent8-stream sudo[347639]: pam_unix(sudo:session):
session open>

# loginctl terminate-session 6
Connection to 192.168.101.180 closed by remote host.
Connection to 192.168.101.180 closed.
```

Для получения отчета о сеансах пользователей, которые уже завершились, необходимо воспользоваться информацией, сохраняемой в файле `/var/log/wtmp`.

Этот бинарный файл имеет ту же структуру, что и `/var/run/utmp`, поэтому его содержимое можно отобразить, указав его в качестве аргумента команды `who`. Однако, для этого предназначена специальная команда `last`.

Пример:

```
$ last
root pts/1 localhost Wed Jan 21 15:21 still logged in
user1 :0 Wed Jan 21 15:10 gone - no logout
reboot system boot 2.4.20-alt5-up Wed Jan 21 15:09 (00:27)
postgres ??? localhost Wed Jan 21 15:09 - 15:09 (00:00)
root pts/1 localhost Wed Jan 21 14:52 - down (00:10)
```

Примечание: Эта команда выводит информацию об открытых и законченных сеансах в обратном хронологическом порядке.

В файл `/var/log/btmp` записывается информация о неудачных попытках входа. Для его просмотра используется команда `last`.

Пример:

```
# last -f /var/log/btmp
admuser ssh:notty 192.168.101.11 Tue Feb 9 21:48 gone - no logout

btmp begins Tue Feb 9 21:48:07 2021
```

Имеется также стандартный файл журнала `/var/log/lastlog`, в

котором также в бинарном виде хранится информация о последних входах в сеанс. Для получения информации, находящейся в этом файле, требуется использовать команду `lastlog` .

Глава 8. Логическая структура файловой системы. Работа с файлами и каталогами.

8.1. Система файлов и каталогов.

- Единое дерево
- Корневой каталог - /
- Файл – основополагающее понятие. Состоит из: имени, метаданных, блоков данных
- Основа дерева – каталоги (специальные файлы)
- Пути: абсолютные и относительные
- Скрытые файлы – начинаются с точки

Логически файловая структура в GNU/Linux организована в виде единой иерархии, напоминающей перевернутое дерево с корнем наверху.

Древовидная структура организована с помощью каталогов, содержащих файлы и подкаталоги.

Каждый каталог может иметь множество подкаталогов, но у каждого подкаталога имеется только один родительский каталог.

На каких физических носителях не находились бы файлы, в GNU/Linux они всегда находятся на одной из ветвей единой древовидной файловой структуры.

Вершиной файловой структуры является корневой каталог (root directory).

Имя корневого каталога: /.

У корневого каталога нет родительского каталога, вернее он сам является для себя родительским.

Пример: Команда, приведенная ниже, выведет содержимое корневого каталога.

```
$ ls /
bin boot dev etc home lib mnt opt proc root sbin swap tmp usr var
```

Примечание: Команда `ls` выводит содержимое каталога, имя которого указано в качестве аргумента. Здесь аргумент – имя корневого каталога `/`.

Файлы в GNU/Linux являются основополагающими объектами, поскольку вся работа с данными, устройствами компьютера, процессами и прочим обеспечивается посредством файлов.

Файл состоит из трех компонентов:

- Имя.
- Inode или метаданные. В метаданных хранятся свойства файла такие как права доступа, размер, указатели на блоки данных и пр.
- Данные (блоки данных). Не обязательный элемент. Например пустой файл не имеет блоков данных.

Обычные файлы (plain files, ordinary files или regular files) обеспечивают хранение данных в компьютере. Они представляют собой именованный блок данных, находящихся в устройстве хранения. Ядро ОС не интерпретирует содержание файлов данного типа.

Древовидная структура образуется за счет использования каталогов, которые могут содержать файлы и другие каталоги.

Каталоги являются особым типом файлов, предназначенным для поддержки иерархической структуры файловой системы. Содержимое файла каталога - это перечень имен файлов, содержащихся в каталоге.

Примечание: Таким образом, для доступа к файлу необходимо пройти один или несколько каталогов.

Последовательность имен каталогов, которые необходимо пройти от корневого каталога для доступа к файлу, называется путем (path).

Для разделения имен вложенных каталогов применяется символ /.

Пример: Путь к файлу `/etc/hostname` начинается, естественно, с корневого каталога, далее путь проходит в каталог `/etc`, в котором и находится указанный файл.

Имя файла может содержать любые символы кроме символов `/` и `\0` (null), а длина имени файла не должна превышать 255 символов.

Следует давать файлам осмысленные имена и избегать излишнего использования метасимволов в именах файлов.

Заглавные и строчные буквы различаются (case sensitive).

Пример: Имена файлов TheFile и thefile относятся к различным файлам.

В различных каталогах могут находиться различные файлы с одинаковыми именами.

Для однозначной идентификации файла необходимо применять полное или абсолютное имя файла. Оно состоит из пути (path) к нему в дереве каталогов и собственно имени файла.

Пути бывают двух типов:

1. Абсолютные – те, которые начинаются с символа косая черта / - корневого каталога. Путь доступа в таких именах начинается от корневого каталога.
2. Относительные - не начинаются с косой черты и, следовательно, показывают путь доступа к файлам относительно текущего каталога.

Имя файла может содержать точки . . В GNU/Linux, в отличие, например, от MS DOS, никакого особого значения точки в именах файлов не имеют.

Часть имени файла, находящуюся после точки, называется суффиксом (расширением) имени файла.

Суффиксы сообщают пользователю о том, какого типа (характера) информация хранится в файле.

Пример: Файл `myarch.gz` является архивом, сжатым утилитой `gzip`. Имя файла может содержать более одного расширения: `tarball.tar.gz`.

Файлы, у которых точка является первым символом в имени, являются скрытыми и командой `ls` не выводятся. Или, что более точно, файловые команды их пропускают, если имена этих файлов не указанны явно.

Список всех файлов (в том числе и скрытых) можно получить, пользуясь

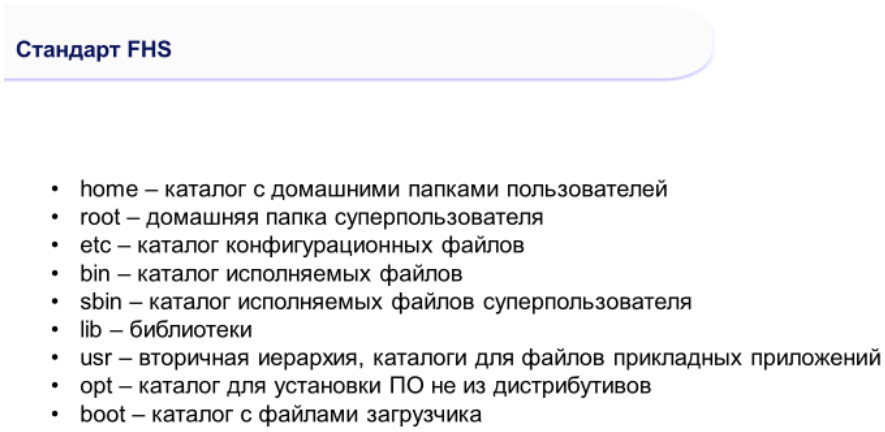
командой `ls` с опцией `-a` (`--all`) или опцией `-A`:

Пример:

```
$ > .hidden
$ ls
$ ls -a
. .. .hidden
$ ls -A
.hidden
```

Примечание: В этом примере команда `ls` не обнаружила никаких файлов в текущем каталоге. Тем не менее, этот каталог содержит файл `.hidden`, являющийся скрытым, так как его имя начинается с точки. Этот файл был создан в каталоге с помощью команды `> .hidden`. Для вывода списка всех файлов используется команда `ls -a`, в том числе и скрытых, в текущем каталоге. Эта команда выводит имя файла `.hidden`. Помимо него выведены еще два имени файлов – . (точка), то есть имя текущего каталога, и .. , являющееся именем родительского каталога. Команда `ls -A` не отобразила имена текущего и родительского каталога, а скрытый файл был ей показан.

8.2. Стандарт FHS

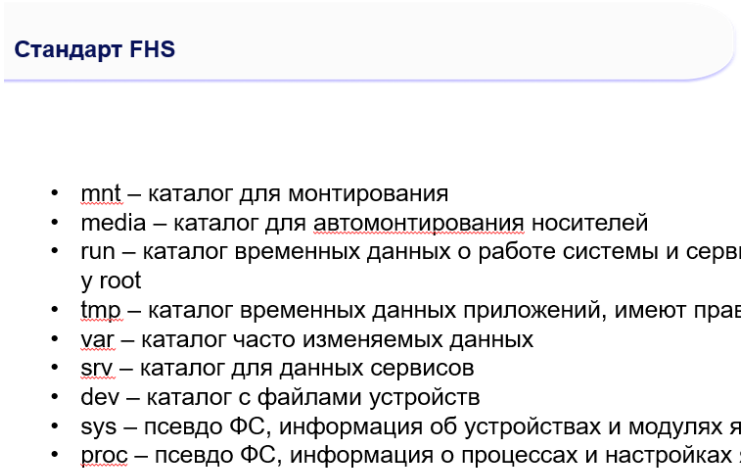


FHS (англ. Filesystem Hierarchy Standard, «стандарт иерархии файловой системы») — стандарт, унифицирующий местонахождение файлов и каталогов с общим назначением в файловой системе UNIX. На данный момент большинство UNIX-подобных систем в той или иной степени следует этим правилам.

Просмотреть подробную информацию о нем можно на сайте https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html

В корневом каталоге не принято хранить пользовательские файлы. Чаще всего в корневом каталоге содержатся исключительно подкаталоги.

Файлы пользователей принято хранить в их домашних каталогах, системные файлы хранятся в специальных каталогах и так далее.



FHS – определяет распределение файлов в системе согласно их предназначения. Так FHS отделяет следующие критерии расположения файлов:

1. часто изменяемые vs относительно статичные данные;
2. данные пользователя vs файлы операционной системы;
3. локальные vs предоставляемые в сетевой доступ;
4. исполняемые файлы vs файлов с данными.

8.3. Получение списков файлов и каталогов.

Текущим каталогом называют каталог, в котором будет производиться работа файловых команд без указания пути (абсолютного или относительного) к файлам.

Если в качестве аргумента файловой команде указано имя файла без пути к нему, то действие команды будет применено к файлу в текущем каталоге.

Команда `pwd` выводит полное имя текущего каталога

Пример:

```
$ pwd
/home/user1
```

Примечание: В этом примере пользователь user1 вывела имя текущего каталога с помощью команды pwd. Имя текущего каталога: /home/user1 – это (вероятнее всего, но совсем не обязательно) домашний каталог пользователя user1.

Для вывода списка файлов и подкаталогов текущего каталога можно использовать команду `ls`.

Пример:

```
$ ls
archive.gz referat.txt text.txt
```

Примечание: Команда ls вывела имена файлов, содержащихся в текущем каталоге: archive.gz, referat.txt и text.txt.

Для вывода содержимого каталога, не являющегося текущим, необходимо указать его имя в качестве аргумента:

Пример:

```
#ls /home/user1
archive.gz referat.txt text.txt
```

Примечание: В этом примере суперпользователь вывел содержимое домашнего каталога пользователя user1, указав его полное имя. То, что команду выполнил суперпользователь указывает решетка # в качестве символа приглашения. При обычных настройках системы безопасности только суперпользователь имеет право просматривать содержимое чужих домашних каталогов.

К текущему каталогу можно обращаться по имени . (точка). Команда ls действует аналогично команде `ls ./` или `ls ..`

Для надежности при указании относительных путей необходимо ставить `./` в начале пути.

Пример: ./myfile – файл, находящийся в текущем каталоге.

Имя `..` соответствует родительскому каталогу текущего каталога.

Пример: для каталога `/home/user1` родительским каталогом является каталог `/home`. Таким образом, если текущим каталогом является `/home/user1`, то команда `ls ..` выведет содержимое каталога `/home`.

Обычно при регистрации в системе нового пользователя ему назначается его домашний каталог, в котором он может хранить личные файлы.

При входе пользователей в систему текущими обычно становятся их домашние каталоги.

Имена домашних каталогов чаще всего совпадают с именами пользователей – владельцев этих каталогов.

Стандартное место для размещения домашних каталогов пользователей - каталог `/home`.

Пример: домашний каталог пользователя `user1` - `/home/user1`.

При пользовании оболочкой Bash существует короткий путь для указания имени домашнего каталога: имя ~ указывает на домашний каталог пользователя, вошедшего в систему, а `~имя_пользователя` – на домашний каталог указанного пользователя.

Пример: команда, приведенная ниже, выведет содержимое домашнего каталога пользователя user1:

```
# ls ~user1
archive.gz referat.txt text.txt
```

Примечание: В этом примере суперпользователь воспользовался более удобным путем обращения к домашнему каталогу пользователя user1 вместо указания его полного имени /home/user1.

При входе в сеанс имя домашнего каталога пользователя сохраняется в переменной окружения `HOME`.

Примечание: Поэтому команда ls $HOME, выполненная каким-либо пользователем, выведет содержимое его домашнего каталога.

Для получения подробных данных о выводимых командой `ls` файлах необходимо воспользоваться опцией `-l`

Пример:

```
$ ls -l /proc/sys
итого 0
dr-xr-xr-x 2 root root 0 Окт 7 06:57 abi
dr-xr-xr-x 2 root root 0 Окт 7 06:57 debug
dr-xr-xr-x 4 root root 0 Окт 7 06:57 dev
dr-xr-xr-x 4 root root 0 Окт 7 06:57 fs
dr-xr-xr-x 3 root root 0 Окт 7 06:57 kernel
dr-xr-xr-x 9 root root 0 Окт 7 06:57 net
dr-xr-xr-x 2 root root 0 Окт 7 06:57 proc
dr-xr-xr-x 2 root root 0 Окт 7 06:57 vm
```

Примечание: В первом столбце выводится тип файла, далее права доступа к файлу, количество имен файла (жестких связей), владелец файла, первичная группа владельца, размер файла, дата изменения и имя файла. Права владения и права доступа будут рассмотрены ниже.

В первом столбце листинга команды `ls -l` выводятся типы файлов, поддерживаемых ядром ОС:

1. `-` - обычные файлы.
2. `d` – каталоги.
3. `l` – символические ссылки (содержат указатели на другие файлы).
4. `b` – блочные устройства (специальные файлы, предназначенные для обращения к устройствам, информация на которые записывается и считывается оттуда блоками, например, флоппи диск).
5. `c` - символьные устройства (специальные файлы, предназначенные для ввода – вывода с неформатированных устройства, таких, как терминал или мышь, обмен с которыми производится посимвольно).
6. `p` – именованный канал (PIPE или FIFO, они являются одним из вариантов организации межпроцессного взаимодействия).
7. `s` – сокеты (socket, предназначенные для организации сетевого межпроцессного взаимодействия).

Другая часто используемая опция команды `ls` – это опция `-F`. При использовании этой опции после имен каталогов выводится `/`, после имен исполняемых файлов - `*`, после символьных ссылок - `@` .

Пример:

```
$ ls -F ~
Desktop/ intro.txt scr1.sh*
```

*Примечание: Команда ls с опцией F вывела содержимое домашнего каталога пользователя с использованием символов подсказки. Здесь Desktop – каталог, так как после его имени выводится знак /. Файл intro.txt – обычный файл. А скрипт scr1.sh является исполняемым файлом, так как после его имени выведен символ *.*

Для получения информации собственно о каталогах, а не о файлах, содержащихся в них необходимо воспользоваться опцией `-d` команды `ls` .

Чаще всего опция `-d` команды `ls` используется для вывода информации о каталоге в подробном формате, то есть совместно с опцией `-l`:

Пример:

```
$ ls -ld /etc
drwxr-xr-x 87 root root 6064 Окт 7 06:16 /etc
```

Примечание: В этом примере получена подробная информация о каталоге /etc. Если бы опция -d отсутствовала, то была бы получена информация не о каталоге, а о файлах, содержащихся в нем.

8.4. Перемещение по дереву каталогов.

Команда `cd` предназначена для смены текущего каталога.

Если она вызвана без аргументов, то текущим становится домашний каталог пользователя.

Пример:

```
$ pwd
/home/user1/books/poetry/Lermontov
$ cd
$ pwd
/home/user1
```

Примечание: В первой строке примера была дана команда вывести имя текущего каталога:/home/user1/books/poetry/Lermontov. Далее была выполнена команда cd без аргументов. Выполненная затем команда pwd вывела имя текущего каталога: /home/user1. То есть команда cd сделала текущим домашний каталог пользователя.

Для перехода в каталог `dir_name` необходимо вызвать команду `cd` с аргументом `dir_name`

Пример:

```
$ pwd
/home/user1
```

```
$ cd /etc
$ pwd
/etc
```

Примечание: В данном примере команда cd /etc сделала текущим каталог /etc.

8.5. Создание и удаление файлов и каталогов.

- touch
- "пустой ввод" >
- rm
- mkdir
- rmdir

Команда `touch file` создает пустой файл с именем `file`

Пример:

```
$ ls
article1.txt article2.txt
$ touch empty
$ ls
article1.txt article2.txt empty
```

Примечание: В этом примере создан пустой файл empty, имя которого выведено последней командой ls среди других имен файлов, находившихся в текущем каталоге до его создания. Преимуществом команды touch является возможность создания сразу многих файлов, если их имена указать в качестве аргументов.

Если в качестве аргумента команды `touch` указан файл, который уже существует, то у этого файла в результате выполнения команды `touch` будет изменена дата модификации.

Пример:

```
$ ls
$ touch f1
$ ls -l
итого 0
```

```
-rw-r--r-- 1 user1 user1 0 Окт 7 08:46 f1
$ touch f1 f2 f3
$ ls -l
итого 0
-rw-r--r-- 1 user1 user1 0 Окт 7 08:47 f1
-rw-r--r-- 1 user1 user1 0 Окт 7 08:47 f2
-rw-r--r-- 1 user1 user1 0 Окт 7 08:47 f3
```

Примечание: Файл f1 был создан командой touch. Затем эта же команда была вызвана с аргументами f1 f2 f3, в результате чего файлы f2 и f3 были созданы, а у файла f1 была изменена дата модификации. Содержимое файла f1 после этого не изменилось.

Более распространенный способ создать пустой файл заключается в использовании перенаправления "пустого вывода" в файл с помощью команды `> newfile`. Если в файле были данные, то они удаляются.

Для удаления файла необходимо воспользоваться командой rm

Пример:

```
$ ls
f1 f2 f3
$ rm -f f1 f2
$ rm -i f3
rm: удалить пустой обычный файл `f3'? y
$ ls
$
```

Примечание: Исходно в текущем каталоге находились три файла f1, f2 и f3. Первые два из них были удалены командой rm -f. Эта команда удаляет файлы без каких-либо дополнительных запросов, так как используется опция -f. Наоборот, если необходимо выводить запрос на удаление каждого файла, указанного в качестве аргумента команды rm, требуется использовать опцию -i.

Команда `rm`, вызванная без опций не задает никаких вопросов по поводу необходимости удаления файлов (по умолчанию действует опция `-f`)

При использовании шаблонов подстановки в качестве аргументов команды `rm` настоятельно рекомендуется предварительно проверить шаблон командой `ls`.

Пример:

```
$ touch file{1,2,3}
$ ls
file1 file2 file3
```

```
$ ls f*[12]
file1 file2
$ rm f*[12]
$
```

Примечание: В этом примере продемонстрирована предварительная проверка шаблона с помощью ls перед вызовом rm.

Также в целях безопасности настоятельно рекомендуется использовать псевдоним для команды rm, активизирующий ее интерактивный режим (опция `-i`) работы по умолчанию.

Пример:

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

Примечание: Приведенный в примере псевдоним следует разместить в файле профиля пользователя .bash_profile, либо в .bashrc. Использование его включит интерактивный режим работы команды rm по умолчанию. Важно понимать, что затруднительно, а часто просто невозможно, восстановить удаленный файл.

Если необходимо рекурсивно удалить каталог со всем его содержимым, необходимо использовать команду `rm` с опцией `-r`

Пример:

```
$ ls -R d1
d1:
direc1 file3

d1/direc1:
Remember
$ rm -rf d1
$ ls -R d1
ls: d1: No such file or directory
```

Примечание: Каталог d1 содержал исходно файлы и подкаталоги. После его удаления командой rm -rf он исчез со всем его содержимым.

Командой `rm -rf` следует пользоваться с особой осторожностью, так как установленная опция `-f` запрещает выводить предупреждающие сообщения об удалении файлов.

Примечание: Рекомендуется перед использованием такой команды переходить в каталог, в котором находится подкаталог, подлежащий удалению, и в качестве аргумента

указывать команде rm -rf относительное имя каталога, подлежащего удалению. Этого правила следует придерживаться обязательно при работе в сеансе суперпользователя, так как он легко может повредить и даже полностью удалить всю файловую систему в результате неосторожного вызова этой команды.

Команда `mkdir dir` создает каталог с именем `dir`:

Пример:

```
$ mkdir dir1
$ cd dir1
$ ls
$ mkdir -p mydir/mydir/mydir
$ ls -R mydir
mydir:
mydir

mydir/mydir:
mydir

mydir/mydir/mydir:
```

Примечание: Приведенный пример демонстрирует как с помощью команды mkdir был создан каталог dir1. Далее показано, что пользователь сделал вновь созданный каталог текущим с помощью команды cd. А затем, используя ключ -p команды mkdir , пользователь создал целую ветвь вложенных каталогов mydir/mydir/mydir. То есть ключ -p позволяет указывать для создания целый путь.

Команда `rmdir` позволяет удалять каталоги, если они пустые.

Пример:

```
$ mkdir emptyDir
$ ls -FR
.:
emptyDir/ mydir/

./emptyDir:

./mydir:
mydir/

./mydir/mydir:
mydir/

./mydir/mydir/mydir:
```

```
$ rmdir *
rmdir: `mydir': Directory not empty
```

Примечание: В этом примере был создан еще один каталог emptyDir. Затем пользователь попытался удалить оба каталога, однако, удален был лишь каталог пустой каталог emptyDir. Каталог mydir был оставлен, поскольку он не пустой.

Команда `rmdir -p` способна удалить ветвь пустых каталогов. Если в пути некоторый каталог окажется не пуст, то будут удалены все пустые каталоги в пути до первого не пустого каталога.

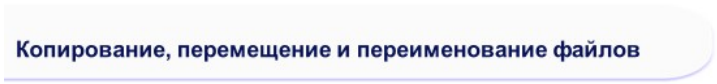
Пример:

```
$ > mydir/f1
$ rmdir -p mydir/mydir/mydir
rmdir: `mydir': Directory not empty
$ ls -FR
.:
mydir/

./mydir:
f1
```

Примечание: В данном случае два последних в пути каталога `mydir` были удалены, так как они пустые. Первый в пути `mydir` каталог содержит файл, созданный командой `> mydir/f1`, поэтому он не был удален.

Команда `rm -rf mydir` удалит каталог со всем его содержимым.

8.6. Копирование, перемещение и переименование файлов.



- cp
- mv

Команда `cp` применяется для копирования:

1. команда `cp srcFile tagFile` копирует `srcFile` в `tagFile`;
2. команда `cp file1 file2 fileN dir` копирует указанные файлы `file1 file2 fileN` в каталог `dir`;
3. команда `cp -r dir1 dir2` копирует каталог `dir1` в каталог `dir2` рекурсивно, делая в каталоге `dir2` подкаталог `dir1` с копиями всех файлов, содержащихся в нем.

Пример:

```
$ ls -FR
.:
mydir/

./mydir:
f1
$ cp mydir/f{1,2}
$ ls -FR
.:
mydir/

./mydir:
f1 f2
```

Примечание: С помощью команды cp mydir/f{1,2} создается копия файла f1 с именем f2 в том же каталоге mydir, где находится исходный файл. Команда cp mydir/f{1,2} эквивалентна команде cp mydir/f1 mydir/f2 , однако она значительно короче.

Пример: копируем каталог mydir со всем содержимым в каталог /tmp.

```
$ ls
mydir
$ cp -r mydir/ /tmp/
$ ls -R /tmp/mydir
/tmp/mydir:
f1 f2
```

Примечание: Пример демонстрирует, что копия каталога mydir является подкаталогом /tmp.

Команда `mv` используется для перемещения:

1. команда `mv oldName newName` переименовывает `oldName` в `newName` ;
2. команда `mv file1 file2 fileN dir` перемещает заданные файлы в каталог `dir` ;
3. команда `mv oldName newName` переименовывает каталог `oldName` в `newName` .

Пример: Переместим каталог /tmp/mydir в домашний каталог.

```
$ mv /tmp/mydir/ ~
$ ls -R ~/mydir
/home/user1/mydir:
f1 f2
```

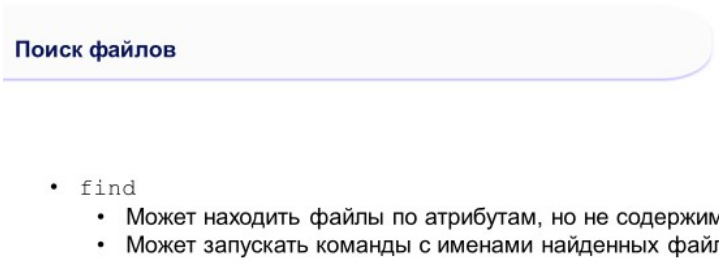
Примечание: Команда mv работает с каталогами точно так же, как и с файлами, то есть для перемещения каталога не надо указывать дополнительных опций.

Пример: Ниже приведен пример, в котором перемещению подвергаются сразу несколько файлов. Для указания имен этих файлов используются символы подстановки.

```
$ mv ~/mydir/f* .
$ ls -R
.:
f1 f2 mydir
./mydir:
f1 f2
```

Примечание: Здесь два файла f1 и f2 из каталога ~/mydir/ перемещены в текущий каталог.

8.7. Поиск файлов.



Команда `find` позволяет производить поиск файлов по указанным критериям.

Формат команды:

```
find место_поиска критерии действия
```

где `место_поиска` – каталог, начиная с которого будет произведен поиск. Поиск также производится и в подкаталогах указанных в качестве мест поиска каталогов.

Критериями поиска могут быть любые атрибуты файла, например, имя файла, его размер, владелец файла, тип, даты доступа, модификации и прочее.

Действие по умолчанию имена найденных файлов выводятся на экране.

Примечание: Сама команда find не обеспечивает поиска файла по его содержимому, но ее можно использовать вместе с утилитой grep

Команда `find` по историческим причинам придерживается нестандартного формата командной строки, в котором после тире указывается длинная опция с дополнительным аргументом.

Наиболее часто используют следующие опции:

1. `-name` – поиск по имени файла или строке в его имени;
2. `-iname` – то же с игнорированием регистра;

3. `-type` – поиск по типу файла;
4. `-size` – для поиска по размеру или диапазону возможных размеров файла;
5. `-perm` – поиск по правам доступа;
6. `-user` и `-group` – по принадлежности файла.

Пример: В текущем каталоге требуется найти все файлы, имя которых начинается со строки d1.

```
$ find . -name "d1*"
./d1
```

Примечание: Обратите внимание, что если критерий поиска по подстроке в имени файла использует шаблоны подстановки, то такой шаблон должен быть экранирован кавычками. В противном случае до исполнения команды Bash заменит шаблоны на имена файлов, подходящие им, и команда будет работать неверно.

Пример: Можно ужесточить критерий поиска, потребовав от предыдущей команды отыскивать только каталоги с заданным именем:

```
$ find . -name "d1*" -type d -ls
12042 0 drwxr-xr-x 2 aberes aberes 192 Окт 7 21:58 ./d1
```

Примечание: Более того, в последнем примере использована специальная настройка для команды find (такой формат работает только с GNU версией этой команды), позволяющая отображать найденные файлы в формате, подобном ls -l.

Если критерии поиска необходимо объединить по логическому условию ИЛИ, то необходимо использовать `-o`.

Пример:

```
$ find . -name "d1*" -o -empty
./d1
./d1/*
./d1/s
./d1/f1
./d1/f2
./d1/s1
./d1/s2
```

Примечание: В этом примере продемонстрирован поиск по двум критериям, объединенным условием ИЛИ. Первый критерий – поиск файлов, начинающихся со строки

d1, а второй – поиск пустых файлов (-empty).

Для поиска файлов определенного типа необходимо использовать критерий `-type` тип , где тип один из:

1. `b` – файл блочного устройства;
2. `c` – файл символьного устройства;
3. `d` – каталог;
4. `f` – обычный файл;
5. `p` – именованный канал;
6. `s` – сокет;
7. `l` – символьная ссылка.

Имеется возможность исполнять команды с найденными find файлами. Для этого необходимо использовать модификатор -exec

Пример:

```
$find ~ -empty -exec rm –f {} \;
```

Примечание: В этом примере производится поиск и удаление всех пустых файлов, начиная с домашнего каталога.

Смысл конструкции команда `{} \;` состоит в следующем: фигурные скобки будут замены именами найденных файлов, которые и будут аргументами команды :

```
команда файл1
команда файл2
…
команда файлN
```

Примечание: Символ обратной косой черты \ здесь применен для экранирования символа точки с запятой от возможной неправильной интерпретации Bash. Символ точка с запятой показывает окончание командной строки.

8.8. Определение типа файлов.

Для определения характера содержимого (типа) файла используется команда `file`, пытающаяся установить тип файла, используя так называемые магические числа (`magic numbers`)

Пример:

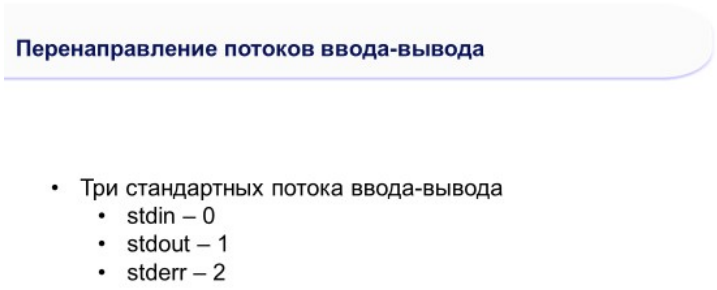
```
$ file tab_d
tab_d: ASCII text
```

Примечание: Файл tab_d, как определила команда file, является текстовым файлом, содержащим текст в формате ASCII.

Перед выводом на экран незнакомого вам файла рекомендуется узнать его тип с помощью команды `file`. В противном случае на экран может быть выведен бинарный файл и настройки терминала могут быть испорчены.

Глава 9. Текстовые файлы и потоки.

9.1. Перенаправление потоков ввода-вывода.



Если процесс требует чтения или записи в файл, то этот файл должен быть сначала открыт, т.е. создан поток (stream) данных.

Процедура открытия файла создает структуру в ядре, называемую файловым дескриптором.

Все файловые дескрипторы пронумерованы.

С любым пользовательским процессом при его создании связываются три файловых дескриптора (потока):

1. стандартный поток ввода (`stdin`), файловый дескриптор которого – 0;
2. стандартный поток вывода (`stdout`), файловый дескриптор – 1;
3. стандартный поток вывода ошибок (`stderr`), файловый дескриптор – 2.

Примечание: Поток ввода открыт на чтение, а потоки вывода и ошибок – на запись. Обычно по умолчанию стандартный поток ввода связан с клавиатурой, а стандартные потоки вывода и ошибок связаны с дисплеем.

Оболочка Bash позволяет перенаправить стандартные потоки в файлы, для чего используются следующие операторы:

1. `< имя_файла` или `0< имя_файла` – перенаправление стандартного потока ввода;
2. `> имя_файла` или `1> имя_файла` – перенаправление стандартного потока вывода;
3. `2> имя_файла` – перенаправление стандартного потока вывода ошибок

Если файла, в который производится запись не существует, то такой файл создается

Пример: с помощью такой команды можно отправить электронное письмо с текстом, содержащемся в файле letter :

```
$ mail -s 'Pismo' user1 < letter
```

Примечание: Эта команда направит электронное письмо пользователю user1. Тема письма (Subject), указана после опции -s, а текст письма передан через стандартный поток ввода из файла letter.

Пример: Следующая команда использует перенаправление стандартного потока вывода в файл ls.txt :

```
$ ls > ls.txt
```

Примечание: В файле ls.txt в результате работы такой команды окажется список файлов в текущем каталоге, выведенный командой ls в стандартный поток вывода.

Пример: Аналогично в файл можно перенаправить ошибки:

```
$ ls -ld /etc /ctc 2> ls.err
drwxr-xr-x 87 root root 6064 Окт 15 18:57 /etc
$ cat ls.err
ls: /ctc: No such file or directory
```

Примечание: В этом примере поток вывода ошибок был перенаправлен в файл ls.err. В силу того, что в качестве аргумента команды ls -ld был задан несуществующий каталог /ctc, то команда вывела соответствующее сообщение об ошибке, которое и было перенаправлено в файл ls.err. Содержимое файла было выведено с помощью команды cat.

Пример: Поток вывода перенаправлен в файл ls.txt одновременно с перенаправлением потока ошибок в файл ls.err.

```
$ ls -ld /etc /ctc > ls.txt 2> ls.err
```

Чтобы перенаправить оба потока вывода в один и тот же файл следует использовать оператор сцепления потоков `&`

Оболочка bash позволяет следующие эквивалентные формы записи перенаправления потоков вывода в один и тот же файл:

```
> имя_файла 2>&1
2> имя_файла 1>&2
>& имя_файла
&> имя_файла
```

Примеры:

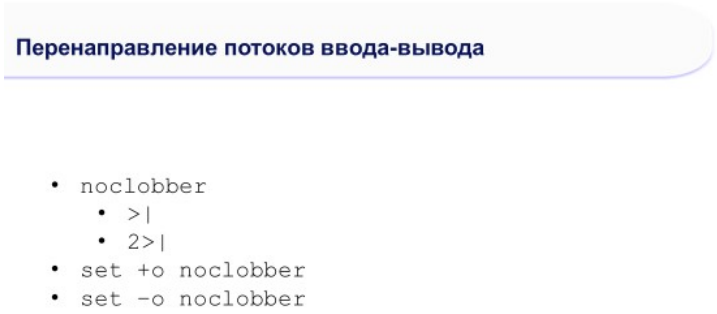
```
$ ls -ld /etc /ctc > ls.txt 2>&1
$ ls -ld /etc /ctc 2> ls.txt >&2
$ ls -ld /etc /ctc &> ls.txt
```

Операции перенаправления потоков вывода и вывода ошибок в файл стирают его содержимое, записывая новое содержимое взамен старого.

Операцию перенаправления можно использовать для стирания содержимого файлов и создания новых пустых файлов.

Пример: для стирания содержимого файла ls.txt можно использовать команду

```
$ > ls.txt
```



Оболочка Bash позволяет исключить стирание содержимого файлов при перенаправлении в них потоков вывода или ошибок с помощью установки флага `noclobber` командой `set -o`.

Пример:

```
$ set -o noclobber
```

Значение всех флагов оболочки можно с помощью команды `set -o`.

Пример:

```
$ set -o
allexport off
braceexpand on
emacs on
errexit off
hashall on
histexpand on
history on
ignoreeof off
interactive-comments on
keyword off
monitor on
noclobber on
noexec off
noglob off
```

```
nolog off
notify on
nounset off
onecmd off
physical off
posix off
privileged off
verbose off
vi off
xtrace off
```

Примечание: Листинг, выведенный командой set -o, демонстрирует, что после выполнения пользователем команды set -o noclobber, данная опция была активизирована (состояние on). Установка этой опции оболочки предотвращает перезапись существующих файлов с помощью операций вывода.

Пример: Попробуем очистить существующий файл dir.txt.

```
$ > dir.txt
bash: dir.txt: cannot overwrite existing file
```

Примечание: Так как установлена опция noclobber, то оболочка не позволила переписать (в данном случае стереть) существующий файл.

Для перезаписи содержимого файла при установленной опции `noclobber` можно воспользоваться операторами:

1. `>|` - для перенаправления потока вывода с гарантированной перезаписью файла;
2. `2>|` - для перенаправления потока вывода ошибок с гарантированной перезаписью файла.

Пример:

```
$ ls -l >| dir.txt
```

Примечание: Эта команда запишет подробную информацию о содержимом текущего каталога в существующий файл dir.txt, не обращая внимания на то, что файл уже существует и установлена опция оболочки noclobber.

Отключить опцию `noclobber` (как и другие опции по аналогии) можно с помощью команды: `set +o noclobber`

Если необходимо добавить в существующий файл информацию из потоков вывода, то можно использовать следующие операторы:

1. `>>` - для перенаправления потока вывода на добавление к файлу;
2. `2>>` - для перенаправления потока вывода ошибок на добавление к файлу

Если файла не существует, то создается новый файл.

Пример: следующая команда добавит в файл dir.txt имя текущего каталога:

```
$ pwd >> dir.txt
```

Некоторые команды, работающие с текстовыми файлами, позволяют вместо имени файла указать стандартный поток ввода. Для чего используется символ –

Пример: команда `vi` - позволяет считать редактируемый текст из стандартного потока ввода вместо открытия файла.

Для завершения потока ввода, производимого с клавиатуры, следует набрать сочетание клавиш `Ctrl-D`, передающее в поток ввода символ конца файла.

Пример:

```
$ view -
Privet
<Ctrl-D>
```

Примечание: В этом примере команда view позволила считать содержимое потока ввода с клавиатуры, окончание которого было обозначено с помощью сочетания Ctrl-D.

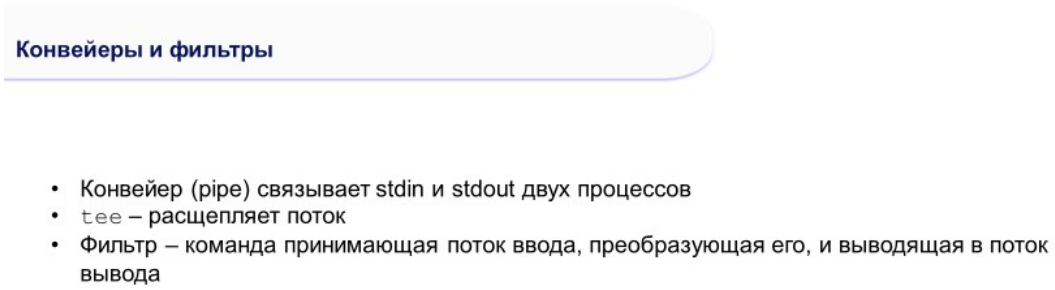
Конструкция `here document` (документ здесь) позволяет вместо символа конца файла (`EOF`), который не может быть использован внутри файла, воспользоваться любым другим удобным символом.

Пример: следующая конструкция обеспечивает посылку письма пользователю user1, где тело письма передается через поток ввода команды mail с помощью here document:

```
$ mail -s HereDoc user1 << .
This is Here Document here!
.
```

Примечание: Обратите внимание, что в предыдущем примере вместо использования символа окончания потока здесь используется символ точки. Этот символ был задан в качестве ограничителя потока ввода в командной строке << . . Это обозначает, что текст, вводимый через стандартный поток ввода, должен быть ограничен символом точка. Символ – ограничитель должен находиться в строке, перед которой чтение стандартного потока ввода прекращается. Символ – ограничитель должен быть единственным символом в строке (не считая символа перевода строки).

9.2. Конвейеры и фильтры.



Конвейер (`pipe`) позволяет направить стандартный поток вывода (`stdout`) одного процесса в стандартный поток ввода (`stdin`) другого процесса.

Для организации конвейера необходимо поставить символ конвейера – вертикальную черту `|` между командами, потоки которых необходимо объединить.

Пример:

```
$ last | view -
```

Примечание: Выводимый командой `last` список входивших в сеанс пользователей передан через конвейер команде просмотра `view` (один из вариантов вызова `vi`). Знак тире после команды `view` сообщает, что данные должны быть введены из стандартного потока ввода, в который передает данные команда `last` через свой поток вывода.

Команда `tee` позволяет организовать вывод информации, полученной из потока ввода, в файлы, указанные как аргументы, и в стандартный поток вывода одновременно.

Пример: команда ps в примере ниже выведет список процессов в файл ps.txt и в стандартный поток вывода.

```
$ ps | tee ps.txt
PID TTY TIME CMD
1720 pts/0 00:00:00 bash
2438 pts/0 00:00:00 ps
2439 pts/0 00:00:00 tee
$ cat ps.txt
PID TTY TIME CMD
1720 pts/0 00:00:00 bash
2438 pts/0 00:00:00 ps
2439 pts/0 00:00:00 tee
```

Примечание: Содержимое файла ps.txt совершенно идентично выводу команды ps. Обратите внимание на присутствие в списке процессов команды tee.

Примечание: Команда tee наиболее часто используется для отладки работы сложных конвейеров, состоящих из множества команд. Эту команду удобно устанавливать в месте конвейера, которое вызывает подозрения. Так как в файле, указанном в качестве аргумента tee, будет находиться та же информация, что была передана в данном месте конвейера, то по ней легко можно будет определить наличие и суть ошибок.

Фильтр – это не интерактивная команда, способная принимать поток данных через стандартный поток ввода, обрабатывать его, и выводить обработанные данные в стандартный поток вывода.

Пример: команда `wc -l` – фильтр, подсчитывающий количество строк в потоке ввода и передающий результат подсчета в поток стандартного вывода. Конвейер, показанный ниже, подсчитывает число процессов, работающих в системе от имени пользователя user1.

```
$ ps -u user1 | wc -l
8
```

Команды, находящиеся в конвейере должны удовлетворять следующим требованиям:

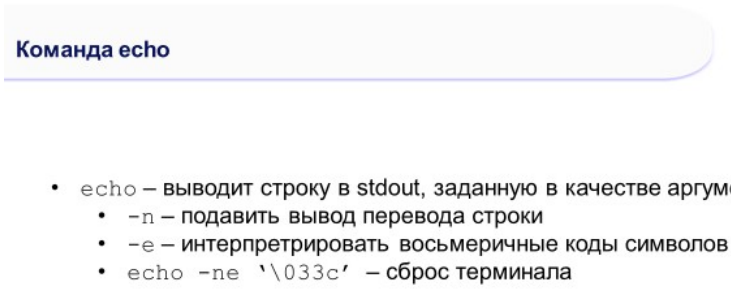
1. Первая команда конвейера должна уметь выводить информацию в стандартный поток вывода.
2. Команды, находящиеся внутри конвейера, должны быть фильтрами.
3. Последняя команда в конвейере должна уметь читать стандартный поток ввода.

Пример: для отправки суперпользователю письма с отсортированным списком пользователей, вошедших в сеанс, можно использовать такой конвейер:

```
$ who | sort | mail -s 'Logged users' root
```

Примечание: В этом примере команды who и mail не являются фильтрами, но поскольку команда who осуществляет вывод в стандартный поток вывода, а mail читает поток ввода, то их можно использовать, соответственно, в начале и конце конвейера. Команда sort, осуществляющая сортировку, является фильтром, то есть она читает из стандартного потока ввода и выводит отсортированный текст в стандартный поток вывода, поэтому она вполне может находиться где-либо в середине конвейера.

9.3. Команда echo.



Команда `echo` выводит на экран текстовую строку, заданную в качестве ее аргумента или значение переменной, перед которой должен стоять символ `$`

Пример:

```
$ echo "It's cool"
It's cool
$echo $PS1
/u:/w$
```

Примечание: В первом примере команда `echo` просто выводит строку, заданную ей в качестве аргумента. Во втором примере выводится значение переменной окружения `PS1`, для чего перед именем переменной поставлен символ извлечения значения переменной `$`.

Команда `echo` автоматически вставляет после выводимой строки символ перевода строки. Для отмены этого можно использовать опцию `-n`.

Опция `-e`, позволяет команде `echo` интерпретировать восьмеричные числа как символы ASCII.

Пример:

```
$ echo -e '\101\t\102'
A B
```

Примечание: В этой команде выводятся буквы A и B, заданные их восьмеричными кодами ASCII, которые разделены символом табуляции.

Пример: Та же команда, но без опции -e выведет следующее:

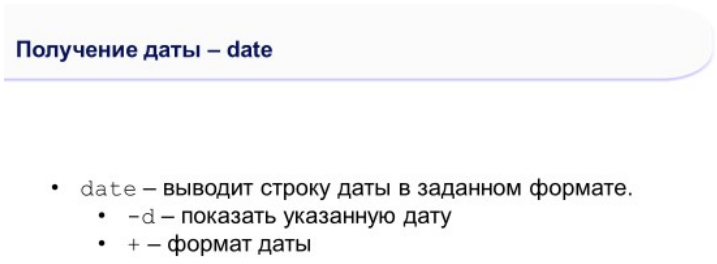
```
$ echo '\101\t\102'
\101\t\102
```

Часто при ошибочном выводе двоичного файла на терминал у последнего сбиваются настройки. Один из способов сброса настроек состоит в посылке на терминал `ESC` последовательности

Пример:

```
$ echo -ne '\033c'
```

9.4. Получение даты – date



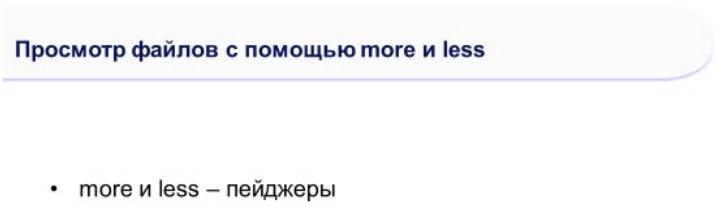
Команда `date` позволяет выводить дату/время (текущую или указанную) в определенном формате, а также задавать системные дату/время

Примеры:

```
$ date
Пн 06 мая 2024 09:04:28 +05

$ date -d yesterday
Вс 05 мая 2024 09:04:50 +05

$ date +"%Y-%m-%d"
2024-05-06
```

Примечание: Для задания системных даты/времени используется опция `-s`

9.5. Просмотр файлов с помощью more и less.



Команды `more` и `less` позволяют постранично просматривать текстовые файлы.

Пример: приведенная ниже команда позволяет просмотреть файл /etc/passwd.

```
$ less /etc/passwd
```

Обе эти команды позволяют читать текст из стандартного потока ввода, причем при чтении из него аргументы – имена файлов задавать не следует.

Обобщающее название этих команд – пейджеры (pager).

Примечание: Первый из них считается стандартным и присутствует во множестве разновидностей UNIX систем. Второй из них - less , разработан позже, и не смотря на свое название (more – больше, less – меньше), обладает гораздо большими возможностями, чем more.

Пейджер `less` используется в GNU/Linux по умолчанию.

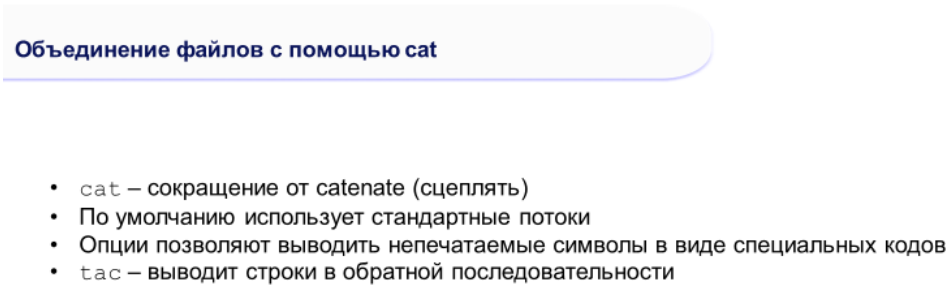
Примечание: Именно он работает, когда пользователь получает помощь в системе `man`, так как страница помощи, найденная `man`, передается для отображения пэйджеру `less`. Одно из наиболее удобных свойств `less` заключается в том, что в отличие от `more` , для промотки экрана в нем можно

пользоваться обычными клавишами управления курсором вверх и вниз, а также клавишами `PgUp` и `PgDn`. Многие из команд управления промотки `more` и `vi` поддерживаются в `less`.

Наиболее часто используемых команд `more` и `less`:

Действие	`more`	`less`
Выход из пэйджера.	`q`	`q`
Получение встроенной помощи по командам.	`h`	`h`
Прокрутка страницы вперед.	`Space`	`PgDn`
Прокрутка страницы назад.	`Ctrl-B`	`PgUp`
Прокрутка на строку вперед.	`Return`	`Ctrl-N`
Прокрутка на строку назад.		`Ctrl-P`
Переход в конец файла.		`G`
Переход в начало файла.		`g`
Поиск по шаблону (регулярному выражению).	`/шаблон`	`/шаблон`

9.6. Объединение файлов с помощью cat.



Команда `cat` выводит в стандартный поток вывода содержимое файла, заданное в качестве аргумента.

Если задано несколько файлов, то их содержимое выводится последовательно.

Примечание: Команда последовательно объединяет в выводимом потоке содержимое этих файлов (название cat – от слова catenate – объединять).

Пример: следующая команда выведет в стандартный поток вывода содержимое файлов `/etc/issue` и `/etc/issue.net`, содержащих приветствие, выводимое на экран перед входом в сеанс, соответственно, на локальном и удаленном компьютерах.

```
$ cat /etc/issue{,.net}
```

Примечание: в этом примере была использована способность Bash, называемая перебором. Фактически, введенная команда эквивалентна следующей:

```
$ cat /etc/issue /etc/issue.net
```

Если аргумент – имя файла не задан, то используется чтение из стандартного потока ввода.

Пример: следующая команда прочитает текст с клавиатуры и запишет его в файл f1.

```
$ cat > f1
Привет!
<Ctrl-D>
$ cat f1
Привет!
```

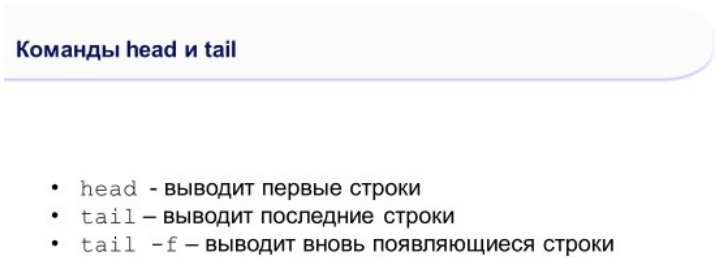
Примечание: В первой команде этого примера в файл f1 с помощью перенаправления стандартного потока вывода записан текст, который был принят командой cat из стандартного потока ввода. Ввод с клавиатуры был завершен с помощью нажатия сочетания клавиш Ctrl-D. Далее, с помощью команды cat было выведено содержимое файла f1.

Команда `cat` может объединять и двоичные файлы, если они указаны как аргументы.

Не следует только выводить содержимое двоичных файлов на терминал, так как это может нарушить его настройки и потребует его инициализации.

У команды `cat` имеется “зеркальный” двойник – команда `tac`, позволяющая выводить строки файла, заданного в качестве аргумента, в обратном порядке.

9.7. Команды head и tail.



Команда `head` предоставляет возможность вывести заданное количество первых строк файла или потока, если файл не указан в качестве аргумента.

По умолчанию выводится десять первых строк.

Требуемое количество строк может быть указано в качестве опции

Пример:

```
$ head -3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

Примечание: В этом примере выведены первые три строки файла /etc/passwd.

Для вывода последних строк из файла используется команда `tail`.

Требуемое количество срок может быть задано через опцию, а по умолчанию выводится десять строк.

Пример: для вывода десяти последних в списке процессов, выводимых командой `ps -e`, можно использовать `tail` :

```
$ ps -e | tail
3308 ? 00:00:00 mapping-daemon
```

```
3310 ? 00:00:00 clock-applet
3312 ? 00:00:00 notification-ar
3314 ? 00:00:00 mixer_applet2
3321 ? 00:00:03 wnck-applet
3325 ? 00:01:59 soffice.bin
3344 ? 00:00:00 getstyle-gnome
3362 ? 00:00:47 xpdf
3653 pts/1 00:00:00 ps
3654 pts/1 00:00:00 tail
```

Примечание: Команда tail предоставляет удобную опцию -f, которая позволяет выводить последние строки заданного файла динамически отображая новые строки, появляющиеся в конце файла. Это бывает полезно, например, при мониторинге файлов журналов.

Опция `-n` команды `tail` позволяет указать не только сколько последних строк показывать, но и с какой строки начать выводить файл

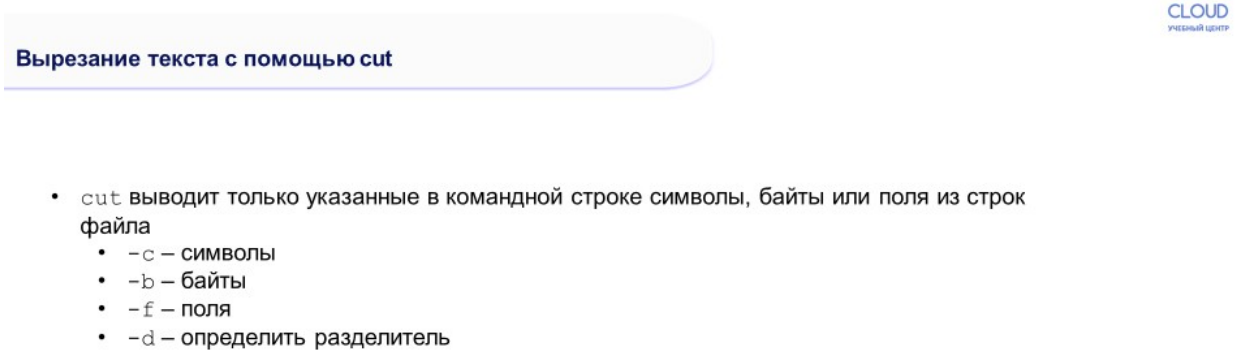
Пример: выведем из потока две последние строки и поток начиная со второй строки

```
$ echo -e 'one\ntwo\nfree\nfour' | tail -n 2
free
four
$ echo -e 'one\ntwo\nfree\nfour' | tail -2
free
four
$ echo -e 'one\ntwo\nfree\nfour' | tail -n +2
two
free
four
```

Примечание: Первые две команды эквивалентны.

Опция `-f` позволяет не выходить из `tail`, а выводить вновь появившиеся строки из файла. Это очень удобно, когда вы смотрите журналы при отладке.

9.8. Вырезание текста с помощью cut.



Команда `cut` выводит только указанные в командной строке символы, байты или поля из строк файла, направляя выводимые данные в стандартный поток вывода.

Если файл для чтения не задан, то осуществляется ввод из `stdin`.

Для указания требуемых символов строки для вывода необходимо использовать опцию `-c`, после которой необходимо указать номера символов через запятую или тире:

Пример:

```
$ ls -ld
drwx------ 51 user1 user1 3752 Окт 22 21:04 .
$ ls -ld | cut -c1-10,35-43
drwx------ 3752
```

Примечание: В этом примере продемонстрировано, как из информации, выводимой командой `ls -ld` были извлечены требуемые диапазоны символов так, что в результате осталась информация только о правах доступа к каталогу и о его размере.

Можно указать номера требуемых байтов в строке для вывода, используя опцию `-b`.

Если строка разделена табуляцией на поля, то с помощью команды `cut -f` можно вывести требуемые поля строк.

Пример: если требуется вывести из файла `/etc/hosts`, содержащего соответствия IP адресов именам хостов, только заданные в нем IP адреса без имен хостов, можно выполнить следующую команду:

```
$ cut -f1 /etc/hosts
127.0.0.1
#212.193.90.39
212.193.90.25
```

Иной разделитель полей можно указать после опции `-d`.

Пример: Получим список пользователей системы и их `UID`. Это можно сделать путем вывода первого и третьего полей файла `/etc/passwd`. Разделитель полей в этом файле – двоеточие.

```
$ cut -f1,3 -d: /etc/passwd
```

9.9. Регулярные выражения

Регулярные выражения

- Метасимволы – шаблоны и квантификаторы

Шаблоны обычные	Квантификаторы обычные
. – любой символ	* - любое количество
^ - начало строки	? – ноль или один раз
$ - конец строки	
[] - диапазон	

Шаблоны расширенные	Квантификаторы расширенные
\< – начало слова	+ - один и более раз
\> – конец слова	{n} – n раз
() – группа символов	{n,m} – от n до m раз
	{n,} – не менее n раз

Регулярные выражения – это специальные шаблоны (pattern), используемые для поиска строк в текстовых файлах и потоках.

Все регулярные выражения строятся из обычных алфавитно-цифровых символов и так называемых метасимволов. Метасимволы позволяют задавать один или более обычных алфавитно-цифровых символов. Регулярные выражения подразделяются на:

- обычные регулярные выражения;
- регулярные выражения с расширенным синтаксисом.

Не все текстовые утилиты, работающие с регулярными выражениями, поддерживают расширенный синтаксис регулярных выражений. Метасимволы, составляющие регулярные выражения представлены двумя классами:

1. Шаблоны – специальные символы, заменяющие один или более обычных символов;
2. Квантификаторы – указатели количества вхождений символа или набора символов, находящихся в регулярном выражении непосредственно перед ними.

Примечание: То есть шаблоны указывают что должно находиться в данном месте искомой строки, а квантификаторы – сколько раз оно должно там встречаться. Помимо перечисленных выше регулярных выражений имеется несколько более расширенных их

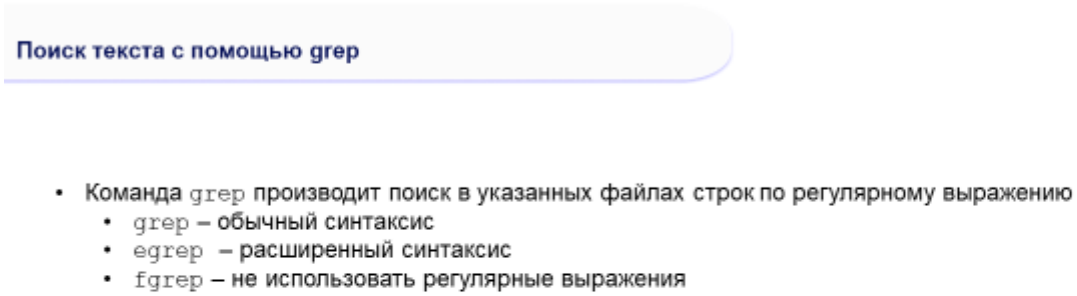
наборов. Например, в языке Perl используется большее количество регулярных выражений. Множества символов, задаваемые в квадратных скобках, имеют следующий смысл: в данном месте должен находиться любой один из символов, входящих в множество.

Пример: множество `[0-3]` включает в себя все числа от нуля до трех, следовательно регулярному выражению `^[0-3]$` удовлетворяют все строки, в которых находится единственный символ – число от нуля до трех (`^` - начало строки, `$` - конец строки).

Помимо множеств, заданных с помощью перечисления символов в квадратных скобках, существуют заранее определенные множества (классы) символов, приведенные в таблице ниже.

Класс	Значение
`[:alnum:]`	Множество алфавитно-цифровых символов.
`[:alpha:]`	Алфавитные символы.
`[:blank:]`	Пустые символы – пробел и табуляция.
`[:cntrl:]`	Символы с восьмеричными кодами от 000 до 037 и 177.
`[:digit:]`	Десятеричные цифры.
`[:graph:]`	Изображаемые символы: алфавитно-цифровые и символы пунктуации.
`[:lower:]`	Буквы в нижнем регистре.
`[:print:]`	Печатаемые символы: алфавитно-цифровые, пробел и символы пунктуации.
`[:space:]`	Табуляция, вертикальная табуляция, пробел, перевод страницы и строки.
`[:upper:]`	Буквы в верхнем регистре.
`[:xdigit:]`	Шестнадцатеричные символы.

9.10. Поиск текста с помощью grep.



Команда `grep` производит поиск в указанных файлах строк по регулярному выражению.

Команда выводит в случае удачного поиска, имя файла и строку, удовлетворяющую заданному регулярному выражению.

Если входные файлы не заданы, то команда производит ввод из стандартного потока ввода.

При этом используются три разновидности команды `grep` (ниже идет речь о GNU версии команды `grep`):

1. `grep` – интерпретирует регулярные выражения с обычным (basic) синтаксисом;
2. `grep -F` или `fgrep` – не интерпретирует регулярные выражения, воспринимая их как обычные текстовые строки;
3. `grep -E` или `egrep` – позволяет работать с расширенным синтаксисом регулярных выражений.

Примечание: Команды `grep`, `egrep` и `fgrep`, разработанные в рамках GNU, реализованы в виде жестких связей. То есть, все эти три команды являются одним файлом. В других разновидностях UNIX систем это не так. К особенностям работы GNU версии этих команд следует отнести способность автоматического переключения `egrep` в режим работы `grep` в тех случаях, когда использование расширенного синтаксиса не возможно и должен быть использован (в силу особенностей реализации библиотек поиска)

обычный синтаксис. Приведенные ниже примеры не всегда будут работать так же в других операционных системах, например, в Solaris и FreeBSD.

Пример: Для поиска всех пользователей системы, использующих оболочку Bash достаточно выполнить команду `grep` с обычным регулярным выражением.

```
$ grep 'bash$' /etc/passwd
root:x:0:0:System Administrator:/root:/bin/bash
user1:x:502:502::/home/user1:/bin/bash
figus:x:503:503::/home/figus:/bin/bash
user1:x:504:504::/home/user1:/bin/bash
```

Примечание: Регулярное выражение взято в кавычки для предотвращения интерпретации символа доллара оболочкой. Доллар здесь обозначает конец строки.

Пример: Для нахождения во всех файлов в каталоге `/etc/sysconfig/` строк, содержащих IP адреса в стандарте Ipv4 можно использовать команду `egrep` с расширенным синтаксисом регулярных выражений:

```
$ egrep '[0-9]{1,3}(\.[0-9]{1,3}){3}' /etc/sysconfig/*
```

Примечание: В этой команде обеспечивается поиск последовательности из одной, двух или трех любых десятичных цифр, которая должна быть троекратно продолжена такой же последовательностью, каждая из которых отделена от предыдущей последовательности точкой.

Командой `fgrep` удобно пользоваться для поиска строк, содержащих символы, имеющие особое значение, как регулярные выражения.

Пример: можно получить список всех процессов, не связанных с какими-либо терминалами:

```
$ ps -A | fgrep '?'
```

Примечание: Команда `fgrep` воспринимает знак вопроса просто как обычный символ, который должен быть найден в строке.

Опция `-i` позволяет производить поиск без учета регистра.

Пример: ниже демонстрируется, как с помощью поиска с игнорированием регистра найти все процессы в системе, где в командной строке есть подстроки `X` или `x` :

```
$ ps -A | grep -i x
  1546 ?        00:00:01 xfs
  1599 ?        00:00:19 X
  1752 ?        00:00:00 xvt
```

Если требуется подсчитать число вхождений регулярного выражения, то для этого используется опция `-c`

Пример:

```
$ egrep -c '^.{1,3}:' /etc/passwd
7
```

Примечание: В приведенном выше примере подсчитано сколько имеется пользователей, длины имен которых не превышают три символа.

Опция `-v` команды `grep` инвертирует алгоритм поиска: команда начинает искать строки, не удовлетворяющие регулярному выражению.

Пример: для того, чтобы найти список всех групп из файла `/etc/group` , в которые не входит пользователь `user1` выполним команду

```
$ grep -v user1 /etc/group
```

Поиск слов выполняется командой `egrep` с использованием метасимволов регулярных выражений, соответственно, \< и \> или с помощью опции `-w`.

Пример: для поиска всех строк в файле `/etc/sysctl.conf` , в которых встречается одиночный слово `Enable` , но не `Enables` , можно использовать команду:

```
$ egrep '\<Enable\>' /etc/sysctl.conf
# Enable the magic-sysrq key
# Enable tcp_syncookies
```

Пример:

```
$ grep -w Enable /etc/sysctl.conf
```

```
# Enable the magic-sysrq key
# Enable tcp_syncookies
```

При необходимости можно также установить режим поиска по целой строке, для чего используется опция `-x` .

Пример: Приведенная ниже команда позволит отыскать в каталоге `/etc` все файлы, которые содержат строки, состоящие из одного любого символа:

```
$ grep -x '.' /etc/*
```

Использование опции `-n` позволяет установить режим вывода номеров строк, в которых найдены искомые строки.

Пример: Для получения строки файла `/etc/hosts` , содержащие подстроки `local` , и номера этих строк выполним команду

```
$ grep -n local /etc/hosts
1:127.0.0.1             localhost.localdomain localhost
```

Примечание: Здесь в первом поле вывода перед двоеточием указан номер строки, в которой найдена искомая подстрока.

Опция `-r` заставляет команду `grep` работать рекурсивно, обрабатывая файлы в подкаталогах.

Пример: в каталоге /usr/share/doc/HOWTO/ требуется найти все файлы, содержащие строку `trainer` :

```
$ grep -r trainer /usr/share/doc/HOWTO/
```

9.11. Потоковый редактор sed.

Потоковые редакторы представляют собой фильтры, не интерактивно (“на лету”) редактирующие проходящий через них поток текста.

Все изменения, вносимые в текст потоковыми редакторами, производятся в потоке и не затрагивают содержимое файла, откуда этот поток текста считан.

Некоторые потоковые редакторы обладают встроенными скриптовыми языками программирования, позволяющими писать сложные сценарии обработки потока текста.

Потоковый редактор `sed` позволяет либо прочитать входной поток для обработки из стандартного потока ввода, либо осуществить чтение и обработку текстового файла, заданного в качестве аргумента.

Обработанный текст передается в стандартный поток вывода.

Если потоковому редактору не указан никакой сценарий обработки текста, то поток выводится в `stdout` без изменений.

Редактор `sed` позволяет удобно работать с целыми строками текста.

Пример: для получения списка процессов без заголовка можно с помощью `sed` удалить первую строку из потока. Для этого необходимо использовать команду d языка программирования `sed`.

```
$ ps | sed 1d
1757 pts/0 00:00:00 bash
1870 pts/0 00:00:00 ps
1871 pts/0 00:00:00 sed
```

Примечание: Команда 1d удалила в потоке первую строку.

Перед командами `sed` нужно указывать строки, с которыми производятся какие-либо действия.

Адреса можно указывать следующим образом:

1. Номером строки
2. Диапазоном строк с указанием первого и последнего номеров строк в диапазоне через запятую

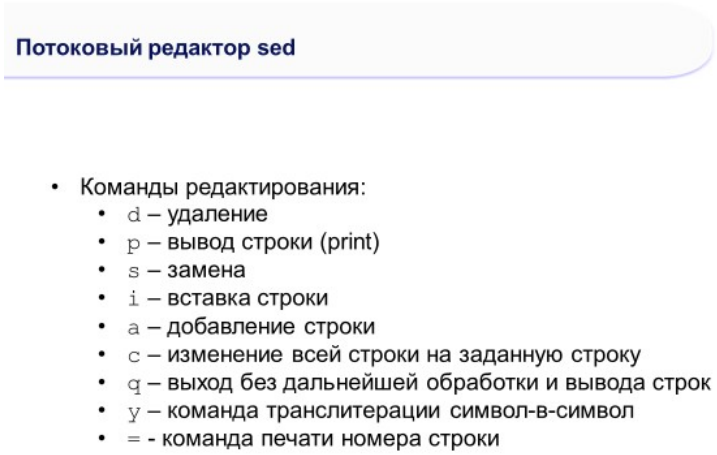
Пример: команда `sed '2,4d'` удалит в потоке строки со второй по четвертую включительно.

3. Можно указать диапазон до последней строки. Последняя строка обозначается символом доллара `$`
4. С помощью регулярного выражения или, в простейшем случае, подстроки, которая должна находиться в адресуемых строках.

Пример: Ниже приведена команда, которая выведет список процессов, не связанных с какими-либо терминалами (их имена содержат подстроку tty), но не псевдотерминалами (например, pts/1). Это достигается с помощью удаления из потока всех строк, в которых имеется подстрока tty. Искомая подстрока (регулярное выражение), используемая для адресации удаляемых в потоке строк, должна быть задана в косых чертах.

```
$ ps -A | sed '/tty/d'
```

Примечание: В результате выполения этой команды будет отображен список процессов, либо не связанных с терминалами вообще, либо связанных с псевдотерминалами.



Наиболее часто используются следующие команды `sed`:

1. `d` – удаление
2. `p` – вывод строки (`print`)
3. `s` – замена искомой подстроки (регулярного выражения) на заданную подстроку
4. `i` – вставка строки
5. `a` – добавление строки
6. `c` – изменение всей строки на заданную строку
7. `q` – выход без дальнейшей обработки и вывода строк
8. `y` – команда транслитерации символ-в-символ
9. `=` - команда печати номера строки

Команда `p` выводит указанные в диапазоне строки в дополнении ко всем строкам текстового потока

Для подавления вывода всех строк текстового потока можно использовать опцию `-n` утилиты `sed`

Пример: чтобы увидеть все строки учетных записей (из файла `/etc/passwd`), начиная с первой и заканчивая строкой, в которой встречается строка daemon выполним команду:

```
$ sed /daemon/q /etc/passwd
root:x:0:0:System Administrator:/root:/bin/bash
```

```
bin:x:1:1:bin:/:/dev/null
daemon:x:2:2:daemon:/:/dev/null
```

Примечание: Из примера видно, что для достижения поставленной цели была использована команда выхода q. Как только sed встретил строку daemon, работа sed была завершена. Обратите внимание на то, что в этом примере файл для чтения задан в качестве аргумента.

Пример: демонстрация того, как `sed` применяется для замены подстрок. Для замены в потоке, полученном при чтении файла `/etc/passwd`, всех вхождений подстроки `bash` на `zsh` выполняем команду

```
$ sed s/bash/zsh/ /etc/passwd
```

Примечание: Здесь после команды s в косых чертах указана подстрока для поиска – bash. Далее указана подстрока замены – zsh.

Пример: Ниже приведен более сложный пример замены подстрок в потоке. Требуется вывести содержание файла /etc/group (файл, содержащий информацию о группах пользователей и членстве в них) с учетом следующих условий:

- должны быть выведены только группы, членом которых является пользователь user1;
- все разделители полей – знаки двоеточия должны быть заменены на символ подчеркивания.

```
$ sed -n /user1/s/:/_/gp /etc/group
adm_x_4_root,adm,daemon,user1
wheel_x_10_root,user1
ftpadmin_x_51_user1
ldap_x_55_user1
users_x_100_user1
webmaster_x_103_user1
user1_x_500_
```

Примечание: В этом примере перед командой s используется адресация с помощью подстроки user1. То есть, во входном потоке выбираются только строки, содержащие подстроку user1. Команда s заменяет все вхождения символа двоеточия в каждой строке на подчеркивание. После косой черты установлен модификатор g. Именно он заставляет команду s заменить все вхождения искомой подстроки (двоеточия) на подстроку – заменитель (подчеркивание). Если бы он отсутствовал, то в каждой адресуемой строке был бы заменен только первый символ двоеточия. После модификатора g следует команда p. Эта команда печатает адресуемые строки, то есть те, в которых здесь произведена

замена. Использование команды p вынуждает добавить опцию -n команды sed, Без этой опции sed без изменений выводит в stdout строки, которые не обрабатывались. Таким образом, если бы не было опции -n, отменяющей вывод необработанных строк, были бы выведены все строки потока, а те, которые были обработаны, были бы выведены дважды.

Пример: демонстрация того, как можно выделить требуемые строки в потоке, вставив перед ними заданную строку (например, строку – разделитель). Для получения списка всех последних входов в систему пользователя user1 так, чтобы после каждой строки, где упоминается этот пользователь, выводилась строка ########################## выполняем команду

```
$ last | sed '/user1/a \
##########################'
```

Примечание: Такой не совсем обычный формат команды связан с тем, что в командной строке оболочки введена команда, которая обычно используется в скриптах sed. В скрипте в первой строке этой команды находилась бы конструкция /user1/a, которая позволила бы адресоваться ко всем строкам, в которых есть строка user1. Команда a заставляет sed добавить после каждой адресуемой строки то, что указано на следующей строке после этой команды.

Пример: Последний пример использования sed показывает содержимое файла сценария sed, реализующий ту же задачу.

```
$ cat sed.scr
/user1/a \
##################
$ last | sed -f sed.scr
```

Примечание: Содержимое файла sed.scr выведено командой cat. Последняя строка примера показывает, что для указания sed имени файла скрипта надо указывать опцию -f.

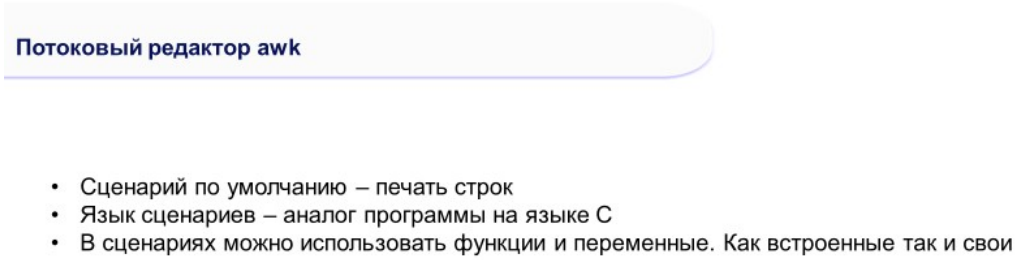
9.12. Потоковый редактор awk.

Утилита `awk`, GNU версия которой называется `gawk`, предназначена для потокового редактирования текста и обладает специализированным языком программирования, исключительно удобным для обработки структурированных данных.

Основная задача `awk` состоит в поиске и обработке строк в потоке.

Программа для `awk` в общем виде выглядит таким образом:

```
условие{ команды }
условие{ команды }
...
условие{ команды }
```



Команды могут находиться в файле сценария, имя которого должно быть указано после опции `-f`, либо могут задаваться в качестве аргумента команды `awk`.

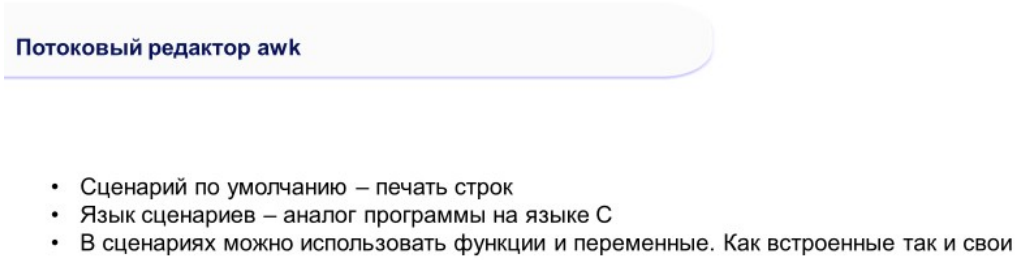
Если условие для команды не указано, то обработке подвергаются все строки.

Если условие указано, а команда – нет, то в `stdout` выводятся только строки, удовлетворяющие условию.

Пример: следующая команда выведет из файла /etc/passwd все строки, содержащие подстроку bash.

```
$ awk '/bash/' /etc/passwd
root:x:0:0:System Administrator:/root:/bin/bash
user1:x:500:500::/home/user1:/bin/bash
user1:x:502:502::/home/user1:/bin/bash
figus:x:503:503::/home/figus:/bin/bash
```

Примечание: В этом случае не была задана команда, но был задан шаблон – строка bash.

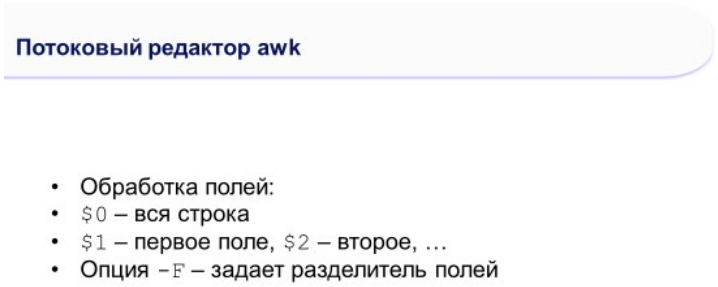


Утилита `awk` позволяет обращаться к полям строк, разделенных с помощью пробелов или табуляций, следующим образом:

1. `$0` – вся строка
2. `$1` – первое поле
3. `$2` – второе и так далее

Утилита `awk` мощный инструмент анализа и манипуляции данными. В рамках данного курса нет возможности изучить все возможности этой программы. Если вы хотите более подробно изучить `awk` почитайте соответствующую литературу, например: https://www.gnu.org/software/gawk/manual/gawk.pdf или `sed` и `awk`: https://github.com/manish-old/ebooks-2/blob/master/O%27Reilly%20-%20sed%20_%20awk%202nd%20Edition.pdf

Если разделитель между полями отличается от пробела или табуляции, то этот символ-разделитель можно указать после опции `-F`



Вывод полей и другой информации обеспечивает команда `print`.

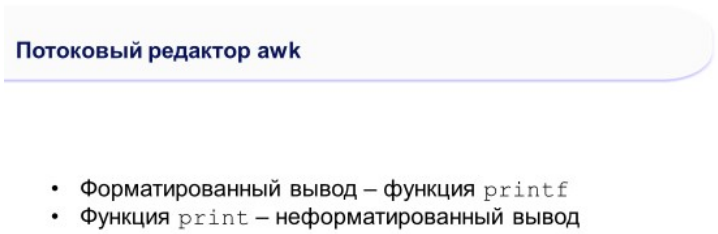
Пример: выведем только имя пользователей, использующих bash, и их UID. Для этого необходимо вывести первое и третье поле этого файла.

```
$ awk -F: '/bash/{print $1,$3}' /etc/passwd
root 0
admin 500
user1 502
user2 503
```

В качестве условия отбора строк можно использовать операции сравнения.

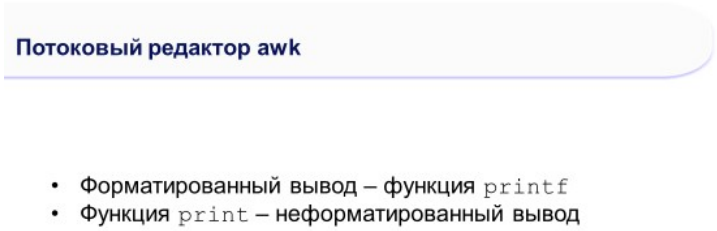
Пример: Для вывода только тех пользователей, чьи UID больше 500 можно отказаться от шаблона и профильтровать строки по условию $3>500.

```
$ awk -F: '$3>500{print $1,$3}' /etc/passwd
user1 502
user2 503
```



Пример: Для вывода только тех пользователей, чьи UID больше 500 с использованием форматированного вывода функцией printf.

```
$ awk -F: '$3>500{printf "%s:%d\n",$1,$3}' /etc/passwd
user1:502
user2:503
```

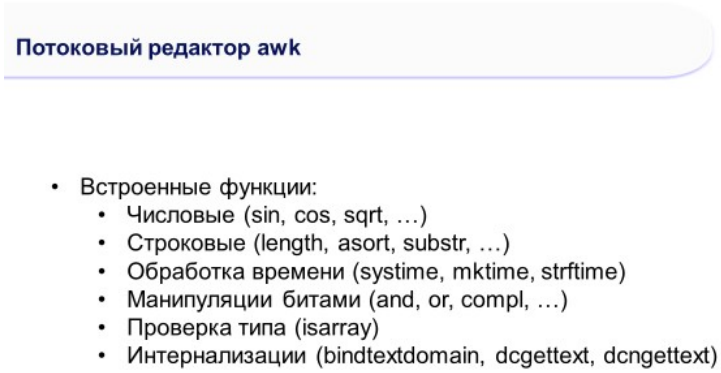


Язык `awk` обладает собственными заранее определенными переменными и позволяет пользователю определять свои переменные:

1. `NR` – номер строки в потоке.
2. `NF` – номер поле в потоке.
3. `FS` – разделитель полей.
4. `FILENAME` – имя файла

Пример: перед именами пользователей выведем номера строк, считанных из файла /etc/passwd.

```
$ awk -F: '$3>500{print NR,$1,$3}' /etc/passwd
48 user1 502
49 figus 503
```



В `awk` имеется большое количество встроенных функций:

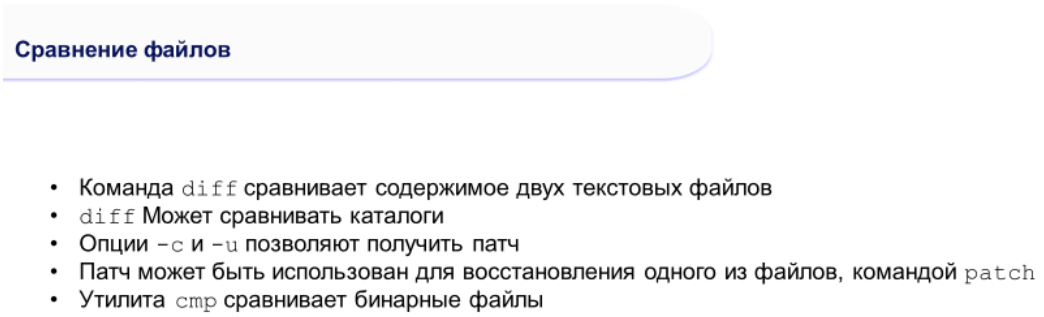
1. Числовые (`sin`, `cos`, `sqrt`, …)
2. Строковые (`length`, `asort`, `substr`, …)
3. Обработка времени (`systime`, `mktime`, `strftime`)
4. Манипуляции битами (`and`, `or`, `compl`, …)
5. Проверка типа (`isarray`)
6. Интернализации (`bindtextdomain`, `dcgettext`, `dcngettext`)

Пример: получим список только тех файлов, длина имен которых больше 15 символов.

```
$ ls | awk 'length($0)>15'
lpi05102003.tar.bz2
Plan_Sem_PM_1.rtf
Plan_Sem_PM_1.sxw
Plan_Sem_PM_2.rtf
Plan_Sem_PM_2.sxw
```

Примечание: В данном случае фильтруются только те строки, длина которых больше 15 символов.

9.13. Сравнение файлов и каталогов.



Команда `diff` принимает в качестве аргумента два имени файлов или каталогов и проводит их сравнение.

Пример:

```
$ ps > ps1.txt
$ ps > ps2.txt
$ diff ps1.txt ps2.txt
3c3
< 3084 pts/0 00:00:00 ps
---
> 3088 pts/0 00:00:00 ps
```

Примечание: В этом примере список текущих процессов был выведен в два различных файла. Так как вызов команды ps производился дважды, то PID процессов у этих команд различался. Это и подтверждает вывод команды diff. Оба получившихся файла содержат по три строки. Последние, третьи, строки отличаются (3c3).

Опция `-q` отменяет вывод отличающихся строк и печатает только сообщение о том, имеются ли отличия.

Пример:

```
$ diff -q ps1.txt ps2.txt
Файлы ps1.txt и ps2.txt различаются
```

Опция `-i` применяется в случае, если необходимо игнорировать регистр в сравниваемых файлах.

Опции `-u` и `-c` выводят информацию, соответственно в обобщенном (unified) и контекстном (context) форматах:

Пример:

```
$ diff -u ps1.txt ps2.txt
--- ps1.txt 2003-11-20 23:18:16 +0500
+++ ps2.txt 2003-11-20 23:18:21 +0500
@@ -1,3 +1,3 @@
PID TTY TIME CMD
2164 pts/0 00:00:00 bash
- 3084 pts/0 00:00:00 ps
+ 3088 pts/0 00:00:00 ps
$ diff -c ps1.txt ps2.txt
*** ps1.txt 2003-11-20 23:18:16 +0500
--- ps2.txt 2003-11-20 23:18:21 +0500
***************
*** 1,3 ****
PID TTY TIME CMD
2164 pts/0 00:00:00 bash
! 3084 pts/0 00:00:00 ps
--- 1,3 ----
PID TTY TIME CMD
2164 pts/0 00:00:00 bash
! 3088 pts/0 00:00:00 ps
```

Особенностью unified и context форматов является вывод информации об именах сравниваемых файлов.

Вывод различий в форматах unified и context может быть потом использован для реконструкции одного из сравниваемых файлов по другому с помощью известных отличий этих файлов.

Для восстановления используется утилита `patch`, позволяющая обновлять содержимое файлов с помощью так называемых “заплаток” (patches).

Примечание: Файлы – “заплатки”, создаются с помощью утилиты `diff`, а обновление файлов с помощью этих “заплаток” производится утилитой `patch`.

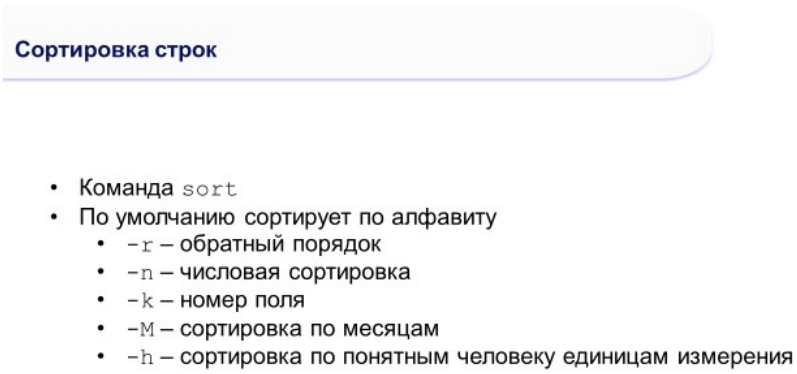
Для по байтного сравнения бинарных файлов удобно применять команду `cmp`.

Пример:

```
$ dd if=/dev/urandom of=r1.bin count=10
10+0 входных записей
10+0 выходных записей
$ dd if=/dev/urandom of=r2.bin count=10
10+0 входных записей
10+0 выходных записей
$ cmp f1.bin f2.bin
$ cmp r1.bin r2.bin
r1.bin r2.bin различаются: байт 1, строка 1
```

Примечание: В этих примерах были созданы четыре бинарные файла: f1.bin и f2.bin, заполненные нулями, и r1.bin и r2.bin, заполненные случайными числами. Размер всех файлов одинаковый. Сравнив эти файлы с помощью cmp, замечаем, что файлы f1.bin и f2.bin не отличаются. В то же время, r1.bin и r2.bin - различны.

9.14. Сортировка строк.



Команда `sort` предназначена для сортировки строк файлов, указанных как аргументы команды. В случае если файлы не указаны, команда читает данные из стандартного потока ввода (`stdin`).

По умолчанию команда `sort` сортирует строки в алфавитном порядке по возрастанию.

При необходимости сортировки строк в порядке убывания используется опция `-r` команды `sort`.

Пример: приведенная ниже команда выводит список имен подкаталогов корневого каталога в обратном алфавитном порядке.

```
$ ls / | sort -r
var
usr
tmp
swap
sbin
root
proc
opt
mnt
lib
home
etc
```

```
dev
boot
bin
```

Команда `sort` способна также сортировать текстовый поток не только по целым строкам, но и по отдельным полям строк.

Разделителем полей по умолчанию считается пробел. Если используется иной разделитель полей, его следует указать после опции `-t`.

Для указания номера поля для сортировки используется опция `-k`.

Опция `-n` позволяет задать команде sort не алфавитный, а числовой порядок сортировки.

Пример: Следующая команда показывает, как отсортировать первые десять строк файла `/etc/passwd`, содержащего учетные записи пользователей, в порядке возрастания их `UID`.

```
$ head /etc/passwd | sort -t: -k3 -n
root:x:0:0:System Administrator:/root:/bin/bash
bin:x:1:1:bin:/:/dev/null
daemon:x:2:2:daemon:/:/dev/null
adm:x:3:4:adm:/var/adm:/dev/null
lp:x:4:7:lp:/var/spool/lpd:/dev/null
sync:x:5:0:sync:/:/bin/sync
shutdown:x:6:0:shutdown:/:/sbin/shutdown
halt:x:7:0:halt:/:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/dev/null
news:x:9:13:news:/var/spool/news:/dev/null
```

Примечание: В этом примере опция разделителем полей является двоеточие, что и установлено опцией -t. Сортировка была выполнена по третьему полю (опция -k3), в котором содержатся UID пользователей. Сортировка была проведена в числовом порядке упорядочения, что было установлено опцией -n.

9.15. Вывод неповторяющихся строк.

Фильтр `uniq` позволяет удалять повторения строки из потока и чаще всего используется после фильтра `sort`, так как входной поток должен быть заранее отсортирован.

Пример: Следующая команда выведет строки файлов f1 и f2 так, что если какая-либо строка имеется в обоих файлах, то в поток вывода попадет только одна ее копия:

```
cat f1 f2 | sort | uniq
```

Опция `-c` команды `uniq` позволяет подсчитать количество вхождений каждой строки во входном потоке.

Опция `-d` позволяет вывести только дублирующихся строк, что позволяет, например, узнать какие строки имеются одновременно в разных файлах,

Опция `-u` выводит только уникальные (недублированные) строки.

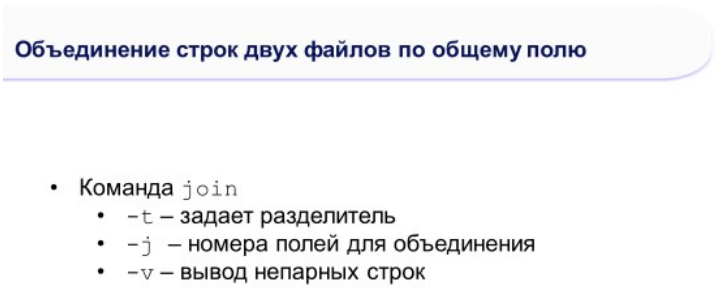
Для игнорирования регистра при сравнении строк можно установить опцию `-i`.

Если строки входного потока разделены на поля, то можно пропустить заранее заданное количество полей до определения уникальности строки. Для пропуска первых полей, разделенных пробелами, необходимо указать их

количество после опции `-f`.

Пример: команда `uniq -f6` пропустит шесть первых полей входного потока при определении уникальности строк.

9.16. Объединение строк двух файлов по общему полю.



Команда `join файл1 файл2` построчно объединяет содержимое заранее отсортированных файлов по общему полю. Выходная информация направляется в стандартный поток вывода (`stdout`).

По умолчанию предполагается, что поля отделяются друг от друга пробелами или табуляцией.

Если не указано иное, то объединение производится по первым полям строк файлов. Опция `-t` позволяет указать символ – разделитель полей, отличный от пробела или табуляции.

Если необходимо произвести объединение строк файлов не по первым полям строк, то номера этих полей необходимо указать после опций `j1` – для первого файла и `j2` – для второго файла. Содержимое файлов должно быть отсортировано по тем полям строк, по которым производится объединение сток.

Для того, чтобы вывести вместе с объединенными строками непарные строки, которые обычно не выводятся, необходимо использовать опцию `-a`. После этой опции необходимо указать номер файла, из которого будут выведены непарные строки.

Опция `-v` позволяет вместо объединенных строк вывести только непарные строки того файла, номер которого указан после этой опции.

9.17. Подсчет количества и нумерация строк.

Команда `wc` позволяет подсчитывать количество символов, слов и строк в файле,указанном в качестве аргумента. Если файл не указан, то команда `wc` читает стандартный поток ввода.

Пример:

```
$ wc /etc/hosts
3 8 92 /etc/hosts
```

Примечание: В этом примере подсчитано количество строк, слов и символов в файле /etc/hosts.

При необходимости можно установить вывод только числа:

1. строк – с использованием опции `-l`
2. слов – при установленной опции `-w`
3. символов – при использовании опции `-c`

Пример: для определения количества пользователей, работающих в настоящее время в сеансе, можно использовать команду:

```
$ who | wc -l
1
```

Команда `nl` позволяет пронумеровать строки в тексте, считанном из файла или из потока ввода.

Команда `cat -b` также нумерует строки, однако возможности команды `nl` намного шире.

Опции `nl` позволяют устанавливать формат нумерации срок, особым образом нумеровать пустые строки, устанавливать шаг нумерации и тому подобное.

9.18. Замена символов в строках с помощью команды tr.

Команда `tr` читает поток ввода и позволяет:

1. Заменять символы в потоке. Для этого необходимо задать два набора символов. Символы из первого набора будут заменены соответствующими по порядку символами из второго набора.
2. При использовании опции `-d` удалять в потоке символы, указанные в наборе.
3. Исключать повторения символов в потоке. Символы, повторы которых должны быть исключены, задаются в наборе. Для использования этого режима работы `tr` следует использовать опцию `-s`.
4. Заменять символы в потоке с последующим устранением их повторов. При этом задаются два набора символов и устанавливается опция `-s`. На первом шаге команда `tr` заменяет символы из первого набора на соответствующие символы из второго, а затем устраняет повторения символов, заданных во втором наборе.
5. Удалять символы из потока с последующим устранением повторов если установлены опции `-d` и `-s`. При этом следует использовать два набора символов: первый – для указания удаляемых символов, и второй, указывающий на символы, повторы которых должны быть устранены.

Пример: замена символов

```
$ echo tarelka | tr a-z A-Z
TARELKA
```

Примечание: Здесь были заданы два набора символов – все буквы английского алфавита в нижнем регистре, которые заменяются на буквы в верхнем регистре.

Пример: Для удаления символов перевода строки во входном тексте можно использовать команду, показанную ниже:

```
$ ls / | tr -d '\n'
binbootdevetchomelibmntoptprocrootsbinswaptmpusrvar
```

Примечание: Здесь в качестве входного потока был использован вывод команды ls /, из которого были удалены все переносы строки.

Пример: Для демонстрации работы опции -s подходит следующая команда:

```
$ echo root | tr -s o
rot
```

Примечание: В этом примере были устранены повторения символа o.

Опция `-c` команды `tr` позволяет инвертировать смысл задаваемого множества символов, то есть удалить при использовании `-d` все, кроме символов, указанных в наборе.

Пример: команда `tr -dc 0-9` удалит во входном потоке все, кроме цифр.

В случае, если в первом наборе встречаются повторяющиеся символы, то символу из первого набора будет сопоставлен тот символ из второго набора, который встретился последним.

С опцией `-t` команда `tr` обрезает длину первого набора по длине второго, для того, чтобы количество символов в них равнялось.

Можно также указать класс символов из набора предопределенных символов:

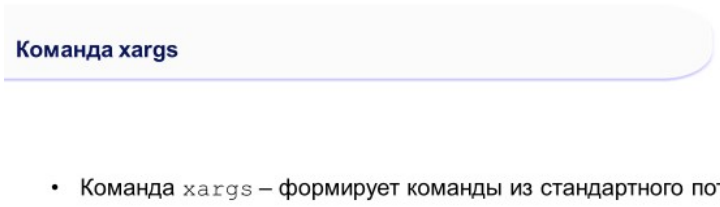
Класс	Символы
[:alnum:]	Символы алфавита в любом регистре и цифры.
[:alpha:]	Символы алфавита в любом регистре.
[:blank:]	Пустое множество.
[:cntrl:]	Управляющие символы.
[:digit:]	Десятичные цифры.
[:graph:]	Все символы, которые могут быть напечатаны кроме пробела.
[:lower:]	Алфавитные символы в нижнем регистре.
[:print:]	Все символы, которые могут быть напечатаны.
[:punct:]	Все символы пунктуации.
[:space:]	Горизонтальное или вертикальное пустое (пробел, табуляция).
[:upper:]	Алфавитные символы в верхнем регистре.
[:xdigit:]	Шестнадцатеричные цифры.

Пример:

```
$echo "str" | tr [:lower:] [:upper:]
STR
```

Примечание: Здесь символы в нижнем регистре заменены символами в верхнем регистре.

9.19. Команда xargs.



Команда `xargs` использует данные, передаваемые ей из стандартного потока ввода, в качестве аргументов для конструирования команды, по умолчанию – `echo`.

Пример: часто команда `xargs` используется с командой `find`. Обе приведенные ниже команды делают одно и то же – ищут и удаляют core-файлы, остающиеся в системе после программных сбоев:

```
$ find /usr -type f -name "core.*" -exec rm -f {} \;
$ find /usr -type f -name "core.*" | xargs rm -f
```

Примечание: В первом случае обработка найденных файлов производится командой find, которая вызывает команду rm. Во втором случае имена найденных файлов отправляются через конвейер команде rm. В силу особенностей обработки потоков и запуска процессов в Linux второй вариант предпочтительнее с точки зрения быстродействия.

Глава 10. Текстовые редакторы.

10.1. Обзор текстовых редакторов

- ed
- emacs
- sed
- awk
- vim
- nano

1. `ed` — первый стандартный текстовый редактор операционной системы UNIX. Имеет командно-ориентированный интерфейс, поскольку создавался в те времена, когда мониторов не существовало и стандартным средством ввода-вывода был телетайп.
2. `emacs` — семейство многофункциональных расширяемых текстовых редакторов. Оригинальный EMACS был написан в 1976 году Ричардом Столлманом и Гаем Стилом как набор макросов для редактирования в редакторе TECO.
3. `sed` (от англ. Stream EDitor) — потоковый текстовый редактор (а также язык программирования), применяющий различные предопределённые текстовые преобразования к последовательному потоку текстовых данных. Первоначально был написан как UNIX-утилита Ли Макмэхоном из Bell Labs в 1973—74 годах. Сейчас `sed` доступен фактически для любой операционной системы, поддерживающей работу с командной строкой.
4. `awk` — с-подобный сценарный язык построчного разбора и обработки входного потока (например, текстового файла) по заданным шаблонам (регулярным выражениям). Может использоваться в сценариях командной строки

5. `vim` — свободный текстовый редактор, созданный на основе более старого `vi`, разработанного Биллом Джойем. Автор Vim'а, Брам Моленар, создал его из порта редактора Stevie для Amiga и в 1991 году выпустил общедоступную версию. `Vim` предназначен для использования как в интерфейсе командной строки, так и в качестве отдельного приложения в графическом пользовательском интерфейсе.
6. `nano` — консольный текстовый редактор для UNIX и Unix-подобных операционных систем, основанный на библиотеке ncurses и распространяемый под лицензией GNU GPL. Это свободный клон текстового редактора Pico, входившего в состав e-mail клиента Pine.

10.2. Работа в редакторе nano

`nano` — консольный текстовый редактор для Unix и Unix-подобных операционных систем, основанный на библиотеке curses и распространяемый под лицензией GNU GPL. Команда вызова редактора выглядит следующим образом:

```
nano
```

В процессе работы с файлом вы можете в любой момент выходить из него, используя сочетание клавиш `Ctrl+X`. Ваши данные не потеряются: после нажатия комбинации редактор спросит, хотите ли вы сохранить изменения:

Также вы можете сразу дать название своему файлу. После `Ctrl+X` нажмите латинскую букву у на клавиатуре, а затем введите имя файла:

Чтобы открыть созданный ранее файл, добавьте его имя в команду. Возьмём в качестве примера наш файл my_file.php — нам нужно будет ввести следующую команду:

```
nano my_file.php
```

Если файл находится в другом каталоге, укажите путь к нему:

```
nano /path/to/my_file.php
```

А эта команда позволит открыть файл сразу в нужном месте, на конкретной строке или столбце:

```
nano +line,column my_file.php
```

Открывайте любой файл указанным выше способом и работайте с ним — пишите, редактируйте, перемещайтесь по тексту. Вам не нужно будет переключаться в режим редактирования, все изменения будут сразу отображаться на экране.

Сочетания клавиш, в которых присутствует `Ctrl`, в редакторе отображаются в виде каретки (`^`) и символа. Например, для вставки содержимого из буфера обмена используется команда `Ctrl+U` — Nano представляет её как `^U`.

В некоторых сочетаниях есть клавиша Meta — обычно это кнопка `Alt`. Такие комбинации редактор отображает в виде символа `M` и команды. Например, нажав клавиши `Alt+W`, вы можете повторить последний поиск.

Комбинации клавиш, которые используются чаще всего, вы увидите в нижней части консоли.

Ознакомиться с остальными комбинациями вы можете в любое время, открыв справку редактора — для этого нажмите `Ctrl+G` либо `F1`:

Для быстрого перемещения по тексту используйте следующие сочетания клавиш:

- `Ctrl +F` (`^F`) — перейти на один символ вперёд;
- `Ctrl +B` (`^B`) — вернуться на один символ назад;
- `Ctrl + Space` (`^Space`) — переместиться вперёд на одно слово;
- `Alt+Space` (`M-Space`) — вернуться на одно слово назад;
- `Ctrl +A` (`^A`) — перейти к началу строки;
- `Ctrl+E` (`^E`) — перейти в конец строки;
- `Ctrl +P` (`^P`) — перейти на предыдущую строку;
- `Ctrl +N` (`^N`) — перейти к следующей строке;
- `Ctrl +V` (`^V`) — перейти на следующую страницу;
- `Ctrl +Y` (`^Y`) — перейти на предыдущую страницу.

Команды для поиска:

- `Ctrl +W` (`^W`) — найти определённое слово или фразу;
- `Alt+W` (`MW`) — перейти к следующему результату поиска;
- `Ctrl +T` (`^T`) + номер строки — найти определённые номера строк;
- `Alt+R` (`MR`) — использовать для поиска regex (регулярные выражения).

Комбинации для выделения, копирования, вырезания и вставки текста:

- `Alt+A` (`MA`) — нажмите эту комбинацию клавиш, предварительно перейдя к началу текста. Далее используйте кнопки курсора, перемещаясь по тому фрагменту в тексте, который вы хотите выделить.
- `Alt+6` (`M-6`) — копировать выделенного текста;
- `Ctrl+K` (`^K`) — вырезать выделенный текст;
- `Ctrl+U` (`^U`) — вставить текст.

Чтобы заменить текст, используйте сочетание клавиш `Ctrl +W` (`^W`) для перехода к панели поиска, далее нажмите `Ctrl +R` (`^R`). Введите часть текста, которую хотите удалить, а затем введите символы, которые должны быть на месте старого фрагмента текста.

Чтобы вставить содержимое другого файла в редактируемый файл, используйте комбинацию `Ctrl + R` (`^R`). У вас откроется внизу панель, в ней укажите путь и имя файла, который хотите вставить в текущий:

С помощью сочетания клавиш `Ctrl+O` (`^O`) вы сможете сохранить свой файл и переименовать его, если нужно. Далее нажмите `Y`. Вы увидите количество строк в файле:

Комбинация `Alt+B` (`MB`) тоже может пригодиться — её используют для создания резервной копии файла в случаях, когда нужно хранить несколько временных версий одного файла. Если нажать это сочетание клавиш повторно, можно отключить резервное копирование.

Для выхода из редактора нажмите сочетание клавиш `Ctrl+X`. Как мы уже с вами отмечали, редактор представляет его в виде `^X`. Если редактируемый файл не был сохранён, то при выходе Nano спросит, хотите ли вы сохранить его.

10.3. Запуск vi и режимы его работы.

- Разные способы запуска
- Четыре режима работы:
  - Командный;
  - Вставки;
  - Двоеточия;
  - Визуальный.
- Встроенная помощь :help

Полноэкранный текстовый редактор `vi` (от слова visual) был разработан для работы с исходными текстами программ и конфигурационными файлами.

Команды его подобраны таким образом, что он будет работать даже на терминалах, не обладающих клавишами управления курсором.

Существуют две версии редактора:

1. Классический `vi`
2. Улучшенный `vim` (vi Improved). В последнем добавлены дополнительные команды и возможности. И vim удобнее при использовании.

Способы запуска `vi` :

1. `vi` – в таком случае `vi` будет запущен для ввода нового файла.
2. `vi -` - файл для редактирования будет считан из стандартного потока ввода (`stdin`).
3. `vi filename` – файл будет открыт для редактирования.
4. `view filename` или `vi -R filename` – файл будет открыт для просмотра.
5. `vi +[num] filename` – курсор будет поставлен на строку с номером `num` или на последнюю строку, если `num` не указан.
6. `vi +/regexp filename` – файл будет открыт для редактирования и курсор будет установлен на строку, где будет найдено вхождение

регулярного выражения `regexp`.

Редактор `vi` обладает тремя различными режимами работы:

1. Командный режим, в котором `vi` оказывается при его запуске и при нажатии на клавишу `ESC` .

Примечание: В этом режиме осуществляется перемещение курсора, просмотр и редактирование текста.

2. Режим ввода текста, в который `vi` переходит при вызове любой из команд вставки, замещения или добавления текста, например, `i` .

Примечание: В этом режиме не рекомендуется, а часто и невозможно, перемещение курсора по тексту. Он используется исключительно для ввода нового текста. Для выхода из этого режима используется клавиша `ESC`.

3. Режим двоеточия, или, иначе, режим последней строки, в который `vi` переходит при нажатии на клавишу двоеточия : в командном режиме.

Примечание: В этом режиме работают такие команды, как открытие нового файла или установка нумерации строк.

4. В визуальный режим `vi` переходит после команд `v`, `V` или `C^v`. В этом режиме вы можете выделять и манипулировать блоками текста.

Система помощи может быть вызвана с помощью команды `:help`, которую необходимо набрать после нажатия на `ESC`.

Если необходимо ввести новый текст, то надо в командном режиме нажать клавишу `i` (`insert`), `vi` перейдет в режим вставки, в котором можно набирать текст. Выйти из режима вставки можно с помощью нажатия `ESC`.

Добиться отмены нежелательных изменений, ошибочно внесенных в текст, можно выполнив в командном режиме команду `u`.

Для выхода из `vi` без сохранения изменений следует нажать клавишу `ESC`, а затем `:q!`

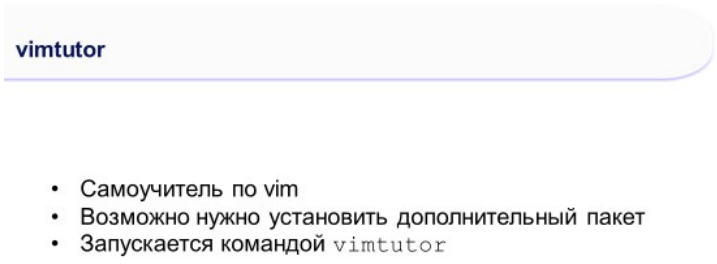
Для сохранения изменений в тексте можно использовать команду `ZZ`, которая обеспечит выход из `vi` с сохранением изменений в редактируемом файле.

Минимально необходимый набор команд:

1. `ESC` — переход в командный режим. Всегда начинайте любое действие с `ESC`.

2. `i` — переход в режим вставки (редактирования). Чтобы закончить редактирование — `ESC`.
3. `:q` и `:q!` - выход и выход без сохранения соответственно.
4. `:wq` или `ZZ` — выйти и сохранить файл.
5. `u` — отмена изменений.

10.4. Учебное пособие vimtutor



Команда `vimtutor` запускает учебник по `vim`. При этом сначала происходит создание копии файла учебника, чтобы его можно было редактировать без опасения потерять исходный файл.

Программа `vimtutor` полезна для новичков, желающих научиться самым основным командам `vim`.

Необязательный параметр `[язык]` представляет собой двухсимвольный код языка, например `ru` или `it`. Если параметр `[язык]` не указан, то используется язык активной в настоящий момент локали. Если учебник на этом языке не существует, то по умолчанию используется учебник на английском языке.

При работе с учебником `vim` всегда запускается в режиме совместимости с `vi`.

Для запуска `vimtutor` возможно нужно установить дополнительный пакет.

Глава 11. Работа с дисками и файловыми системами

11.1. Таблица разделов.

Большинство сценариев использования диска в Linux предполагает создание разделов.

Разделы на дисках создаются для логического разделения информации.

Linux может работать с диском и напрямую без создания разделов. Например при использовании в LVM (Logical Volume Manager).

В Linux в основном используются две схемы разбиения диска на разделы:

MBR (Master Boot Record) – используется для загрузки компьютера и описания разделов на диске.

GPT (GUID Partition Table) – новая схема разбиения дисков. Позволяет создать больше разделов на бóльших дисках.

Linux может создавать и использовать и другие типы разбиения диска, но они на практике почти не применяются: SGI и Sun.

Количество основных разделов в MBR от 1 до 4.

Обычно каждый раздел предназначен для размещения одной операционной системы, отдельной файловой системы, или, например, области размещения страниц подкачки.

MBR находится в нулевом секторе диска, а таблица разделов в нем указывает на начало, конец, размер и тип каждого раздела.

Тип раздела позволяет ОС, в которую он подключен, определить как работать с разделом. Или, другими словами, автоматизировать операции по подключению разделов на этапе загрузки.

Если вы ссылаетесь на разделы в каких-нибудь командах или утилитах, то тип раздела, как правило, игнорируется.

Примечание: Например если вы подключаете диск с «неизвестным» разделом в Windows, то вы даже удалить этот раздел не сможете стандартными средствами Windows. Linux при работе с разделами на тип реагирует «либерально», т. е. Вы можете вручную его всегда подключить или удалить. Но если тип будет не правильный, то во время загрузки нужный вам раздел будет проигнорирован. Например раздел, который не имеет тип fd не будет рассматриваться как часть RAID массива.

Для обеспечения возможности создания большого количества разных файловых систем на жестком диске Linux (как и Windows) поддерживает концепцию логических разделов.

Один из основных разделов (primary partition) может быть объявлен, как расширенный (extended). В расширенном разделе может быть создано неограниченное количество логических разделов (logical partitions).

Разбиение диска с помощью MBR имеет существенное ограничение (2 ТВ) на размер диска, который можно разбить. Это одна из причин почему был предложен новый метод деления диска на разделы — GPT (GUID Partition Table).

GPT — часть стандарта EFI.

В GPT для обратной совместимости сохраняется MBR, в котором создается один раздел на весь диск с типом 0xee.

После MBR в первом секторе содержится оглавление таблицы разделов. В оглавлении содержится информация о положении таблицы разделов, а также о количестве и размере записей.

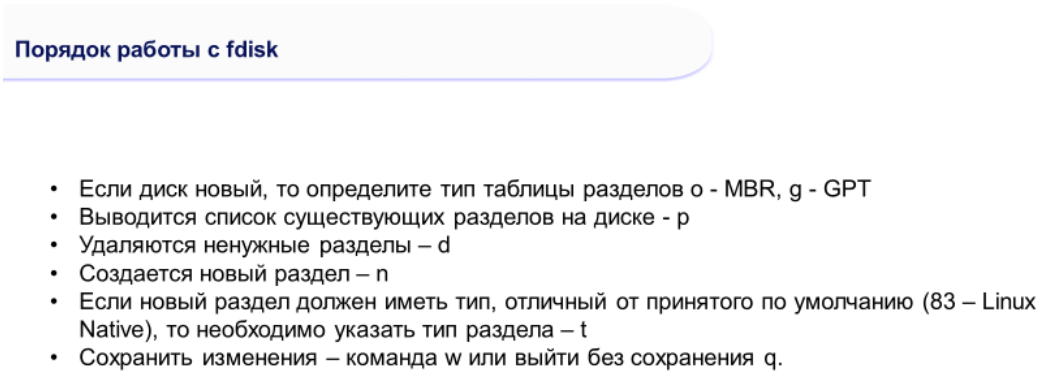
Всего в GPT резервируется 128 записей.

GPT обеспечивает дублирование — оглавление и таблица разделов записаны как в начале, так и в конце диска.

Теоретически, GPT позволяет создавать разделы диска размером до 9,4 ЗБ (9,4 × 1021 байт), в то время как MBR может работать только до 2,2 ТБ (2,2 × 1012 байт).

Для получения таблицы разделов на диске можно воспользоваться командой `fdisk -l`.

11.2. Создание разделов с использованием fdisk.



В Linux вы можете манипулировать разделами с помощь разнообразных утилит. Сред них основные:

1. `fdisk`
2. `sfdisk`
3. `cfdisk`
4. `parted/gparted`
5. `blivet-gui`

Интерактивная утилита `fdisk` позволяет оперировать с дисковыми разделами жестких магнитных дисков и обладает специальным набором собственных команд.

Утилита `sfdisk`, в отличие от `fdisk` , является утилитой для не интерактивного редактирования таблицы разделов на жестком диске.

Примечание: При неосторожном ее использовании легко можно утратить все данные на жестком диске, так как данные для редактирования таблицы разделов задаются в командной строке `sfdisk`.

Популярна также утилита `cfdisk`, которая позволяет редактировать таблицу разделов диска с помощью простого меню-образного интерфейса.

Утилита `parted` помимо создания и удаления разделов может

перемещать разделы по диску. Может работать как интерактивно, так и не интерактивно. `gparted` — графическая утилита.

`blivet-gui` относительно новая графическая утилита для работы с разделами и файловыми системами.

Команда `fdisk` доступна только администратору. Для редактирования таблицы разделов на диске эту команду следует запустить в интерактивном режиме, указав файл устройства для требуемого жесткого диска в качестве аргумента команды:

Пример:

```
# fdisk /dev/sda

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write
them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x806fa0ce.

Command (m for help): m

Help:

DOS (MBR)
a toggle a bootable flag
b edit nested BSD disklabel
c toggle the dos compatibility flag

Generic
d delete a partition
F list free unpartitioned space
l list known partition types
n add a new partition
p print the partition table
t change a partition type
v verify the partition table
i print information about a partition

Misc
m print this menu
u change display/entry units
x extra functionality (experts only)

Script
```

```
I load disk layout from sfdisk script file
O dump disk layout to sfdisk script file

Save & Exit
w write table to disk and exit
q quit without saving changes

Create a new label
g create a new empty GPT partition table
G create a new empty SGI (IRIX) partition table
o create a new empty DOS partition table
s create a new empty Sun partition table
```

Примечание: В этом примере команда fdisk была выполнена с аргументом - файлом устройства первого SCSI диска. Далее была выполнена встроенная команда m, отобразившая список встроенных команд fdisk.

Список основных команды утилиты `fdisk`:

1. `q` - завершение работы без сохранения изменений;
2. `l` - вывод списка возможных типов разделов;
3. `g` - преобразование в GPT диск;
4. `d` - удаление раздела (для удаления будет запрошен номер удаляемого раздела);
5. `n` - создание нового раздела;
6. `t` - установка типа вновь созданного раздела (для установки типа необходимо ввести номер типа раздела);
7. `a` - выбор активного раздела;
8. `w` – запись измененной таблицы разделов.

Встроенная команда `p` утилиты `fdisk` выводит информацию о таблице разделов на диске.

Пример:

```
Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x806fa0ce
```

Обычная последовательность работы с утилитой `fdisk` для создания

нового раздела на жестком диске:

Выводится список существующих разделов на диске - команда `p`.

Удаляются ненужные разделы (при этом данные на них теряются безвозвратно) – команда `d`.

Создается новый раздел – команда `n`. При этом будет запрошены: тип раздела (`p` – первичный, `e` – расширенный, `l` – логический), номер раздела, первый и последний цилиндры раздела; при этом последний цилиндр нового раздела можно указать абсолютно или, после знака `+`, размер раздела в килобайтах (например, `+15000k`) или мегабайтах (`+1200M`).

Если новый раздел должен иметь тип, отличный от принятого по умолчанию (`83 - Linux Native`), то необходимо указать тип раздела – команда `t` (получение списка возможных типов разделов – команда `L`).

Когда все требуемые разделы созданы, то необходимо сохранить изменения – команда `w`.

Пример:

```
Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x806fa0ce

Command (m for help): n
Partition type
p primary (0 primary, 0 extended, 4 free)
e extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-41943039, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-41943039, default
41943039): +5G

Created a new partition 1 of type 'Linux' and of size 5 GiB.

Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
```

```
Disk identifier: 0x806fa0ce

Device Boot Start End Sectors Size Id Type
/dev/sda1 2048 10487807 10485760 5G 83 Linux

Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): L

0 Empty 24 NEC DOS 81 Minix / old Lin bf Solaris
1 FAT12 27 Hidden NTFS Win 82 Linux swap / So c1 DRDOS/sec (FAT-
2 XENIX root 39 Plan 9 83 Linux c4 DRDOS/sec (FAT-
3 XENIX usr 3c PartitionMagic 84 OS/2 hidden or c6 DRDOS/sec (FAT-
4 FAT16 <32M 40 Venix 80286 85 Linux extended c7 Syrinx
5 Extended 41 PPC PReP Boot 86 NTFS volume set da Non-FS data
6 FAT16 42 SFS 87 NTFS volume set db CP/M / CTOS / .
7 HPFS/NTFS/exFAT 4d QNX4.x 88 Linux plaintext de Dell Utility
8 AIX 4e QNX4.x 2nd part 8e Linux LVM df BootIt
9 AIX bootable 4f QNX4.x 3rd part 93 Amoeba e1 DOS access
a OS/2 Boot Manag 50 OnTrack DM 94 Amoeba BBT e3 DOS R/O
b W95 FAT32 51 OnTrack DM6 Aux 9f BSD/OS e4 SpeedStor
c W95 FAT32 (LBA) 52 CP/M a0 IBM Thinkpad hi ea Rufus alignment
e W95 FAT16 (LBA) 53 OnTrack DM6 Aux a5 FreeBSD eb BeOS fs
f W95 Ext'd (LBA) 54 OnTrackDM6 a6 OpenBSD ee GPT
10 OPUS 55 EZ-Drive a7 NeXTSTEP ef EFI (FAT-12/16/
11 Hidden FAT12 56 Golden Bow a8 Darwin UFS f0 Linux/PA-RISC b
12 Compaq diagnost 5c Priam Edisk a9 NetBSD f1 SpeedStor
14 Hidden FAT16 <3 61 SpeedStor ab Darwin boot f4 SpeedStor
16 Hidden FAT16 63 GNU HURD or Sys af HFS / HFS+ f2 DOS secondary
17 Hidden HPFS/NTF 64 Novell Netware b7 BSDI fs fb VMware VMFS
18 AST SmartSleep 65 Novell Netware b8 BSDI swap fc VMware VMKCORE
1b Hidden W95 FAT3 70 DiskSecure Mult bb Boot Wizard hid fd Linux
raid auto
1c Hidden W95 FAT3 75 PC/IX bc Acronis FAT32 L fe LANstep
1e Hidden W95 FAT1 80 Old Minix be Solaris boot ff BBT
...
Hex code (type L to list all codes): 82
Changed type of partition 'Linux' to 'Linux swap / Solaris'.

Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x806fa0ce

Device Boot Start End Sectors Size Id Type
/dev/sda1 2048 10487807 10485760 5G 82 Linux swap / Solaris
```

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
[76896.597480] sda: sda1
Syncing disks.
```

При изменении типа таблицы разделов все существующие разделы удаляются.

Для создания GPT разделов нужно явно сменить тип разбиения командой `g`.

Пример: изменение таблицы разделов на GPT и создание нового GPT раздела.

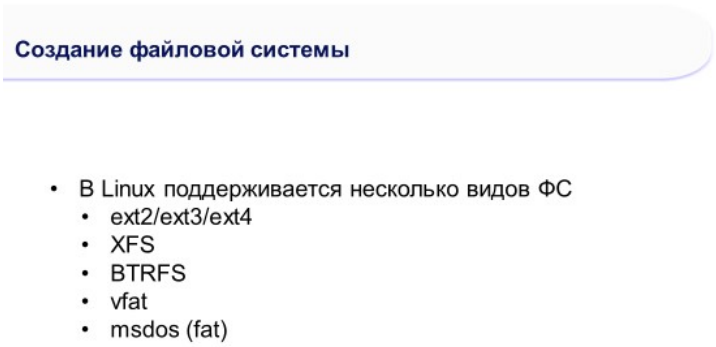
```
# fdisk /dev/sda

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write
them.
Be careful before using the write command.

[77210.824171] sda: sda1

Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x806fa0ce

Device Boot Start End Sectors Size Id Type
/dev/sda1 2048 10487807 10485760 5G 82 Linux swap / Solaris

Command (m for help): g
Created a new GPT disklabel (GUID: 419B9A4F-B2BD-9D47-A19F-
7466E0249CC5).
The old dos signature will be removed by a write command.

Command (m for help): p

Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
```

```
Disk identifier: 419B9A4F-B2BD-9D47-A19F-7466E0249CC5

Command (m for help): n
Partition number (1-128, default 1): 1
First sector (2048-41943006, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-41943006, default
41943006): +5G

Created a new partition 1 of type 'Linux filesystem' and of size 5
GiB.

Command (m for help): p
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 419B9A4F-B2BD-9D47-A19F-7466E0249CC5

Device Start End Sectors Size Type
/dev/sda1 2048 10487807 10485760 5G Linux filesystem

Command (m for help): t
Selected partition 1
Partition type (type L to list all types): L
1 EFI System C12A7328-F81F-11D2-BA4B-00A0C93EC93B
2 MBR partition scheme 024DEE41-33E7-11D3-9D69-0008C781F39F
3 Intel Fast Flash D3BFE2DE-3DAF-11DF-BA40-E3A556D89593
4 BIOS boot 21686148-6449-6E6F-744E-656564454649
5 Sony boot partition F4019732-066E-4E12-8273-346C5641494F
6 Lenovo boot partition BFBFAFE7-A34F-448A-9A5B-6213EB736C22
7 PowerPC PReP boot 9E1A2D38-C612-4316-AA26-8B49521E5A8B
8 ONIE boot 7412F7D5-A156-4B13-81DC-867174929325
9 ONIE config D4E6E2CD-4469-46F3-B5CB-1BFF57AFC149
10 Microsoft reserved E3C9E316-0B5C-4DB8-817D-F92DF00215AE
11 Microsoft basic data EBD0A0A2-B9E5-4433-87C0-68B6B72699C7
12 Microsoft LDM metadata 5808C8AA-7E8F-42E0-85D2-E1E90434CFB3
13 Microsoft LDM data AF9B60A0-1431-4F62-BC68-3311714A69AD
14 Windows recovery environment DE94BBA4-06D1-4D40-A16A-
BFD50179D6AC
15 IBM General Parallel Fs 37AFFC90-EF7D-4E96-91C3-2D7AE055B174
16 Microsoft Storage Spaces E75CAF8F-F680-4CEE-AFA3-B001E56EFC2D
17 HP-UX data 75894C1E-3AEB-11D3-B7C1-7B03A0000000
18 HP-UX service E2A1E728-32E3-11D6-A682-7B03A0000000
19 Linux swap 0657FD6D-A4AB-43C4-84E5-0933C84B4F4F
20 Linux filesystem 0FC63DAF-8483-4772-8E79-3D69D8477DE4
21 Linux server data 3B8F8425-20E0-4F3B-907F-1A25A76F98E8
22 Linux root (x86) 44479540-F297-41B2-9AF7-D131D5F0458A
23 Linux root (ARM) 69DAD710-2CE4-4E3C-B16C-21A1D49ABED3
```

```
24 Linux root (x86-64) 4F68BCE3-E8CD-4DB1-96E7-FBCAF984B709
25 Linux root (ARM-64) B921B045-1DF0-41C3-AF44-4C6F280D3FAE
26 Linux root (IA-64) 993D8D3D-F80E-4225-855A-9DAF8ED7EA97
27 Linux reserved 8DA63339-0007-60C0-C436-083AC8230908
28 Linux home 933AC7E1-2EB4-4F13-B844-0E14E2AEF915
29 Linux RAID A19D880F-05FC-4D3B-A006-743F0F84911E
30 Linux extended boot BC13C2FF-59E6-4262-A352-B275FD6F7172
31 Linux LVM E6D6D379-F507-44C2-A23C-238F2A3DF928
32 FreeBSD data 516E7CB4-6ECF-11D6-8FF8-00022D09712B
33 FreeBSD boot 83BD6B9D-7F41-11DC-BE0B-001560B84F0F
34 FreeBSD swap 516E7CB5-6ECF-11D6-8FF8-00022D09712B
35 FreeBSD UFS 516E7CB6-6ECF-11D6-8FF8-00022D09712B
36 FreeBSD ZFS 516E7CBA-6ECF-11D6-8FF8-00022D09712B
37 FreeBSD Vinum 516E7CB8-6ECF-11D6-8FF8-00022D09712B
38 Apple HFS/HFS+ 48465300-0000-11AA-AA11-00306543ECAC
39 Apple UFS 55465300-0000-11AA-AA11-00306543ECAC
40 Apple RAID 52414944-0000-11AA-AA11-00306543ECAC
41 Apple RAID offline 52414944-5F4F-11AA-AA11-00306543ECAC
42 Apple boot 426F6F74-0000-11AA-AA11-00306543ECAC
43 Apple label 4C616265-6C00-11AA-AA11-00306543ECAC
44 Apple TV recovery 5265636F-7665-11AA-AA11-00306543ECAC
45 Apple Core storage 53746F72-6167-11AA-AA11-00306543ECAC
46 Solaris boot 6A82CB45-1DD2-11B2-99A6-080020736631
47 Solaris root 6A85CF4D-1DD2-11B2-99A6-080020736631
48 Solaris /usr & Apple ZFS 6A898CC3-1DD2-11B2-99A6-080020736631
49 Solaris swap 6A87C46F-1DD2-11B2-99A6-080020736631
50 Solaris backup 6A8B642B-1DD2-11B2-99A6-080020736631
51 Solaris /var 6A8EF2E9-1DD2-11B2-99A6-080020736631
52 Solaris /home 6A90BA39-1DD2-11B2-99A6-080020736631
53 Solaris alternate sector 6A9283A5-1DD2-11B2-99A6-080020736631
54 Solaris reserved 1 6A945A3B-1DD2-11B2-99A6-080020736631
55 Solaris reserved 2 6A9630D1-1DD2-11B2-99A6-080020736631
56 Solaris reserved 3 6A980767-1DD2-11B2-99A6-080020736631
57 Solaris reserved 4 6A96237F-1DD2-11B2-99A6-080020736631
58 Solaris reserved 5 6A8D2AC7-1DD2-11B2-99A6-080020736631
59 NetBSD swap 49F48D32-B10E-11DC-B99B-0019D1879648
60 NetBSD FFS 49F48D5A-B10E-11DC-B99B-0019D1879648
61 NetBSD LFS 49F48D82-B10E-11DC-B99B-0019D1879648
62 NetBSD concatenated 2DB519C4-B10E-11DC-B99B-0019D1879648
63 NetBSD encrypted 2DB519EC-B10E-11DC-B99B-0019D1879648
64 NetBSD RAID 49F48DAA-B10E-11DC-B99B-0019D1879648
65 ChromeOS kernel FE3A2A5D-4F32-41A7-B725-ACCC3285A309
66 ChromeOS root fs 3CB8E202-3B7E-47DD-8A3C-7FF2A13CFCEC
67 ChromeOS reserved 2E0A753D-9E48-43B0-8337-B15192CB1B5E
68 MidnightBSD data 85D5E45A-237C-11E1-B4B3-E89A8F7FC3A7
69 MidnightBSD boot 85D5E45E-237C-11E1-B4B3-E89A8F7FC3A7
70 MidnightBSD swap 85D5E45B-237C-11E1-B4B3-E89A8F7FC3A7
71 MidnightBSD UFS 0394EF8B-237E-11E1-B4B3-E89A8F7FC3A7
72 MidnightBSD ZFS 85D5E45D-237C-11E1-B4B3-E89A8F7FC3A7
```

```
73 MidnightBSD Vinum 85D5E45C-237C-11E1-B4B3-E89A8F7FC3A7
74 Ceph Journal 45B0969E-9B03-4F30-B4C6-B4B80CEFF106
75 Ceph Encrypted Journal 45B0969E-9B03-4F30-B4C6-5EC00CEFF106
76 Ceph OSD 4FBD7E29-9D25-41B8-AFD0-062C0CEFF05D
77 Ceph crypt OSD 4FBD7E29-9D25-41B8-AFD0-5EC00CEFF05D
78 Ceph disk in creation 89C57F98-2FE5-4DC0-89C1-F3AD0CEFF2BE
79 Ceph crypt disk in creation 89C57F98-2FE5-4DC0-89C1-
5EC00CEFF2BE
80 VMware VMFS AA31E02A-400F-11DB-9590-000C2911D1B8
81 VMware Diagnostic 9D275380-40AD-11DB-BF97-000C2911D1B8
82 VMware Virtual SAN 381CFCCC-7288-11E0-92EE-000C2911D0B2
83 VMware Virsto 77719A0C-A4A0-11E3-A47E-000C29745A24
84 VMware Reserved 9198EFFC-31C0-11DB-8F78-000C2911D1B8
85 OpenBSD data 824CC7A0-36A8-11E3-890A-952519AD3F61
86 QNX6 file system CEF5A9AD-73BC-4601-89F3-CDEEEEE321A1
87 Plan 9 partition C91818F9-8025-47AF-89D2-F030D7000C2C

Partition type (type L to list all types): 20
Changed type of partition 'Linux filesystem' to 'Linux
filesystem'.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
[77277.892973] sda: sda1
Syncing disks.
```

11.3. Создание файловой системы.



Для обеспечения возможности хранения информации в виде файлов в разделе должна быть создана файловая система, то есть должно быть произведено форматирование раздела.

При форматировании формируются суперблок, массив индексных дескрипторов и выделяется пространство для блоков данных.

Изначально в GNU/Linux основной файловой системой являлась EXT2. В настоящий момент помимо EXT2 широко используются следующие файловые системы:

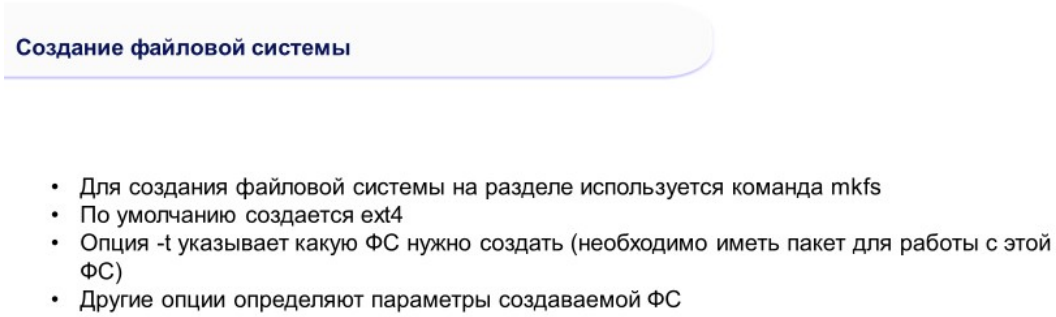
1. EXT3 - осовремененная версия EXT2 с поддержкой журналирования и с улучшенными показателями времени восстановления после сбоя, продвигаемая Red Hat;
2. EXT4 - следующая ступень развития файловых систем EXT;
3. XFS - высокопроизводительная файловая система для больших и очень больших объемов хранимой информации, разработанная в Silicon Graphics, распространяемая как Open Source (начиная с ядра 2.4.24 ее поддержка имеется в ядре Linux без необходимости установки специального патча);
4. BTRFS - современная файловая система, обеспечивающая такие возможности как отказоустойчивость на уровне ФС, снимки, оптимизация для дисков SSD, динамическое выделение inode.

5. JFS - высокопроизводительная файловая система от IBM, используемая при необходимости хранения очень больших объемов информации.

Чтобы подключить и использовать определенную файловую систему нужна поддержка этой ФС в ядре.

Для создания ФС поддержка со стороны ядра не обязательна, но необходимо иметь соответствующие утилиты для обслуживания этой ФС.

Примечание: если у вас есть подходящие утилиты вы можете создать на диске какую угодно ФС, но не факт, что вы сможете ее использовать (монтировать).



Для создания файловой системы используется команда `mkfs`, при вызове которой в качестве аргумента должен быть задан файл устройства, соответствующий разделу жесткого диска (могут быть использованы файлы устройств для дискет и прочих блочных устройств).

Команда `mkfs` по умолчанию создает файловую систему EXT4.

Пример: для создания файловой системы EXT4 на втором первичном разделе диска следует выполнить такую команду (форматирование дисков - привилегия администратора):

```
# mkfs /dev/sda2
```

При успешном выполнении команды `mkfs` на экран будет выведена информация о размере блоков и их количестве в данной файловой системе, о местоположении копий суперблока и прочее.

Для создания иной файловой системы ее тип необходимо указать после опции `-t` команды `mkfs`.

Пример: для создания на том же разделе диска файловой системы EXT2 надо выполнить такую команду:

```
# mkfs -t ext2 /dev/sda2
```

Вместо опции `-t` для форматирования файловых систем можно использовать специализированные утилиты:

1. `mke2fs` - для создания EXT2, EXT3 и EXT4 файловых систем;
2. `mkntfs` - для создания NTFS;
3. `mkdosfs` - для форматирования под FAT.

Альтернативный способ создания разных файловых систем – использование команд `mkfs.*` :

```
1./sbin/mkfs.ext2
2./sbin/mkfs.ext3
3./sbin/mkfs.ext4
4./sbin/mkfs.xfs
```

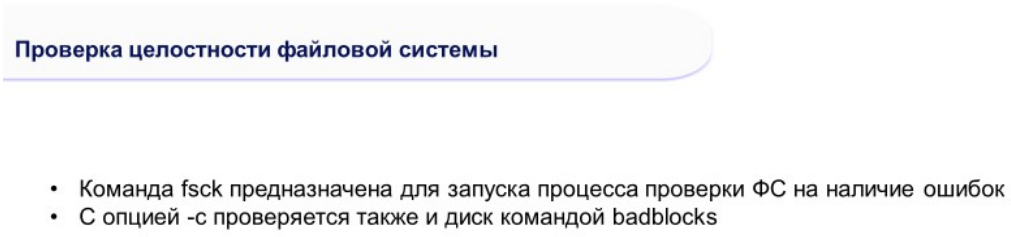
Примечание: Эти команды, в свою очередь, являются символьными ссылками, либо же жесткими связями со специализированными утилитами, упомянутыми выше.

Пример:

```
# ls -F /sbin/mkfs.*
/sbin/mkfs.btrfs*         /sbin/mkfs.ext4*         /sbin/mkfs.minix*
/sbin/mkfs.vfat@
/sbin/mkfs.cramfs*         /sbin/mkfs.fat*         /sbin/mkfs.msdos@
/sbin/mkfs.xfs*
/sbin/mkfs.exfat@ /sbin/mkfs.gfs2* /sbin/mkfs.ntfs@
/sbin/mkfs.ext2* /sbin/mkfs.hfsplus* /sbin/mkfs.reiserfs@
/sbin/mkfs.ext3* /sbin/mkfs.jffs2* /sbin/mkfs.ubifs*
```

Команда `mkfs` предоставляет возможность использовать опцию `-c`, которая заставляет команду перед созданием файловой системы проверять поверхность диска на наличие плохих блоков.

11.4. Проверка целостности файловой системы.



Целостность структуры и данных файловой системы может быть нарушена в результате сбоя системы.

Примечание: например, ставшего следствием сбоя питания или же аппаратной неисправности.

Если работа операционной системы была прервана и файловая система не была размонтирована, то в суперблоке файловой системы остается специальный флаг (dirty – грязный или not clean), который сообщает о том, что в файловой системе возможны нарушения.

Если файловая система была размонтирована правильно и сбоев не было, то говорят, что файловая система находится в состоянии clean - “чистая”.

Сбой файловой системы обычно сопровождается тем, что информация, находящаяся в кэше (дисковых буферах), не синхронизируется с информацией в файловой системе.

Сбои системы могут привести к следующим проблемам в файловой системе:

1. Искажению или потере информации в блоках данных
2. Появлению в системе блоков данных, которые считаются занятыми, хотя на них не указывает ни один индексный дескриптор.
3. Наличию перекрестных ссылок на блоки данных.

4. Появлению метаданных с ненулевым счетчиком ссылок, на которые не ссылаются никакие файлы.
5. Наличию противоречивых записей в каталогах и т.п.

Различают два класса возможных нарушений в файловой системе:

1. Нарушение целостности данных.
2. Нарушение целостности структуры файловой системы.

Для защиты целостности данных могут быть использованы разнообразные методы резервного хранения данных (backup).

Примечание: Утилиты восстановления целостности файловой системы не могут гарантировать восстановление данных после сбоя. Гарантом сохранности данных является только наличие правильно выполненных резервных копий данных.

Для восстановления структуры файловых систем в GNU/Linux используется утилита `fsck`

Для проверки различных типов файловых систем используются специализированные утилиты `fsck.*` является или опция `-t`

Внимание! Проверка целостности файловой системы должна осуществляться лишь на размонтированной файловой системе. Невыполнение этого требования может привести к полной потере данных на файловой системе!

Пример:

```
# ls -F /sbin/fsck.*
/sbin/fsck.btrfs*          /sbin/fsck.ext3*          /sbin/fsck.hfs@
/sbin/fsck.ntfs@
/sbin/fsck.cramfs*       /sbin/fsck.ext4*       /sbin/fsck.hfsplus*
/sbin/fsck.reiserfs@
/sbin/fsck.exfat@         /sbin/fsck.fat*         /sbin/fsck.minix*
/sbin/fsck.vfat@
/sbin/fsck.ext2*         /sbin/fsck.gfs2*         /sbin/fsck.msdos@
/sbin/fsck.xfs*
```

Если вызвать утилиту fsck без опции -t, указывающей тип файловой системы, то будет вызвана утилита e2fsck, предназначенная для проверки файловой системы EXT.

Пример:

```
# fsck.ext3 /dev/sda1
e2fsck 1.44.6 (5-Mar-2019)
/dev/sda1: clean, 11/327680 files, 39535/1310720 blocks
```

Примечание: Эта команда проверит целостность файловой системы ext3 на первом первичном разделе первого SCSI диска в системе.

Полная проверка осуществляется не всегда, а только при условии:

1. Наличия флага dirty в суперблоке, что бывает при сбое или выключении питания без размонтирования файловой системы.
2. Достижения максимального разрешенного количества монтирований файловой системы без проверки ее целостности.
3. Достижения максимального срока без проверки целостности файловой системы.

Если необходимо выполнить полную проверку файловой системы в отсутствии любого из приведенных выше условий, то надо использовать опцию `-f` команды `e2fsck`.

Для проверки поверхности диска перед проверкой файловой системы требуется использовать опцию `-c` команды `e2fsck`.

Пример: приведенная ниже команда выполнит полную проверку файловой системы, так как превышено максимально разрешенное для данной файловой системы число монтирований без проверки целостности файловой системы.

```
# e2fsck /dev/sda1
e2fsck 1.32 (09-Nov-2002)
USBDISK has been mounted 27 times without being checked, check
forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
USBDISK:     3585/2442240     files     (2.2%     non-contiguous),
3004291/19534832 blocks
```

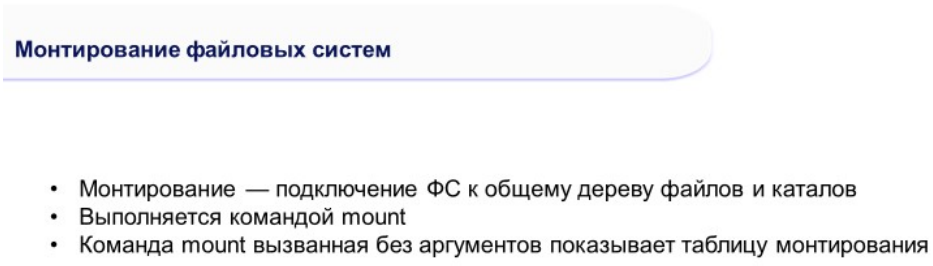
Команда `e2fsck` используется специальный каталог `lost+found`, находящийся в корневом (верхнем) каталоге файловой системы устройства для сохранения потерянных цепочек блоков данных (файлов, у которых нет имени).

Для проверки целостности файловой системы XFS применяется команда `xfs_repair`.

Пример:

```
# xfs_repair /dev/sda2
Phase 1 - find and verify superblock...
Phase 2 - using internal log
- zero log...
- scan filesystem freespace and inode maps...
- found root inode chunk
Phase 3 - for each AG...
- scan and clear agi unlinked lists...
- process known inodes and perform inode discovery...
- agno = 0
- agno = 1
- agno = 2
- agno = 3
- process newly discovered inodes...
Phase 4 - check for duplicate blocks...
- setting up duplicate extent list...
- check for inodes claiming duplicate blocks...
- agno = 0
- agno = 1
- agno = 2
- agno = 3
Phase 5 - rebuild AG headers and trees...
- reset superblock...
Phase 6 - check inode connectivity...
- resetting contents of realtime bitmap and summary inodes
- traversing filesystem ...
- traversal finished ...
- moving disconnected inodes to lost+found ...
Phase 7 - verify and correct link counts...
done
```

11.5. Монтирование файловых систем.



В GNU/Linux все файловые системы, доступные для работы с ними, должны быть “подцеплены” к логической структуре файлов и каталогов.

Процесс “подцепления” файловой системы, существующей на дисковом или ином блочном устройстве, в общее дерево файлов и каталогов называется монтированием.

Каталог, в который произошло “подцепление” отформатированного устройства, называется точкой монтирования.

За исключением файловых систем, для которых установлены специальные настройки в, например, файле `/etc/fstab`, монтирование файловых систем производится суперпользователем.

Стандарт FHS предписывает, что точки монтирования временных файловых систем должны находиться в каталоге `/mnt`.

Временные файловые системы для сменных носителей должны в соответствии с FHS находится в каталоге `/media`

Примечание: каталог /media/cdrom может быть точкой монтирования для CD-ROM, а /media/usbflash – для флоппи диска.

Команда `mount` монтирует файловую систему указанного с помощью опции `-t` типа (по умолчанию ext2) в каталог - точку монтирования.

Первый аргумент команды `mount` - файл блочного устройства, на котором находится монтируемая файловая система.

Второй аргумент - точка монтирования этой файловой системы.

Пример: для монтирования USB диска с файловой системой EXT2 следует выполнить команду:

```
# mount /dev/sda1 /mnt/usbflash
```

Если команда `mount` вызвана без аргументов, то она показывает список смонтированных файловых систем, то есть имена файлов устройств и соответствующих им точек монтирования.

Пример:

```
# mount | grep ^/dev
/dev/vda2 on / type ext4 (rw,relatime,seclabel)
/dev/vda1 on /boot type ext4 (rw,relatime,seclabel)
/dev/sda1 on /media/usbflash type ext3 (rw,relatime,seclabel)
```

Последние версии `mount` умеют сами определять тип подключаемой ФС, если нет, то при монтировании, например, CD ROM необходимо указать тип файловой системы iso9660 :

Пример:

```
# mount -t iso9660 /dev/cdrom /media/cdrom
```

Монтирование файловой системы подменяет индексный дескриптор каталога - точки монтирования. До монтирования индексный дескриптор соответствует каталогу, находящемуся в файловой системе, к которой монтируется устройство. После монтирования - индексный дескриптор каталога - точки монтирования принадлежит уже смонтированной файловой системе.

Пример:

```
# ls -ldi /media/usbflash/
529793 drwxr-xr-x. 2 root root 4096 Feb 10 12:13 /media/usbflash/

# mount /dev/sda1 /media/usbflash
```

```
# ls -ldi /media/usbflash/
2 drwxr-xr-x. 3 root root 4096 Feb 10 08:47 /media/usbflash/
```

Примечание: Из этого примера заметно, что до монтирования временной файловой системы в каталог /media/usbdisk, индексный дескриптор каталога был 529793 , а после монтирования 2. У всех корневых каталогов файловых систем на диске индексный дескриптор имеет номер 2. У виртуальных файловых систем корневой каталог имеет номер inode 1.

Пример:

```
# ls -ldi /boot /
2 dr-xr-xr-x. 18 root root 4096 Nov 9 2019 /
2 dr-xr-xr-x. 6 root root 4096 Feb 9 10:39 /boot

# mount | grep ^/dev
/dev/vda2 on / type ext4 (rw,relatime,seclabel)
/dev/vda1 on /boot type ext4 (rw,relatime,seclabel)
/dev/sda1 on /media/usbflash type ext3 (rw,relatime,seclabel)

# ls -ldi /proc
1 dr-xr-xr-x. 91 root root 0 Feb 9 10:50 /proc
```

Примечание: В этом примере демонстрируется то, что для каждой файловой системы inode корневого каталога, то есть точки монтирования - 2.

Как и операцию монтирования, размонтировать файловые системы (без специальных настроек в `/etc/fstab`) имеет право только суперпользователь.

Для размонтирования файловой системы применяется команда `umount`, которой в качестве аргумента должен быть задан единственный аргумент - либо точка монтирования, либо файл устройства.

Пример:

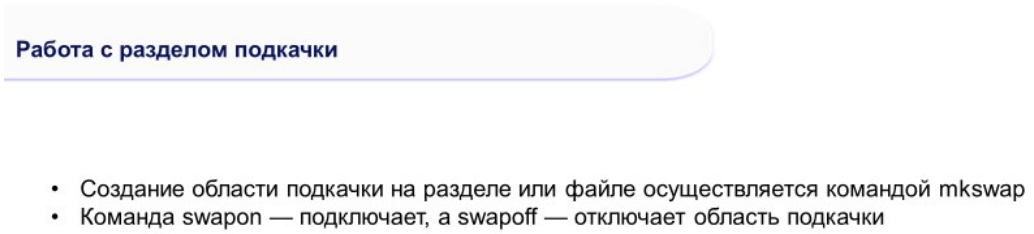
```
# ls /media/usbflash/
lost+found somedocs.txt
[root@lin00 ~]# umount /media/usbflash/
[root@lin00 ~]# ls /media/usbflash/
```

Примечание: В этом примере показана работа команды umount. После ее выполнения в каталоге - точке монтирования больше нет доступа к файлам на временной файловой системе.

Хорошим правилом является следующее: не следует хранить какие-либо

файлы в каталогах, являющихся точками монтирования временных файловых систем.

11.6. Работа с разделом подкачки.



Раздел или файл подкачки необходимы при работе GNU/Linux для обеспечения временного перемещения страниц памяти из ОЗУ в этот раздел или файл.

Такое перемещение происходит при недостатке физической памяти.

Процесс обмена страницами памяти между ОЗУ и разделом подкачки называется swapping, а раздел подкачки называется swap – разделом.

Получить информацию об использовании раздела подкачки можно с помощью команды `swapon -s`.

Пример:

```
$ /sbin/swapon -s
Filename Type Size Used Priority
/dev/hda5 partition 248968 0 -1
```

Примечание: Информация, полученная от команды `swapon -s`, демонстрирует следующее: имеется раздел подкачки в первом логическом разделе Primary Master IDE диска. Размер раздела – 248 Мб, из них использовано в настоящий момент – 0.

Столбец приоритет отображает порядок использования swap разделов. Сначала будут использованы разделы подкачки с большим номером приоритета, затем – с меньшим.

При необходимости можно добавить в систему дополнительные области подкачки. Они могут быть размещены как в разделах дисков, так и в обычных файлах. Создавать, подключать и отключать разделы подкачки имеет право суперпользователь.

Примечание: Если в системе не хватает ОЗУ для выполнения каких-либо приложений, то создание раздела подкачки может быть единственно возможным методом решения этой проблемы.

Если в системе имеется несколько жестких дисков, то рекомендуется для оптимизации производительности системы разместить разделы подкачки на нескольких дисках. Для создания файла подкачки необходимо создать файл, заполненный нулями.

Пример: Приведенная ниже команда создает 128 Мб файл, заполненный нулями:

```
$ dd if=/dev/zero of=swap.file bs=1k count=131072
131072+0 входных записей
131072+0 выходных записей
$ ls -l swap.file
-rw-r--r-- 1 user1 user1 134217728 Дек 19 17:43 swap.file
```

Команда `mkswap` создает в файле или разделе (указанном при помощи файла устройства) область подкачки, специальным образом размечая ее.

Пример:

```
$ /sbin/mkswap swap.file
Setting up swapspace version 1, size = 134213 kB
```

Примечание: Эта команда создала в файле swap.file область подкачки.

Создать область подкачки, в разделе можно лишь тогда, когда тип раздела установлен Linux Swap (82 тип в команде fdisk).

Для создания раздела подкачки в разделе выполняется та же команда `mkswap`

Пример:

```
# mkswap -c /dev/sda2
```

Примечание: В этом примере создается раздел подкачки на втором первичном разделе первого SCSI диска. При этом используется опция `-c`, которая перед созданием области подкачки проверяет поверхность диска на наличие плохих блоков.

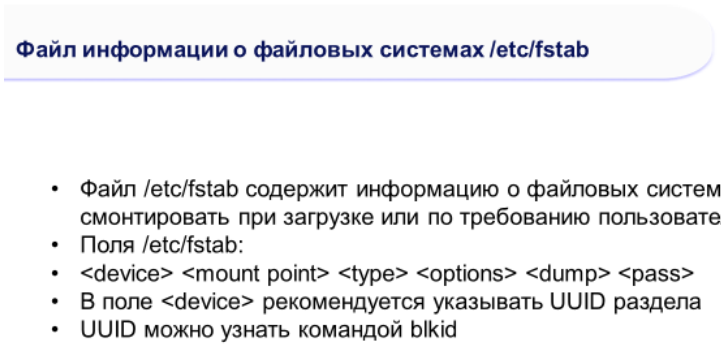
Подключить созданный раздел или файл подкачки можно с помощью команды `swapon` , а отключить – с помощью команды `swapoff`.

Пример:

```
# swapon ~user1/swap.file

# swapon -s
Filename Type Size Used Priority
/dev/sda5 partition 248968 0 -1
/home/user1/swap.file file 131064 0 -2

# swapoff ~user1/swap.file

# swapon -s
Filename Type Size Used Priority
/dev/sda5 partition 248968 0 -1
```

Примечание: В этом примере был подключен дополнительный файл подкачки с помощью команды `swapon`, а затем этот файл был отключен командой `swapoff`.

11.7. Файл информации о файловых системах /etc/fstab.



Конфигурационный файл `/etc/fstab` (filesystem table) содержит информацию о файловых системах, которые нужно смонтировать при загрузке или по требованию пользователя.

Информация в файле `/etc/fstab` представлена в виде таблицы:

Пример:

```
# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Sat Nov 9 04:18:20 2019
#
# Accessible filesystems, by reference, are maintained under
'/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for
more info.
#
# After editing this file, run 'systemctl daemon-reload' to update
systemd
# units generated from this file.
#
UUID=1a5655e7-613b-4733-ba6b-e598633402fb / ext4 defaults 1 1
UUID=897234d2-2307-406f-990d-a9620bcb4d1f /boot ext4 defaults 1 2
UUID=5a4f53ca-c983-4def-8535-e2541d8419bb swap swap defaults 0 0
```

Строки, начинающиеся с символа # являются комментариями.

Таблица, содержащаяся в файле `/etc/fstab` , состоит из колонок полей, назначение которых следующее:

Первое поле (`fs_spec`) содержит указатель на монтируемое устройство. Здесь может быть имя файла устройства, UUID файловой системы, UUD GPT, метка GPT или метка тома в файловой системе.

Второе (`fs_file`) указывает каталог - точку монтирования

Третье (`fs_vfstype`) содержит тип файловой системы на данном носителе.

Четвертое (`fs_mntops`) содержит опции команды mount , которые должны быть использованы при монтировании данной файловой системы. Для разделов swap это поле должно содержать sw.

Пятое (`fs_freq`) указывает надо ли для данной файловой системы производить автоматическое резервное копирование (backup) командой dump . Если в этом поле находится 1, то резервное копирование производится, если 0 – нет.

Шестое (`fs_passno`) предназначено для порядка проверки целостности файловых систем при загрузке операционной системы. Для корневой файловой системы в этом поле должно быть установлено значение 1. Для других файловых систем, которые необходимо проверять при загрузке, следует указать 2. Если проверка не требуется, то в этом поле ставится 0 .

Ниже приведена таблица, содержащая часто используемые опции команды `mount`, указываемые в поле `fs_mntops` файла `/etc/fstab` .

Опция	Назначение
`defaults`	Установки: `rw,suid,dev,exec,auto,nouser,asynch`.
`asynch`	Асинхронный режим ввода/вывода.
`auto`	Монтировать во время загрузки.
`noauto`	Не монтировать во время загрузки.
`exec`	Разрешение выполнения файлов с машинным кодом.
`noexec`	Запрет выполнения файлов с машинным кодом.
`suid`	Бит SUID устанавливать можно.
`nosuid`	Запрещена установка битов SUID.
`user`	Обычный пользователь может монтировать устройство.
`nouser`	Монтировать разрешено только суперпользователю.
`ro`	Режим только для чтения.
`rw`	Разрешено как чтение, так и запись.

Наличие записи в файле /etc/fstab означает, что данная файловая система может быть смонтирована без указания обоих аргументов командной строки mount – файла устройства и каталога – точки монтирования. Для монтирования файловой системы, указанной в /etc/fstab , достаточно указать либо точку монтирования, либо файл устройства.

В ОС построенных на systemd для монтирования разделов на основе /etc/fstab генерируются специальные юниты типа mount.

```
# systemctl list-units -t mount
UNIT LOAD ACTIVE SUB DESCRIPTION
-.mount loaded active mounted Root Mount
boot.mount loaded active mounted /boot
dev-hugepages.mount loaded active mounted Huge Pages File System
dev-mqueue.mount loaded active mounted POSIX Message Queue File
System
run-user-0.mount loaded active mounted /run/user/0
sys-kernel-config.mount loaded active mounted Kernel Configuration
File System
sys-kernel-debug.mount loaded active mounted Kernel Debug File
System

LOAD = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization
of SUB.
SUB = The low-level unit activation state, values depend on unit
type.

7 loaded units listed. Pass --all to see loaded but inactive
units, too.
To show all installed unit files use 'systemctl list-unit-files'.

# systemctl status [-].mount
● -.mount - Root Mount
Loaded: loaded (/etc/fstab; generated)
Active: active (mounted) since Tue 2021-02-09 10:50:18 +05; 1 day
6h ago
Where: /
What: /dev/vda2
Docs: man:fstab(5)
man:systemd-fstab-generator(8)
```

Юнит для монтирования можно создать самостоятельно или написать специальные опции монтирования в файл /etc/fstab , которые сформируют юнит монтирования с нужными вам параметрами. (см. man 5 systemd.mount).

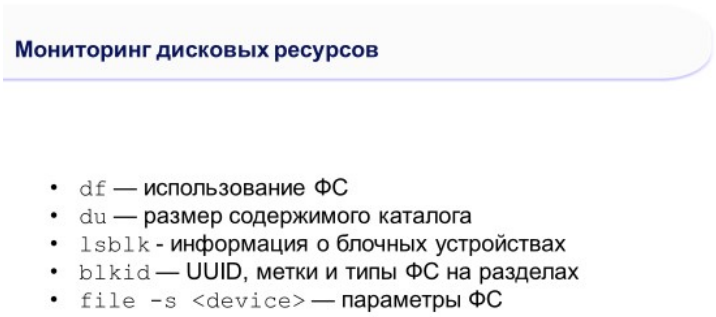
Использование `/etc/fstab` считается предпочтительным.

Пример: создадим юнит, который описывает новую точку монтирования:

```
# mkfs.ext4 /dev/sda1
mke2fs 1.44.6 (5-Mar-2019)
/dev/sda1 contains a ext3 file system
last mounted on /media/usbflash on Wed Feb 10 12:16:19 2021
Proceed anyway? (y,N) y
Creating filesystem with 1310720 4k blocks and 327680 inodes
Filesystem UUID: a13a401f-cdf5-48e3-a277-21409ad76de0
Superblock backups stored on blocks:
...

# mkdir /storage1

# vi /etc/systemd/system/storage1.mount
# cat /etc/systemd/system/storage1.mount
[Unit]
Description=Persistent Mount Point for directory /storage1

[Mount]
What=/dev/disk/by-uuid/a13a401f-cdf5-48e3-a277-21409ad76de0
Where=/storage1
Type=ext4
Options=defaults

[Install]
WantedBy=sysinit.target

# systemctl start storage1.mount

# systemctl status storage1.mount
● storage1.mount - Persistent Mount Point for directory /storage1
Loaded:   loaded   (/etc/systemd/system/storage1.mount;   disabled;
vendor preset:>
Active: active (mounted) since Wed 2021-02-10 17:49:04 +05; 6s ago
Where: /storage1
What: /dev/sda1
Tasks: 0 (limit: 12472)
Memory: 76.0K
CGroup: /system.slice/storage1.mount

Feb 10 17:49:04 lin00 systemd[1]: Mounting Persistent Mount Point
for directory>
Feb 10 17:49:04 lin00 systemd[1]: Mounted Persistent Mount Point
for directory
```

```
# systemctl enable storage1.mount
Created                                                    symlink
/etc/systemd/system/sysinit.target.wants/storage1.mount          →
/etc/systemd/system/storage1.mount.

# systemctl status storage1.mount
● storage1.mount - Persistent Mount Point for directory /storage1
Loaded:   loaded   (/etc/systemd/system/storage1.mount;   enabled;
vendor preset: >
Active: active (mounted) since Wed 2021-02-10 17:49:04 +05; 2min
17s ago
...
```

11.8. Мониторинг дисковых ресурсов.



Команда `lsblk` выдает информацию о дисках, разделах и точках монтирования.

Пример:

```
# lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 20G 0 disk
├─sda1 8:1 0 5G 0 part /storage1
└─sda2 8:2 0 2G 0 part /storage2
sr0 11:0 1 1024M 0 rom
vda 253:0 0 20G 0 disk
├─vda1 253:1 0 1G 0 part /boot
├─vda2 253:2 0 17G 0 part /
└─vda3 253:3 0 2G 0 part [SWAP]

# lsblk -f
NAME FSTYPE LABEL UUID MOUNTPOINT
sda
├─sda1 ext4 a13a401f-cdf5-48e3-a277-21409ad76de0 /storage1
└─sda2 xfs e5b66aa7-c101-4bcd-bb63-2d10d2e4aa44 /storage2
sr0
vda
├─vda1 ext4 BOOT 897234d2-2307-406f-990d-a9620bcb4d1f /boot
├─vda2 ext4 ROOT 1a5655e7-613b-4733-ba6b-e598633402fb /
└─vda3 swap SWAP1 5a4f53ca-c983-4def-8535-e2541d8419bb [SWAP]
```

Команда `findmnt` показывает точки монтирования в удобном для восприятия виде.

Пример:

```
# findmnt
TARGET SOURCE FSTYPE OPTIONS
/ /dev/vda2 ext4 rw,relatime,seclabel
├─/sys sysfs sysfs rw,nosuid,nodev,noexec,
│          ├─/sys/kernel/security         securityfs         securit
rw,nosuid,nodev,noexec,
│ ├─/sys/fs/cgroup tmpfs tmpfs ro,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/systemd cgroup cgroup rw,nosuid,nodev,noexec,
│ │ │ ├─/sys/fs/cgroup/perf_event      cgroup      cgroup
rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/pids cgroup cgroup rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/freezer cgroup cgroup rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/cpuset cgroup cgroup rw,nosuid,nodev,noexec,
│ │ │ ├─/sys/fs/cgroup/net_cls,net_prio     cgroup     cgroup
rw,nosuid,nodev,noexec,
│ │ │ ├─/sys/fs/cgroup/cpu,cpuacct      cgroup      cgroup
rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/rdma cgroup cgroup rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/devices cgroup cgroup rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/memory cgroup cgroup rw,nosuid,nodev,noexec,
│ │ ├─/sys/fs/cgroup/blkio cgroup cgroup rw,nosuid,nodev,noexec,
│ │ └─/sys/fs/cgroup/hugetlb cgroup cgroup rw,nosuid,nodev,noexec,
│ ├─/sys/fs/pstore pstore pstore rw,nosuid,nodev,noexec,
│ ├─/sys/fs/bpf bpf bpf rw,nosuid,nodev,noexec,
│ ├─/sys/fs/selinux selinuxfs selinux rw,relatime
│ ├─/sys/kernel/debug debugfs debugfs rw,relatime,seclabel
│ └─/sys/kernel/config configfs configf rw,relatime
├─/proc proc proc rw,nosuid,nodev,noexec,
│         └─/proc/sys/fs/binfmt_misc        systemd-1        autofs
rw,relatime,fd=35,pgrp=
├─/dev devtmpfs devtmpf rw,nosuid,seclabel,size
│ ├─/dev/shm tmpfs tmpfs rw,nosuid,nodev,seclabe
│ ├─/dev/pts devpts devpts rw,nosuid,noexec,relati
│ ├─/dev/mqueue mqueue mqueue rw,relatime,seclabel
│ └─/dev/hugepages hugetlbfs hugetlb rw,relatime,seclabel,pa
├─/run tmpfs tmpfs rw,nosuid,nodev,seclabe
│ └─/run/user/0 tmpfs tmpfs rw,nosuid,nodev,relatim
├─/boot /dev/vda1 ext4 rw,relatime,seclabel
├─/storage1 /dev/sda1 ext4 rw,relatime,seclabel
└─/storage2 /dev/sda2 xfs rw,relatime,seclabel,at
```

Команда `blkid` находит и печатает атрибуты блочных устройств.

Пример:

```
# blkid
/dev/vda1:        LABEL="BOOT"        UUID="897234d2-2307-406f-990d-
a9620bcb4d1f" TYPE="ext4" PARTUUID="706c87eb-01"
/dev/vda2:        LABEL="ROOT"        UUID="1a5655e7-613b-4733-ba6b-
e598633402fb" TYPE="ext4" PARTUUID="706c87eb-02"
/dev/vda3:       LABEL="SWAP1"        UUID="5a4f53ca-c983-4def-8535-
e2541d8419bb" TYPE="swap" PARTUUID="706c87eb-03"
/dev/sda1: UUID="a13a401f-cdf5-48e3-a277-21409ad76de0" TYPE="ext4"
PARTUUID="87ce5033-6121-0b42-b875-05ff23508aad"
/dev/sda2:  UUID="e5b66aa7-c101-4bcd-bb63-2d10d2e4aa44"  TYPE="xfs"
PARTUUID="fea45c49-5143-a34b-8b92-9e78bc409109"

# blkid -i /dev/sda1
/dev/sda1:     MINIMUM_IO_SIZE="512"     PHYSICAL_SECTOR_SIZE="512"
LOGICAL_SECTOR_SIZE="512"

# blkid -t TYPE="ext4"
/dev/vda1:        LABEL="BOOT"        UUID="897234d2-2307-406f-990d-
a9620bcb4d1f" TYPE="ext4" PARTUUID="706c87eb-01"
/dev/vda2:        LABEL="ROOT"        UUID="1a5655e7-613b-4733-ba6b-
e598633402fb" TYPE="ext4" PARTUUID="706c87eb-02"
/dev/sda1: UUID="a13a401f-cdf5-48e3-a277-21409ad76de0" TYPE="ext4"
PARTUUID="87ce5033-6121-0b42-b875-05ff23508aad"
```

Команда `df` выводит количество свободного места в блоках на указанном устройстве, а если оно не указано, то на всех смонтированных файловых системах.

Удобно использовать опцию `-h`, для отображения информации в понятных для пользователя единицах (human readable format):

Пример:

```
$ df -h
Filesystem Size Used Avail Use% Mounted on
devtmpfs 975M 0 975M 0% /dev
tmpfs 989M 0 989M 0% /dev/shm
tmpfs 989M 17M 973M 2% /run
tmpfs 989M 0 989M 0% /sys/fs/cgroup
/dev/vda2 17G 3.2G 13G 21% /
/dev/vda1 976M 163M 747M 18% /boot
tmpfs 198M 0 198M 0% /run/user/0
/dev/sda1 4.9G 20M 4.6G 1% /storage1
/dev/sda2 2.0G 47M 2.0G 3% /storage2
```

Для получения информации о наличии свободных индексных дескрипторов необходимо вызвать команду `df -i` :

Пример:

```
$ df -i
Filesystem Inodes IUsed IFree IUse% Mounted on
devtmpfs 249450 376 249074 1% /dev
tmpfs 253090 1 253089 1% /dev/shm
tmpfs 253090 513 252577 1% /run
tmpfs 253090 17 253073 1% /sys/fs/cgroup
/dev/vda2 1114112 138925 975187 13% /
/dev/vda1 65536 320 65216 1% /boot
tmpfs 253090 5 253085 1% /run/user/0
/dev/sda1 327680 11 327669 1% /storage1
/dev/sda2 1048576 3 1048573 1% /storage2
```

Примечание: Обозначения K и M присутствующие в листинге выше следует понимать, как тысячи и миллионы индексных дескрипторов.

Для того, чтобы узнать сколько пространства занимают файлы в каталоге следует использовать команду `du`, отображающую количество блоков, занимаемое каждым каталогом и каждым его подкаталогом.

Можно использовать `du -h` для отображения информации в удобных единицах:

Пример:

```
$ du -h /etc/rc.d
161K /etc/rc.d/init.d
1.0K /etc/rc.d/rc0.d
1.0K /etc/rc.d/rc1.d
1.0K /etc/rc.d/rc2.d
1.0K /etc/rc.d/rc3.d
1.0K /etc/rc.d/rc4.d
1.0K /etc/rc.d/rc5.d
1.0K /etc/rc.d/rc6.d
18K /etc/rc.d/scripts
207K /etc/rc.d
```

Также можно отобразить лишь суммарную информацию о каталоге, без вывода подробностей о подкаталогах. Для этого используется `du -s`

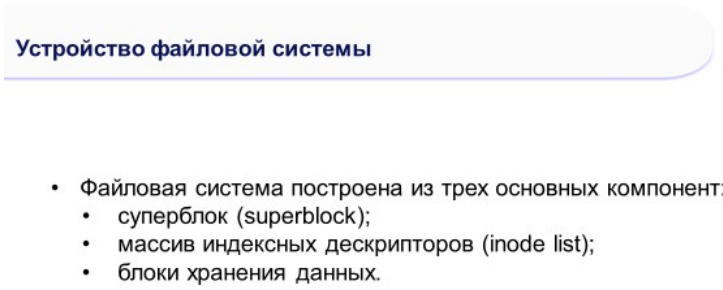
Пример:

```
$ du -sh ~
467M /home/user1
```

Примечание: В этом примере получена информация о суммарном пространстве, используемом домашним каталогом пользователя.

Глава 12. Файловая система Linux.

12.1. Устройство файловой системы.



Файловая система построена из трех основных компонент:

- суперблок (superblock);
- массив индексных дескрипторов (inode list);
- блоки хранения данных.

Суперблок содержит основную информацию, необходимую для монтирования и работы файловой системы:

- тип файловой системы;
- размер и количество блоков в файловой системе;
- количество индексных дескрипторов;
- время последнего монтирования;
- информация о том была ли размонтирована файловая система;
- счетчик числа монтирований;
- список свободных блоков и индексных дескрипторов, и т.п.

В случае если суперблок файловой системы испорчен, то монтирование файловой системы без его восстановления невозможно.

Современные файловые системы специально сохраняют в заранее известных блоках диска копии суперблока, обеспечивая возможность их использования при восстановлении структуры файловой системы.

В индексных дескрипторах хранятся метаданные (атрибуты) файлов:

1. Владелец и группа пользователей файла;
2. Права доступа к файлу;
3. Тип файла;
4. Количество имен у файла (link count);
5. Дата доступа к файлу (файл открыт на чтение);
6. Дата модификации (файл открыт на запись);
7. Дата изменения метаданных;
8. Количество блоков, занятое файлом;
9. Указатели на блоки данных файла.

Каталоги предоставляют собой особый тип файлов, содержащие таблицу, в которой содержатся имена файлов, находящихся в данном каталоге, и соответствующие им номера индексных дескрипторов (inode).

Записи, хранящиеся в каталоге, связывают имена файлов с их индексными дескрипторами, а те, в свою очередь, предоставляют информацию о местонахождении блоков данных файла.

Одному и тому же индексному дескриптору может соответствовать несколько имен файлов.

Если у одного и того же файла имеется несколько имен, то говорят, что между этими именами существует жесткая связь (hard link).

Количество разных имен файл фиксируется счетчиком имен файла (link counter) в его индексном дескрипторе.

Примечание: То есть, разные имена файлов указывают на одни метаданные, и, следовательно, на одни и те же блоки данных. Все имена файла совершенно эквивалентны и изменение содержимого этих файлов будет совершенно синхронным. Нет никакой возможности определить, какое имя у файла было исходно, а какое появилось потом.

У каждого каталога имеется как минимум два имени (то есть счетчик имен всегда не меньше двух):

1. Обычное имя каталога, находящееся в родительском каталоге (например, `/home/user1`).
2. Имя точка (.) - имя текущего каталога.

Примечание: При появлении в любом каталоге подкаталога, количество имен у этого каталога увеличивается на единицу, так как в дочернем каталоге всегда содержится имя две точки (..) - имя родительского каталога.

Количество имен у файла можно определить с помощью команды `ls -l`,

а получить номера индексных дескрипторов можно, используя `ls -i`.

Пример:

```
$ ls -ldi /etc
6 drwxr-xr-x 87 root root 6064 Дек 14 00:13 /etc
```

Примечание: Из полученного листинга видно, что номер inode каталога /etc равен 6, а количество имен у этого каталога равняется 87, следовательно, количество подкаталогов в нем равняется 85.

```
$ ls -l /etc | grep -c '^d'
85
```

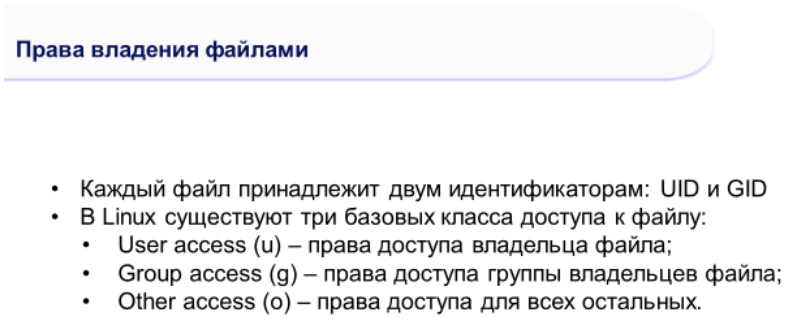
Примечание: Эта команда подтверждает, что приведенное выше правило подсчета имен каталогов верно.

Подробную информацию об индексном дескрипторе файла можно получить, используя команду stat.

Пример:

```
$ stat /etc
File: `/etc'
Size: 6064 Blocks: 12 IO Block: 4096 Directory
Device: 305h/773d Inode: 6 Links: 87
Access: (0755/drwxr-xr-x) Uid: ( 0/ root) Gid: ( 0/ root)
Access: 2003-06-17 23:19:59.000000000 +0600
Modify: 2003-12-14 00:13:23.000000000 +0500
Change: 2003-12-14 00:13:23.000000000 +0500
```

12.2. Права владения файлами.



Каждый файл имеет два идентификатора определяющего его принадлежность – владелец файла и группа пользователей.

Эта информация сохраняется не в самом файле, а в его метаданных (inode).

Любой файл принадлежит одному единственному пользователю. Этот пользователь называется владельцем (или пользователем — user) файла.

Когда любой пользователь системы создает файл, то права владения этим файлом принадлежат именно этому пользователю.

Первичная группа пользователя (GID) в обычных условиях определяет группу пользователей, которая будет установлена для вновь создаваемого файла.

Права доступа к файлам могут быть определены с помощью команды `ls -l`.

Пример:

```
$ id
uid=501(user1) gid=100(users) группы=100(users)
$ > file
$ ls -l file
-rw-r--r-- 1 user1 users 0 Дек 12 18:03 file
```

Примечание: Пример демонстрирует, что в сеансе находится пользователь user1, который создал файл. Из вывода команды ls -l заметно, что созданный файл принадлежит пользователю user1 (третий столбец листинга) и имеет группу пользователей users (четвертый столбец). Первичная группа пользователя – users была установлена на файл в качестве группы его пользователей.

Примечание: Реально, в файловой системе сохраняется информация не об именах пользователя и группы, а UID и GID пользователя, создавшего файл.

В Linux существуют три базовых класса доступа к файлу:

1. User access (u) – права доступа владельца файла;
2. Group access (g) – права доступа группы владельцев файла;
3. Other access (o) – права доступа для всех остальных.

Примечание: Никаких других категорий не предусматривается, поэтому каждый пользователь может быть либо владельцем файла, либо входить в группу пользователей, либо относиться к категории всех остальных. Сами по себе права владения файлами не предоставляют информации о том, что может пользователь делать с данным файлом. Система прав доступа к файлу, описываемая ниже, определяет какие возможности работы с файлом имеет конкретный пользователь системы.

12.3. Права доступа, устанавливаемые на файлы.

Права доступа к файлу хранятся в метаданных файла и кодируются тремя триадами бит.

Права доступа можно задавать в символической и восьмеричной нотациях.

Символическая нотация основана на буквенных обозначениях прав владения и прав доступа, а восьмеричная связана с фактическим представлением этих прав в виде триад бит.

Порядок триад:

1. старшая триада соответствует правам доступа владельца файла (u);
2. средняя триада - правам доступа группы владельцев (g);
3. младшая триада – правам доступа всех остальных пользователей (o).

Порядок битов в триадах:

1. установленный в 1 старший бит в каждой триаде (4 в восьмеричной нотации) обозначает разрешение на чтение данного файла и в символической нотации обозначается r--;
2. установленный в 1 средний бит в каждой триаде (2 в восьмеричной нотации) обозначает разрешение на изменение данного файла: -w-;
3. установленный в 1 младший бит в каждой триаде (1 в восьмеричной нотации) обозначает разрешение на исполнение данного файла: --x.

Пример: запись `rxwxr-x--x` (`751`) обозначает, что пользователь файла имеет все права на доступ к нему (`rxw` или `7`), группа пользователей имеет права на чтение и исполнение файла (`r-x` или `5`), все остальные имеют права на исполнение файла (`--x` или `1`).

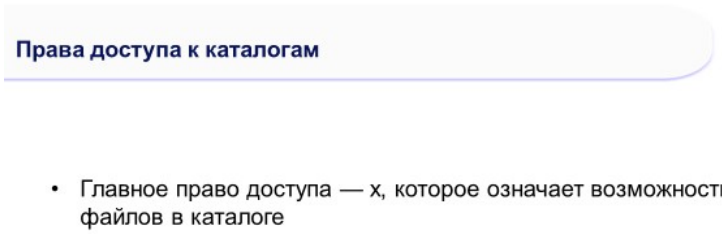
Символьная и восьмеричная нотация записи прав доступа абсолютно эквивалентны.

Восьмеричное значение триады бит получается сложением степеней двойки, соответствующих номеру бита в триаде.

Пример: права доступа в символьной нотации rwxr-xr-- в восьмеричной нотации записываются как 754, где 7 = 22+21+20, 5 = 22+20, 4 = 22.

Для того чтобы увидеть права доступа к файлу, достаточно набрать команду `ls -l`, при этом права доступа к файлам выводятся в первой колонке.

12.4. Права доступа к каталогам.



Права доступа, устанавливаемые на каталоги имеют несколько иной смысл, чем права на файлы.

Для каталогов используются следующие права:

1. `x` – (search) – право обращаться к метаданным файлов в каталоге, что предоставляет возможность использовать имя этого каталога в имени пути до нужного файла.
2. `r` – право на чтение имен файлов, находящихся в каталоге, то есть на выполнение команды `ls`.
3. `w` – право на запись в каталог, то есть право переименовывать, удалять, создавать файлы и прочее.

Без наличия права на search (`x`), установленного на каталог, работа с находящимися внутри файлами невозможна.

Примечание: Поэтому, для каталогов права доступа должны быть либо нечетные, либо они должны отсутствовать.

Ниже приведены права доступа к каталогам, которые имеют практический смысл:

1. 0 (`---`) – прав нет.
2. 1 (`--x`) – имеется право перехода в каталог, можно обращаться к

файлам в нем, однако, нельзя выполнять команду `ls` и осуществлять какие-либо манипуляции с файлами, например, переименование или удаление.

3. 3 (`-wx`) - в каталог можно переходить, разрешены любые манипуляции с файлами и можно к ним обращаться, однако получить список файлов командой `ls` невозможно.
4. 5 (`r-x`) - в каталог можно переходить и получать подробную информацию о файлах командой `ls -l`, разрешено обращаться к файлам, однако, нельзя осуществлять какие-либо манипуляции с файлами, например, переименование или удаление.
5. 7 (`rwx`) – полные права.

Для получения прав доступа к каталогу следует использовать команду `ls -ld`.

Пример:

```
$ ls -ld
drwxr-x--x 10 user1 users 4096 Дек 12 18:03 .
```

Примечание: В примере, приведенном выше, на текущий каталог установлены права 751, то есть владелец этого каталога (user1) имеет все права на этот каталог, группа (users) не имеет прав переименовывать и удалять файлы, поскольку не имеет прав на запись, а для всех остальных каталог является "темным". Все остальные могут переходить в этот каталог и могут обращаться к файлам, находящихся в нем, однако, при этом они должны знать, какие имена имеют файлы, к которым им необходим доступ. Это связано с тем, что командой ls остальные пользоваться не могут, так как прав на чтение каталога нет. Производить какие-либо манипуляции с файлами они также не имеют права, так как права на запись в каталог нет.

12.5. Изменение прав владения файлами.

Права владения файлами могут быть изменены с помощью следующих команд:

1. `chown` - эта команда позволяет менять как владельца файла или каталога, так и группу пользователей файла;
2. `chgrp` - позволяет менять группу пользователей файла.

В Linux этим команды обычно может выполнять только суперпользователь, так как передача прав владения, разрешенная для обычных пользователей, представляет собой существенную угрозу безопасности.

Примечание: При необходимости разрешить какому-либо уполномоченному пользователю исполнять эти команды, на них необходимо установить специальный бит (например, `SUID` - бит подмены владельца процесса), о которых будет рассказано в конце этой главы. Однако, даже не смотря на возможность для обычного пользователя с помощью такой манипуляции изменять права владения файлами, обычный пользователь может менять владельца или группу только у тех файлов, которыми он владеет.

Пример: Приведенная ниже команда меняет владельца файла:

```
# ls -l f1
-rw-r--r-- 1 tania prof 8 Окт 22 21:04 f1

# chown user1 f1
```

```
# ls -l f1
-rw-r--r-- 1 user1 prof 8 Окт 22 21:04 f1
```

Примечание: Из приведенного выше примера заметно, что первый аргумент команды - имя нового владельца файла, а далее идут файлы или каталоги, права на владение которыми изменяются.

Пример: Ниже приведен пример смены группы пользователей файлов f1 и text.c :

```
# ls -l f1 text.c
-rw-r--r-- 1 user1 prof 8 Окт 22 21:04 f1
-rw-r--r-- 1 prof prof 175 Дек 13 21:24 text.c

# chgrp tania f1 text.c

# ls -l f1 text.c
-rw-r--r-- 1 user1 tania 8 Окт 22 21:04 f1
-rw-r--r-- 1 prof tania 175 Дек 13 21:24 text.c
```

С помощью команды `chown` можно одновременно изменить владельца и группу (через двоеточие или точку) одновременно, причем новый владелец вовсе не обязан быть членом той группы, которая будет установлена на файл

Пример:

```
# chown tania:sys f1
# ls -l f1
-rw-r--r-- 1 tania sys 8 Окт 22 21:04 f1
```

Примечание: В этом примере продемонстрировано, как одновременно поменять владельца и группу с помощью команды chown . В команде новый владелец и новая группа указываются через знак двоеточия в соответствии с POSIX. Однако, в Linux допускается использование BSD стиля, в котором вместо двоеточия указывают точку. Например, приведенная ниже команда с помощью chown изменит только группу в BSD стиле (точка вместо двоеточия):

Пример:

```
# chown .adm f1
# ls -l f1
-rw-r--r-- 1 tania adm 8 Окт 22 21:04 f1
```

Опция -c GNU версий команд chown и chgrp позволяет получать подробную информацию об изменяемых правах владения

Пример:

```
# chgrp -c tania f1
изменена группа `f1' на tania
```

Обе команды chown и chgrp имеют опцию -R, позволяющую рекурсивно изменять права владения на каталоги и их содержимое.

Пример:

```
# ls -Rl scores/
scores/:
итого 1
drwxrwxr-x 2 prof prof 80 Авг 24 16:20 rnd_tutorial

scores/rnd_tutorial:
итого 4
-rw-rw-r-- 1 prof prof 1040 Авг 24 16:20 000.score

# chown -R tania.tania scores/

# ls -Rl scores/
scores/:
итого 1
drwxrwxr-x 2 tania tania 80 Авг 24 16:20 rnd_tutorial

scores/rnd_tutorial:
итого 4
-rw-rw-r-- 1 tania tania 1040 Авг 24 16:20 000.score
```

Примечание: В этом примере владельцем и группой пользователей стали tania (владелец и группа). В команде был применен BSD стиль указания владельца и группы (точка вместо двоеточия).

При использовании рекурсивной смены прав владения бывает очень удобно получать подробную информацию об этом процессе. Для этого предназначена опция -v команд chown и chgrp

Пример:

```
# chgrp -Rv users scores/
изменена группа `scores/' на users
```

```
изменена группа `scores//rnd_tutorial' на users
изменена группа `scores//rnd_tutorial/000.score' на users
```

Примечание: Пример демонстрирует, что с опцией -v для каждого файла, на который изменяются права владения, выдается подробная информация.

Примечание: Внимание! Неосторожное использование команд chown и chgrp, особенно с опцией -R, может привести к выводу всей системы из строя!

12.6. Установка прав доступа.

Команда `chmod` предназначена для изменения прав доступа к файлам и каталогам, указанным в качестве аргументов.

Права доступа должны быть указаны либо в восьмеричной, либо в символьной нотации.

Права указывают в качестве первого аргумента команды.

Изменять права доступа к файлу могут только суперпользователь и владелец файла.

Пример: Ниже приведен пример использования команды `chmod` для изменения прав доступа к файлу в восьмеричной нотации:

```
$ ls -l text.c
-rw-r--r-- 1 prof tania 175 Дек 13 21:24 text.c

$ chmod 660 text.c

$ ls -l text.c
-rw-rw---- 1 prof tania 175 Дек 13 21:24 text.c
```

Примечание: Использование команды chmod 660 text.c позволило установить права 660 на файл text.c .

Команда `chmod` позволяет также устанавливать права на доступ к файлу,

указывая их в символической нотации. Для этого применяется следующая форма команды `chmod класс_изменение_права файлы`

класс может принимать следующие значения:

1. `u` – доступ владельца;
2. `g` – доступ группы владельцев;
3. `o` – доступ всех остальных;
4. `a` – доступ всех групп пользователей.

изменение может принимать следующие значения:

1. `+` - разрешить;
2. `-` - запретить;
3. `=` - установить.

Права может принимать следующие значения:

1. `r` – чтение;
2. `w` – запись;
3. `x` – выполнение.

Если используются операции разрешения (+) или запрета (-) прав на файл, то они не изменяют те биты прав доступа, которые не относятся к требуемому изменению.

Пример: если для файла text.c требуется удалить право на изменение для группы, и добавить право на чтение для всех трех категорий пользователей (владелец, группа, все остальные), то выполним следующую команду:

```
$ chmod g-w,a+r text.c
$ ls -l text.c
-rw-r--r-- 1 prof tania 175 Дек 13 21:24 text.c
```

Примечание: приведенная выше команда отобрала права на запись для группы пользователей файла f1, но это не отразилось на правах на чтение этого файла, установленных для группы.

Использование операции назначения (=) стирает те права, которые были установлены ранее и назначает новые.

Пример: установим на файл text.c права на чтение и запись для владельца и группы, и запретим всем остальным какой-либо доступ к файлу:

```
$ chmod ug=rw,o= text.c
$ ls -l text.c
```

```
-rw-rw---- 1 prof tania 175 Дек 13 21:24 text.c
```

Аналогично командам `chown` и `chgrp` , команда `chmod` способна рекурсивно изменять права доступа к каталогам и всему их содержимому, если она вызвана с опцией `-R`.

Примечание: этой возможностью следует пользоваться с особой осторожностью, принимая во внимание концептуальные отличия прав на файлы от прав на каталоги - на файлы в большинстве случаев устанавливаются четные права (отсутствие прав на исполнение), а на каталоги наоборот - нечетные (без права на search каталоги не позволят обращаться к метаданным файлов внутри них).

Пример: Приведенная ниже команда снимает права на запись для каталога scores :

```
$ ls -lR scores
scores:
итого 1
drwxrwxr-x 2 prof users 80 Авг 24 16:20 rnd_tutorial

scores/rnd_tutorial:
итого 4
-rw-rw-r-- 1 prof users 1040 Авг 24 16:20 000.score

$ chmod -R g-w scores

$ ls -lR scores
scores:
итого 1
drwxr-xr-x 2 prof users 80 Авг 24 16:20 rnd_tutorial

scores/rnd_tutorial:
итого 4
-rw-r--r-- 1 prof users 1040 Авг 24 16:20 000.score
```

Примечание: Заметно, что и с самого каталога, и с всего его содержимого были удалены права на запись для группы пользователей.

Примечание: В обычной практике права на каталоги и на файлы устанавливаются отдельно. При необходимости рекурсивного изменения прав на каталог и его содержимое опцию -R команды chmod обычно не используют. Вместо этого пользуются командой find с установкой -exec chmod (или xargs chmod).

Пример: в каталоге scores требуется установить для файлов права 644, не затрагивая при этом права на каталоги:

```
$ find scores -type f -exec chmod 644 {} \;
$ ls -lR scores
scores:
итого 1
drwxr-xr-x 2 prof users 80 Авг 24 16:20 rnd_tutorial

scores/rnd_tutorial:
итого 4
-rw-r--r-- 1 prof users 1040 Авг 24 16:20 000.score
```

GNU версия команды `chmod` позволяет использовать опцию `-v` для получения информации о файлах, права доступа к которым изменяются, и опцию `-c` для получения подробностей изменения прав.

Пример:

```
$ chmod -Rv g-w scores
права доступа `scores' изменены на 0755 (rwxr-xr-x)
права доступа `scores/rnd_tutorial' изменены на 0755 (rwxr-xr-x)
права доступа `scores/rnd_tutorial/000.score' изменены на 0644
(rw-r--r--)
$ chmod -c g-w text.c
права доступа `text.c' изменены на 0640 (rw-r-----)
$ chmod -v 660 text.c
права доступа `text.c' изменены на 0660 (rw-rw----)
```

Примечание: опция -v выдает подробную информацию всегда, а -c только тогда, когда права действительно изменяются.

12.7. Автоматическая установка прав доступа к вновь создаваемым файлам.

Команда `umask` предназначена для автоматической установки прав доступа к вновь создаваемым файлам и каталогам.

Команда `umask` позволяет задавать значение битовой маски, которая будет “вычитаться” из прав `777` для каталогов и `666` для файлов.

При вызове этой команды без аргумента она возвратит текущее значение маски

Пример:

```
$ umask
0022
```

Установка другого значения `umask` никоим образом не отразится на уже существующих файлах и каталогах, она участвует только в процессе определения прав на вновь создаваемые файлы и каталоги.

Пример:

```
$ umask 002
$ mkdir dir1
$ > file1
```

```
$ ls -ld dir1 file1
drwxrwxr-x 2 prof prof 48 Дек 14 20:43 dir1
-rw-rw-r-- 1 prof prof 0 Дек 14 20:43 file1
$ umask 077
$ mkdir dir2
$ > file2
$ ls -ld dir2 file2
drwx------ 2 prof prof 48 Дек 14 20:44 dir2
-rw------- 1 prof prof 0 Дек 14 20:44 file2
```

Примечание: В этом примере продемонстрировано, что при установленном значении umask равном 002 на каталоги устанавливаются права 775, а на файлы - 664. В то же время umask , установленная в 077 дает в результате, соответственно, 700 - для каталогов и 600 - для файлов.

Ниже приведена таблица наиболее часто применяемых значений umask.

umask	Каталоги	Файлы
002	775	664
007	770	660
022	755	644
027	750	640
077	700	600

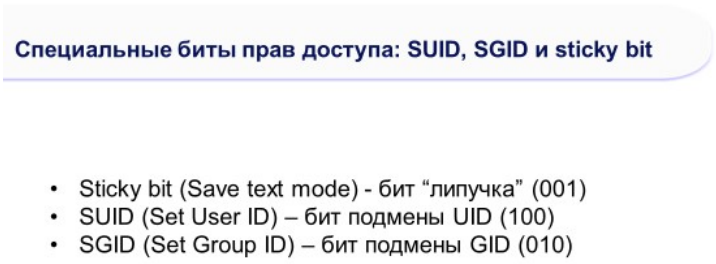
Значение umask можно задавать также и в символьной нотации

Пример:

```
$ umask g=rwx,g=rx,o=
$ umask
0027
```

Примечание: При задании значения umask в символьной нотации всего лишь требуется указать в качестве аргумента права, которые должны будут иметь новые каталоги.

12.8. Специальные биты прав доступа: SUID, SGID и sticky bit.



Помимо битов, устанавливающих разрешения на доступ к файлу, существуют специальные атрибуты, которых образуют еще одна триаду битов:

1. `Sticky bit` (Save text mode) - бит “липучка”;
2. `SUID` (Set User ID) – бит подмены UID;
3. `SGID` (Set Group ID) – бит подмены GID.

`Sticky bit` кодируется восьмеричной 1 (двоичная 001), `SGID` кодируется восьмеричной 2 (010), а `SUID` - 4 (100).

В символьной нотации применяются символы `T` для `Sticky bit`, `S` для `SUID` и `SGID`. Эти символы всегда выводятся в позиции, где должен находиться флаг разрешения на исполнение (`x`).

Если одновременно установлены и бит `x` и бит `S`, то отображается символ `s`

Биты `T` и `x` всегда устанавливаются вместе, поскольку бит `T` используется только для каталогов. Поэтому в символьном отображении прав доступа встречается только символ `t`

`SUID` отображается в виде буквы `s` или `S` в старшей триаде бит, отображающей права владельца

`SGID` отображается в виде буквы `s` или `S` в средней триаде бит, отображающей права группы

Sticky bit отображается в виде буквы t в младшей триаде бит, отображающей права для всех остальных.

Пример:

```
$ ls -ld /tmp /bin/ping
-rws--x--x 1 root root 32908 Окт 16 2002 /bin/ping
drwxrwxrwt 94 root root 3488 Дек 14 20:31 /tmp
```

Примечание: Приведенный выше пример демонстрирует, что на файл системной команды ping установлен бит SUID (символ s в старшей триаде вместо прав на исполнение), а на каталог /tmp установлен Sticky bit (символ t в триаде бит для прав всех остальных).

Примечание: Атрибут Sticky bit для файлов не используется, в ранних версиях UNIX он был предназначен для того, чтобы оставить в памяти образ программы (Save text mode).

Процесс наследует права доступа к системным ресурсам от пользователя (UID), запустившего процесс, и его первичной группы (GID), если для исполняемых файлов, не установлены биты SUID и/или SGID

При установленном на исполняемый файл бите SUID процесс исполняется не от имени пользователя, запустившего его, а от имени владельца исполняемого файла команды.

При установленном бите SGID процесс исполняется не от имени первичной группы пользователя, запустившего процесс, а от имени группы пользователей файла.

У каждого процесса имеется четыре идентификатора:

1. RUID - Real UID, который всегда равен UID пользователя, выполнившего команду.
2. RGID - Real GID, который всегда равен GID пользователя, выполнившего команду.
3. EUID - Effective UID, который либо равен RUID, либо если на исполняемый файл установлен бит SUID, то UID владельца файла.
4. EGID - Effective GID, который либо равен RGID, либо если на исполняемый файл установлен бит SGID, то GID владельца файла.

Примечание: В подавляющем большинстве случаев подмена владельца или группы осуществляется на root или какого-либо высокопривелигированного пользователя или группу. Например, при выполнении команды ping (см. пример выше), несмотря на то, что ее запустил обычный пользователь, она будет исполняться от имени root, так как он владеет

ее исполняемым файлом. Это используется, например, в таких программах, как passwd, которые требуют временного предоставления доступа обычному пользователю к тем ресурсам, к которым он не имеет доступа. Естественно, такие программы требуют особого подхода к разработке, так как предоставляют серьезную угрозу для безопасности системы. На файлы скриптов Shell биты SUID и SGID устанавливать можно, но они действовать не будут.

Установка `Sticky bit` на каталог, в отношении которого пользователь имеет права на чтение и на запись, позволяет запретить удалять и изменять пользователю чужие файлы в этом каталоге.

Примечание: Это используется при установке прав доступа к каталогу /tmp, открытому на запись всем, поскольку иначе пользователь может удалить чужие временные файлы, находящиеся в этом каталоге, что может повлечь плачевные последствия.

При установке атрибута `SGID` на каталог, вновь созданные файлы в этом каталоге будут наследовать группу владельцев по группе владельцев каталога (так называемый “стиль BSD”), вместо `RGID` процесса, создающего файл по версии System V.

Пример:

```
$ cd dir1

$ ls -ld
drwxrwsr-x 2 tania users 48 Дек 14 20:43 .

$ id
uid=500(prof) gid=500(prof) группы=500(prof),100(users)

$ > file

$ ls -l
итого 0
-rw-r----- 1 prof users 0 Дек 14 22:00 file
```

Примечание: В каталоге, на который установлен бит SGID, был создан файл. При этом группа владельцев файла была назначена не по первичной группе пользователя, создавшего файл, а по группе пользователей каталога, в котором файл был создан.

Команда `chmod` позволяет установить особые биты доступа на файлы и каталоги.

Для установки специальных битов в символьном режиме команда `chmod`

должна быть выполнена со следующими аргументами:

1. `u+s` - для установки на файл бита `SUID`;
2. `g+s` - для установки на файл или каталог бита `SGID`;
3. `o+t` - для установки на каталог бита `Sticky bit`.

Для установки специальных битов в числовом режиме команде `chmod` в качестве первого аргумента передается число, состоящее из 4 цифр, где левая цифра представляет собой сумму специальных битов.

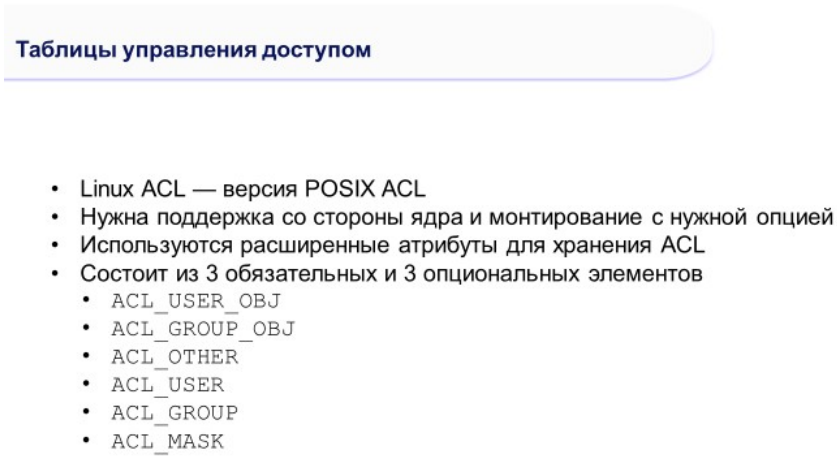
Пример: Приведенная ниже команда `chmod 2775 d1` устанавливает бит `SGID` на каталог:

```
$ mkdir d1
$ ls -ld d1
drwxr-x--- 2 prof prof 48 Дек 14 22:31 d1
$ chmod 2770 d1
$ ls -ld d1
drwxrws--- 2 prof prof 48 Дек 14 22:31 d1
```

Ниже приведена таблица, в которой указаны результаты установки различных специальных прав доступа на файлы и каталоги.

Права	Эффект для каталогов	Эффект для файлов
`-rws--x--x`		Комада исполняется от имени владельца файла.
`-rwx--s--x`		Комада исполняется от имени группы пользователей файла.
`drwxrws---`	На файлы, создаваемые в каталоге, будет установлена такая же группа, как у каталога.	
`drwxrwxrwt`	В каталоге можно удалять или переименовывать только собственные файлы.	

12.9. Общие сведения о ACL



Стандартная система доступа к файлам Linux-систем предусматривает 12 бит, которые определяют вид доступа для трех категорий пользователей: владельца, группы владельцев и всех остальных.

Кроме этого в такой системе нет наследования прав доступа.

Такая система хорошо работает в простых ситуациях, но в сложных случаях она создает проблемы для администраторов.

Для решения данных проблем можно воспользоваться Linux ACL. (Access Control Lists - списки контроля доступа) - версией POSIX ACL для Linux.

Linux ACL — это набор патчей для ядра операционной системы и программ для работы с файловой системой и несколько утилит, дающих возможность устанавливать права доступа к файлам не только для пользователя-владельца и группы-владельца файла, но и для любого пользователя или группы.

В версии ядра 2.6 ACL включен в стандартную поставку.

Linux ACL использует расширенные атрибуты для хранения данных о правах доступа к файлам пользователей и групп.

Расширенные атрибуты — это пара “имя/значение’, привязанная к определенному файлу.

Список расширенного контроля доступа существует для каждого inode и состоит из шести компонентов:

Первые три являются копией стандартных прав доступа к файлу. Они содержатся в единственном экземпляре в ACL и есть у каждого файла в системе:

1. `ACL_USER_OBJ` — режим доступа к файлу пользователя-владельца;
2. `ACL_GROUP_OBJ` — режим доступа к файлу группы-владельца;
3. `ACL_OTHER` — режим доступа к файлу остальных пользователей.

Следующие компоненты устанавливаются для каждого файла в отдельности и могут присутствовать в ACL в нескольких экземплярах:

1. `ACL_USER` — содержит UID и режим доступа к файлу пользователя, которому установлены права, отличные от основных.
2. `ACL_GROUP` — то же самое, что и `ACL_USER`, но для группы пользователей;
3. `ACL_MASK` — маска действующих прав доступа для расширенного режима.

На каждого пользователя со своими правами на данный файл хранится отдельная запись. Не может существовать более одной записи на одного и того же пользователя.

При установке дополнительных прав доступа присваивается значение и элементу `ACL_MASK`.

Каталоги также могут иметь список контроля доступа по умолчанию. В отличие от основного ACL, он действует на создаваемые внутри данного каталога файлы и каталоги.

При создании файла внутри такого каталога файл получает ACL, равный ACL. по умолчанию (`default ACL`) этого каталога - наследование.

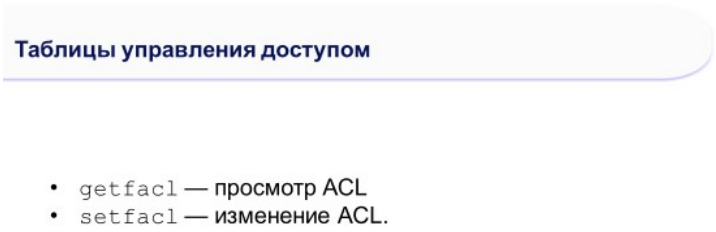
Поддержка ACL должна быть включена в ядре для нужной вам ФС.

Пример: проверяем в каких ФС есть поддержка ACL.

```
$ grep ACL /boot/config-$(uname -r)
CONFIG_XILINX_EMACLITE=m
CONFIG_EXT4_FS_POSIX_ACL=y
CONFIG_REISERFS_FS_POSIX_ACL=y
CONFIG_JFS_POSIX_ACL=y
CONFIG_XFS_POSIX_ACL=y
CONFIG_BTRFS_FS_POSIX_ACL=y
CONFIG_F2FS_FS_POSIX_ACL=y
CONFIG_FS_POSIX_ACL=y
CONFIG_NTFS3_FS_POSIX_ACL=y
CONFIG_TMPFS_POSIX_ACL=y
```

```
CONFIG_JFFS2_FS_POSIX_ACL=y
CONFIG_EROFS_FS_POSIX_ACL=y
CONFIG_NFS_V3_ACL=y
CONFIG_NFSD_V2_ACL=y
CONFIG_NFSD_V3_ACL=y
CONFIG_NFS_ACL_SUPPORT=m
CONFIG_CEPH_FS_POSIX_ACL=y
CONFIG_9P_FS_POSIX_ACL=y
```

12.10. Установка и изменение прав доступа



Управление списками контроля доступа производится при помощи двух утилит: `getfacl` и `setfacl`.

С помощью `getfacl` можно просмотреть текущие параметры доступа любого файла.

Пример: при вызове `getfacl` для домашнего каталога пользователя user1 мы получим следующее:

```
$getfacl /home/user1
file: home/user1
owner: user1
group: users
user::rwx
group::---
other:---
```

Примечание: Как можно видеть, каталог /home/user1 принадлежит пользователю user1, группе users и значение прав доступа к каталогу — 0700. Каталог имеет только основные параметры доступа, поскольку изначально дополнительные права не устанавливаются.

Дополнительные права доступа к файлу устанавливаются и изменяются при помощи утилиты `setfacl`.

Для этого используется следующий формат вызова:

`setfacl -<опции> <ACL_структура>, <ACL_структура>,…, <ACL_структура> <имя_файла> <имя_файла>`

ACL_структура представляет собой одну из следующих конструкций:

1. `[d[efault]:] [u[ser]:] <пользователь> [:<режимы_доступа>]` — определяет режим доступа к файлу или каталогу пользователя. Если пользователь не указан, определяет режим доступа пользователя-владельца;
2. `[d[efault]:] g[roup]: <группа> [:<режимы_доступа>]` — то же, что и предыдущая конструкция, но для группы;
3. `[d[efault]:] m[ask][:] [:<режимы_доступа>]` — определяет действующие права доступа;
4. `[d[efault]:] o[ther][:] [:<режимы_доступа>]` — определяет режим доступа для остальных пользователей.

Параметр `default` определяет режим наследование прав доступа.

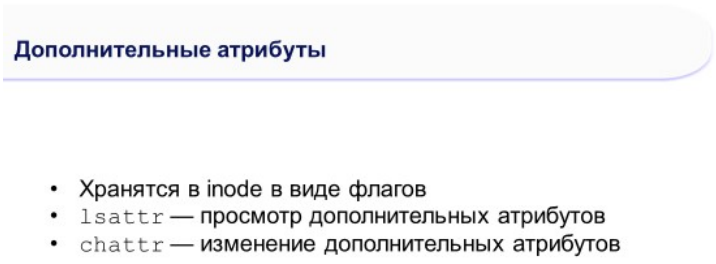
Для установки и изменения ACL. используются следующие опции:

1. `--set-acl` — заменяет полностью ACL-файл, на указанный в командной строке;
2. `-m` — изменяет режимы доступа к файлу (каталогу);
3. `-x` — убирает правила доступа из ACL.

Пример: вот что мы получим, применив `setfacl` к каталогу user1:

```
$setfacl                                                    \
--set-acl=u::rwx,g::---,o::---,u:user2:rwx,g:users2:rx,u:user3:---
\
/home/user1
$getfacl /home/user1
file: home/user1
owner: user1
group: users
user::rwx
user:user2:rwx
user:user3:---
group::---
group:users2:r-x
mask:rwx
other:---
```

12.11. Дополнительные атрибуты



Файловая система ext поддерживает несколько дополнительных атрибутов.

По умолчанию эти атрибуты при создании нового файла не устанавливаются, и их активизация производится вручную.

Атрибуты хранятся в блоке информационного дескриптора (inode) под заголовком flag (флаг) в виде шестнадцатеричного числа:

1. «A» отключает обновление поля времени последнего доступа к файлу (поле atime), что позволяет снизить нагрузку на жесткий диск.
2. «S» предписывает операционной системе асинхронно сохранять на диске все модификации файла.
3. «a» запрещает выполнять над файлом какие-либо операции, кроме добавления данных. Установить может только суперпользователь.
4. «c» включает сжатие файла. При записи файла на диск такой файл автоматически сжимается, а при чтении разжимается.
5. «d» используется программой dump, которая игнорирует файлы с этим атрибутом и не осуществляет их резервное копирование.
6. «i» блокирует любую возможность изменения файла. Файлы, обладающие таким атрибутом, являются неизменяемыми (immutable), их нельзя удалить, изменить или переименовать, кроме того, на него нельзя сослаться при помощи ссылки. Все эти действия блокируются не

только для обычных пользователей, но даже и для пользователя root. Только суперпользователь может установить этот атрибут.

7. «s» предписывает ядру сопровождать удаление файла записью нулей в дисковые блоки, принадлежащие этому файлу.
8. «u» предназначен для восстановления файла после того, как файл удален обычным образом.

Примечание: не следует слишком полагаться на этот атрибут, так как зачастую восстановить удается лишь первый блок файла (обычно это 4096 байт). Если вы хотите обеспечить возможность восстановления удаляемых файлов, более правильным решением будет не удалять файл сразу, а перемещать его в специальный каталог, где он будет находиться некоторое время, по истечении которого cron удалит его по-настоящему.

Для настройки перечисленных ранее атрибутов используется команда chattr.

Как и для команды `chmod`, требуемая операция указывается при помощи оператора, за которым следуют символьные обозначения атрибутов.

- «+» предписывает команде установить указанные атрибуты;
- «–» – сбросить атрибуты;
- «=» – установить указанные атрибуты и сбросить все остальные.

Параметр `-R`, обозначает, что команда будет применена рекурсивно в отношении всех подкаталогов.

Параметр `-V`, обозначает вывод версии программы `chattr`, а также дополнительных сообщений во время ее работы.

Параметр `-v`. При использовании этого параметра после него следует указать число, которое будет установлено в качестве номера версии индексного дескриптора.

Примечание: Никакого смысла, кроме того, который вы сами ему дадите, в номере версии нет — это просто число, которое можно записать в индексный дескриптор. Это число никак не связано с самим файлом. При создании файла ему выделяется индексный дескриптор из числа свободных, номер версии этого индексного дескриптора устанавливается равным единице.

Пример: Если выполнить команду `mv foo foo2`, то файл `foo2` унаследует свой индексный дескриптор от файла `foo`. Это означает, что если файл `foo2` до этого существовал, его индексный дескриптор (а значит, и номер версии) будет заменен. Однако если при существующем `foo2` выполнить `cp foo foo2`, индексный дескриптор `foo2` останется прежним (если файл `foo2` до этого не существовал, ему будет выделен свободный индексный дескриптор с номером версии, равным 1). К примеру `Caldera` использует эти номера при установке системы, давая всем устанавливаем файлам уникальный номер версии. Делается это для того, чтобы файлы, созданные после первоначальной установки, можно было отличить от файлов, созданных во время установки. Например, если обновить пакет (`rpm -U`), то номер версии новых файлов будет равен не заданному во время установки уникальному номеру, а простой единице.

Команда `lsattr` выводит список файлов и их ext2-атрибутов.

Подобно команде `chattr`, команда `lsattr` поддерживает параметр `-R`, при наличии которого она рекурсивно обрабатывает все подкаталоги.

Параметр `-a` означает отображение информации обо всех файлах, включая файлы, имена которых начинаются с точки.

При использовании опции `-d` выводятся сведения только о каталогах, но не о файлах.

Параметр `-l` позволяет получить сведения о файлах в расширенном формате, где каждый атрибут отображается не с помощью буквы, а с помощью слов.

Параметр `-v` включает вывод версии файлов.

Пример: установка и проверка дополнительных атрибутов

```
# ls
file.txt
# chattr +i file.txt
# lsattr
----i--------- ./file.txt
# rm -rf file.txt
rm: cannot remove `file.txt': Operation not permitted
```

Примечание: обратите внимение после установки атрибута immutable (i) root не смог удалить этот файл.

12.12. Общепринятые соглашения об именовании файлов.

Несмотря на то, что при именовании файлов в Linux можно использовать практически любые символы кроме косой черты / , рекомендуется использовать алфавитно-цифровые символы, символ подчеркивания, точку и дефис.

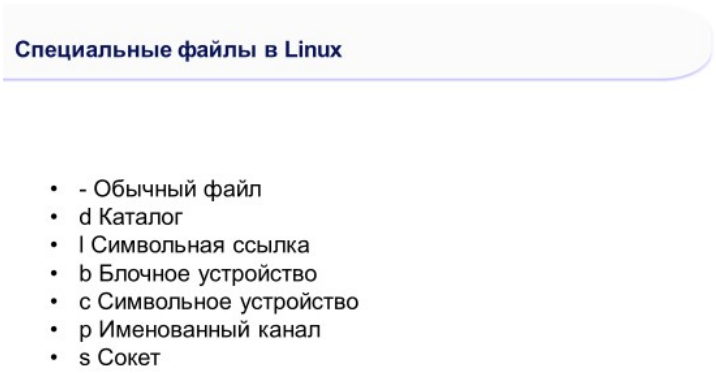
В отличие от многих операционных систем, например, MS DOS, Linux не придает особого значения точке в имени файла. Тем не менее, пользоваться точками в именах файлов удобно для указания после нее суффикса имени файла, показывающего тип данных, хранящихся в данном файле.

Ниже перечислены часто используемые общепринятые суффиксы, которые не стоит без особой цели использовать для файлов с другим типом содержимого.

Суффикс	Содержимое файла
.c	Исходный код программы на языке C.
.cpp	Исходный код программы на языке C++.
.h	Заголовочный файл для программы на C или C++ (header).
.o	Объектный код.
.a	Статическая библиотека.
.so	Разделяемая библиотека (shared object).
.sh	Shell скрипт
.csh	C Shell скрипт.

.pl	Программа на Perl.
.pm	Скомпилированная в байт-код программа на Perl.
.py	Программа на Python.
.rpm	Программный пакет в формате RPM (Red Hat Package Manager).
.src.rpm	Пакет с исходным кодом в формате RPM.
.deb	Программный пакет в формате Debian.
.tar	Архив tar (tape archive).
.cpio	Архив cpio.
.gz	Файл, сжатый gzip.
.bz2	Файл, сжатый bzip2.
.Z	Файл, сжатый compress.

12.13. Специальные файлы в Linux.



В GNU/Linux используются следующие типы специальных файлов (в столбце обозначение приводятся символы из первого столбца вывода команды `ls -l`, обозначающие тип файла):

Обозначение	Тип файла	Назначение
-	Обычный файл.	Для хранения данных.
d	Каталог.	Организует древовидную файловую структуру.
l	Символьная ссылка.	Указывает на другой файл.
b	Блочное устройство.	Обеспечивает доступ к блочным устройствам.
c	Символьное устройство.	Обеспечивает доступ к символьным устройствам.
p	Именованный канал.	Передает поток от одного процесса к другому.
s	Сокет.	Для передачи данных по сетевому протоколу.

Специальные файлы, не являющиеся обычными обрабатываются ядром операционной системой особым образом.

Примечание: Так, например, операционная система не пытается вывести на экран двоичное содержимое файла каталога, вместо этого при обращении к нему с помощью ls отображаются имена файлов в нем в понятной человеку форме.

В файле символьной ссылки находится строка, представляющая собой имя файла, на которое эта ссылка указывает.

Примечание: Когда пользователь обращается к символьной ссылке операционная система вместо того, чтобы показывать ее содержимое, на самом деле обращается к файлу, на который ссылка указывает.

Пример:

```
$ ls -l doc
lrwxrwxrwx 1 prof prof 14 Май 1 2003 doc -> /usr/share/doc
$ cd doc
$ pwd
/home/prof/doc
```

Примечание: В этом примере с помощью команды ls -l демонстрируется, что файл символьной ссылки doc указывает на каталог /usr/share/doc . Несмотря на то, что doc является не каталогом, а ссылкой на каталог, можно сделать команду cd и оказаться в каталоге, на который ссылка указывает. При этом создается полная иллюзия, будто бы был осуществлен переход в каталог doc, что демонстрирует команда pwd.

Специальные файлы блочные устройства предназначены для обращения к блочным устройствам, то есть к тем, с которых информация считывается и записывается блоками строго определенной длины.

Примечание: Такие устройства требуют форматирования, и к этой категории относятся, например, магнитные и лазерные диски.

Специальные файлы символьных устройств предназначены для обращения к устройствам, которые принимают и передают информацию посимвольно, а не блоками.

Примечание: Примерами таких устройств являются терминалы, мыши, модемы и т.п.

Специальные файлы блочных и символьных устройств – пустые файлы.

Вместо размера для файлов устройств в индексном дескрипторе хранятся два числа – мажор (major number, старшее число) и минор (minor number, младшее число).

Пример:

```
$ ls -l /dev/fd0 /dev/tty1
```

```
brw-rw---- 1 prof floppy 2, 0 Янв 20 2003 /dev/fd0
crw------- 1 root root 4, 1 Дек 14 23:55 /dev/tty1
```

Примечание: Из листинга, приведенного выше, заметно, что мажор файла блочного устройства (символ b в первом столбце) флоппи диска /dev/fd0 равен 2, а минор - 0. А для символьного файла устройства (символ c в первом столбце) первого виртуального терминала /dev/tty1 мажор - 4, минор - 1.

Мажор файла устройства соответствует номеру драйвера устройства в ядре операционной системы

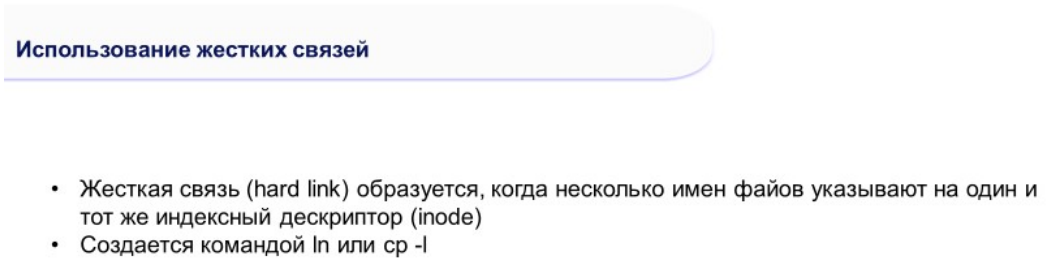
Минор - дополнительный параметр для драйвера, который указывает с каким именно устройством нужно работать драйверу.

Файлы именованных каналов предназначены для передачи потока вывода одного процесса на поток ввода другого процесса.

Примечание: Если один процесс записывает какие-либо данные в именованный канал, то его работа прерывается до тех пор, пока другой процесс не считает требуемые данные из этого именованного канала. Именованные каналы были введены в BSD.

Сокеты являются файлами, подобными именованным каналам, однако являются двунаправленными и могут быть как локальными, так и сетевыми.

12.14. Использование жестких связей.



Жесткая связь (hard link) образуется, когда несколько имен файлов указывают на один и тот же индексный дескриптор (inode).

Команда `ls -l` во втором столбце показывает количество имен у файла (link count).

Примечание: Если у файла link count имеет значение, большее единицы, значит у файла имеются жесткие связи.

Пример:

```
$ ls -l f1 link1
-rw-r--r-- 2 tania tania 8 Окт 22 21:04 f1
-rw-r--r-- 2 tania tania 8 Окт 22 21:04 link1
$ ls -i f1 link1
13598 f1 13598 link1
```

Примечание: Здесь между файлами f1 и link1 существует жесткая связь, то есть это два разных имени файла. Команда `ls -l` демонстрирует, что количество имен у файла - два (цифра во втором столбце вывода `ls -l`), права доступа и владения, размер файла и дата модификации совершенно одинаковы, поскольку метаданные одни и те же. Команда `ls -i` показывает, что номера inode у этих файлов одинаковы.

Поскольку метаданные у файлов, между которыми установлена жесткая

связь, одинаковы, то и блоки данных тоже одинаковы, следовательно любые изменения, производимые с одним файлом, зеркально отразятся на файле, жестко связанным с ним.

Фактически жесткая связь это один файл с несколькими именами.

Пример:

```
$ echo 'This is a hard link' > f1
$ cat link1
This is a hard link
```

Примечание: В этом примере информация в файл была записана с использованием имени файла f1, а чтение информации было произведено из link1.

Удаление одной жесткой связи с файлом уменьшает количество имен (link count) на единицу, однако реально файл удаляется только тогда, когда счетчик количества имен становится равным нулю.

Пример:

```
$ ls -l f1 link1
-rw-rw-rw- 2 tania tania 20 Дек 15 22:03 f1
-rw-rw-rw- 2 tania tania 20 Дек 15 22:03 link1

$ rm -f f1

$ ls -l f1 link1
ls: f1: No such file or directory
-rw-rw-rw- 1 tania tania 20 Дек 15 22:03 link1
```

Примечание: Пример, приведенный выше, демонстрирует, что при удалении одного из имен файла счетчик имен уменьшился на единицу.

Для создания жесткой связи с файлом применяется команда ln. Первый аргумент команды - имя файла, для которого создается новое имя, а второй - имя жесткой связи.

Пример:

```
$ ln file1 newname
$ ls -li file1
4676 -rw-rw-r-- 2 prof prof 0 Дек 14 20:43 file1
$ ls -li newname
4676 -rw-rw-r-- 2 prof prof 0 Дек 14 20:43 newname
```

Примечание: В этом примере создано новое имя для файла file1 - newname. Комада ls -li демонстрирует, что inode у этих файлов одинаковые.

Жесткие связи могут быть установлены только в пределах одной файловой системы.

Примечание: То есть нельзя установить жесткую связь между файлом на жестком диске и, например, дискете. Это вызвано тем, что у жестко связанных файлов должен быть один и тот же индексный дескриптор, что невозможно если файловые системы разные.

Установить жесткую связь с каталогом с помощью команды ln нельзя.

Пример:

```
$ ls -ld LPI
drwxrwx--- 10 prof users 544 Сен 2 10:15 LPI
$ ln LPI l1
ln: `LPI': не допускается создавать жесткие ссылки на каталоги
```

У каждого каталога имеется как минимум два имени - имя каталога, записанное в его родительском каталоге, и имя точка.

У родительского каталога после создания нового подкаталога количество имен увеличивается на единицу, так как в дочернем каталоге имеется имя две точки, являющееся жесткой связью с именем родительского каталога

Пример:

```
$ ls -ldi
3555 drwx------ 16 user1 user1 1120 Дек 15 22:15 .

$ mkdir dir1

$ ls -ild dir1
1598 drwxr-x--- 2 user1 user1 48 Дек 15 23:03 dir1

$ ls -ial dir1
итого 2
1598 drwxr-x--- 2 user1 user1 48 Дек 15 23:03 .
3555 drwx------ 17 user1 user1 1144 Дек 15 23:03 ..

$ ls -ldi
3555 drwx------ 17 user1 user1 1144 Дек 15 23:03 .
```

Примечание: В этом примере был создан каталог dir1 , inode которого 1598. Имя точка, находящееся в каталоге dir1 - имя текущего каталога, имеет тот же индексный дескриптор 1598. Таким образом, у нового каталога dir1 имеется два имени. Имя родительского каталога - две точки, находящиеся в каталоге dir1 , имеет inode 3555, значение которого совпадает с inode родительского каталога. Сравнив вывод первой и последней команд, становится заметно, что после создания каталога dir1 количество имен у его родительского каталога увеличилось на единицу.

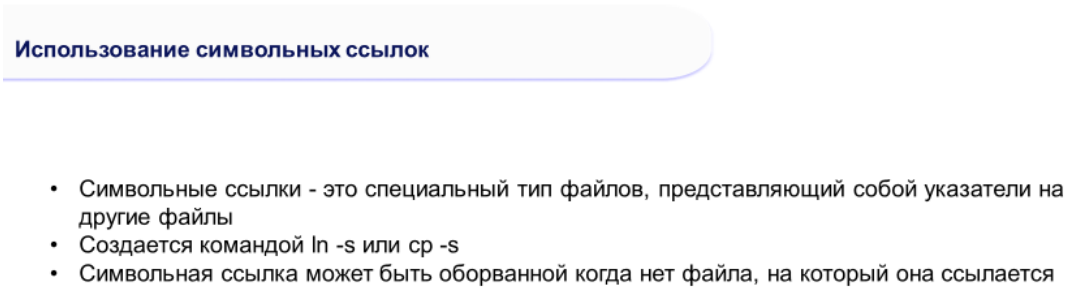
Команда `cp` имеет специальную опцию `-l` , которая позволяет (в пределах одной файловой системы) вместо копирования создавать жесткие связи с исходными файлами.

Пример:

```
$ ls -li f???
4629 -rw-r--r-- 1 user1 user1 0 Дек 11 12:30 f112
4631 -rw-r--r-- 1 user1 user1 0 Дек 11 12:30 f113
4632 -rw-r--r-- 1 user1 user1 0 Дек 11 12:30 f117
$ cp -l f??? dir1
$ ls -li dir1
итого 0
4629 -rw-r--r-- 2 user1 user1 0 Дек 11 12:30 f112
4631 -rw-r--r-- 2 user1 user1 0 Дек 11 12:30 f113
4632 -rw-r--r-- 2 user1 user1 0 Дек 11 12:30 f117
```

Примечание: В этом примере показано, что при использовании команды cp с опцией -l для исходных файлов были созданы жесткие связи с такими же именами в каталоге dir1.

12.15. Использование символьных ссылок.



Символьные ссылки - это специальный тип файлов, представляющий собой указатели на другие файлы.

Символьные ссылки можно создавать с помощью команды `ln -s` , где первый аргумент - имя уже существующего файла, а второй аргумент - либо имя символьной ссылки, либо каталога, где будет образован файл символьной ссылки с таким же именем, что и исходный файл.

Пример:

```
$ ln -s file1 slink1

$ ls -li file1 slink1
4639 -rw-rw-r-- 1 user1 user1 0 Дек 11 15:55 file1
1604 lrwxrwxrwx 1 user1 user1 5 Дек 16 01:37 slink1 -> file1

$ echo 123 > file1

$ cat slink1
123
```

Примечание: В этом примере на файл file1 создана символьная ссылка slink1 . Метаданные у file1 и slink1 различаются, поскольку это совершенно разные файлы. Тем не менее, к файлу можно обращаться с помощью ссылки.

Символьные ссылки можно устанавливать на каталоги

Пример:

```
$ ln -s ~/dir1 /tmp

$ ls -ldi dir1 /tmp/dir1
4635 drwxr-xr-x 2 user1 user1 144 Дек 16 00:26 dir1
163286 lrwxrwxrwx 1 user1 user1 16 Дек 16 01:12 /tmp/dir1 ->
/home/user1/dir1
```

Примечание: В этом примере создается символьная ссылка на каталог dir1 в каталоге /tmp . Заметно, что символьная ссылка имеет совершенно иной тип файла и номер inode, местоположение ее не ограничивается пределами одной файловой системы.

При создании символьной ссылки в другом каталоге следует указывать полное имя исходного файла. Иначе:

- либо ссылка будет оборванной (или иначе – висящей, broken link), то есть символьная ссылка будет указывать на несуществующий файл;
- либо ссылка будет указывать на существующий файл в целевом каталоге, имя которого (случайно или намеренно) совпадет с исходным файлом, однако, ссылка будет указывать не на исходный файл.

Символьная ссылка может оказаться оборванной в случае, если:

- файл, на который она указывает перемещен, переименован или удален;
- нет достаточных прав доступа на файл, указываемый символьной ссылкой;
- файл находится в файловой системе, которая в настоящий момент не смонтирована.

Пример:

```
$ mv file1 file11
$ cat slink1
cat: slink1: No such file or directory
```

Примечание: После переименования файла file1 в file11 ссылка slink1 стала оборванной.

Символьная ссылка может быть создана на любой специальный файл.

Пример:

```
$ ln -s /dev/fd0 .
$ ls -l fd0
lrwxrwxrwx 1 user1 user1 8 Дек 16 01:13 fd0 -> /dev/fd0
```

Размер файла символьной ссылки равна длине имени файла, на который она указывает.

Команда `cp` обладает специальной опцией `-s` , которая позволяет вместо копирования файлов создавать на них символьные ссылки

Пример:

```
$ cp -s ~/f??? /tmp
$ ls -l /tmp/f???
lrwxrwxrwx 1 user1 user1 16 Дек 16 01:52 /tmp/f112 ->
/home/user1/f112
lrwxrwxrwx 1 user1 user1 16 Дек 16 01:52 /tmp/f113 ->
/home/user1/f113
lrwxrwxrwx 1 user1 user1 16 Дек 16 01:52 /tmp/f117 ->
/home/user1/f117
lrwxrwxrwx 1 user1 user1 16 Дек 16 01:52 /tmp/f122 ->
/home/user1/f122
```

Примечание: В этом примере на группу файлов были созданы символьные ссылки в каталоге /tmp.

Если используется команда `cp` , для которой в качестве аргумента указаны файлы символьных ссылок, то будут скопированы не файлы символьных ссылок, а файлы-оригиналы, на которые они указывают.

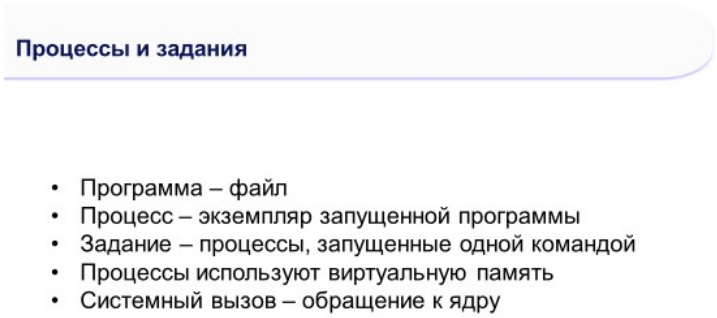
Если же необходимо скопировать не исходные файлы, а именно символьные ссылки, то следует использовать опцию `-d` команды `cp`

Пример:

```
$ cp -d /tmp/f122 Documents/
$ ls -l Documents/f122
lrwxrwxrwx 1 user1 user1 16 Дек 16 02:00 Documents/f122 ->
/home/user1/f122
```

Глава 13. Процессы.

13.1. Процессы и задания.



GNU/Linux – многопользовательская и многозадачная операционная система.

Примечание: Это значит, что одновременно в системе может выполняться множество программ, запущенных различными пользователями. Для обеспечения такой работы центральный процессор компьютера последовательно переключается на обработку программного кода различных приложений, системных и прочих программ, создавая впечатление, что все они выполняются одновременно. Ядро GNU/Linux спроектировано таким образом, что имеется гарантия выполнения центральным процессором кода каждой работающей в системе программы за некоторое конечное время. Естественно, чем больше загружена система, тем большим становится этот промежуток времени. Сильно загруженная система может (с точки зрения пользователя) потерять интерактивность, то есть она может настолько медленно отвечать на запросы пользователя, что он будет уверен в том, что система не работает.

Программы представляют собой исполняемые файлы двух типов:

1) Скомпилированные бинарные файлы, содержащие инструкции на машинном языке.
2) Интерпретируемые сценарии (например, сценарии Bash или программы Perl).

Примечание: Когда от пользователя поступает команда выполнить

некоторую программу, операционная система в начале определяет с помощью “магических чисел” (magic numbers см. man file) к какому из этих двух типов относится данная программа. Если она относится к первому типу, то программа пополняет очередь других работающих программ, ожидающих своей очереди постановки на исполнение процессором. В противном случае сначала запускается программа – интерпретатор, предназначенный для исполнения данного вида сценариев (например, /bin/bash), который, в свою очередь, относится к программам первого типа.

Программа – это некоторый код, хранящийся в обычном (регулярном) исполняемом файле.

Процесс – это экземпляр программы, находящийся на исполнении в процессоре, либо ожидающий этого момента в очереди заданий.

Примечание: Программа – это исполняемый файл, а процесс – это исполняющийся машинный код.

Задание (task, job) – это процесс(ы), запущенный(е) пользователем с помощью одной команды.

Процессоры, аппаратно обеспечивающие многозадачность, поддерживают не менее двух режимов (уровней) выполнения.

Примечание: Процесоры архитектуры IA-32 поддерживаю четыре уровня выполнения.

Нулевой уровень соответствует привилегированному режиму, который используется ядром GNU/Linux. Пользовательские процессы исполняются в непривилегированном режиме и не могут повредить ядро. GNU/Linux использует концепцию виртуальной памяти, т.е. адреса ячеек памяти, с которыми работает процесс не являются адресами физической памяти. Адреса виртуального пространства трансформируются в реальные (физические) адреса с помощью набора карт трансляции адресов, которые реализуются в виде таблиц страниц. Страница – это выделенный и защищенный блок памяти фиксированного размера.

Примечание: Современные процессоры архитектуры IA-32 поодерживают страницы размеров 4Kb и 4Gb.

Регистры блока управления памятью (часть процессора, memory management unit, MMU) хранят данные необходимые для трансляции адресов

для процесса, который в данный момент исполняется на процессоре. При вытеснении процесса новым процессом происходит замена указателей на новые карты трансляции (так называемое, переключение контекста)

Регистры MMU доступны только в режиме ядра.

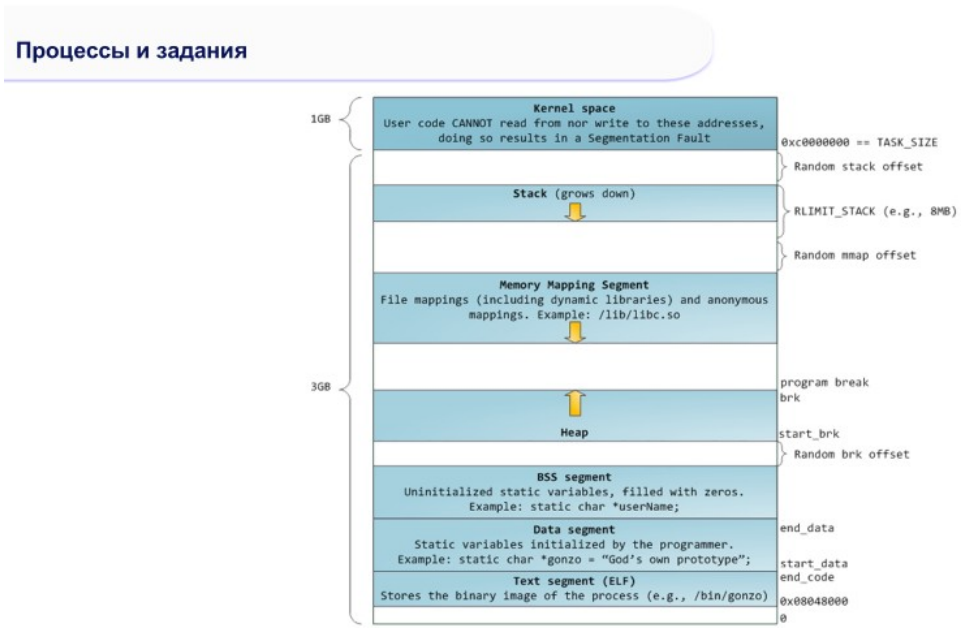
Примечание: Это делает возможным изоляцию процессов, поскольку каждый пользовательский процесс имеет доступ только к своим страницам памяти и не может получить доступа к чужим.

Каждый процесс работает в своем собственном виртуальном адресном пространстве. Общий размер всех виртуальных пространств может превышать объем физической памяти. Механизм подкачки (swapping) используется для освобождения места в физической памяти и помещения неиспользуемых страниц памяти или процессов целиком в область подкачки (swap), которая находится на жестком диске. Определенная часть виртуального пространства процесса отображается на пространство, в котором находится ядро.

Процессы не могут напрямую обращаться к ядру, обращение происходит с помощью интерфейса системных вызовов. Обращение к системному вызову переводит процесс в режим ядра, которое производит работу от имени процесса, после завершения работы системного вызова исполнение процесса возвращается в пользовательский режим. При запуске процесс переводится в состояние готовности к дальнейшей работе. По истечении некоторого промежутка времени ядро выбирает запущенный ранее процесс для исполнения, переключает контекст и передает управление процессу.

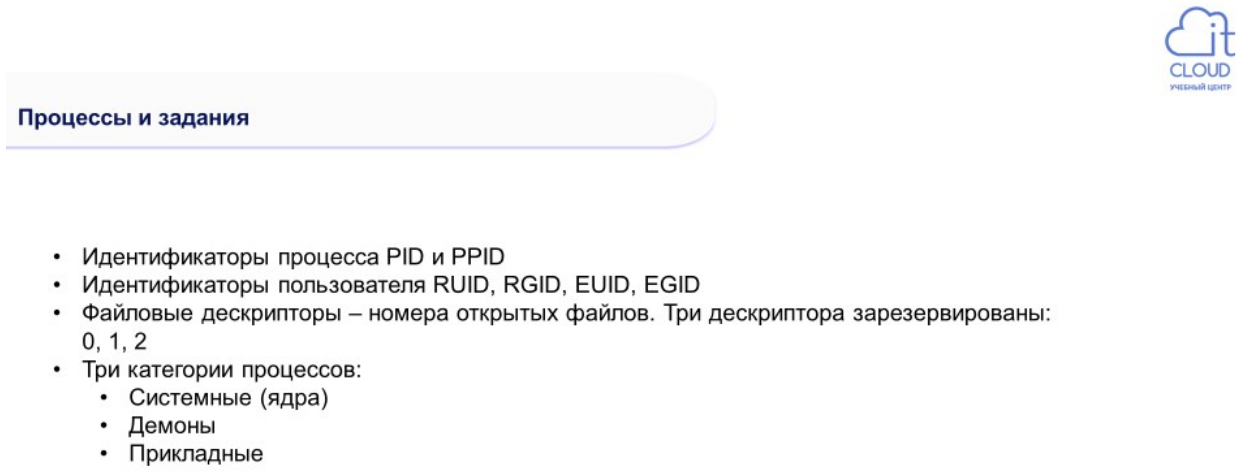
Основные состояния процесса: готов к работе (R, running), спящее (S, sleeping), выполнение. Процессор выделяет каждому процессу кванты времени (time slices), в течение которых инструкции процесса выполняются процессором. Только один процесс в каждый момент времени может выполняться на процессоре, но одновременно может существовать много процессов

Используемая в Linux модель управления пользовательскими процессами относится к классу вытесняющей многозадачности (preemptive multitaskings). В рамках этой модели каждый процесс имеет свой уровень важности в системе или приоритета. Более высоко приоритетные процессы могут вытеснять с исполнения на процессоре менее приоритетные процессы.



Адресное пространство процесса состоит из следующих областей:

1. Заголовок процесса, содержащий, в частности, специальные сигнатуры, показывающие формат процесса или магическое число (например, формат ELF или COFF). Заголовок процесса содержит размеры областей, точку входа (адрес первой инструкции).
2. Инструкции (text).
3. Инициализированные данные (data)
4. Неинициализированные данные (bss, block static storadge)
5. Разделяемая память (shared memory)
6. Разделяемые библиотеки (shared libraries)
7. Куча (heap) – область памяти, предназначенная для динамически выделяемой памяти, выделение и увеличение размера кучи производится ядром
8. Стек (stack) – структура типа LIFO (Last Input – First Output), предназначенная для вызовов подпрограмм и хранения некоторых переменных, выделяется ядром.



Управляющая информация о процессе поддерживается с помощью структур:

1. область u (u-area)
2. структуры proc (p-area)

Структура proc называется таблицей процессов и находится в системном пространстве (пространстве ядра, kernel space)

Область u является частью пространства пользовательского процесса, но процесс не может изменять информацию в этой области.

- В области u хранится следующая информация:
    1. аппаратный контекст (значения регистров MMU)
    2. указатель на proc
    3. RUID и EUID – реальный и эффективные идентификаторы пользователя (кто запустил процесс и от имени кого идет выполенние)
    4. RGID и EGID – реальный и эффективные идентификаторы группы
    5. таблица дескрипторов открытых файлов
- В p области хранится
    1. PID (Process ID) – уникальный порядковый номер (идентификатор) процесса в системе.
    2. PPID (Parent Process ID) – PID родительского процесса.
    3. Текущее состояние процесса
    4. приоритеты и флаги

5. расположение области u

При создании процесса ядро ассоциирует с ним три стандартных потока (открытых файла):

1. Стандартный поток ввода (stdin) – дескриптор 0;
2. Стандартный поток вывода (stdout) – дескриптор 1;
3. Стандартный поток ошибок (stderr) – дескриптор 2.

Процессы создаются другими процессами с помощью системного вызова fork().

Процесс, созданный другим процессом, называется по отношению к последнему потомком или дочерним процессом. И, наоборот, тот процесс, который создал некоторый дочерний процесс, является для него родительским.

У каждого процесса есть один и только один родитель, но у процесса может быть более одного потомка.

Иерархия процессов представляет собой перевернутое дерево, в корне которого находится процесс init (PID=1)

Когда процесс – родитель завершает свою работу, он должен завершить работу своих дочерних процессов.

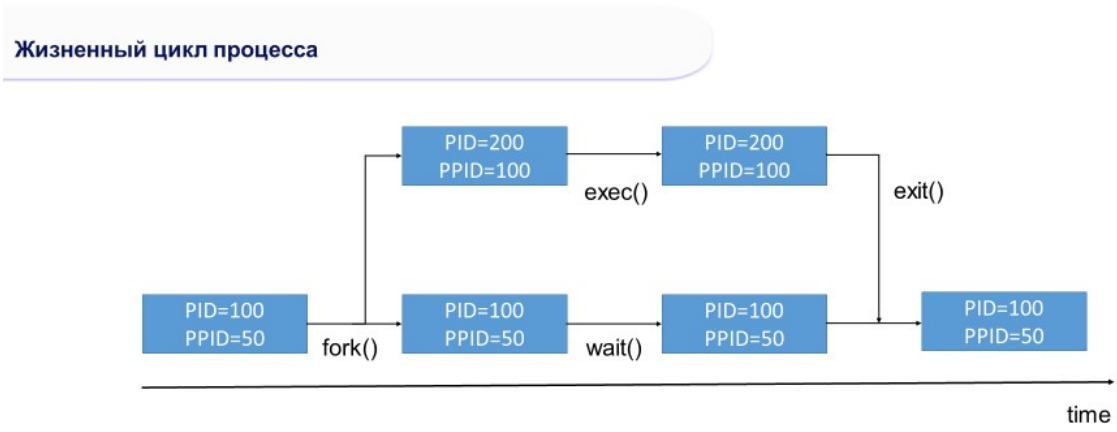
Процессы можно разделить на три типа:

1. Процессы ядра или системные процессы.

Примечание: Они происходят в kernel space и не видны пользователю. Примером такого процесса в Linux является процесс постановки процессов в очередь (sheduling). Процессы ядра, разумеется, не связаны ни с каким терминалом, однако, они при желании могут выводить информацию на терминал. Для этого в коде ядра есть специальные подпрограммы. Процессы ядра всегда располагаются в оперативной памяти.

2. Демоны (сервисы) – это фоновые неинтерактивные процессы несвязанные с терминалами и запускаемые системными пользователями.
3. Прикладные процессы - порождаются в рамках сеанса работы пользователя, связаны с терминалом.

13.2. Жизненный цикл процесса.



Родительский процесс создает дочерний процесс с помощью системного вызова `fork()`.

Процесс-потомок является точной копией родительского процесса.

Вызов `fork()` приводит к тому, что ядро дублирует адресное пространство процесса, произведшего этот вызов, в свободное адресное пространство в памяти.

В дочернем процессе, как правило, незамедлительно выполняется системный вызов `exec()`, который заменяет код только что созданного процесса на код новой программы

Примечание: вызов `exec()` может не осуществляться, если дочерний процесс должен выполнять ту же работу, что и родительский процесс, т.е. если не нужно менять код дочернего процесса.

В то же время работа родительского процесса приостанавливается системным вызовом `wait()` до завершения работы дочернего процесса.

Примечание: родительский процесс может и не останавливаться на время работы дочернего, если дочерний процесс запускается в фоновом режиме (background)

Процесс завершается путем обращения к системному вызову `exit()`

Системный вызов `exit()` освобождает адресное пространство,

закрывает все открытые файлы и переводит процесс в состояние “зомби” (zombie, defunct), будит, если нужно, родительский процесс.

Вызов `exit()` не освобождает структуру proc. За эту операцию отвечает родительский процесс.

Примечание: Вполне возможна ситуация, когда дочерний процесс уже завершил свою работу, а родительский процесс еще не успел произвести системный вызов wait(). В таком случае информация об уже несуществующем дочернем процессе (структура proc) сохраняется в таблице процессов, а запись в таблице о завершившемся дочернем процессе так и остается помеченной как defunct или, иначе, процесс – зомби (zombie). Структура proc, помеченная как zombie, остается в таблице до перезагрузки системы

Если родительский процесс завершается раньше процесса-потомка, то потомок удочеряется процессом `init`.

13.3. Фоновый режим выполнения заданий.

В GNU/Linux пользователь может запускать процессы или в интерактивном (foreground) или в фоновом (background) режиме.

Только одно задание в рамках сеанса может находиться в интерактивном режиме , все остальные активные задания выполняются в фоновом режиме.

Для запуска команды в фоновом режиме в конце командной строки необходимо поставить символ &, при этом выводится номер задания:

Пример:

```
$find ~/ -name "*t" &
[1]546
```

Примечание: В этом примере команда find запущена в фоновом режиме, так как в конце командной строки установлен символ &. Номер задания выводится в квадратных скобках, в этом примере – 1. Число, выводящееся после квадратных скобок - PID процесса задания.

Для мониторинга состояний фоновых заданий предназначена команда `jobs`, которая позволяет просмотреть статус фоновых заданий.

Она отображает номер задания, имя команды и статус задания:

Пример:

```
$find / –empty –user basile –exec rm –f {} \; &
[1]548
$find / –empty –user user1 –exec rm –f {} \; &
[2]551
$jobs
[1]-Done find / –empty –user basile rm –f {} \;
[2]+Running find / –empty –user user1 rm –f {} \;
```

Примечание: В этом примере были запущены два задания на поиск и удаление пустых файлов пользователей basile и user1. Заданиям назначены номера 1 и 2. Команда jobs показала, что задание 1 выполнено, а задание 2 выполняется.

Обозначения `%%` и `%+` показывают последнее запущенное фоновое задание, а `%–` - предпоследнее задание.

В выводе команды `jobs` символы `+` и `–` для индикации последнего задания и предпоследнего заданий.

Команда `fg %номерзадания` переводит задание с номером `номерзадания` в интерактивный режим из фонового (можно пользоваться командой `%номерзадания`).

Пример: Так, команды `fg %1` и `%1` переводят задание с номером 1 в интерактивный режим.

Наоборот, для перевода задания в фоновый режим необходимо приостановить его выполнение сочетанием `Ctrl-Z`, а затем выполнить команду `bg` с аргументом `%номерзадания` (можно пользоваться командой `%номерзадания &`).

Пример: Команды `bg %1` и `%1 &` переводят (после приостановки нажатием Ctrl-Z) задание с номером 1 в фоновый режим из интерактивного режима.

Вместо использования конструкции `%номерзадания` можно обращаться к заданиям по первым символам их команд, предваряя их символом `%`:

Пример:

```
$jobs
[2]+Running find / –empty –user user1 rm –f {} \;
```

```
$fg %fi
```

Примечание: В этом примере выполняющееся в фоновом режиме задание было переведено в интерактивный режим командой fg %fi, которая использовала идентификацию задания не по его номеру, а по двум первым буквам в имени команды.

Для прекращения работы фонового задания необходимо использовать команду `kill %номерзадания`

Пример:

```
$jobs
[2]+Running find / –empty –user user1 rm –f {} \;
$kill %2
$jobs
[2]+Terminated find / –empty –user user1 rm –f {} \;
```

Примечание: В этом примере задание с номером 2 было завершено командой `kill`.

13.4. Мониторинг процессов.

- ps
- top
- /proc
- fuser

Основным инструментом для исследования процессов является команда `ps`, которая выводит мгновенное состояние процессов.

Команда `ps` без аргументов выводит список процессов, связанных с текущим терминалом.

Пример:

```
$ ps
PID TTY TIME CMD
1751 pts/0 00:00:00 bash
1890 pts/0 00:00:00 ps
```

Столбец `PID` отображает идентификаторы процессов.

Столбец `TTY` – имена терминалов

Столбец `TIME` – суммарное процессорное время, затраченное процессом с момента его старта.

Столбец `CMD` – командная строка, соответствующая данному процессу.

Более подробную информацию можно получить с помощью опции подробного вывода (full format) `-f`.

Пример:

```
$ ps -f
UID PID PPID C STIME TTY TIME CMD
user1 1751 1745 0 21:01 pts/0 00:00:00 bash
user1 1970 1751 0 23:14 pts/0 00:00:00 ps -f
```

`UID` – владелец процесса.

`STIME` показывает время запуска процесса

столбец `C` – уровень загрузки процессора

Величины, находящиеся в столбце `C` целочисленные, они участвуют в вычислении приоритета процесса планировщиком. Чем выше эта величина, тем ниже приоритет.

Примечание: Когда процесс ожидает своей очереди постановки на процессор, эта величина постепенно снижается на единицу на каждом цикле работы планировщика, то есть приоритет процесса постепенно повышается. После работы процесса на процессоре в этом столбце устанавливается положительное значение, которое затем постепенно снижается.

Детальную информацию о процессах можно получить, используя опцию `-l` (long format).

Пример:

```
$ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 500 1751 1745 0 76 0 - 781 11c541 pts/0 00:00:00 bash
0 R 500 1988 1751 0 77 0 - 617 - pts/0 00:00:00 ps
```

Столбец `F` – показывает флаги процесса, отображающие его особые свойства (см. `man ps`).

Столбец `S` – статус процесса, который может быть:

1. `D` – процесс приостановлен и не может быть прерван (например, ожидает окончания ввода-вывода).
2. `R` – процесс выполняется или находится в очереди.
3. `S` – процесс “спит” (доступ к процессору не требуется).
4. `T` – процесс трассируется.
5. `Z` – процесс defunct (zombie).

Столбец NI показывает величину Nice Number. Эта константа устанавливается пользователем или администратором и участвует в вычислении приоритета процесса планировщиком.

Примечание: Другое ее название – относительный приоритет.

Столбец SZ – количество памяти, занимаемое процессом

WCHAN – это адрес или имя функции ядра, обслуживающей текущее состояние спящего процесса (статус S).

Опция -e позволяет вывести список всех процессов в системе.

Примечание: Также для этого можно пользоваться опцией -A. Чаще всего для получения списка всех процессов используют команду ps -ef, дающую подробную информацию о процессах.

Наиболее популярные опции программы ps для предназначенные для фильтрации выводимой информации:

1. -u – фильтрация по UID.
2. -t – по терминалу.
3. -p – по PID искомого процесса.
4. -C – по командной строке.

Пример: Требуется вывести список процессов, запущенных на второй виртуальной консоли:

```
$ ps -ft tty2
UID PID PPID C STIME TTY TIME CMD
root 1557 1 0 06:18 tty2 00:00:00 /sbin/mingetty tty2
```

Все приведенные выше опции команды ps соответствуют POSIX формату, однако GNU версия программы ps поддерживает также опции и в BSD стиле.

Наиболее популярным набором опций команды ps в таком формате является ps aux – она выводит список всех процессов в системе с указанием их владельцев.

Пример: Ниже приведен небольшой фрагмент из ее вывода:

```
$ ps aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.1 1268 60 ? S Oct08 0:04 init
```

Примечание: Эта команда отобразила в виде процентов уровень загрузки процессора и памяти данным процессом. Столбец VSZ – объем используемой процессом виртуальной памяти и RSS – физической памяти.

Опции POSIX, BSD и длинные опции команды `ps` можно комбинировать.

Пример:

```
$ ps -f U root
UID PID PPID C STIME TTY STAT TIME CMD
root 1 0 0 06:18 ? S 0:04 init
root 2 1 0 06:18 ? SW 0:00 [keventd]
root 3 1 0 06:18 ? SWN 0:00 [ksoftirqd_CPU0]
root 4 1 0 06:18 ? SW 0:00 [kswapd]
root 5 1 0 06:18 ? SW 0:00 [bdflush]
root 6 1 0 06:18 ? SW 0:00 [kupdated]
root 7 1 0 06:18 ? SW 0:00 [kinoded]
root 8 1 0 06:18 ? SW< 0:00 [mdrecoveryd]
root 11 1 0 06:18 ? SW 0:00 [kreiserfsd]
root 846 1 0 06:18 ? SW 0:00 [khubd]
root 1016 1 0 06:18 ? S 0:00 /sbin/cardmgr
root 1290 1 0 06:18 ? SW 0:00 [kapmd]
root 1314 1 0 06:18 ? S 0:00 crond
root 1499 1 0 06:18 ? S 0:00 /usr/lib/postfix/master
root 1518 1 0 06:18 ? S 0:00 gpm -m /dev/psaux -t imps2
root 1556 1 0 06:18 tty1 S 0:00 /sbin/mingetty tty1
root 1557 1 0 06:18 tty2 S 0:00 /sbin/mingetty tty2
root 1558 1 0 06:18 tty3 S 0:00 /sbin/mingetty tty3
root 1559 1 0 06:18 tty4 S 0:00 /sbin/mingetty tty4
root 1560 1 0 06:18 tty5 S 0:00 /sbin/mingetty tty5
root 1561 1 0 06:18 tty6 S 0:00 /sbin/mingetty tty6
root 1563 1 0 06:18 ? S 0:00 kdm -nodaemon
root 1590 1563 0 06:18 ? R 0:03 /etc/X11/X -auth /var/run/xaut
root 1591 1563 0 06:18 ? S 0:00 -:0
```

Примечание: В этом примере для выбора процессов root была использована опция в BSD стиле U, а POSIX опция -f была использована для указания подробного формата вывода.

Для постоянного мониторинга процессов используется утилита `top`, которая отображает исполняющиеся процессы, использующие большую часть процессорного времени и наиболее сильно использующие память.

Утилита `top` регулярно обновляет информацию о процессах.

Для выхода из `top` необходимо набрать `q`.

Команда `top` позволяет, не выходя из интерактивного просмотра процессов, посылать процессам сигналы с помощью нажатия на клавишу `k`.

Нажатие на `i` отключает вывод утилитой `top` неактивных процессов.

В первой строке экрана вывода команды приводятся данные о средней загруженности системы load averages за последние 1, 5 и 15 минут. Далее статистка процессов, загруженность процессора, памяти и пространства подкачки.

Для получения информации о процессах можно использовать каталог `/proc`.

`/proc` – это псевдофайловая система, порождаемая ядром.

`/proc` позволяет также получать и устанавливать (суперпользователю) параметры ядра на лету.

Пример:

```
$ ls /proc
1 1508 1590 3710 846 driver kcore mtrr sys
1016 1518 1591 3718 966 execdomains kmsg net sysvipc
11 1537 1620 4 967 fb ksyms partitions tty
1260 1556 1687 5 apm filesystems loadavg pci uptime
1277 1557 1745 5090 asound fs locks scsi version
1290 1558 1751 5233 bus ide mdstat self
1296 1559 2 6 cmdline interrupts meminfo slabinfo
1314 1560 3 7 cpuinfo iomem misc splash
1499 1561 3429 7156 devices ioports modules stat
1507 1563 3709 8 dma irq mounts swaps
```

Примечание: Подкаталоги /proc с именами, состоящими из цифр, соответствуют исполняемым процессам.

Пример: Определим PID текущей оболочки и исследуем соответствующий ее процессу каталог.

```
$ ps
PID TTY TIME CMD
1751 pts/0 00:00:00 bash
7157 pts/0 00:00:00 ps
$ ls /proc/1751
cmdline cwd environ exe fd maps mem mounts root stat statm status
$ cat /proc/1751/cmdline
bash
```

```
$ cat /proc/1751/status
Name: bash
State: S (sleeping)
Tgid: 1751
Pid: 1751
PPid: 1745
TracerPid: 0
Uid: 500 500 500 500
Gid: 500 500 500 500
FDSize: 256
Groups: 500 4 10 51 55 100 103
VmSize: 3124 kB
VmLck: 0 kB
VmRSS: 716 kB
VmData: 628 kB
VmStk: 24 kB
VmExe: 428 kB
VmLib: 1668 kB
SigPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 8000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 000000000000000
```

Примечание: Видно, что файл cmdline содержит в себе командную строку, породившую процесс, а файл status – подробную информацию о статусе процесса.

Подкаталог `fd` предназначен для мониторинга файлов, открытых процессом.

В каталоге `fd` содержатся символические ссылки на реально открытые файлы процессом. Имена этих символических ссылок соответствуют номерам файловых дескрипторов открытых файлов.

Команда `fuser` позволяет определить, какие процессы используют какой-либо файл.

Пример: пользователь смонтировал дискету в каталоге `/mnt/floppy`, перешел в этот каталог и получил информацию о процессе, использующем этот каталог.

```
$ mount /mnt/floppy
$ cd /mnt/floppy/
$ /sbin/fuser .
.: 1751c
```

```
$ ps
PID TTY TIME CMD
1751 pts/0 00:00:00 bash
7188 pts/0 00:00:00 ps
```

Примечание: Команда fuser вывела PID процесса, соответствующему оболочке, так как для нее этот каталог является текущим. Этот факт команд fuser подтверждает, выводя букву с после PID процесса.

Команда `fuser` выводит следующие символы после PID процессов:

1. `c` – текущий каталог.
2. `e`- исполняемый файл в момент его работы.
3. `f` – открытый файл.
4. `r` – корневой каталог.
5. `m` – разделяемая библиотека, либо отображаемый в память файл.

Опция `-m` команды `fuser` позволяет указать, что имя файла является именем смонтированного блочного устройства:

Пример:

```
$ /sbin/fuser -m /dev/fd0
/dev/fd0: 1751c
```

Примечание: В этом примере указана опция -m, так как в качестве аргумента команды используется имя смонтированного блочного устройства, соответствующего флоппи диску.

Для просмотра списка процессов в виде иерархического дерева, отображающего отношения родительских и дочерних процессов, необходимо выполнить команду `pstree`

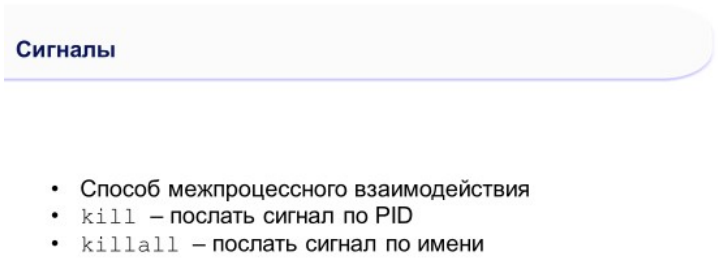
Пример:

```
$pstree
init-+-2*[automount]
|-cron
|-6*[getty]
|-inetd
|-kdm-+-XF86_SVGA
| `-kdm-+-kwm-+-kaudioserver---maudio
| | |-kbgndwm
| | |-kfm---konsole---bash---pstr
| | |-kpanel
```

```
| | |-krootwm
| | `-kwmsound
| `-netserv
|-kflushd
|-klogd
|-kpiod
|-kswapd
|-lpd
|-portmap
|-proftpd
|-sleep
|-syslogd
`-xfs
```

Примечание: Иерархический список процессов, выведенный командой pstree.

13.5. Сигналы.



Сигналы – это один из способов межпроцессного взаимодействия.

Примечание: Они обеспечивают возможность передачи процессами друг-другу команд и сообщений, таких, как, например, сообщение о завершении работы дочернего процесса, или команда перечитать файл конфигурации, или сообщение об ошибке операции с плавающей запятой. Одно из главных направлений использования сигналов состоит в снятии процессов с исполнения.

Список сигналов, используемых в системе, можно получить с помощью команды

```
kill -l
```

Подробное описание сигналов доступно с помощью `man 7 signal`.

Наиболее часто приходится использовать следующие сигналы:

1. 1 – `HUP` – разрыв связи с терминалом (Hang Up – положить трубку). Многие демоны используют этот сигнал, как команду перечитать их конфигурационный файл и продолжить работу с измененными настройками. Оболочка Bash реагирует на этот сигнал завершением сеанса.
2. 2 – `INT` – клавиатурное прерывание процесса. В GNU/Linux генерируется при нажатии `Ctrl-C`.
3. 3 – `QUIT` – “отключение” клавиатуры. Получение этого сигнала обычно

снимает процесс с исполнения.

4. 15 – `TERM` – сигнал для завершения процесса. Получив этот сигнал приложение должно (но не обязано) завершить свою работу корректно, закрыв потоки ввода-вывода. Этот сигнал команда `kill` посылает по умолчанию.
5. 9 – `KILL` – безусловное и немедленное снятие процесса с исполнения без корректного завершения процесса.

Примечание: Каким образом то или иное приложение реагирует на получение некоторого сигнала зависит от того, как эта программа написана. В программе получение сигнала может перехватываться и обрабатываться специальным образом. Сигнал `KILL` не может быть перехвачен. Этот сигнал приводит к немедленному и, таким образом, часто некорректному снятию процесса с исполнения. При этом файлы, открытые процессом не закрываются нормальным способом, что может привести к существенной потере данных или даже сбою настроек терминала.

Команда `kill` посылает заданный сигнал процессу, номер которого указывается после дефиса.

Если номер сигнала или его имя не заданы после дефиса, то команда `kill` посылает целевым процессам сигнал 15 (`TERM`).

Пример: Посылаем сигнал оболочке Bash:

```
$ ps
PID TTY TIME CMD
2179 pts/0 00:00:00 bash
2180 pts/0 00:00:00 ps
$ kill 2179
$ kill -2 2179

$ kill -1 2179

login:
```

Примечание: Этот пример демонстрирует, что оболочка Bash игнорирует сигналы 15 (TERM) и 2 (INT) , а получив сигнал 1 (HUP) она завершает работу и выходит из сеанса.

Посылать сигналы процессам могут только их владелец и суперпользователь.

Если родительским процессом получен сигнал, приводящий к его остановке, то сняты с выполнения будут также и все его дочерние процессы.

Команда `killall` послать требуемый сигнал процессам, в командных

строках которых присутствует заданная строка.

Пример: следующая команда приведет к немедленному останову всех копий демона `httpd`:

```
# killall -9 httpd
```

Примечание: В результате передачи сигнала 9 (KILL) всем процессам httpd они будут сняты с исполнения, так как перехватить такой сигнал невозможно.

13.6. Перехват и обработка сигналов в Bash.

- `trap`

Встроенная команда оболочки Bash `trap` позволяет перехватывать сигналы и реагировать на них каким-либо заданным способом.

Первым аргументом является команда, которую следует выполнить при получении оболочкой сигнала.

Второй аргумент задает сигнал, который должен быть обработан.

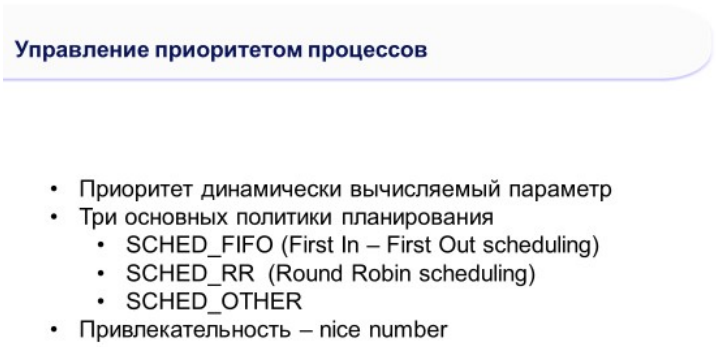
Пример: выполняются следующие команды для установки ловушек сигналов `INT`, `QUIT` и на событие выхода из оболочки `EXIT`:

```
$ trap "echo Получен сигнал INT" INT
$ trap "echo А это был QUIT" QUIT
$ trap "echo Пока!" EXIT
$ trap -p
trap -- 'echo Пока!' EXIT
trap -- 'echo Получен сигнал INT' SIGINT
trap -- 'echo А это был QUIT' SIGQUIT
$ Получен сигнал INT

$ ps
PID TTY TIME CMD
1811 pts/0 00:00:00 bash
1812 pts/0 00:00:00 ps
$ kill -3 1811
А это был QUIT
```

```
$ exit
Пока!
login:
```

Примечание: Команды trap установили ловушки для сигналов – команды echo. Для сигнала INT на экран будет выведено Получен сигнал INT, для QUIT - А это был QUIT, а при выходе из оболочки на экране будет отображаться Пока! , Команда trap -p вывела список установленных обработчиков сигналов. Далее пользователь нажал сочетание Ctrl-C, передающее сигнал INT оболочке. При этом сигнал был перехвачен обработчиком и сработала соответствующая команда echo. Далее оболочке был передан сигнал QUIT, и снова сработал соответствующий обработчик. Выход из оболочки также привел к срабатыванию ловушки.

13.7. Управление приоритетом процессов.



Часть ядра, называемая планировщиком (scheduler), определяет, какой из готовых к работе процессов (runnable process) будет выполняться на центральном процессоре (CPU)

Планировщик в ядре Linux поддерживает три различных класса (политики планирования, scheduling policy) процессов:

1. SCHED_FIFO (First In – First Out scheduling)
2. SCHED_RR (Round Robin scheduling)
3. SCHED_OTHER

Классы SCHED_FIFO и SCHED_RR используются для процессов программного (псевдо) реального времени (soft real time)

Процессы класса SCHED_OTHER используются для стандартных процессов, работающих в режиме разделения времени и является классом процесса по умолчанию.

Команда `chrt` позволяет запустить процесс с заданным классом или изменить класс у существующего процесса.

Опция `-c` команды `ps` выводит класс процесса в столбце `CLS`

Пример:

```
# ps -c
PID CLS PRI TTY TIME CMD
4644 - 24 pts/6 00:00:00 bash
4730 - 24 pts/6 00:00:00 ps

# chrt 10 bash
[root@trainer root]# ps -c
PID CLS PRI TTY TIME CMD
4644 - 24 pts/6 00:00:00 bash
4731 RR 50 pts/6 00:00:00 bash
4763 RR 50 pts/6 00:00:00 ps
```

Каждому процессу присваивается некоторое значение статического приоритета (static sched_priority)

Статический приоритет может изменяться в пределах от 0 до 99.

Процессы класса SCHED_OTHER имеют статический приоритет равный 0. Процессы класса SCHED_FIFO и SCHED_RR имеют статический приоритет от 1 до 99. Для вывода статического приоритета можно использовать опцию `-o rtprio` команды `ps`

Пример:

```
# ps -o pid,rtprio,cmd
PID RTPRIO CMD
4644 0 bash
4731 10 bash
4764 10 ps -o pid,rtprio,cmd
```

Планировщик для каждого статического приоритета поддерживают свою очередь процессов готовых к вполнению (runnable process). При выборе процесса для исполнения планировщик просматривает непустую очередь с наивысшим статическим приоритетом, исполняя процесс, находящийся в начале очереди. Планирование имеет вытесняющий характер. Если у готового к работе процесса более высокий статический приоритет, то он вытеняет с процессора текущий процесс. Политика планирования (класс) определяет перемещение процесса внутри очереди.

Процесс класса SCHED_FIFO занимает процессор до тех пор, пока не будет вытеснен более приоритетным процессом или не будет заблокирован в ожидании завершения выполнения запроса на операцию ввода/вывода.

Процесс класса SCHED_RR практически соответствует классу SCHED_FIFO за исключением того, что каждому процессу этого класса позволено занимать процессор не более максимального кванта (slice, quant) времени.

Все процессы класса SCHED_OTHER находятся в одной очереди, соответствующей статическому приоритету 0.

Выбор процесса из очереди с нулевым статическим приоритетом происходит на основе динамического приоритета. Динамический приоритет вычисляется с использованием постоянного числа nice и фактора, характеризующего как давно готовый к работе процесс не был на процессоре.

Пример:

```
$ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 500 1747 1741 0 74 0 - 783 11c541 pts/0 00:00:00 bash
0 R 500 2510 1747 4 77 0 - 618 - pts/0 00:00:00 ps
```

Чем ниже значение nice number, тем более высокий приоритет будет у процесса.

В GNU/Linux значение nice number задается в пределах от -20 до 19.

По умолчанию nice number устанавливается в 0.

Обычных пользователи могут только увеличивать nice, уменьшать nice может только суперпользователь.

Значение nice number можно установить с помощью команды `nice`, после которой в качестве аргумента задается команда, которая должна быть исполнена с измененным приоритетом.

- nice
- renice

По умолчанию команда `nice` увеличивает значение nice number на 10.

Если требуется указать иное значение увеличения nice number , то его следует указать после опции `-n`.

Команда `nice`, вызванная без аргументов, отображает заданное значение nice number для данной оболочки.

Пример: Запустим Bash с пониженным приоритетом:

```
$ nice -n 19 bash
$ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 500 1747 1741 0 74 0 - 784 11c541 pts/0 00:00:00 bash
0 S 500 2559 1747 0 78 19 - 780 11c541 pts/0 00:00:00 bash
0 R 500 2560 2559 0 79 19 - 618 - pts/0 00:00:00 ps
$ nice
19
```

Примечание: Здесь продемонстрировано, как с помощью команды nice -n 19 bash, была запущена оболочка Bash с пониженным приоритетом. В поле NI для этой оболочки команда ps -l выводит значение 19. Поле PRI этого же листинга показывает, что приоритет запущенной оболочки Bash действительно снижен (78 у Bash с пониженным приоритетом против 74 у исходной оболочки).

Для установки иного значения nice number для уже исполняющегося процесса следует использовать команду `renice`.

Пример: для изменения nice number оболочки Bash из предыдущего примера, суперпользователь может выполнить следующую команду:

```
# renice 10 2559
2559: old priority 19, new priority 10
# ps -l -p 2559
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 500 2559 1747 0 76 10 - 782 read_c pts/1 00:00:00 bash
```

Примечание: Данная команда установила новое значение приоритета (поле NI) для оболочки, которая была исходно запущена с nice number 19. Заметно, что приоритет процесса (см. поле PRI) также изменился.

С помощью команды `renice` можно изменять приоритет всех процессов для заданного после опции `-u` пользователя, или после `-g` группы пользователей.

Пример:

```
# renice 0 -u user1
```

Примечание: Эта команда установит значение nice number для всех процессов, принадлежащих пользователю user1, в 0.