Часть 1. Теоретические материалы

Оглавление

01. Введение. История Linux. Дистрибутивы.

Семейство ОС типа UNIX

Операционная система — это комплекс программ, который обеспечивает управление аппаратными средствами компьютера, организует работу с файлами (в том числе запуск и управление выполнением программ), а также реализует взаимодействие с пользователем, т. е. интерпретацию вводимых пользователем команд и вывод результатов обработки этих команд.

Без операционной системы компьютер вообще не может функционировать в качестве такового. В таком случае он представляет собой не более чем совокупность неработающих электронных устройств, непонятно зачем собранных воедино.

На сегодняшний день наиболее известными операционными системами для компьютеров являются семейства операционных систем Microsoft Windows и UNIX. Первые ведут свою родословную от операционной системы MS-DOS, которой оснащались первые персональные компьютеры фирмы IBM. Операционная система UNIX была разработана группой сотрудников Bell Labs под руководством Денниса Ричи, Кена Томпсона и Брайана Кернигана (Dennis Ritchie, Ken Thompson, Brian Kernighan) в 1969 году. Но в наши дни, когда говорят об операционной системе UNIX, чаще всего имеют в виду не конкретную ОС, а скорее целое семейство UNIX-подобных операционных систем. Само же слово UNIX (заглавными буквами) стало зарегистрированной торговой маркой корпорации AT&T.

В конце 70-х годов (теперь уже прошлого столетия) сотрудники Калифорнийского университете в Беркли внесли ряд усовершенствований в исходные коды UNIX, включая работу с протоколами семейства TCP/IP. Их разработка стала известна под именем BSD ("Berkeley Systems Distribution"). Она распространялась под лицензией, которая позволяла дорабатывать и усовершенствовать продукт, и передавать результат третьим лицам (с исходными кодами или без них) при условии, что будет указано, какая часть кода разработана в Беркли.

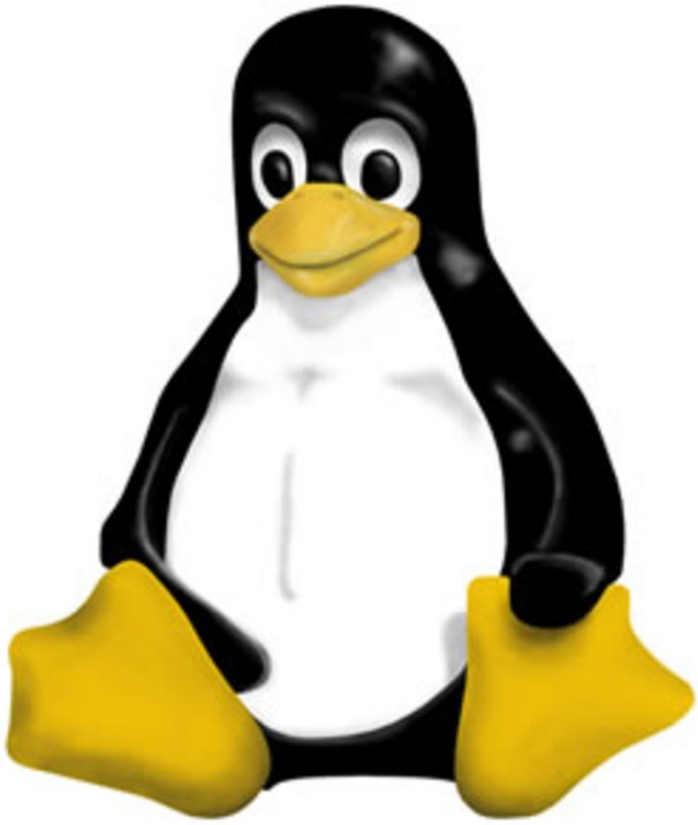
Операционные системы типа UNIX, в том числе и BSD, изначально разрабатывались для работы на больших многопользовательских компьютерах — мейнфреймах. Но персональные компьютеры постепенно наращивали мощь своего аппаратного обеспечения, и в наши дни они уже превосходят по возможностям те мейнфреймы, для которых в 70-х годах разрабатывалась ОС UNIX. И вот, в начале 90-х годов студент хельсинкского университета Линус Торвальдс (Linus Torvalds) приступил к разработке UNIX-подобной ОС для IBM-совместимых персональных компьютеров.

История Linux

История Linux начинается в 1991 году, когда финский программист Линус Торвальдс стал разрабатывать ядро операционной системы для своего компьютера. Свои наработки он выложил на сервере, и это стало ключевым событием в истории Linux. Сначала десятки, потом сотни и тысячи разработчиков поддержали его проект - общими усилиями на свет появилась полноценная операционная система.

Как уже было сказано, на Linux значительно повлияла система Unix, это заметно даже по названию. Впрочем, изначально проект назывался Freax - от слов “free” (бесплатный) и “freak” (странный), но в дальнейшем название было изменено на гибрид имени создателя (Линус) и Unix.

Эмблемой Linux стал Такс (Tux) - пингвин, нарисованный в 1996 году программистом и дизайнером Ларри Юингом. Впрочем, идею использовать именно пингвина придумал сам Линус Торвальдс. Теперь Такс является символом не только Linux, но и свободного программного обеспечения в целом.



Первая официальная версия Linux 1.0 вышла в 1994 году; вторая версия пошла в 1996 году. Товарный знак Linux был зарегистрирован на год раньше, в 1995.

С самого начала и по сей день Linux распространяется как свободное программное обеспечение с лицензией GPL. Это значит, что исходный код операционной системы может увидеть любой пользователь - и не только увидеть, но и доработать его. Единственное условие - измененный, модифицированный код должен быть так же доступен всем и распространяться по лицензии GPL. Это важно, так как дает возможность разработчикам использовать код и в то же время не бояться проблем из-за авторских прав.

Своему успеху Linux во многом обязан GNU: на момент выхода Linux существовало уже много свободного распространяемых утилит этого проекта, которые можно было использовать с разработанным ядром.

По факту Linux до сих представляет собой ядро Unix-подобной операционной системы, которое выполняет различные низкоуровневые задачи. В то же время проект GNU нуждался в ядре - разработка Линуса Торвальдса была очень своевременной.

Сейчас благодаря своей гибкости Linux используется на множестве разных устройств, начиная от компьютеров и заканчивая серверами и мобильными устройствами.

Популярные дистрибутивы Linux

Дистрибутив Linux – это определение операционной системы, которая использует ядро Linux, и которую можно установить на машину пользователя. В дистрибутивах обычно содержатся не только ядро и сама операционная система, но и полезные приложения: редакторы, проигрыватели, инструменты для работы с базами данных и другое программное обеспечение.

То есть, как уже было сказано в начале статьи, дистрибутив Linux – это операционная система, которая состоит из ядра Linux и утилит, которые разрабатываются в рамках GNU.

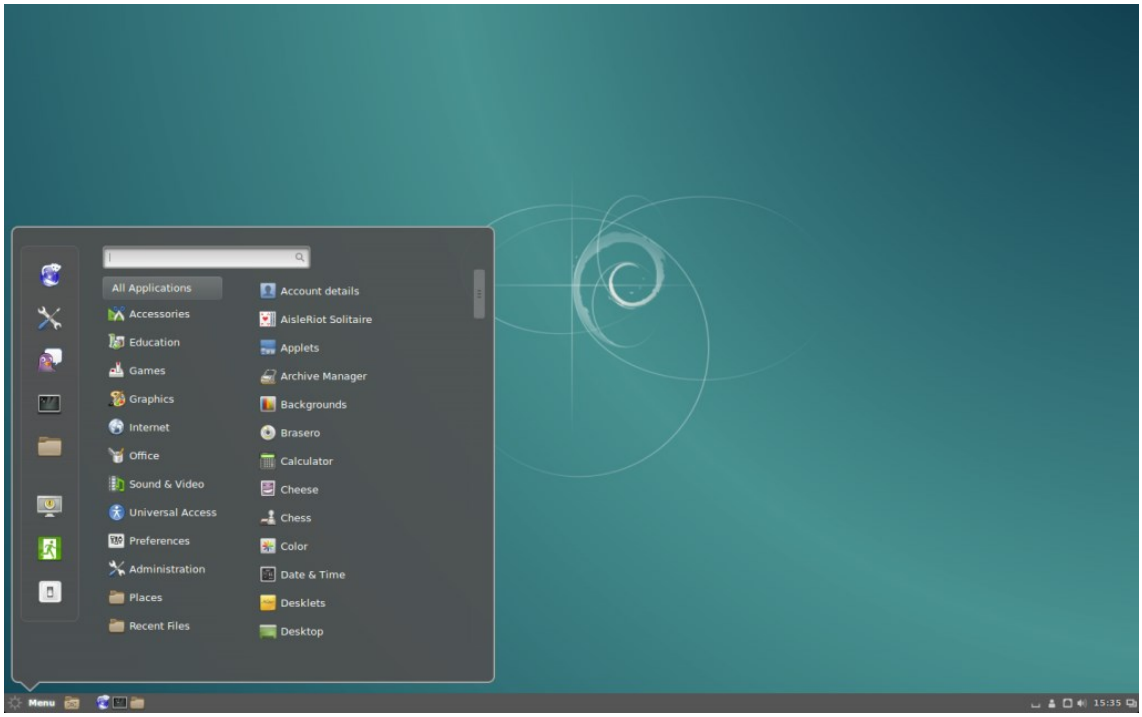
Количество существующих дистрибутивов Linux превышает 600 разновидностей, более 300 из которых постоянно дорабатываются и обновляются. Чтобы представить все разнообразие дистрибутивов пройдите по ссылке, будьте готовы к тому, что придётся значительно изменить масштаб страницы.

Ubuntu

Ubuntu - один из самых распространенных дистрибутивов, легко устанавливается и интуитивно понятен в работе. Отлично подходит для персональных компьютеров, ноутбуков и серверов. Разрабатывается и спонсируется компанией Canonical Ltd, но имеет активную поддержку и со стороны свободного сообщества. Самая популярная операционная система для веб-серверов.

Debian

Debian - еще один популярный дистрибутив GNU/Linux, который оказал существенное влияние на развитие всех GNU/Linux операционных систем в целом. Основные черты Debian: широкие возможности, за счёт наличия множества репозиториев и большого количества программ, высокое качество версий - это самый стабильный дистрибутив из всех существующих.

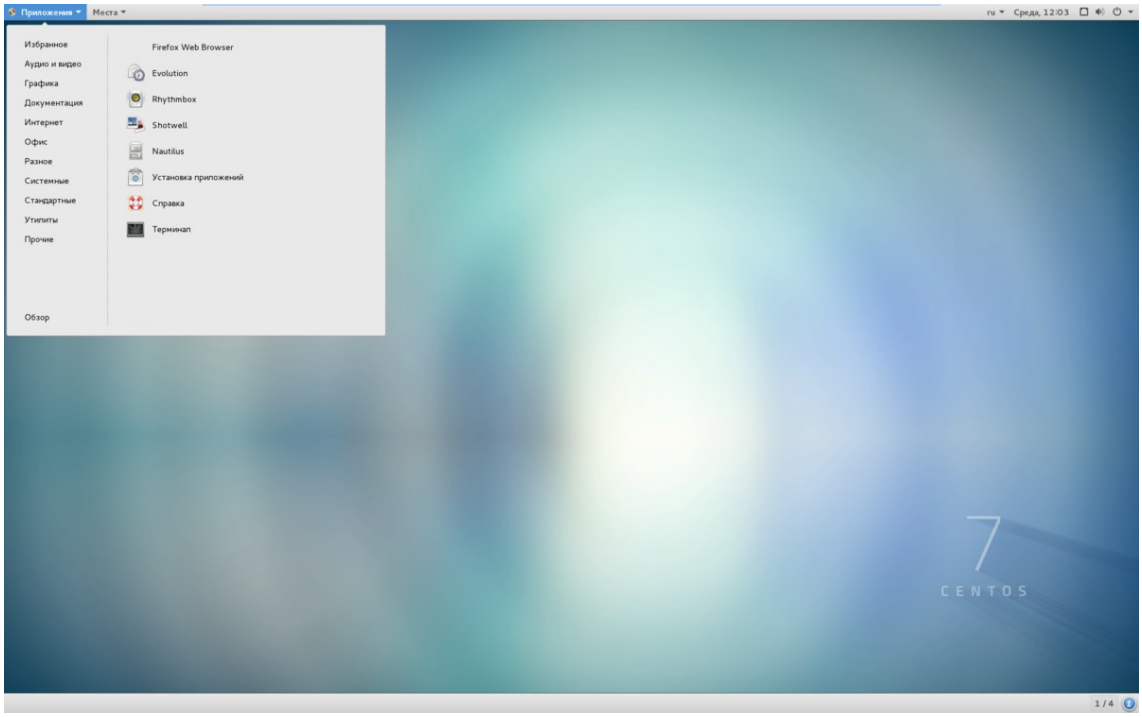


Linux Mint

Linux Mint - дистрибутив, основанный на Ubuntu и Debian. Linux Mint обладает красивым и удобным дизайном и подойдет даже начинающим пользователям. Поэтому его часто устанавливают на домашние компьютеры для того, чтобы иметь простую и удобную систему. Дистрибутив имеет поддержку различных мультимедийных форматов, в том числе включает проприетарные программы (Adobe Flash), поэтому хорошо подходит для работы с мультимедиа.

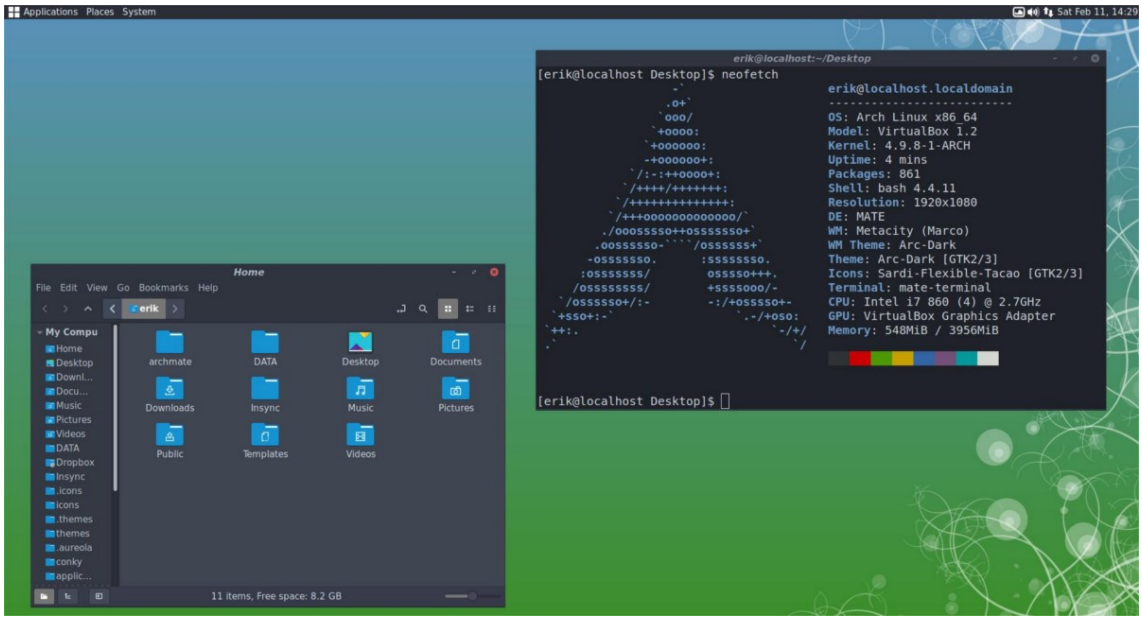
Centos

Centos - является бесплатной версией дистрибутива RedHat Enterprise Linux. Отличается характерной стабильностью и может отлично работать на компьютерах с 64-битной и 32-битной архитектурой. Дистрибутив комплектуется не всегда свежими версиями программ, в том числе ядро Linux тоже не всегда новое. Поэтому данная система не подходит для тех, кто любит ежедневные обновления.



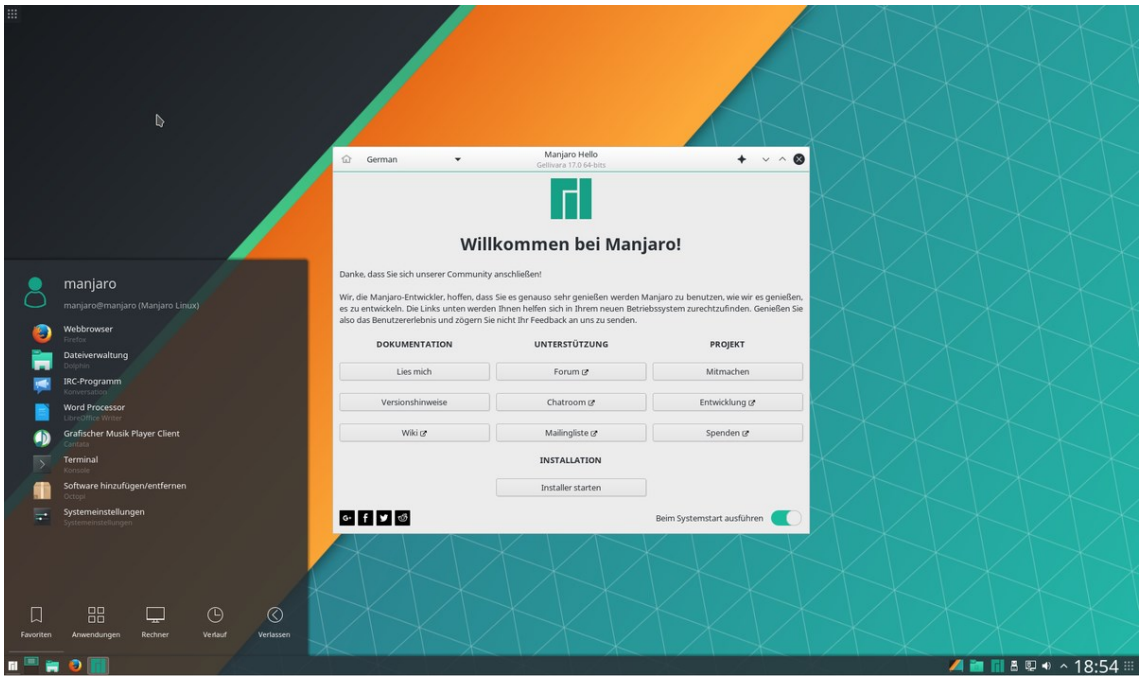
Arch Linux

Arch - мощный дистрибутив, базирующийся на принципах простоты, современности, прагматизма, гибкости и идеи, что в центре внимания должен быть пользователь. Однако принцип простоты распространяется не на использование системы, а на ее внутреннюю организацию (принципы KISS и Unix-way). Поэтому Arch рассчитан на опытных пользователей, которые самостоятельно настроят и установят необходимые им утилиты.



Manjaro

Manjaro - дистрибутив, основанный на Arch Linux. Благодаря большому количеству предустановленных программ (например, для офисной работы) он достаточно дружественен к новичкам, но в то же время имеет возможность тонкой настройки, множество пакетов, и стабилен в целом.



Linux в Сбербанке и СБТ.

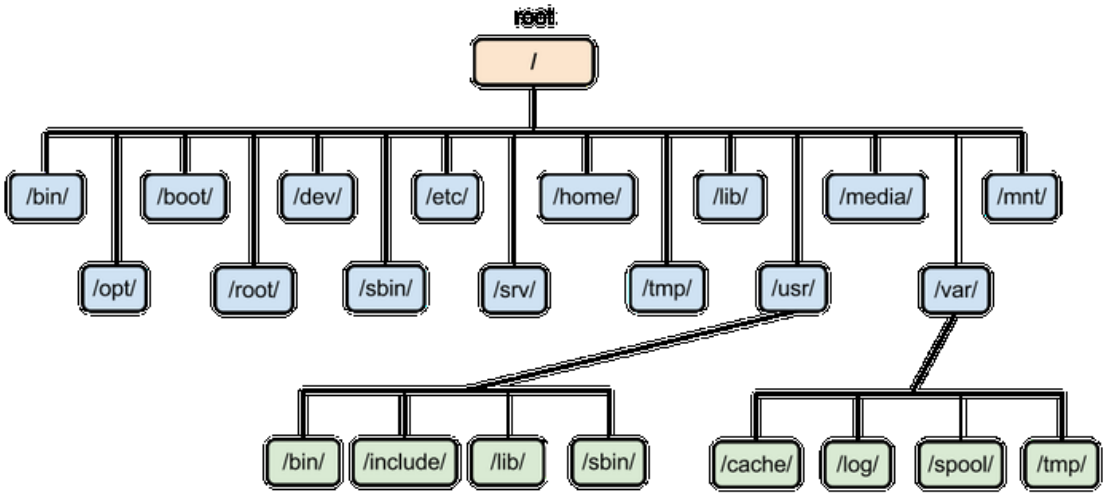
В нашей компании, в основном, используются три дистрибутива: RedHat Enterprise Linux, Centos и Ubuntu.
В прошлом году проходил проект по замене Windows на рабочих станциях на Linux, его результатом стала возможность официально установить себе Ubuntu Linux сотруднику Блока Т.

02. Работа с файловой системой

Дерево каталогов *nix

В целом структура каталогов *nix - систем представляет из себя дерево с единым корнем, обозначаемым "/" (корневой каталог). В корневом каталоге находятся базовые подкаталоги (на схеме - голубого цвета), в них могут быть файлы или другие подкаталоги (и так далее по вложенной структуре).

Структура каталогов, как правило, в большинстве ОС семейства Linux, выглядит так:



После загрузки *nix ОС (или подключения к серверу) и аутентификации пользователь попадает в командную оболочку UNIX (терминал). После запуска командная оболочка переходит в домашний каталог.

Допустим, мы зашли как пользователь **user.** После ввода команды **pwd** получим путь до папки, в которой находимся. В данном случае мы видим путь до домашней папки этого пользователя:

```
~ $ pwd
/home/user
```

Чтобы перейти в корневой каталог нужно выполнить следующую команду:

```
~ $ cd /
```

Ниже перечислены основные системные каталоги, которые находятся в корневом, и их краткое описание, какую функцию они несут:

/bin	Данный каталог содержит исполняемые файлы ("binaries" - "двоичные", "исполняемые") самых необходимых утилит. Отсюда и его название. В этом каталоге содержатся программы, которые могут понадобиться для устранения неполадок в системе или при восстановлении после сбоя.
/boot	"Boot" - в этой папке расположены файлы, необходимые для загрузки ядра - и, обычно, само ядро. Без необходимости и соответствующих знаний не рекомендуется работать с этими файлами.
/dev	Каталог предназначен для хранения файлов особого типа - они предназначены для обращения к различным системным ресурсам и

	устройствам ("devices" - "устройства"). К примеру, файлы /dev/sdXN соответствуют подключенным дискам, где X - буквы из латинского алфавита [a-z], обозначающие диск; а N - числа, обозначающие номера разделов диска
/etc	Этот каталог предназначен для хранения системных конфигурационных файлов. Эти файлы содержат информацию о специфических настройках данной системы.
/home	В этом каталоге расположены домашние каталоги пользователей, отсюда и название "home".
/lib	Этот каталог содержит коллекцию библиотек (сокращение от "libraries"). Под термином библиотека мы будем понимать набор функций, объединенных в одну смысловую единицу. Компоновка таких библиотек позволяет использовать функции многократно в своих программах без необходимости дублирования кода..
/mnt	Этот каталог предназначен для монтирования (от англ. "mount") - временного подключения файловых систем, например, на флеш-карте.
/proc	В этом каталоге все файлы "виртуальные" - они располагаются не на диске, а в оперативной памяти. В этих файлах содержится информация о программах (процессах), выполняемых в данный момент в системе.
/root	Это домашний каталог главного пользователя (администратора) системы - пользователя root. Этот каталог должен пристутствовать в любой ситуации, в отличии от каталога home, который можно разместить на съемном носителе
/sbin	Здесь располагаются системные утилиты (system binaries): в дополнение к исполняемым файлам каталога /bin здесь находятся программы, которые необходимы для загрузки, резервного копирования, восстановления системы.
/tmp	Каталог временных файлов: в этих файлах программы могут хранить необходимые для работы непостоянные данные. Обычно этот каталог очищается при каждой загрузке системы.
/usr	Каталог /usr - это "государство в государстве". Здесь можно найти такие же подкаталоги bin, etc, lib, sbin, как и в корневом каталоге. Однако в корневой каталог попадают только утилиты, **необходимые** для загрузки и восстановления системы в аварийной ситуации - **все остальные** программы и данные располагаются в подкаталогах /usr. Прикладных программ в современных системах обычно установлено очень много, поэтому этот раздел файловой системы может быть очень большим.
/var	В этом каталоге размещаются данные, которые создаются в процессе работы различными программами и предназначены для передачи другим программам и системам. Название этого каталога - сокращение от "variable" ("переменные" данные).
/var/log	В данном каталоге хранятся логи работы сервисов и, как правило, установленных приложений

Имена файлов и каталогов: допустимые имена, кодировки и русские символы, расширения

В Unix-системах имена файлов и каталогов имеют ограничения длины - 255 символов. Символы могут быть любые, за исключением "/", т.к. этот спецсимвол используется в качестве разделителя в составном имени (включает подкаталоги и имя файла). Также имеется набор спецсимволов, использовать которые не рекомендуется: « * \ & < > ; ( ) | ». Эти символы могут быть по-своему интерпретированы командными оболочками, в таком случае работа с ними может привести к неожиданным результатам.

Расширение файла — это последовательность символов, стоящих после последней точки в имени. Основное предназначение расширения - указать тип содержимого файла (например: .docx - для файлов, созданных в Microsoft WORD, .py - исполняемые Python-скрипты). Содержимое файла может не соответствовать расширению, так как расширение можно беспрепятственно изменить. Расширение можно оставить пустым. Для определения типа содержащихся в файле данных имеется утилита **file**:

```
[user]$ file -- new.txt
new.txt: ASCII text, with no line terminators
```

Создать файл можно при использовании любого текстового редактора, например, vim, nano.
Также его можно создать при помощи команды ***touch***:

```
[user]$ touch filename
```

Полный и относительный путь, домашняя и текущая директория

Расположение любого файла или каталога в системе точно и однозначно описывается с помощью полного пути. Полный путь — это составное имя файла или каталога. Он начинается от корневого каталога и состоит из списка всех подкаталогов, стоящих на пути к искомому включительно. Разделителем простых имен каталогов в полном пути служит специальный символ " **/** " ("слэш").

Файловая система в *nix системах организована в виде дерева. Появление циклов в такой структуре невозможно, поэтому полный путь к файлу или каталогу будет конечным.

Пример полного пути к файлу file.my, который находится в директории home:

```
/home/file.my
```

Относительный путь к файлу или каталогу строится по тем же принципам, но не от корневого каталога, а от директории, в которой мы находимся в текущий момент. Текущая директория обозначается спецсимволом "**.**"

Пример относительного пути к файлу file.my, который находится в директории home (текущая директория - корневая):

```
./home/file.my
```

Пример относительного пути к файлу file.my, который находится в директории home (текущая директория - home):

```
./file.my
```

Как мы уже знаем, домашняя директория — это каталог, в котором хранятся данные пользователя. В относительном пути домашняя директория обозначается специальным знаком тильда "**~**"

```
/tmp $ cd ~
~ $ pwd
/home/user
```

Права доступа

В операционной системе Linux есть много отличных функций безопасности, одна из самых важных — это система прав доступа к файлам. Идеология ядра UNIX изначально подразумевала проектирование многопользовательской системы (в отличие от Windows), поэтому права доступа к файлам продуманы очень хорошо.

Основные права доступа к файлам в Linux

Каждый файл имеет три параметра доступа:

- **Чтение** - разрешает получать содержимое файла (но не изменять его). Для каталога позволяет получить список файлов и каталогов, расположенных в нем;
- **Запись** - разрешает изменять содержимое файла (добавлять или редактировать данные). Для каталога позволяет создавать и удалять в нем файлы и каталоги;
- **Выполнение** - разрешает загрузить файл в память и попытаться запустить его как исполняемую программу. Для каталога означает право переходить в этот каталог.

Каждый файл имеет три категории пользователей, для которых можно устанавливать различные сочетания параметров доступа:

- **Владелец** - набор прав для владельца файла (пользователя, который его создал, или сейчас установлен быть его владельцем). Обычно владелец имеет все права: чтение, запись и выполнение.
- **Группа** - набор прав для группы владельца файла (или другой назначенной группы пользователей). Пользователи, занимающиеся общими задачами, могут объединяться в группы. Каждый пользователь обязательно принадлежит к одной или нескольким группам.
- **Остальные** - все пользователи, кроме владельца и группы файла.

Именно с помощью этих наборов полномочий устанавливаются права права доступа к файлам в Linux. Каждый пользователь может получить полный доступ к файлам, владельцем (автором) которых он является. Права доступа к остальным файлам могут быть ограничены. Только суперпользователь **Root** может работать со всеми файлами без ограничений.

Можно переназначить владельца файла, а также группу.

Со временем возможностей такой системы стало не хватать и было добавлено еще несколько специальных параметров.

Специальные права доступа к файлам в Linux

Для того, чтобы позволить обычным пользователям выполнять программы от имени суперпользователя без знания его пароля была придумана такие флаги (параметры, принимающие значение 0 или 1), как SUID и SGID биты. Рассмотрим их подробнее.

- **SUID** (set user ID) - если этот бит установлен, то при выполнении программы, id пользователя, от которого она запущена заменяется на id владельца файла. Фактически, это позволяет обычным пользователям запускать программы от имени суперпользователя;
- **SGID** (set group ID) - этот флаг работает аналогичным образом, только разница в том, что пользователь считается членом группы, с которой связан файл, а не групп, к которым он действительно принадлежит. Если SGID флаг установлен на каталог, все файлы, созданные в нем, будут связаны с группой каталога, а не пользователя. Такое поведение используется для организации общих папок;

- **Sticky-bit** - этот бит тоже используется для создания общих папок. Если он установлен, то пользователи могут только создавать, читать и выполнять файлы, но не могут удалять файлы, принадлежащие другим пользователям.

Как посмотреть права доступа к файлам в Linux

Права доступа к файлам в Linux можно посмотреть с помощью файлового менеджера, но так вы получите неполную информацию. Для максимально подробной информации обо всех параметрах, в том числе специальных флагах, нужно использовать команду **ls** с параметром **-l**. Все файлы из каталога будут выведены в виде списка, и там будут показаны все атрибуты и биты.

Чтобы узнать права на файл Linux, выполните такую команду в папке, где находится этот файл:

```
~/ $ ls -l

-rw-r--r-- 1 user user 492 окт 17 16:08 dir.txt
drwxrwxr-x 5 user user 132 окт 26 16:59 StarTrek
-rw-r--r-- 1 user user 0 окт 16 17:56 Eddard_Stark_biography.txt
-rw-r----- 1 user user 0 окт 16 17:59 log1.txt
-rwxrwxrw- 1 user user 0 окт 16 17:45 script.sh
-rw-r--r-- 1 user user 0 окт 16 17:56 stardust.mpeg
-rw-r--r-- 1 user user 0 окт 16 17:56 STARS.txt
-rw-r--r-- 1 user user 0 окт 16 17:55 star_trek_OST.mp3
-rw-r--r-- 1 user user 0 окт 16 17:55 Star_Wars.avi
```

Рассмотрим подробнее, что значат условные значения флагов прав:

- --- - нет прав, совсем;
- --x - разрешено только выполнение файла как программы, но не изменение (запись) и не чтение;
- -w- - разрешена только запись и изменение файла;
- -wx - разрешено изменение и выполнение. В случае с каталогом вы не можете посмотреть его содержимое;
- r-- - права только на чтение;
- r-x - только чтение и выполнение, без права на запись;
- rw- - права на чтение и запись, но без выполнения;
- rwx - все права;
- --s - установлен SUID или SGID бит, первый отображается в поле для владельца, второй для группы;
- --t - установлен sticky-bit, а значит пользователи не могут удалить этот файл.

Первая группа из 10 символов необходима для отображения прав на использование файла или директории. Первый символ - тип файла (есть множество обозначений, например, **d** - директория, **x** - исполняемый файл). Следующие 9 символов разбиты на группы по 3. Каждая группа имеет вид **rwx**, на месте буквы может стоять прочерк. Первый символ - **r** - означает наличие прав на чтение. Символ **w** - наличие прав на запись, символ **x**- на исполнение. Соответственно, прочерк обозначает отсутствие соответствующих прав. Первая тройка символов говорит о наличии соответствующих прав у создателя файла, вторая тройка - у группы пользователей, которая сопоставлена данному файлу, третья - у остальных пользователей.

Далее идет информация о создателе файла и наименовании группы, к которой он принадлежит.

Чтобы изменить права на файл в Linux вы можете использовать утилиту **chmod**. Она позволяет менять все флаги, включая специальные. Рассмотрим ее синтаксис:

```
~ $ chmod опции права /путь/к/файлу
```

Нас интересует опция **-R**. С помощью нее вы можете применять изменения ко всем файлам и каталогам рекурсивно (не нужно вручную устанавливать права для вложенных файлов/каталогов).
Категория указывает, для какой группы пользователей нужно применять изменения. Доступно три категории:

- **u** - владелец файла;
- **g** - группа файла;
- **o** - другие пользователи.
- **a** - для всех групп пользователей

Действие может быть одно из двух, либо добавить флаг "**+**", либо убрать флаг - "**-**". Что касается самих прав доступа, то они аналогичны выводу утилиты ls:

- **r** - чтение,
- **w** - запись,
- **x** - выполнение,
- **s** - suid/sgid, в зависимости от категории, для которой вы его устанавливаете,
- **t** - устанавливает sticky-bit.

Например, создадим файл new.txt и дадим всем пользователям полный доступ к файлу:

```
~ $ touch new.txt
~ $ chmod ugo+rwx new.txt
```

Или заберем все права у группы и остальных пользователей:

```
~ $ chmod go-rwx new.txt
```

Дадим группе право на чтение и выполнение:

```
~ $ chmod g+rx new.txt
```

Остальным пользователям только чтение:

```
~ $ chmod o+r new.txt
```

Как правило, в Linux, используют несколько другой способ работы с правами файла, и вместо буквенных аргументов используют цифры.

Значимый символ представляется в виде единицы в двоичном представлении. Таким образом переведя двоичное число в десятичное мы получаем числовое представление прав доступа.

Они соотносятся с буквенными определениями, которые уже были описаны выше, следующим образом:

- r (читать) заменяют на 4 (100 в двоичном виде - единица слева)
- w (запись) заменяют на 2 (010 в двоичном виде - единица посередине)

- x (исполнение) заменяют на 1 (001 в двоичном виде - единица справа)
- 0 означает – ничего не делать (то, что в буквенной записи обозначается дефисом)

Допустим у нас есть с файл со следующими правами доступа: **rwxr--r--** Если заменить в строчке буквы и дефисы на цифры как описано выше и сложить цифры в каждой тройке, то получим цифровой вид этой записи: `744`. Т.е. получается, что сумма этих цифр и показывает **chmod** по отношению к файлам или папке. Например:

7 (rwx) = 4 + 2 +1 (полные права, 111 в двоичном виде)
5 (r-x)= 4 + 0 + 1 (чтение и выполнение, 101 в двоичном виде)
6 (rw-) = 4 + 2 + 0 (чтение и запись, 110 в двоичном виде)
4 (r--) =4 + 0 + 0 (только чтение, 100 в двоичном виде)

В таблице ниже приведены все возможные комбинации привилегий, записанные в цифровом виде:

«Права»	«Цифровая форма»	«Двоичная форма»	«Символьная форма»
Ничего нельзя делать	0	000	—
Только чтение	4	100	r--
Только запись	2	010	-w-
Только выполнение	1	001	--x
Чтение и запись	6	110	rw-
Чтение и выполнение	5	101	r-x
Чтение, запись, выполнение	7	111	rwx

В следующей таблице даны примеры комбинаций записей в цифрах, применительно к группам пользователей, и их пояснение:

«Права»	«Владелец»	«Группа»	«Остальные»
777	читать записывать исполнять	читать записывать исполнять	Читать записывать исполнять
776	читать записывать исполнять	читать записывать исполнять	Читать записывать
775	читать записывать исполнять	читать записывать исполнять	Читать исполнять
774	читать записывать исполнять	читать записывать исполнять	Читать
766	читать записывать исполнять	читать записывать	Читать записывать
655	читать записывать	читать исполнять	Читать исполнять
644	читать записывать	читать	Читать

Например, чтобы назначить для всех полный доступ к файлу new.txt и разрешить его исполнять, нужно выполнить следующую команду:

```
~ $ chmod 777 new.txt
~ $ ls -l new.txt
-rwxrwxrwx 1 user user 0 ноя 2 12:53 new.txt
```

Чтобы исполнять файл мог только владелец, а все останые могли только читать его и редактировать, нужно использовать следующее сочетание:

```
~ $ chmod 766 new.txt
~ $ ls -l new.txt
-rwxrw-rw- 1 user user 0 ноя 2 12:53 new.txt
```

"Владение" объектами. Команда chown.

Как уже было сказано ранее, для файлов и каталогов есть три категории пользователей, на которые можно назначить права. Это "пользователь", "группа" и "остальные".

"Пользователь" — это и есть владелец файла. Точнее, не он сам, а его идентификатор пользователя (UID)

С группой чуть сложнее. Пользователь может входить в несколько групп одновременно, но в любой момент времени только одна группа для него будет основной, текущей. Это та группа, на которую указывает идентификатор группы (GID) для этого пользователя. Именно для этой группы и выставляются права по умолчанию, когда пользователь создает файл или каталог.

Остальные — это, соответственно, все остальные пользователи и группы.

Обычно владелец файла — это тот, кто его изначально создал, и группа - его текущая группа, но в некоторых случаях это нужно и можно поменять. Для этого и нужна команда **chown**.

Вот несколько примеров ее использования:

1. Вы создали файл **myfile**, используя **sudo** или находясь в системе под пользователем root, так что владельцем файла является **root**. Но файл должен принадлежать обычной учетной записи пользователя **myuser**.

   Для смены владельца используется **chown**:

```
~ $ sudo chown myuser myfile
```

- Вы - владелец файла **myfile**, но хотите передать его во владение другому пользователю в системе - **anotheruser**. Также вы хотите сменить группу владения на группу того пользователя, **anothergroup**.

  chown используется для смены как пользователя, так и группы:

```
~ $ sudo chown anotheruser:anothergroup myfile
```

- Вы перенесли целый каталог с файлам под названием **otherfiles**, с другого компьютера. Все файлы и подкаталоги принадлежат вашей учетной записи в другой системе, и вы хотите поменять пользователя и группу владения на ваши для всего содержимого каталога.

  Для этого используется рекурсивный режим, с опцией **-R**:

```
~ $ sudo chown -R myuser:mygroup otherfiles
```

Некоторые особенности команды **chown**:

- Пользователя и группу можно указывать как по имени, так и по номеру;
- Только суперпользователь может менять владельца файла. Самостоятельно владелец файла не может передать права другому пользователю - только если у него есть доступ к аккаунту root или с помощью sudo.
- Владелец файла может поменять группу владения, но только если пользователь принадлежит этой группе. Суперпользователь (root) может поменять группу владения на любую группу. Однако другие члены группы владения, отличные от владельца файла, не могут поменять его группу владения.
- Группу владения можно также поменять командой **chgrp**. Она функционально идентична команде **chown** и использует те же системные выводы.
- Некоторые операции с файлами могут выполняться только владельцем или суперпользователем (root). Например, только root или владелец могут менять параметры "atime" и "mtime" (время последнего доступа и время последнего изменения), например, командой **touch**.
- В жизни практически всегда команда **chown** выполняется суперпользователем, либо с утилитой **sudo**.

Перечень основных команд.

2. **pwd** - вывод полного или абсолютного пути текущей директории
3. **ls** - вывод в консоль содержания текущей директории
   ключи команды:
   -F - позволяет отличать файлы от каталогов. К каждому объекту будет добавлен суффикс, один из специализированных символов */=>@|-
   -a - позволяет включить в список вывода файлы, начинающиеся со спецсимволов
4. **cd** - перемещение по дереву каталогов
   .. - обозначение родительского каталога
   ~ - обозначение домашней директории
5. **mkdir** *имя_директории* - создание каталога
6. **mv** *источник цель* - перемещение файлов и каталогов
7. **cp** - копирование файлов и каталогов
8. **rm** *имя_файла* - удаление файла
9. **rmdir** *имя_директории* - удаление каталога
10. **chmod** - изменение прав доступа к файлу
11. **chown** - смена владельца файла

03. Работа с Bash

Основные сведения

Bash - это одна из самых популярных командных оболочек *nix систем. По своей сути bash является интерпретатором команд пользователя. Такие команды могут быть введены как с помощью текстового окна, так и заранее могут быть описаны в исполняемом файле. Такие файлы называются скриптом или сценарием. Подобно остальным *nix-оболочкам, bash поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, контроль за порядком выполнения, операторы ветвления и цикла.

Настройка

Для удобства работы с командной строкой есть возможность настроить строку приглашения ввода команды - добавить некоторые атрибуты и цвет. Для корректировки таких настроек служит переменная PS1. Вот пример вывода текущего значения этой переменной:

```
[user]$echo $PS1
[\u]\$
```

Этот формат говорит нам о том, что сначала в квадратных скобках отображается имя пользователя, затем знак приглашения к вводу текста. Вот еще некоторые данные, которые мы можем добавить:

\u - Имя пользователя.

\h - имя хоста.

\w - Текущий абсолютный путь. Используйте **\W** для текущей директории.

\$ - Символ приглашения.

\[и **\]** - Эти теги должны обрамлять цветовые коды, чтобы bash понимал как правильно позиционировать курсор.

Попробуем добавить абсолютный путь к отображаемой информации:

```
[user]$export PS1='[\u:\w]\$'
[user:~/Documents]$
```

Так же мы можем менять цвет отображаемой информации. Для этого необходимо заключить тег нужного нам атрибута в квадратные скобки с указанием шрифта (простой, жирный, курсив и тд) и цвета отображения. Например,

\e[1;31m \u **] -** эта запись говорит нам о том, что имя пользователя должно быть отображено жирным шрифтом (код 1 - это жирный шрифт) и красным цветом (31m). Таким образом, начало тега окрашивания имеет вид **\e[1;31m,** где первое число означает стиль написания, второе число означает цвет. Если мы не закроем квадратную скобку после тега \u, то вся остальная информация будет отображена с применением того же стиля.

Полную информацию о цветах и тегах можно найти здесь.

Горячие клавиши

Tab - Автодополнение строки за курсором.
Ctrl + a - Перемещает курсор в начало строки.
Ctrl + r - Поиск по набранным ранее командам.

Ctrl + d - Закрывает текущую оболочку
Ctrl + e - Перемещает курсор в конец строки.
Ctrl + x, **Ctrl + v** - Вывод на экран информации о версии текущего экземпляра bash.

Встроенные переменные

$BASH путь к исполняемому файлу bash
$BASH_VERSION версия Bash, установленного в системе
$EDITOR заданный по умолчанию редактор
$HOME домашний каталог пользователя
$OSTYPE тип операционной системы
$PATH путь поиска (включает в себя каталоги /usr/bin/, /usr/X11R6/bin/, /usr/local/bin и т. д.)
$PPID PID (идентификатор) родительского процесса
$PWD рабочий (текущий) каталог

Стартовые скрипты

Bash при запуске вызывает команды из множества различных скриптов.

Когда bash вызывается как интерактивная оболочка входа в систему, первым делом он читает и вызывает команды из файла /etc/profile, если этот файл существует. После чтения этого файла он смотрит следующие файлы в следующем порядке: ~/.bash_profile, ~/.bash_login и ~/.profile, читает и вызывает команды из первого, который существует и доступен для чтения. При выходе bash читает и выполняет команды из файла ~/.bash_logout.

Когда запускается интерактивная оболочка, но не для входа в систему, bash читает и исполняет команды из файлов /etc/bash.bashrc и ~/.bashrc, если они существуют. Это может быть отменено опцией `-norc`. Опция `-rcfile file` заставит bash использовать команды из файла file вместо /etc/bash.bashrc и ~/.bashrc.

Дополнительная информация

Если данной информации вам показалось мало и хочется узнать больше о возможностях оболочки, то рекомендуем почитать статьи ниже:

- Wiki
- Оболочка Bash — шпаргалка для начинающих
- Bash-скрипты: начало
- Advanced Bash-Scripting Guide
- Особенности работы оболочки bash.

04. Перенаправление ввода/вывода, каналы и фильтры

Стандартный ввод/вывод

Потоки ввода-вывода

Когда программа запускается на выполнение, в ее распоряжение предоставляются три потока (или канала):

- **стандартный ввод** (standard input или **stdin**). По этому каналу данные передаются программе;
- **стандартный вывод** (standard output или **stdout**). По этому каналу программа выводит результаты своей работы;
- **стандартный поток сообщений об ошибках** (standard error или **stderr**). По этому каналу программы выдают информацию об ошибках.

Из стандартного ввода программа может только читать, а два других потока могут использоваться программой только для записи.

По умолчанию стандартный ввод связан с клавиатурой, а стандартный вывод и поток сообщений об ошибках направлены на терминал пользователя. Т.е., вся выходная информация запущенной пользователем команды или программы, а также все сообщения об ошибках выводятся в окно терминала. Однако, как мы увидим чуть позже, выходные сообщения можно перенаправить (например, в файл).

Для демонстрации работы стандартного потока ошибок, выполним команду **ls** с неверным аргументом - передадим в качестве аргумента имя несуществующего файла.

В таком случае **ls** выведет сообщение об ошибке в стандартный поток ошибок.

```
[user]$ ls notfilename
ls: невозможно получить доступ к 'notfilename': Нет такого файла или
каталога
```

В данном случае стандартный поток ошибок неотличим от выходного потока, поскольку сообщение об ошибке мы видим в окне терминала.

Работу со стандартными входным и выходным потоками лучше всего проиллюстрировать на примере команд **echo** и **cat**.

Команда echo

Команда **echo** предназначена для передачи на стандартный вывод строки символов, которая задана ей в качестве аргумента. После этого она выдает сигнал перевода строки и завершается. Попробуйте выполнить команду

```
[user]$ echo "Hello world!"
Hello world!
```

Команда cat

Cat выводит содержимое перечисленных файлов. По умолчанию выход команды **cat** направляется в выходной поток. Чтобы убедиться, что эта команда воспринимает входной поток (как будто это файл), запустите команду **cat** без аргументов.

```
[user]$ cat
|
```

В результате курсор переместится в новую строку. В это время команда ожидает поступления символов во входном потоке.
Введите любой символ и вы увидите, что после перевода строки он сразу же появился на экране. Это говорит о том, что программа сразу же направила его в выходной поток. Если продолжить ввод символов, они также появятся на экране.

Как правило клавиатура настроена на построчный ввод - если нажать клавишу <Enter>, то последняя набранная строка передается команде **cat**, которая вновь выводит данные на монитор через стандартный вывод.

Таким образом, каждая строка будет показана дважды: один раз при наборе и второй раз — командой **cat**.

```
[user]$ cat
abcd
abcd
|
```

Если нажать комбинацию клавиш <Ctrl>+<D>, которая служит командой окончания процедуры ввода, то вновь вернемсь к командной строке.
Можно использовать комбинацию клавиш <Ctrl>+<C>, которая является в оболочке командой завершения работы запущенной программы.

Если команде **cat** в качестве аргумента задать имя файла, содержимое файла будет направлено во входной поток команды **cat** и будет выдано в ее выходной поток. Это частные случаи **перенаправления ввода**, очень полезного механизма оболочки. Рассмотрим его подробнее.

Перенаправление ввода/вывода, каналы и фильтры

Ввод/вывод программы связаны со стандартными потоками, в оболочке существуют специальные средства для перенаправления ввода/вывода.

Операторы >, < и >>

Для обозначения перенаправления используются символы "**>**", "**<**" и "**>>**". Чаще всего используется перенаправление вывода команды в файл. Вот соответствующий пример:

```
[user]$ ls -l > /home/user/dir.txt
```

По этой команде в файле /home/user/dir.txt будет сохранен перечень файлов и подкаталогов того каталога, который был текущим на момент выполнения команды **ls**;

при этом, если указанного файла не существовало, то он будет создан;
если он существовал, то будет перезаписан;
если хотите, чтобы вывод команды был дописан в конец существующего файла, то надо вместо символа **>** использовать **>>**.

Наличие пробелов до или после символов **>** или **>>** несущественно и служит только для удобства пользователя.

Посмотреть содержимое файла dir.txt при помощи команды **cat** можно следующим образом:

```
[user]$ cat dir.txt
итого 4
-rw-r--r-- 1 user user    596 окт 17 15:59 dir.txt
```

```
-rw-r--r-- 1 user user   2,6K окт 16 17:56 Eddard_Stark_biography.txt
-rw-r--r-- 1 user user     1K окт 16 17:59 log1.txt
-rw-r--r-- 1 user user    156 окт 16 17:58 log2.txt
-rw-r--r-- 1 user user   200M окт 16 17:56 stardust.mpeg
-rw-r--r-- 1 user user   1,2K окт 16 17:56 STARS.txt
-rw-r--r-- 1 user user   8,9M окт 16 17:55 star_trek_OST.mp3
-rw-r--r-- 1 user user   1,3G окт 16 17:55 Star_Wars.avi
```

Оператор **>** служит для перенаправления выходного потока. Входной поток перенаправляет оператор **<**.

Пример команды подсчёта слов в файле, в данном примере файл передаётся команде при помощи оператора **<**:

```
[user]$ wc -w < /home/user/report.txt
12
```

Этот вариант перенаправления часто применяют к тем командам, которые воспринимают ввод (или ожидают ввода) с клавиатуры. Вместо ввода с клавиатуры в команду подается информация из заранее заполненного файла. Так можно автоматизировать рутинные операции, например заполнение конфигурационных параметров.

Сам по себе символ перенаправления не может использоваться, следующая команда выдаст ошибку:

```
[user]$ file1 > file2
file1: команда не найдена
```

Перенаправить можно не только стандартный ввод и вывод, но и стандартный поток сообщений об ошибках. Для этого надо указать перед символом перенаправления номер перенаправляемого потока.

Стандартный ввод **stdin** имеет номер **0**, стандартный вывод **stdout** — номер **1**, стандартный поток сообщений об ошибках **stderr** — номер **2**. Полный формат команды перенаправления имеет вид:

```
[user]$ command N > M
```

где N и M — номера стандартных потоков (0,1,2) или имена файлов. Употребление в некоторых случаях символов **<**, **>** и **>>** без указания номера потока или имени файла возможно только потому, что вместо отсутствующего номера по умолчанию подставляется **1**, т. е. стандартный вывод. Так, оператор **>** без указания номера интерпретируется как **1 >**.

Существует возможность не просто перенаправить поток, но и сделать копию его содержимого. Для этого служит специальный символ **&**, который ставится перед номером потока, на который идет перенаправление:

```
[user]$ command N > &M
```

Такая команда означает, что выход потока с номером N направляется как на стандартный вывод, так и дублируется в поток с номером M. Например, для того, чтобы сообщения об ошибках дублировались на стандартный вывод, надо дать команду **2>&1**, в то время как **1>&2** дублирует **stdout** в **stderr**.

```
[user]$ command 2>&1
```

Эта возможность особенно полезна при перенаправлении вывода в файл: мы будем видеть сообщения на экране и одновременно сохранять их в файле.

Пример: команда ниже будет выводить в файл результат поиска файла *.profile* в корневом каталоге **/**, а также ошибки, которые возникали в процессе поиска. В данном случае, ошибки перенаправляются из **stderr** в **stdout** (**2>&1**) :

```
~ $ find / -name .profile > results.txt 2>&1
~ $ cat results.txt
find: '/boot/efi': Отказано в доступе
find: '/home/oem/.config/mc/mcedit': Отказано в доступе
/home/oem/.profile
.....................................................
```

Вариант ниже будет выводить поток сообщений из стандартного вывода (**stdout**) в стандартный поток ошибок (**stderr**):

```
[user]$ command 1 >&2
```

Если мы воспользуемся таким перенаправлением в примере выше, то получим вывод всего в консоль, сам же файл results.txt будет пустым :

```
~ $ find / -name .profile > results.txt 1>&2
find: '/boot/efi': Отказано в доступе
find: '/home/oem/.config/mc/mcedit': Отказано в доступе
/home/oem/.profile
.....................................................
~ $ cat results.txt
```

Если же мы хотим разделить вывод результата работы **find** в файл result.txt, а ошибки - в отдельный файл, то это будет выглядеть следующим образом:

```
~ $ find / -name .profile 2> error.txt 1> results.txt
```

Оператор |

Особым вариантом перенаправления вывода является организация программного канала, который называют конвейером. Для этого две или несколько команд, таких, что вывод предыдущей служит вводом для следующей, соединяются символом вертикальной черты — "**|**". При этом стандартный выходной поток команды, расположенной слева от символа **|**, направляется на стандартный ввод программы, расположенной справа от символа **|**. Например:

```
[user]$ cat myfile | grep Linux | wc -l
4
```

Эта строка означает, что вывод команды **cat**, т. е. текст из файла myfile, будет направлен на вход команды **grep**, которая выделит только строки, содержащие слово "Linux". Вывод команды **grep** будет направлен на вход команды **wc -l**, которая подсчитает число таких строк и выведет его как результат.

Программные каналы (конвейеры) используются для того, чтобы скомбинировать несколько маленьких программ, каждая из которых выполняет только определенные преобразования над своим входным потоком. Так создается общая команда, результатом которой будет более сложное преобразование.

Оболочка одновременно вызывает на выполнение все команды, включенные в конвейер, запуская для каждой из команд отдельный экземпляр оболочки. Как только первая программа начинает выдавать что-либо в свой выходной поток, следующая команда сразу начинает его обрабатывать. Точно так же каждая следующая команда выполняет свою операцию, ожидая данные от предыдущей команды и выдавая свои результаты на вход последующей. Если нужно, чтобы какая-то команда полностью завершилась до начала

выполнения последующей, после команд конвейера (соединенных **|**), используйте "**;**" (точка с запятой). Перед каждой точкой с запятой оболочка будет останавливаться и ожидать, пока завершится выполнение всех предыдущих команд, включенных в конвейер. Например, команда ниже позволит просмотреть два лога один за другим, при этом будет возможность пролистывать каждый лог отдельно. Как только будет завершен просмотр перового файла, начнётся просмотр следующего:

```
~ $ cat /var/log/dpkg.log | less; cat /var/log/boot.log | more
```

Статус выхода (логическое значение, возвращаемое после завершения работы программы) из канала совпадает со статусом выхода, который возвращает последняя команда конвейера.
Перед первой командой конвейера можно поставить символ "**!**", тогда статус выхода из конвейера будет логическим отрицанием статуса выхода из последней команды. Оболочка ожидает завершения всех команд конвейера прежде чем установить возвращаемое значение.

Фильтры

Приведенный выше пример с командой **grep** можно использовать для иллюстрации еще одного важного понятия - программы-фильтра. Фильтры — это команды (или программы), которые воспринимают входной поток данных, производят над ним некоторые преобразования и выдают результат на стандартный вывод (откуда его можно перенаправить куда-то еще по желанию пользователя). К числу команд-фильтров относятся команды **cat**, **more**, **less**, **wc**, **cmp**, **diff,** а также следующие команды.

Команды-фильтры

Команда	**Краткое описание**
grep, fgrep, egrep	Ищут во входном файле или данных со стандартного ввода строки, содержащие указанный шаблон, и выдают их на стандартный вывод
tr	Заменяет во входном потоке все встречающиеся символы, перечисленные в заданном перечне, на соответствующие символы из второго заданного перечня
comm	Сравнивает два файла по строкам и выдает на стандартный вывод 3 колонки: в одной — строки, которые встречаются только в 1 файле, во второй — строки, которые встречаются только во 2-ом файле: и в третьей — строки, имеющиеся в обоих файлах
pr	Форматирует для печати текстовый файл или содержимое стандартного ввода
sed	Строковый редактор, использующийся для выполнения некоторых преобразований над входным потоком данных (берется из файла или со стандартного ввода)

Особым фильтром является команда **tee**, которая "раздваивает" входной поток, с одной стороны направляя его на стандартный вывод, а с другой — в файл (имя которого вы должны задать). Легко видеть, что по своему действию команда **tee** аналогична оператору перенаправления **1>&file**.

Возможности фильтров можно существенно расширить за счет использования регулярных выражений, позволяющих организовать поиск по различным, зачастую очень сложным, шаблонам.

Команда cat

Команда **cat** часто используется для создания файлов (можно воспользоваться и командой **touch**). По команде **cat** на стандартный вывод передается содержимое указанного файла (или нескольких файлов, если их имена последовательно заданы в качестве аргументов

команды). Если вывод команды **cat** перенаправить в файл, то можно получить копию исходного файла:

```
[user]$ cat file1 > file2
```

Изначально предназначение команды **cat** как раз и предполагало перенаправление вывода, так как эта команда создана для конкатенации (con**cat**enation), т. е. объединения нескольких файлов в один:

```
[user]$ cat file1 file2 ... fileN > new-file
```

Возможности перенаправления ввода и вывода этой команды используются для создания новых файлов. Для этого на вход команды **cat** направляют данные со стандартного ввода (т. е. с клавиатуры), а вывод команды — в новый файл:

```
[user]$ cat > newfile
```

После того, как вы напечатаете все, что хотите, нажмите комбинацию клавиш <Ctrl>+<D> или <Ctrl>+<C>, и все, что вы ввели, будет записано в newfile. Конечно, таким образом создаются в основном короткие текстовые файлы.

Команды *more* и *less*

Команда **cat** позволяет вывести на стандартный вывод (на экран) содержимое любого файла, однако она используется для этих целей очень редко, разве что для вывода небольших по объему файлов. Дело в том, что содержимое большого файла мгновенно "проскакивает" на экране, и пользователь видит только последние строки. Поэтому **cat** используется в основном по прямому назначению — для конкатенации файлов, а для просмотра содержимого файлов (конечно, текстовых) используются команды **more** и **less** (или текстовые редакторы).

```
[user]$ more filename
[user]$ less filename
```

Команда-фильтр **more** выводит содержимое файла на экран отдельными страницами размером, как раз, в целый экран. Для того, чтобы увидеть следующую страницу, надо нажать на клавишу пробела. Нажатие на клавишу <Enter> приводит к смещению на одну строку. Кроме клавиш пробела и <Enter> в режиме паузы еще некоторые клавиши действуют как управляющие (например, клавиша <B> возвращает вас на один экран назад). Необходимо запомнить, что выйти из режима просмотра можно с помощью клавиши <Q>, иначе придется долго нажимать пробел, пока вы не доберетесь до конца файла. Полный перечень возможностей и опций команды можно посмотреть в интерактивном руководстве **man** или **info**.

Утилита **less**, разработанная в рамках проекта GNU, содержит все функции и команды управления выводом программы **more** и некоторые дополнительные: например, позволяет использовать клавиши управления курсором (<Стрелка вверх>, <Стрелка вниз>, <PgUp>, <PgDown>) для перемещения по тексту.

Команды **more** и **less** позволяют производить поиск подстроки в просматриваемом файле, причем команда **less** позволяет производить поиск как в прямом, так и в обратном направлении. Для организации поиска строки символов, например "stringtofind", надо набрать в командной строке программы в нижней части экрана (там, где двоеточие) /stringtofind. Если искомая строка будет найдена, будет отображена соответствующая часть текста, а найденная строка будет находиться в самом верху экрана.

06. Работа с редактором vi

В ОС семейства Linux есть несколько консольных текстовых редакторов, самые популярные из них - **vi** и **nano**.

Сейчас мы рассмотрим текстовый редактор **vi**.

Перед использование vi для редактирования файлов вам следует узнать, как в **vi** перемещаться по файлу.
У **vi** множество команд перемещения, многие из них мы сейчас рассмотрим.
Открытие файла выполняется следующей командой:

```
[user]$ vi filename.txt
```

После загрузки **vi**, на экране вы должны увидеть часть загруженного вами текстового файла.

В отличии от большинства редакторов **vi** после загрузки находится в специальном «**командным режиме**». Это значит, что если вы нажмете клавишу <l>(строчная L), вместо появления буквы «l» на месте курсора вы увидите, что курсор сдвинулся на один символ вправо. В командном режиме знаки, набираемые на клавиатуре, используются как команды для **vi**, а не как помещаемые в текст символы. Один из наиболее важных типов команд — это команды перемещения.

Перемещаемся по документу

Находясь в командном режиме, вы можете использовать клавиши <h>,<j>,<k> и <l> для перемещения курсора влево, вниз, вверх и вправо соответственно. Если вы используете современную версию **vi**, вы можете также использовать клавиши со стрелками. Клавиши <h>,<j>,<k> и <l> предпочтительнее, так как, освоив их, вы сможете перемещаться по файлу, не размахивая руками над клавиатурой. Для перемещения по текстовому файлу используйте следующие клавиши:

- **j** – перемещает курсор вниз;
- **k** – перемещает курсор вверх;
- **h** – перемещает курсор влево;
- **l** – перемещает курсор вправо.

vi не даст вам перескочить на предыдущую строку, нажимая <h>, если вы находитесь в начале строки.
Точно также он не позволит вам перескочить на следующую строку, нажимая <l> в конце строки.

vi предоставляет специальные команды для прыжков в начало и конец текущей строки:

- **0 (ноль)** – перескочить на первый символ в строке;
- **$** – перескочить на последний символ в строке.

Поскольку у **vi** так много команд перемещения, его можно использовать в качестве инструмента просмотра — «pager»а (как команды **more** или **less**).
Используя **vi** для просмотра, вы очень быстро запомните все команды перемещения.
Вы также можете использовать <Ctrl>+<F> и <Ctrl>+<B> для перемещения вперед и назад сразу на страницу.
Современные версии **vi** (такие как **vim**) могут позволить использовать для этих целей клавиши <PageUp> и <PageDown>.

Перемещение по словам

vi также позволяет вам перемещаться влево и вправо по словам:

- **w** – на первый символ следующего слова;
- **e** – на последний символ слова (если находимся на последнем символе, то на последний символ следующего слова);
- **b** – на первый символ текущего слова (если находимся на первом символе, то на первый символ предыдущего слова);.

vi считает конструкции типа «foo-bar-oni» пятью различными словами! Это происходит потому, что **vi** по умолчанию разделяет слова пробелами или знаками пунктуации (которые тоже считает словами).
Поэтому foo-bar-oni считается пятью словами: «foo»,«-»,«bar»,«-» и «oni».
В **vi** есть понятие «большого слова». «Большие слова» разделены только пробелами и началами строк.
Это значит, что foo-bar-oni состоит из пяти **vi**-слов, но является только одним "большим словом" **vi**.

Чтобы переместиться на следующее или предыдущее большое слово, вы можете использовать «заглавные» команды перемещения по словам. Результат будет аналогичным. Нажав следующие клавиши, вы попадете на:

- **W** — первый символ следующего большого слова;
- **E** — следующий последний символ большого слова;
- **B** — предшествующий первый символ большого слова.

Выход из редактора

Мы рассмотрели основные команды перемещения, однако осталась еще пара команд, которые вам необходимо знать. Напечатав:

- **:q** вы должны выйти из **vi**. Если не получится, значит вы каким-то образом ухитрились изменить файл;
- **:q!** вы выйдете из редактора, отбросив изменения в файле.

Теперь вы должны оказаться в командной строке системы.

Любая команда в vi, начинающаяся с двоеточия («:»), называется командой ex-режима (an ex-mode command). vi имеет встроенный не экранный редактор **ex**. Он может использоваться для выполнения операций редактирования строк. Дополнительно, как мы только что видели, он может использоваться для завершения работы. Если вы случайно нажмете <Q>, находясь в командном режиме, то можете оказаться в **ex**-режиме. В этом случае вы столкнетесь с приглашением «**:**», и нажатие на Enter будет сдвигать вверх содержимое экрана. Для возврата к привычному **vi**-режиму просто наберите **vi** и нажмите <Enter>.

Сохранение и редактирование

Save(сохранить) и Save as...(сохранить как...)

Мы уже видели, как используя **ex**-команду **:q** выйти из **vi**.

Для того, чтобы изменения после выхода сохранились, используйте команды:

- **:w** – записать или сохранить файл;
- **:w filename** – для сохранение под другим именем;

- **:x** или **:wq** – сохранить и выйти.

В **vim** (и других потомках **vi**, типа **elvis**) вы можете держать открытыми одновременно несколько "вкладок". Введите команду

- **:sp filename.txt** для открытия файла filename.txt в новом окне;
- **:sp** (без имени файла) откроет дополнительное окно для активного буфера.

Для перехода между окнами нажмите дважды <Ctrl>+<w>). Любая из команд **:q, :q!, :w** и **:x** относится только к активному окну.

Простое редактирование

Изучим простые команды редактирования. "Простыми" они считаются из-за того, что оставляют вас в командном режиме. Более сложные команды автоматически переводят вас в режим ввода текста, позволяющий добавлять в буфер символы, набираемые на клавиатуре; их рассмотрим позднее.

- **x** – удаление символа под курсором;
- **J** – присоединение следующей строки к текущей;
- **r+символ** – замена символа под курсором на указанный символ;
- **dd** – удаление текущей строки.

Повторение и удаление

Повторить любую команду редактирования вы можете, нажав клавишу <.> (точка). Если попробуете, то можете увидеть, что печать "**dd…**" удалит 4 строки, а "**J……**" объединит 7 строк. **vi** обеспечивает вас различными удобными средствами сокращения трудозатрат.

Удалять текст вы можете также комбинируя команду **d** с любыми командами перемещения. Например, **dw** удалит часть текста от текущей позиции курсора до начала следующего слова; **d)** удалит вплоть до следующего конца предложения, и **d}** удалит весь остаток абзаца. Поэкспериментируйте с командой **d** и другими командами редактирования, пока не почувствуете себя с ними уверенно.

Режим ввода текста

Мы уже рассмотрели, как в **vi** перемещаться, выполнять чтение\запись файлов и основные операции редактирования.

В **vi**, в режиме ввода текста, вы можете вводить текст «прямо на экран», как в большинстве других экранных редакторов. Чтобы это сделать:

- **i** или **a** – вернуться/войти в режим ввода.

- **Esc** – вернуться в командный режим;

Если вы хотите более подробно узнать и разобраться по работе с расширенной версией редактора, то можете почитать статью здесь.

07. Команды архивирования файлов

При работе с Linux вы, может быть, еще не скоро встретитесь с необходимостью работать с большинством консольных команд, поскольку имеются такие оболочки, как **Midnight Commander** или графические оболочки типа **KDE**. Но с командами архивирования (точнее, разархивирования) вам работать придется обязательно, хотя бы потому, что вы будете часто встречать архивированные файлы в Интернете.

Основным средством архивирования в UNIX (а, следовательно, и в Linux) является комплекс из двух программ — **tar** и **gzip**. Хотя никто не запрещает пользоваться **arj, pkzip**, **lha**, **rar** и т. д. — версии этих программ для Linux общедоступны. Просто уж исторически сложилось, что пользователи Unix чаще применяют именно tar и gzip, и именно в таком формате распространяется большая часть программного обеспечения для Unix. Поэтому овладеть работой с **tar** и **gzip** — дело чести любого пользователя Linux.

Архиватор tar

У читателя, привыкшего к архиваторам типа arj, которые собирают файлы в единый архив и сразу "сжимают" их, может возникнуть вопрос "А зачем использовать две программы?" Все дело в том, что tar расшифровывается как Tape ARchiver, он не сжимает данные, а лишь объединяет их в единый файл с последовательным доступом для последующей записи на ленту. По умолчанию этот архивный файл создается на ленточном накопителе, точнее на устройстве /dev/rmt0. Если вы хотите создать архивный файл на диске, то необходимо использовать команду tar с опцией f, после которой указывается имя архивного файла.

У программы tar имеется 8 опций, отличающихся от остальных тем, что при вызове программы должна обязательно задаваться одна из этих опций. Эти опции определяют основные функции программы.

Таблица 4.5. *Основные опции программы tar*

Опция	Значение
-A, --catenate, --concatenate	Добавляет файлы в существующий архив
-c, --create	Создает новый архив
-d, --diff, --compare	Найти различия между архивом и файловой системой
--delete	Удалить из архива (не может использоваться с магнитной лентой!)
-r, --append	Дописывает файлы в конец архива
-t, --list	Выводит список файлов архива
-u, --update	Добавляет только файлы, которые новее, чем имеющаяся в архиве копия
-x, --extract, --get	Извлечь файлы из архива

Если вы работаете с файлами архивов на дисках, а не с ленточным устройством, то, очевидно, обязательной будет и опция f. Другие опции не являются обязательными, они служат только для конкретизации задания программе. Например, опция v заставляет программу выводить список обрабатываемых файлов.

Однобуквенные опции программы tar могут перечисляться друг за другом (вы увидите это в приводимых ниже примерах).

Я не буду давать здесь описание всех опций команды tar, просто приведу несколько командных строк для выполнения самых необходимых действий с архивами.

Чтобы создать один tar-архив из нескольких файлов, используется команда:

```
[user]$ tar -cf имя_архива файл1 файл2
```

где опция -c сообщает программе, что необходимо создать (create) архив, а опция f говорит о том, что архив должен создаваться в виде файла (имя которого должно следовать сразу за этой опцией).

В именах файлов, которые сохраняются в архиве, можно использовать шаблоны имен файлов, в том числе просто символы-заместители * и ?. Благодаря этому можно очень короткой командой отправить в архив сразу много файлов. Например, для того, чтобы создать архив, содержащий все файлы одного из подкаталогов (пусть это будет sub_dir) текущего каталога, достаточно дать команду

```
[user]$ tar -cvf имя_архива ./sub_dir/*
```

или даже просто

```
[user]$ tar -cvf имя_архива sub_dir
```

По этой команде в архиве будут сохранены не только файлы, расположенные непосредственно в подкаталоге sub_dir, но и рекурсивно все файлы из подкаталогов каталога sub_dir. При этом в архиве сохраняется вся структура подкаталогов каталога sub_dir.

Заметим, что если в только что приведенном примере вместо * поставить *.*, то будут сохранены только те файлы, которые расположены непосредственно в подкаталоге sub_dir, а подкаталоги каталога sub_dir архивированы не будут . Если в том же примере не указать имя подкаталога, то будут архивироваться все файлы (и подкаталоги) текущего каталога. Но если вы дадите команду следующего вида

```
[user]$ tar -cvf имя_архива ./.*
```

то в архиве будут сохранены не только все файлы (и подкаталоги) текущего каталога, но и файлы из родительского каталога, а хотели ли вы этого?

Теперь вы знаете как создать архив, а для того, чтобы распаковать (извлечь) файлы из архива, нужно дать команду:

```
[user]$ tar -xvf имя_архива файлы
```

Получить список файлов архива можно командой:

```
[user]$ tar -tf имя_архива | less
```

В заключение раздела заметим, что всегда можно получить подсказку по использованию программы tar, дав команду

```
[user]$ tar --help
```

Архиватор gzip

Хотя программа tar создает архивы, она, как было сказано, не сжимает архивы, а просто соединяет отдельные файлы в единый архивный файл. Для сжатия этого файла часто применяют команду gzip. В простейшем случае она вызывается в следующем формате:

```
[user]$ gzip файл
```

В командной строке можно указать сразу несколько имен файлов или шаблон имени файла. Но в этом случае каждый из указанных файлов будет заархивирован отдельно (общий архив не создается).

Для того, чтобы распаковать архив, используйте команду

```
[user]$ gzip -d файл_архива
```

или

```
[user]$ gunzip файл_архива
```

Исходные файлы после сжатия удаляются, остается только архивный файл (файлы перемещаются в архив), а при разархивации удаляется архив.

Перечислим кратко другие полезные опции программы gzip.

Основные опции программы gzip

Опция	Значение
-h, --help	Вызов краткой помощи по использованию программы
-l, --list	Выдает имя файла, содержащегося в архиве, его объем и степень сжатия
-L, --license	Отображает номер версии и лицензию на программу
-N, --name	Сохранять (или восстанавливать) исходное имя и время создания файла
-n, --no-name	Не сохранять (не восстанавливать) исходное имя и время создания файла
-q, --quiet	Подавляет выдачу на экран предупреждающих сообщений
-r, --recursive	Рекурсивно обрабатывать подкаталоги (используется в случае. когда задан шаблон имен обрабатываемых файлов)
-S .suf, --suffix .suf	Добавить суффикс .suf к имени сжатого файла (вместо добавляемого по умолчанию суффикса gz; но учтите, что при разархивации файлов с суффиксами, отличными от gz, программа вас не поймет)
-t, --test	Протестировать архивный файл
-v, --verbose	Выдача дополнительных сообщений в процессе работы программы
-V, --version	Отобразить версию программы
-1, --fast	Быстрое сжатие
-9, --best	Более высокая степень сжатия

Поскольку программа gzip не умеет сохранять в одном архиве несколько файлов, то обычно ее применяют для сжатия архивов, созданных программой tar. Более того, среди опций программы tar имеется специальная опция -z, позволяющая сразу после создания сжать его с помощью программы gzip. Для выполнения такого сжатия надо использовать команду tar примерно следующим образом:

```
[user]$ tar -czf имя_архива шаблон_имен_файлов (или имя_каталога)
```

Только имейте в виду, что в этом случае суффикс .gz не добавляется автоматически к имени создаваемого архива, поэтому лучше сразу задать имя архива с указанием обеих суффиксов: имя.tar.gz.

Архиватор bzip2

В последнее время все чаще вместо программы gzip используется архиватор bzip2, который обеспечивает более высокую степень сжатия и работает несколько быстрее. Команда bzip2 обычно не устанавливается автоматически при инсталляции Linux. Но она имеется на дистрибутивном диске в виде rpm-пакета и ее легко установить.

Работает bzip2 примерно так же, как команда gzip, т. е. замещает каждый файл, имя которого задано в командной строке, сжатой версией, добавляя к имени файла суффикс .bz2.

Сжатый файл имеет то же самое время модификации, права доступа и, по возможности, того же владельца, что и исходный файл, что дает возможность восстановить эти атрибуты при извлечении файлов из архива.

Команда bunzip2 (или bzip2 -d) разархивирует указанные в командной строке файлы. Если эти файлы не были созданы программой bzip2, они не будут разархивироваться, будет выдано соответствующее сообщение. При разархивации bzip2 пытается угадать имя разархивируемого файла по следующим правилам:

- filename.bz2 заменяется на filename;
- filename.bz заменяется на filename;
- filename.tbz2 заменяется на filename.tar;
- filename.tbz заменяется на filename.tar;
- любое другое "имя" заменяется на "имя.out".

Опции командной строки для bzip2 очень похожи на опции команды gzip, но все же они не идентичны.
Краткую сводку наиболее необходимых в работе опций.

Основные опции программы bzip2

Опция	Значение
-d, --decompress	Принудительная разархивация. Эта опция необходима в силу того, что bzip2, bunzip2 и bzcat — это на самом деле одна и та же программа, которая сама по расширению имени файла принимает решение о том, какое действие надо выполнить над указанным файлом. Опция -d отключает этот механизм и заставляет программу разархивировать указанные файлы
-z, --compress	Принудительная архивация
-t, --test	Проверка целостности указанного файла(ов) без разархивации
-f, --force	Перезапись существующего файла. По умолчанию bzip2 не перезаписывает существующие файлы. Если вы хотите перезаписать существующий файл, надо задать опцию -f
-k, --keep	Сохранять (не удалять) исходные файлы при архивации или разархивации
-s, --small	Снижает требования к объему используемой оперативной памяти за счет снижения скорости архивации. Эту опцию рекомендуется применять на компьютерах с малым объемом ОЗУ (8 Мбайт и меньше)
-q, --quiet	Не выводить малосущественные сообщения
-v, --verbose	Выводить дополнительную информацию в процессе работе (представляет интерес в диагностических целях)
-L, --license,	Отобразить версию программы и лицензионное соглашение

-V, --version	

Аргументы командной строки, которым предшествует двойное тире и пробел, трактуются как имена файлов, даже если они начинаются с тире. Например,

```
[user]$ bzip2 -- -myfilename
```

08. Работа с сетью (ifconfig)

В *nix системах сетевые интерфейсы Ethernet идентифицируются с присвоением имен в виде ethX, где X является числом, отсчет идет с 0.
В некоторых ОС, например Ubuntu, сетевые интерфейсы могут именоваться инач. В примере ниже видно, что один сетевой интерфейс имеет имя enp4s0, а другой wlp3s0 (для сети Wi-Fi).
Утилита ifconfig используется для назначения адреса сетевому интерфейсу и / или для настройки и чтения параметров сетевого интерфейса.

В некоторых дистрибутивах данная утилита не установлена по умолчанию, для установки данной утилиты нужно установить пакет net-tools:

```
$ sudo apt install net-tools
```

Пример вывода команды приведен ниже

```
$ ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Локальная петля (Loopback))
        RX packets 300628  bytes 85000208 (85.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 300628  bytes 85000208 (85.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.23  netmask 255.255.224.0  broadcast 192.168.1.255
        inet6 fe80::1a39:599b:9312:ab42  prefixlen 64  scopeid 0x20<link>
        ether 54:27:1e:04:6a:f1  txqueuelen 1000  (Ethernet)
        RX packets 9943312  bytes 3370640704 (3.3 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1170227  bytes 287599882 (287.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Для настройки сетевых интерфейсов используются следующие команды.

Вывести информацию по всем интерфейсам:

```
$ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Локальная петля (Loopback))
        RX packets 300628  bytes 85000208 (85.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 300628  bytes 85000208 (85.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


enp4s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether e0:3f:49:d4:da:26  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

```
        inet 192.168.1.23  netmask 255.255.224.0  broadcast 192.168.1.255
        inet6 fe80::1a39:599b:9312:ab42  prefixlen 64  scopeid 0x20<link>
        ether 54:27:1e:04:6a:f1  txqueuelen 1000  (Ethernet)
        RX packets 9943312  bytes 3370640704 (3.3 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1170227  bytes 287599882 (287.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Интерфейсы начинающиеся на enp* - проводные. Интерфейсы начинающиеся на wlp* - беспроводные.

Вывести информацию по указанному интерфейсу wlp3s0:

```
$ ifconfig wlp3s0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.38.134.237  netmask 255.255.224.0  broadcast 10.38.159.255
        inet6 fe80::1a39:599b:9312:ab42  prefixlen 64  scopeid 0x20<link>
        ether 54:27:1e:04:6a:f1  txqueuelen 1000  (Ethernet)
        RX packets 2303439  bytes 773749496 (773.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 266710  bytes 67870054 (67.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Включить сетевой интерфейс wlp3s0:

```
$ sudo ifconfig wlp3s0 up
```

Выключить сетевой интерфейс wlp3s0:

```
$ sudo ifconfig wlp3s0 down
```

Установка IP адреса интерфейсу wlp3s0:

```
$ sudo ifconfig wlp3s0 192.168.1.10
```

Установка маски подсети для интерфейса wlp3s0:

```
$ sudo ifconfig wlp3s0 netmask 255.255.255.0
```

Если аргументы не переданы, ifconfig выдает информацию о состоянии активных интерфейсов.

Если указан один аргумент интерфейс, выдается информация только о состоянии этого интерфейса;

если указан один аргумент -a, выдается информация о состоянии всех интерфейсов, даже отключенных.

Иначе команда конфигурирует указанный интерфейс.

09. Управление процессами

Первым делом научимся определять, какие процессы в системе запущены. Для этого в Linux (как и во всех UNIX-системах) имеется команда ps. Если ее запустить без всяких параметров, то она выдает список процессов, запущенных в текущей сессии. Если вы хотите увидеть список всех процессов, запущенных в системе, надо задать ту же команду с параметром -ax.

Команда ps

Как оказалось, GNU-версия этой программы, входящая в состав Linux, поддерживает опции в стиле трех разных типов UNIX. Опции в стиле Unix98 состоят из одного или нескольких символов, перед которым(и) должен стоять дефис. Опции в стиле BSD имеют аналогичный вид, только используются без дефиса. Опции, характерные только для GNU-версии представляют собой слово, перед которым должно стоять два дефиса. Их нельзя объединять, как однобуквенные опции двух предшествующих типов. Таким образом, существует три равноправных формата задания этой команды:

ps [-опции]

ps [опции]

ps [-- длинное_имя_опции [-- длинное_имя_опции] ...]

При этом опции разных типов нельзя употреблять в одной команде. Дадим краткую характеристику наиболее важных опций.

Первая группа опций регулирует вывод команды. Независимо от наличия опций этой группы команда ps выдает для каждого процесса отдельную строку, но содержимое этой строки может быть разным. В зависимости от заданных опций могут присутствовать следующие поля:

- USER — имя владельца процесса;
- PID — идентификатор процесса в системе;
- PPID — идентификатор родительского процесса;
- %CPU — доля времени центрального процессора (в процентах), выделенного данному процессу;
- %MEM — доля реальной памяти (в процентах), используемая данным процессом;
- VSZ — виртуальный размер процесса (в килобайтах);
- RSS — размер резидентного набора (количество 1K-страниц в памяти);
- STIME — время старта процесса;
- TTY — указание на терминал, с которого запущен процесс;
- S или STAT — статус процесса;
- PRI — приоритет планирования;
- NI — значение nice (см. описание команды nice ниже);
- TIME — сколько времени центрального процессора занял данный процесс;
- CMD или COMMAND — командная строка запуска программы, выполняемой данным процессом;

а также и другие поля, полный список которых приведен на man-странице, посвященной команде ps.

Значения, выводимые в большинстве этих полей вы поймете без дополнительных пояснений. В поле **Статус процесса**, как уже говорилось выше, могут стоять следующие значения:

- R — выполнимый процесс, ожидающий только момента, когда планировщик задач выделит ему очередной квант времени;
- S — процесс "спит";
- D — процесс находится в состоянии подкачки на диске;
- T — остановленный процесс;
- Z — процесс-зомби.

Рядом с указателем статуса могут стоять дополнительные символы из следующего набора:

- W — процесс не имеет резидентных страниц;
- < — высоко-приоритетный процесс;
- N — низко-приоритетный процесс;
- L — процесс имеет страницы, заблокированные в памяти.

Вторая группа опций регулирует то, какие именно процессы включаются в вывод команды. Чтобы получить список всех процессов надо использовать команду ps с опциями ax или –A. Вывод в этих двух случаях отличается только в поле CMD: в первом случае выдается полная командная строка запуска программы, а во втором — только имя запущенной программы.

Описание всех опций программы ps здесь привести невозможно. Поэтому приведем только несколько примеров ее применения, которые покажут, как пользоваться этой командой в типичных ситуациях.

Для того чтобы увидеть все процессы в системе, используя стандартную форму вывода:

```
[user]$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:07 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    6 ?        00:00:00 kworker/0:0H-kb
    8 ?        00:00:00 mm_percpu_wq
    9 ?        00:00:00 ksoftirqd/0
   10 ?        00:00:17 rcu_sched
   11 ?        00:00:00 rcu_bh
.....................................................
```

Можно к той же команде добавить опцию –o, после которой указать через запятую, какие именно поля вы хотите видеть в выводе команды:

```
[user]$ ps -e -o cmd,pid,user
CMD                          PID USER
/sbin/init splash              1 root
[kthreadd]                     2 root
[rcu_gp]                       3 root
[rcu_par_gp]                   4 root
[kworker/0:0H-kb]              6 root
[mm_percpu_wq]                 8 root
[ksoftirqd/0]                  9 root
[rcu_sched]                   10 root
[rcu_bh]                      11 root
```

```
.....................................................
```

Для того, чтобы увидеть все процессы в системе, используя форму вывода BSD-систем:

```
[user]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:07 /sbin/init splash
    2 ?        S      0:00 [kthreadd]
    3 ?        I<     0:00 [rcu_gp]
    4 ?        I<     0:00 [rcu_par_gp]
    6 ?        I<     0:00 [kworker/0:0H-kb]
    8 ?        I<     0:00 [mm_percpu_wq]
    9 ?        S      0:00 [ksoftirqd/0]
   10 ?        I      0:18 [rcu_sched]
   11 ?        I      0:00 [rcu_bh]
.....................................................
```

Для того, чтобы увидеть, сколько % ЦПУ и памяти занимают запущенные вами процессы:

```
[user]$ ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user     11151  0.0  0.0  36040  3368 pts/0    R+   17:06   0:00 ps -u
user     17994  0.0  0.0  22832  6820 pts/0    Ss   15:45   0:00 /bin/bash
```

Чтобы получить весь список запущенных процессов используется ключи aux:

```
[user]$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 226524 10676 ?        Ss   12:08   0:08 /sbin/init
splash
root         2  0.0  0.0      0     0 ?        S    12:08   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   12:08   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   12:08   0:00
[rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   12:08   0:00
[kworker/0:0H-kb]
root         8  0.0  0.0      0     0 ?        I<   12:08   0:00
[mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    12:08   0:00
[ksoftirqd/0]
root        10  0.1  0.0      0     0 ?        I    12:08   0:21 [rcu_sched]
root        11  0.0  0.0      0     0 ?        I    12:08   0:00 [rcu_bh]
.....................................................................
.........................
```

Команда top

Команда ps позволяет вывести список процессов , запущенных в системе, на момент выполнения этой команды.
Команда top отображает состояние процессов и их активность в реальном режиме времени.
На рисунке 3.1 изображено окно терминала, в котором запущена программа top.

Вывод команды top

```
top - 15:43:23 up 1 day,  3:34,  1 user,  load average: 0,63, 0,60, 0,55
Tasks: 434 total,   2 running, 316 sleeping,   0 stopped,   0 zombie
%Cpu(s): 15,8 us,  8,6 sy,  0,0 ni, 73,9 id,  0,0 wa,  0,0 hi,  1,7 si,  0,0 st
КиБ Mem : 16308364 total,   583388 free,  9706452 used,  6018524 buff/cache
КиБ Swap:  7670780 total,  7653360 free,    17420 used.  5896792 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 6230 oem       20   0 1003100 360844  73100 S  40,5  2,2  31:55.27 deepin-wm
15273 oem       20   0 3923776 884236 177396 S  20,6  5,4  30:47.57 firefox
 5008 root      20   0  520360 195428 104608 R  11,3  1,2  53:22.16 Xorg
25697 oem       20   0 1002432 311584  93184 S   6,0  1,9  14:05.29 chrome
 8795 oem       20   0 83,918g 323904 106240 S   4,7  2,0   4:36.44 evolution
 5024 root     -51   0       0      0      0 S   3,3  0,0  11:02.19 irq/23-nvidia
10160 oem       20   0 1787872 466360 188380 S   2,3  2,9  42:11.84 chrome
 9096 oem       20   0 98,494g 105084  79524 S   1,3  0,6   0:40.86 WebKitWebProces
18043 oem       20   0  823080 117984  72708 S   1,3  0,7   1:14.41 chrome
 3375 oem       20   0  622776  36312  28000 S   1,0  0,2   0:00.48 gnome-screensho
 6548 oem       20   0 1234044 282460  46276 S   1,0  1,7   9:47.90 docky
10981 oem       20   0  845940 188724  87724 S   1,0  1,2  15:20.91 chrome
21384 oem       20   0 3627392 1,277g 133564 S   1,0  8,2  17:36.42 Web Content
 1940 message+  20   0   60372   8084   4628 S   0,7  0,0   2:28.56 dbus-daemon
 2553 oem       20   0   40908   4268   3228 R   0,7  0,0   0:00.70 top
 6232 oem       20   0 1628052  40688  22500 S   0,7  0,2   1:10.55 dde-session-ini
15769 oem       20   0 2302832 485932 125116 S   0,7  3,0   1:50.47 Web Content
21006 oem       20   0 1048608 308088  95856 S   0,7  1,9  26:28.54 chrome
21396 oem       20   0 2173016 473520 138200 S   0,7  2,9   2:23.72 Web Content
 1919 avahi     20   0   57172   7516   4184 S   0,3  0,0  25:58.62 avahi-daemon
 3080 root      20   0 1089856  65344  38488 S   0,3  0,4   1:30.56 dockerd
 3883 root      20   0  293536  58104  22172 S   0,3  0,4   7:08.98 mongod
```

В верхней части окна отображается астрономическое время, время, прошедшее с момента запуска системы, число пользователей в системе, число запущенных процессов и число процессов, находящихся в разных состояниях, данные об использовании ЦПУ, памяти и свопа. А далее идет таблица, характеризующая отдельные процессы. Число строк, отображаемых в этой таблице, определяется размером окна: сколько строк помещается, столько и выводится. Графы таблицы обозначены так же, как поля вывода команды ps .

Содержимое окна обновляется каждые 3 секунд. Список процессов может быть отсортирован по используемому времени ЦПУ (по умолчанию), по использованию памяти, по PID, по времени исполнения. Переключать режимы отображения можно с помощью команд, которые программа top воспринимает.
Ниже указаны сочетания клавиш и результат их выполнения, который можно визуально увидеть при работе с программой top:

- **<Shift>+<N>** — сортировка по PID;
- **<Shift>+<A>** — сортировать процессы по возрасту;
- **<Shift>+<P>** — сортировать процессы по использованию ЦПУ;
- **<Shift>+<M>** — сортировать процессы по использованию памяти;
- **<Shift>+<T>** — сортировка по времени выполнения.

Кроме команд, определяющих режим сортировки, команда top воспринимает еще ряд команд, которые позволяют управлять процессами в интерактивном режиме. С помощью клавиши <K> можно завершить некоторый процесс (его PID будет запрошен), а с помощью клавиши <R> можно переопределить значение nice для некоторого процесса. Если нажать "m", то сможешь увидеть информацию о свободной и потребляемой памяти:

```
КиБ Mem : 16305524 total,  3219580 free,  7607492 used,  5478452 buff/cache
КиБ Swap:  7670780 total,  7670780 free,        0 used.  8250172 avail Mem
```

Для вызова справки о работе утилиты, можно нажать клавишу "h":

```
Help for Interactive Commands - procps-ng 3.3.12
```

```
Window 1:Def: Cumulative mode Off.  System: Delay 3,0 secs; Secure mode Off.

  Z,B,E,e   Global: 'Z' colors; 'B' bold; 'E'/'e' summary/task memory scale
  l,t,m     Toggle Summary: 'l' load avg; 't' task/cpu stats; 'm' memory
info
  0,1,2,3,I Toggle: '0' zeros; '1/2/3' cpus or numa node views; 'I' Irix
mode
  f,F,X     Fields: 'f'/'F' add/remove/order/sort; 'X' increase fixed-width

  L,&,<,> . Locate: 'L'/'&' find/again; Move sort column: '<'/'>' left/right
  R,H,V,J . Toggle: 'R' Sort; 'H' Threads; 'V' Forest view; 'J' Num justify
  c,i,S,j . Toggle: 'c' Cmd name/line; 'i' Idle; 'S' Time; 'j' Str justify
  x,y     . Toggle highlights: 'x' sort field; 'y' running tasks
  z,b     . Toggle: 'z' color/mono; 'b' bold/reverse (only if 'x' or 'y')
  u,U,o,O . Filter by: 'u'/'U' effective/any user; 'o'/'O' other criteria
  n,#,^O  . Set: 'n'/'#' max tasks displayed; Show: Ctrl+'O' other filter(s)
  C,...   . Toggle scroll coordinates msg for: up,down,left,right,home,end

  k,r       Manipulate tasks: 'k' kill; 'r' renice
  d or s    Set update interval
  W,Y       Write configuration file 'W'; Inspect other output 'Y'
  q         Quit
          ( commands shown with '.' require a visible task display window )
Press 'h' or '?' for help with Windows,
Type 'q' or <Esc> to continue
```

Сигналы и команда kill

Сигналы — это средство, с помощью которого процессам можно передать сообщения о некоторых событиях в системе. Сами процессы тоже могут генерировать сигналы, с помощью которых они передают определенные сообщения ядру и другим процессам. С помощью сигналов можно осуществлять управление процессами, как приостановка процесса, запуск приостановленного процесса, завершение работы процесса. Всего в Linux существует 64 разных сигнала, их перечень можно посмотреть по команде

```
[user]$ kill -l
 1) SIGHUP  2) SIGINT  3) SIGQUIT   4) SIGILL  5) SIGTRAP
 6) SIGABRT   7) SIGBUS  8) SIGFPE  9) SIGKILL  10) SIGUSR1
11) SIGSEGV  12) SIGUSR2  13) SIGPIPE  14) SIGALRM  15) SIGTERM
16) SIGSTKFLT17) SIGCHLD  18) SIGCONT  19) SIGSTOP  20) SIGTSTP
21) SIGTTIN  22) SIGTTOU  23) SIGURG 24) SIGXCPU  25) SIGXFSZ
26) SIGVTALRM27) SIGPROF  28) SIGWINCH 29) SIGIO  30) SIGPWR
31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42)
SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47)
SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-
12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-
7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-
2
63) SIGRTMAX-1  64) SIGRTMAX
```

Сигналы принято обозначать номерами или символическими именами. Все имена начинаются на SIG, например, сигнал с номером 1 обозначают или как SIGHUP

Когда процесс получает сигнал, то возможен один из двух вариантов развития событий. Если для данного сигнала определена подпрограмма обработки, то вызывается эта

подпрограмма. В противном случае ядро выполняет от имени процесса действие, определенное по умолчанию для данного сигнала. Вызов подпрограммы обработки называется *перехватом* сигнала. Когда завершается выполнение подпрограммы обработки, процесс возобновляется с той точки, где был получен сигнал.

Можно заставить процесс игнорировать или блокировать некоторые сигналы. Игнорируемый сигнал просто отбрасывается процессом и не оказывает на него никакого влияния. Блокированный сигнал ставится в очередь на выдачу, но ядро не требует от процесса никаких действий до разблокирования сигнала. После разблокирования сигнала программа его обработки вызывается только один раз, даже если в течение периода блокировки данный сигнал поступал несколько раз.

В таблице ниже приведены некоторые из часто встречающихся сигналов.

Сигналы

№	**Имя**	**Описание**	**Комбинация клавиш**
1	SIGHUP	Hangup. Отбой	
2	SIGINT	Interrupt. В случае выполнения простых команд вызывает прекращение выполнения, в интерактивных программах — прекращение активного процесса	<Ctrl>+<C> или <Del>
3	SIGQUIT	Как правило, сильнее сигнала Interrupt	<Ctrl>+<\>
4	SIGILL	Illegal Instruction. Центральный процессор столкнулся с незнакомой командой (в большинстве случаев это означает, что допущена программная ошибка). Сигнал отправляется программе, в которой возникла проблема	
8	SIGFPE	Floating Point Exception. Вычислительная ошибка, например, деление на ноль	
9	SIGKILL	Всегда прекращает выполнение процесса	
11	SIGSEGV	Segmentation Violation. Доступ к недозволенной области памяти	
13	SIGPIPE	Была предпринята попытка передачи данных с помощью конвейера или очереди FIFO, однако не существует процесса, способного принять эти данные	
15	SIGTERM	Software Termination. Требование закончить процесс (программное завершение)	
17	SIGCHLD	Изменение статуса порожденного процесса	
18	SIGCONT	Продолжение выполнения приостановленного процесса	
19	SIGSTOP	Приостановка выполнения процесса	
20	SIGTSTP	Сигнал останова, генерируемый клавиатурой. Переводит процесс в фоновый режим	<Ctrl>+<Z>

Некоторые сигналы можно сгенерировать с помощью определенных комбинаций клавиш. Но такие комбинации существуют не для всех сигналов. Остальные сигналы можно посылать при помощи команды kill. Для посылки сигнала процессу (или группе процессов) можно воспользоваться командой kill в следующем формате:

```
[user]$ kill [-сигн] PID [PID..]
```

где сигн — это номер сигнала, по-умолчанию, если никакая опция не указана, то посылается сигнал 15 (TERM — программное завершение процесса).

Самый частоиспользуемый сигнал 9 (KILL). При помощи этого сигнала и обладая правами пользователя root(суперпользователь), можно завершить любой процесс. В большинстве случаев рекомендуется использовать сигналы TERM или QUIT, которые завершают процесс более корректно.

Часто команду kill применяют с правами суперпользователя(root). Как правило это делается для уничтожения процессов-зомби, зависших процессов (они показываются в листинге команды ps как <exiting>), процессов, которые занимают слишком много процессорного времени или слишком большой объем памяти.

Перевод процесса в фоновый режим

Программы, запускаемые из командной строки, начинают работать в приоритетном режиме. Это позволяет вам видеть данные, выводимые программой, и взаимодействовать с ней. Однако в некоторых случаях вам хотелось бы, чтобы программа работала, не захватывая полностью ваш терминал. Это называется работой программы в фоновом режиме, и для осуществления этой задачи существует несколько способов.

Первым способом перевода процесса в фоновый режим является добавление в командную строку амперсанда **&** при запуске программы.

```
[user]$ programa &
```

programa запустится и будет работать в фоновом режиме, появится доступ к командной строке терминала и можно продолжить дальнейшую работу.

В оболочке bash имеются две встроенные команды, которые служат для перевода процессов на передний план или возврата их в фоновый режим. Но прежде, чем рассказывать об этих командах, надо рассказать о команде jobs. Она всегда вызывается без аргументов и показывает задания, запущенные из текущего экземпляра shell. В начале каждой строки вывода этой команды указывается порядковый номер задания в виде числа в квадратных скобках. После номера указывается состояние процесса: stopped (остановлен), running (выполняется) или suspended (приостановлен). В конце строки указывается команда, которая исполняется данным процессом. Один из номеров выполняющихся заданий помечен знаком +, а еще один — знаком -. Процесс, помеченный знаком +, будет по умолчанию считаться аргументом команд fg или bg, если они вызываются без параметров. Процесс, помеченный знаком -, получит знак +, если только завершится по какой-либо причине процесс, который был помечен знаком +.

А теперь можно рассказать и о командах fg и bg, которые служат для перевода процессов на передний план или возврата их в фоновый режим. В качестве аргумента обеим этим командам передаются номера тех заданий, которые присутствуют в выводе команды jobs. Если аргументы отсутствуют, то подразумевается задание, помеченное знаком +. Команда fg переводит указанный в аргументе процесс на передний план, а команда bg — переводит процесс в фоновый режим. Одной командой bg можно перевести в фоновый режим сразу несколько процессов, а вот возвращать их на передний план необходимо по одному.

Команда nohup

Предположим, вы запустили из оболочки bash несколько процессов, часть из них в фоновом режиме. И по каким-то причинам завершили текущую сессию работы в оболочке. При завершении сессии оболочка посылает всем порожденным ею процессам сигнал "отбой", по которому некоторые из порожденных ею процессов могут завершиться, что не всегда желательно. Если вы хотите запустить в фоновом режиме программу, которая должна выполняться и после вашего выхода из оболочки, то ее нужно запускать с помощью утилиты nohup**.** Делается это так:

nohup команда &

Запускаемый таким образом процесс будет игнорировать посылаемые ему сигналы (если это возможно, см. табл. 8.1). Стандартный выходной поток и стандартный поток ошибок при таком запуске команд перенаправляются в файл nohup.out или $HOME/nohup.out.

Команда nohup имеет побочный эффект, заключающийся в том, что значение nice для запускаемого процесса увеличивается на 5, т. е. процесс выполняется с более низким приоритетом.

Дополнение

В некоторых случаях может быть полезна команда lsof, fuser, об их работе применении можно узнать по ссылкам и из man.

10. Управление службами

В операционной системе Linux, так же как и в Windows, кроме обычных программ, которые могут взаимодействовать с пользователем есть еще один вид программ. Это работающие в фоне службы. Они следят за состоянием системы, обеспечивают автоматическое подключение внешних устройств и сети, позволяют процессам взаимодействовать с оборудованием (dbus), а также в виде служб реализованы различные веб-серверы и серверы баз данных. В отличие от пользовательских программ, службы выполняются в фоне, и пользователь не имеет к ним прямого доступа. Пользователь еще не вошел в систему, только началась загрузка а основные службы уже запущенны и работают.

Рассмотрим управление службами Linux. Мы не будем трогать уже устаревшие системы, такие как SysVinit, сосредоточимся только на Systemd, она также имеется в используемом для нашей практики дистрибутиве Ubuntu Linux.

Чтобы всем этим управлять нужна основная служба - система инициализации, которая будет запускать службы Linux в нужный момент, следить чтобы они нормально работали, записывать сообщения логов, и самое главное позволять останавливать службы. Раньше, для управления службами использовались скрипты. Т.к. службу можно запустить из терминала, каждая служба запускалась в фоновом режиме одна за другой, без возможности параллельного запуска и возвращала свой PID(идентификатор) процесса скрипту инициализации, он сохранялся и потом с помощью этого PID можно было проверить работает ли служба и остановить службу Linux если это нужно. Все это можно сделать и вручную.

Потом на смену этому методу пришла новая модель и система инициализации systemd. Система инициализации запускается сразу после загрузки ядра и начинает инициализировать службы, теперь появилась возможность параллельной инициализации, а также зависимостей между службами. Таким образом, теперь можно определить сложное дерево порядка запуска служб. После запуска systemd собирает весь вывод службы в лог, и следит за ее работой, если служба аварийно завершилась, то автоматически ее перезапускает.

В Systemd есть специальный инструмент для управления службами в Linux - systemctl. Эта утилита позволяет делать очень много вещей, начиная от перезапуска службы Linux и проверки ее состояния, до анализа эффективности загрузки службы. Синтаксис следующий:

```
$ sudo systemctl опции команда служба служба …
```

Опции настраивают поведение программы, подробность вывода, команда - указывает что нужно сделать со службой, а служба, это та самая служба, которой мы собираемся управлять. В некоторых случаях утилита может использоваться без указания команды и службы.

Опций достаточно большое количество, подробнее можно ознакомиться в официальной документации. Мы будем использовать несколько из них:

- **start** - запустить службу Linux
- **stop** - остановить службу Linux
- **restart** - перезапустить службу
- **status** - посмотреть состояние и вывод службы
- **enable** - добавить службу в автозагрузку

Примеры команд с использованием данных опций:

```
$ sudo systemctl start nginx
```

```
$ sudo systemctl stop nginx
```

```
$ sudo systemctl status nginx
```

```
$ sudo systemctl enable nginx
```

Менеджер инициализации systemd фактически уже стал стандартом в современных Linux-системах. На данный момент он используется во многих популярных дистрибутивах: Debian, RHEL/CentOS, Arch, Ubuntu. systemd имеет собственную систему ведения логов, названную журналом (journal), которая использует принципиально иной (по сравнению с традиционным инструментом syslog) подход к логгированию. Специализированный компонент journal собирает все системные сообщения (сообщения ядра, различных служб и приложений). При этом специально настраивать отправку логов не нужно: приложения могут просто писать в stdout и stderr, а journal сохранит эти сообщения автоматически. **systemd** сохраняет логи в бинарной базе, что существенно упрощает систематизацию и поиск. Хранение логов в бинарных файлах также позволяет избежать сложностей с использованием парсеров для разных видов логов. При необходимости логи можно без проблем переконвертировать в другие форматы. Journal может работать как совместно с syslog, так и полностью заменить его.
Для просмотра логов используется утилита journalctl:

```
$ sudo journalctl
```

У данного решения есть возможность фильтровать логи по различным параметрам:

- дате и времени
- приложения
- службам
- процессам
- пользователям и группам
- по пути
- просмотр сообщений ядра
- сообщений по уровню ошибки

Нас, на данном этапе, интересует два:

- по службам

```
$ sudo journalctl -u имя_службы
```

- по ошибкам, когда команда выводит логи с ошибками

```
$ sudo journalctl -xe
```

11. SSH-консоль

SSH-соединение (консольный доступ к серверу) позволяет вам выполнять программы и управлять файлами непосредственно на сервере. Это может быть полезно, если вам необходимо произвести действия с файлами на сервере, не скачивая их к себе на компьютер (например, просмотр логов доступа, перекодирование файлов и др.)

Как правило, в дистрибутивах Linux ssh-клиент уже стоит по-умолчанию и можно им пользовать через консоль.

Поскольку у большинства участников будет использоваться ОС Windows мы будем использовать Putty.

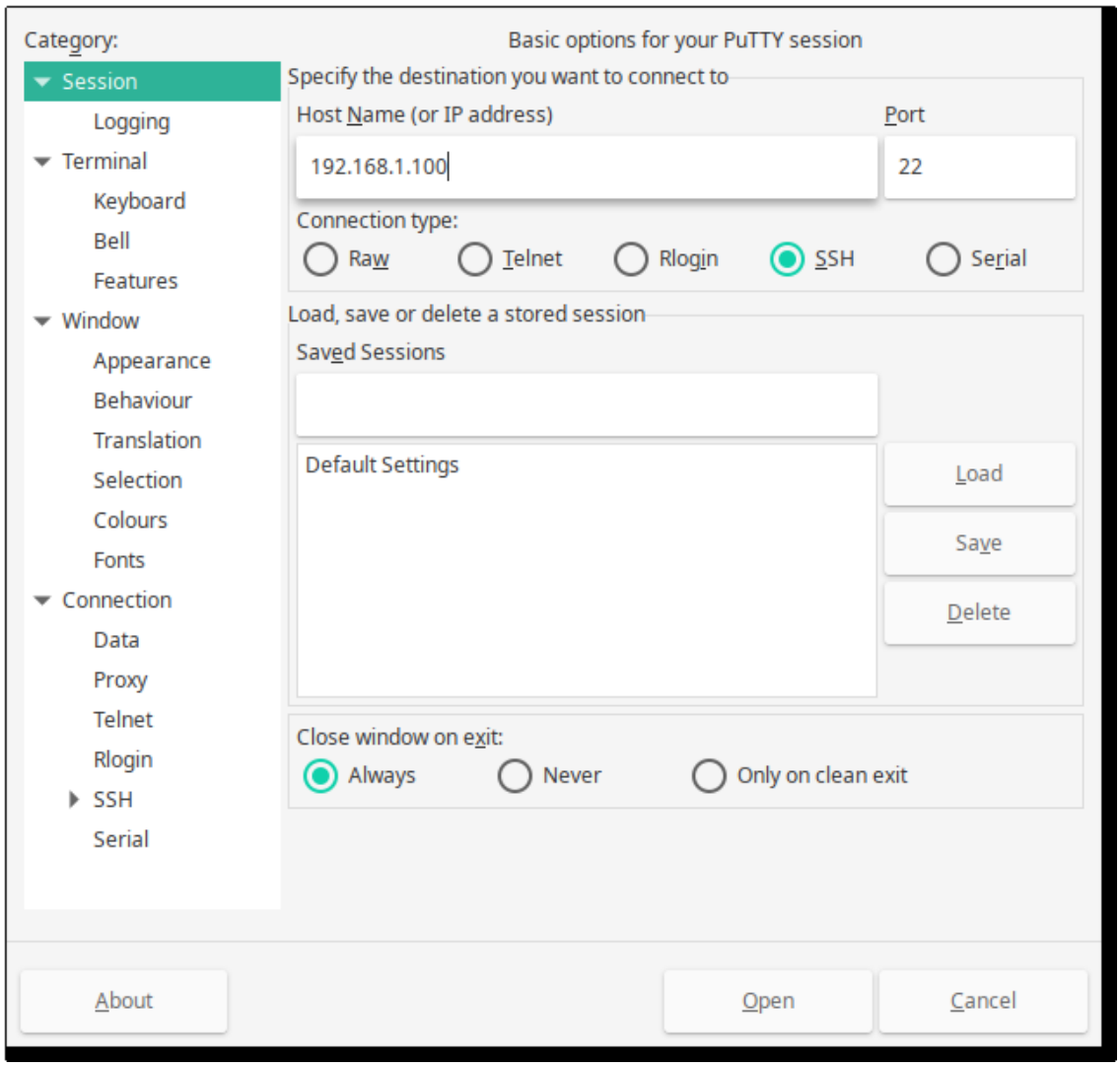
PuTTY позволяет подключиться и управлять удаленным узлом (например, сервером). В PuTTY реализована только клиентская сторона соединения — сторона отображения, в то время как сама работа выполняется на стороне сервера.

Есть несколько вариантов подключения:

1. по IP-адресу или имени хоста
2. по ключам

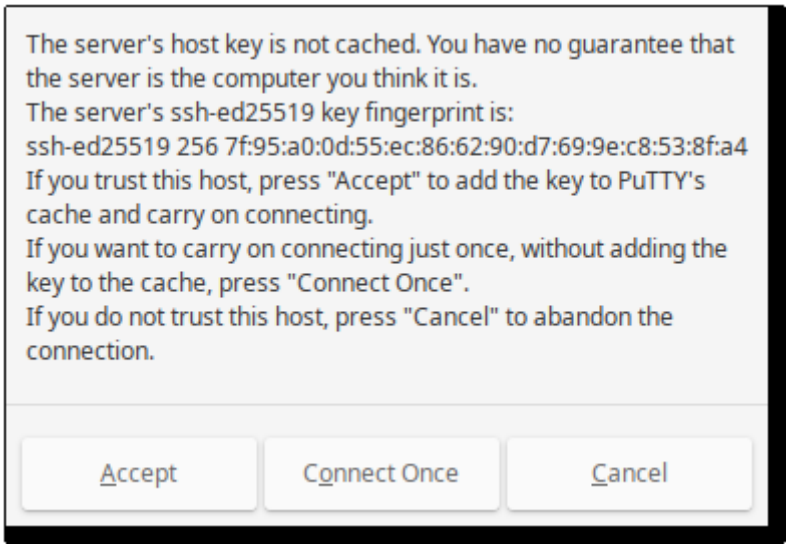
Подключение по IP-адресу или имени хоста.

Для подключения по IP-адресу или хосту достаточно ввести их в соотвествующее поле:



И нажать Open.

При первом подключении появится запрос на добавления нового ключа:



Принимаем запрос (Accept).

Далее последует запрос ввода логина и пароль для пользователя:

После успешнного ввода логина и пароль мы получим доступ к консоли серера, или виртуальной машины.

Подключение с использованием ключа.

Для использования подкючения по SSH при помощи ключенй, необходимо сгенерировать ключевую пару: открытый ключ выслать администратору удаленной машины, а приватный указать в настройках ssh-клиента. Подробнее о процессе создания ключевой пары можно прочитать на странице ssh-ключи, генерация

Передача файлов.

Протокол **SSH** позволяет не только защищенно администрировать компьютеры и получать доступ к сервисам сети через туннели, но еще и передавать файлы. Делается это очень просто.
Для этого нужна консольная утилита **pscp**.exe, входящая в очень полезный пакет утилит **putty**. Это свободный пакет, его легко можно найти в сети.

Синтаксис pcsp выглядит следующим образом:

```
pscp опции путь_файлу имя_пользователя@хост:/путь/к/файлу/на/удаленном/хосте
```

Например, чтобы передать файл text.txt в домашний каталог пользователя user нужно выполнить следующую команду:

```
pscp test.txt user@192.168.1.100:/home/user/
```

Важный нюанс: в данном случае утилита pcsp и файл test.txt находятся в одном каталоге.

В качестве альтернативного решения, можно использовать WinSCP.

Более подробную информацию по работе с Putty можно узнать из её документации.

12. Справочные страницы man

Большинство программ и утилит в Linux имеют своё руководство. Как правило, это более подробное описание справки команды, которая вызывается при помощи ключа --help.

Чтобы получить доступ к расширенной справке достаточно ввести следующую команду:

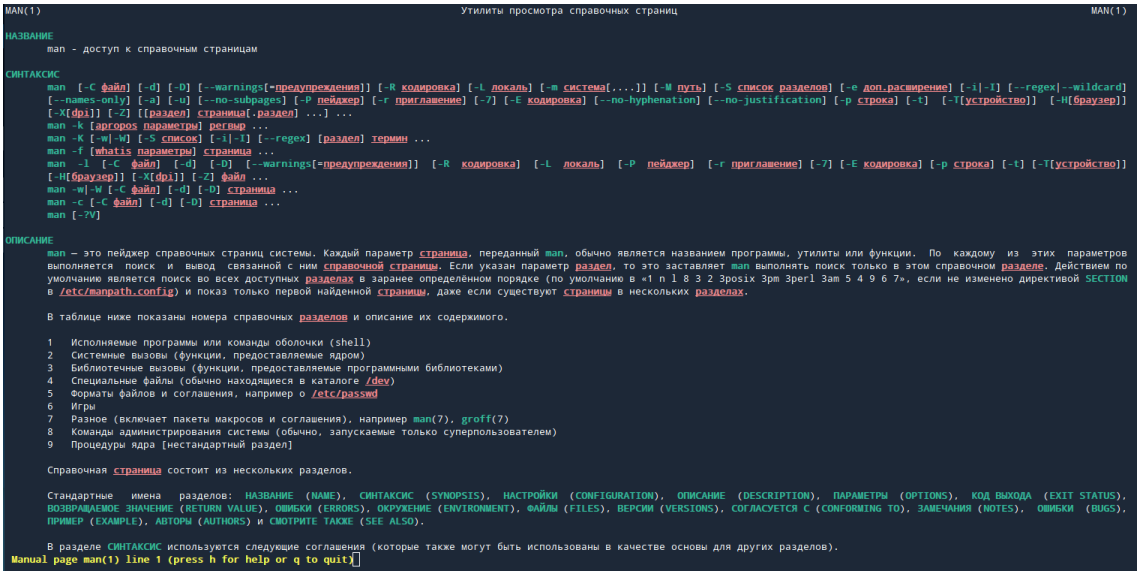
```
[user]$ man имя_команды
```

Для примера, вывод команды справки для команды man выглядит так

Команда:

```
[user]$ man man
```

Вывод:



Навигация по файлу справки

Для более удобной навигации по справке есть горячие:

- <PageUp> или <k> — прокрутка текста вверх.
- <PageDown> или <j> — прокрутка текста вниз.
- <Ctrl> + <u> — прокрутка страницы в начало.
- <Ctrl> + <d> — прокрутка страницы в конец.
- <?> - поиск текста. После активации этой клавиши необходимо ввести искомое слово и нажать Enter. Все вхождения будут подсвечены.
- <n> — перейти к следующему вхождению искомого слова.
- <N> — перейти к предыдущему вхождению искомого слова.
- <q> — выйти из просмотра справочной информации.
- <h> — просмотр всех доступных горячих клавиш.

13. Мониторинг

Работа с логами

В процессе работы как сервисами, так и с приложения всегда есть вероятность того, что что-то пойдёт не так. Чтобы помочь разобраться с тем, в чём же может быть проблема, в Linux существуют различные инструменты. Кроме приложений, ведется логирование других событий системы.

Логи пишет практически любой сервис. Для их хранения используется каталог /var/log.

Ниже вывод списка логов, но он неполный, для каждой системы будет отличаться в зависимости от установленного ПО:

```
$ ls –ls /var/log ls -lh /var/log/

итого 110M
drwxr-xr-x 1 root              root      644 ноя  1 00:10 account
-rw-r--r-- 1 root              root        0 ноя  1 00:10 alternatives.log
-rw-r----- 1 root              adm         0 дек 15  2017 apport.log
drwxr-xr-x 1 root              root      768 ноя  1 10:52 apt
-rw-r--r-- 1 root              root        0 июл  6 00:08 aptitude
drwxr-x--- 1 root              root      106 апр 16  2018 audit
-rw-r----- 1 syslog            adm      1,4M ноя  1 12:26 auth.log
-rw-r--r-- 1 root              root     122K окт 30 14:54 boot.log
-rw-r--r-- 1 root              root      60K июн  2  2017 bootstrap.log
-rw-rw---- 1 root              utmp        0 ноя  1 00:10 btmp
drwxr-xr-x 1 root              root      152 ноя  1 00:10 ConsoleKit
drwxr-xr-x 1 root              root      448 ноя  1 00:10 cups
-rw-r----- 1 root              adm        31 июн  2  2017 dmesg
-rw-r--r-- 1 root              root     8,3K ноя  1 10:52 dpkg.log
-rw-r--r-- 1 root              root     2,0M окт 29 14:51 faillog
-rw-r----- 1 syslog            adm       12M ноя  1 12:25 kern.log
-rw-rw-r-- 1 root              utmp      18M окт 29 14:51 lastlog
drwxr-xr-x 1 root              adm       140 окт 31 09:07 nginx
drwxr-xr-x 1 ntp               ntp         0 апр  5  2017 ntpstats
-rw-r----- 1 syslog            adm      9,7M ноя  1 12:26 syslog
drwxr-xr-x 1 root              root        0 мая 12  2016 sysstat
-rw------- 1 root              root     8,4K окт 29 14:51 tallylog
-rw-r----- 1 syslog            adm       11M ноя  1 12:25 ufw.log
-rw-rw-r-- 1 root              utmp        0 ноя  1 00:10 wtmp
-rw-r--r-- 1 root              root     119K окт 31 17:36 Xorg.0.log
-rw-r--r-- 1 root              root      59K окт 30 14:54 Xorg.0.log.old
-rw-r--r-- 1 root              root      11K окт 30 14:52 Xorg.1.log
-rw-r--r-- 1 root              root      11K окт 30 14:52 Xorg.1.log.old
-rw-r--r-- 1 root              root      65K окт 30 14:53 Xorg.failsafe.log
-rw-r--r-- 1 root              root      48K окт 30 14:52
Xorg.failsafe.log.old
```

Список может быть очень длинным и не уместиться на одном экране. Также, для некоторых сервисов, логи могут лежать во вложенных каталогах. Например, логи NGINX лежат в каталоге /var/log/nginx

Рассмотрим несколько примеров логов:

- /var/log/messages - содержит глобальные системные логи Linux, в том числе те, которые регистрируются при запуске системы. В этот лог записываются несколько типов сообщений: это почта, cron, различные сервисы, ядро, аутентификация и другие./var/log/dmesg - содержит сообщения, полученные от ядра. Регистрирует много сообщений еще на этапе загрузки, в них отображается информация об аппаратных устройствах, которые инициализируются в процессе загрузки. Можно сказать это еще один лог системы Linux. Количество сообщений в логе ограничено, и когда файл будет

переполнен, с каждым новым сообщением старые будут перезаписаны. Вы также можете посмотреть сообщения из этого лога с помощью команды dmseg.

- /var/log/lastlog - Отображает информацию о последней сессии всех пользователей. Это нетекстовый файл, для его просмотра необходимо использовать команду lastlog.
- /var/log/user.log - Информация из всех журналов на уровне пользователей.
- /var/log/Xorg.x.log - Лог сообщений X сервера.
- /var/log/alternatives.log - Информация о работе программы update-alternatives. Это символические ссылки на команды или библиотеки по умолчанию.
- /var/log/btmp - лог файл Linux содержит информацию о неудачных попытках входа. Для просмотра файла удобно использовать команду last -f /var/log/btmp
- /var/log/faillog - лог системы Linux, содержит неудачные попытки входа в систему. Используйте команду faillog, чтобы отобразить содержимое этого файла.

Ниже описаны примеры, как можно посмотреть логи, используя стандартные инструменты ОС Linux.

Смотрим лог /var/log/messages, с возможностью прокрутки:

```
~ $ less /var/log/messages
```

Просмотр логов Linux, в реальном времени:

```
~ $ tail -f /var/log/messages
```

Открываем лог файл dmesg:

```
~ $  cat /var/log/dmesg
```

Выводим только ошибки из /var/log/messages:

```
~ $  grep -i error /var/log/messages
```

IOSTAT

iostat – утилита, предназначенная для мониторинга использования дисковых разделов, входящая в набор sysstat.

Пример вызова iostat без ключей:

```
~ $ iostat
Linux 4.19.0-041900-generic (hostname)  02.11.2018   _x86_64_   (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          12,80    0,03    4,64    1,15    0,00   81,38

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0,00         0,04         0,00       1069          0
loop1             0,00         0,04         0,00       1053          0
loop2             0,00         0,04         0,00       1055          0
loop3             0,00         0,01         0,00        338          0
loop4             0,00         0,04         0,00       1081          0
loop5             0,00         0,04         0,00       1067          0
loop6             0,00         0,04         0,00       1050          0
loop7             0,00         0,04         0,00       1063          0
sda              20,63       135,23       629,11    3407874   15853545
sdb               0,03         0,85         0,08      21473       2048
loop8             0,00         0,04         0,00       1063          0
```

```
loop9             0,00         0,04         0,00       1064          0
loop10            0,00         0,04         0,00       1068          0
loop11            0,00         0,04         0,00       1067          0
loop12            0,87         0,91         0,00      23029          0
loop13            0,00         0,01         0,00        330          0
loop14            0,00         0,04         0,00       1080          0
loop15            0,00         0,00         0,00        112          0
loop16            0,00         0,04         0,00       1054          0
loop17            0,00         0,04         0,00       1066          0
loop18            0,00         0,04         0,00       1044          0
loop19            0,00         0,04         0,00       1075          0
loop20            0,00         0,04         0,00       1055          0
loop21            1,10         1,14         0,00      28839          0
loop22            0,00         0,04         0,00       1069          0
loop23            0,67         0,71         0,00      17902          0
loop24            0,00         0,04         0,00       1064          0
loop25            0,00         0,04         0,00       1067          0
loop26            0,00         0,00         0,00          8          0
```

По-умолчанию, iostat выводит довольно мало информации. Рассмотрим наиболее полезные ключи:

-d – отображать только использование дисков;
-c – отобразить только использование CPU;
-j – отобразить имя раздела (ID | LABEL | PATH | UUID);
-k – отобразить данные в килобайтах;
-m – отобразить данные в мегабайтах;
-p – отобразить статистику по указанному блочному устройству;
-t – отобразить время выполнения теста;
-x – отобразить расширенную статистику;

Кроме того, iostat можно вызвать с двумя ключами, передав количество выполнения и паузу между ними.

Например, выполнить 2 раза с интервалом в 1 секунду, только для диска sda:

```
~ $ iostat -d -t -p sda 1 2

Linux 4.19.0-041900-generic (hostname)  02.11.2018   _x86_64_   (4 CPU)

02.11.2018 18:29:29
Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda              20,59       134,68       628,72    3408034   15909265
sda4              0,00         0,00         0,00          2          0
sda2              0,00         0,14         0,01       3432        268
sda5             13,26        40,92       221,11    1035532    5595000
sda3              6,85        93,20       407,60    2358268   10313992
sda1              0,01         0,20         0,00       5168          5
sda6              0,00         0,13         0,00       3352          0
```

Вывод с опцией -x предоставит больше информации:

```
~ $ iostat -d -t -p sda -x

Linux 4.19.0-041900-generic (hostname)  02.11.2018   _x86_64_   (4 CPU)

02.11.2018 18:36:59
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm
%wrqm r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
sda              1,97   18,59    132,33    626,88     0,04     2,63   2,03
12,42   82,74   39,75   0,90    67,34    33,73   3,81   7,83
```

```
sda4              0,00    0,00      0,00      0,00     0,00     0,00   0,00
0,00   81,50    0,00   0,00     1,00     0,00  82,00   0,00
sda2              0,00    0,00      0,13      0,01     0,00     0,00   0,00
26,67   31,29  134,36   0,00    35,38    24,36  30,04   0,01
sda5              1,14   12,08     40,21    220,17     0,03     0,60   2,64
4,74   86,93   51,67   0,72    35,34    18,22   3,85   5,09
sda3              0,81    6,04     91,57    406,69     0,01     2,03   1,17
25,18   77,98   15,17   0,16   113,01    67,34   2,85   1,95
sda1              0,01    0,00      0,20      0,00     0,00     0,00   3,42
0,00   27,77    0,00   0,00    36,65     1,67  22,25   0,01
sda6              0,00    0,00      0,13      0,00     0,00     0,00   0,00
0,00   33,03    0,00   0,00    38,53     0,00  30,85   0,01
```

В таблице ниже есть расшифровка каждого параметра указанного в название столбца:

Колонка	Описание
rrqm/s	обобщенное количество запросов на чтение в секунду;
wrqm/s	обобщенное количество запросов на запись в секунду;
r/s	количество запросов на чтение в секунду;
w/s	количество запросов на запись в секунду;
rMB/s	количество МБ при чтении с диска в секунду;
wMB/s	количество МБ при записи на диск в секунду;
avgrq-sz	средний размер (в секторах) запросов к диску;
avgqu-sz	средний размер очереди запросов к диску;
await	среднее время (милисекунды) на обработку запросов к диску (включает в себя время, потраченное в очереди на обработку и время на обработку запроса);
r_await	среднее время (милисекунды) на обработку запросов чтения к диску (включает в себя время, потраченное в очереди на обработку и время на обработку запроса);
w_await	среднее время (милисекунды) на обработку запросов запи ик диску (включает в себя время, потраченное в очереди на обработку и время на обработку запроса);
svctm	среднее время (милисекунды) I/O запросов (не обращайте внимания на неё – будет удалена в ближайшем будущем);
%util	% CPU, затраченный на передачу I/O запросов к диску (“пропускная способность” диска);

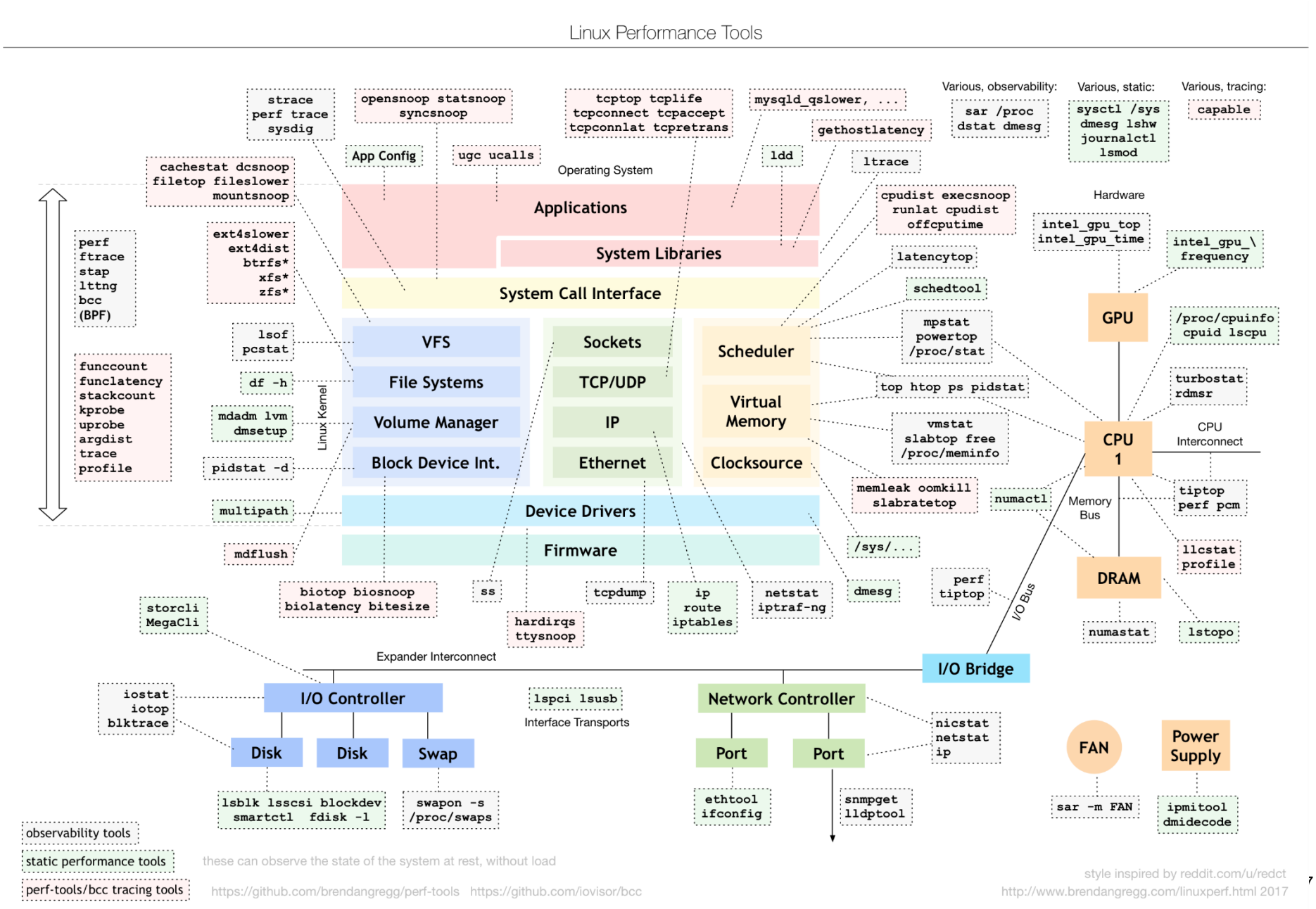
В обычном виде столбцов меньше, и выражены они других единицах:

Колонка	Описание
tps	transfers per second – активность I/O операций в секунду, несколько логических запросов могут быть объедены в один;
Blk_read/s	количество запросов на чтение в секунду, выраженное в блоках (512 bytes);
Blk_wrtn/s	количество запросов на запись в секунду, выраженное в блоках (512 bytes);
Blk_read	общее количество прочитанных блоков;

Blk_wrtn	общее количество записанных блоков;

Дополнительно можно почитать в руководстве к данной утилите (man iostat), или , например, по ссылке.

И картинка для тех, кому хочется еще больше углубиться в измерение производительность системы и ее мониторингу, а по ссылке можно найти еще больше информации:


Linux Performance Tools
Various, observability:
Various, static:
Various, tracing:
sar /proc dstat dmesg
sysctl /sys dmesg lshw journalctl lsmod
capable
strace perf trace sysdig
opensnoop statsnoop syncsnoop
tcptop tcplife tcpconnect tcpaccept tcpconnlat tcpretrans
mysqld_qslower, ...
gethostlatency
App Config
ugc ucalls
ldd
ltrace
Operating System
cachestat dcsnoop filetop fileslower mountsnoop
Applications
System Libraries
System Call Interface
cpudist execsnoop runlat cpudist offcputime
Hardware
intel_gpu_top intel_gpu_time
intel_gpu_\ frequency
perf ftrace stap lttng bcc (BPF)
ext4slower ext4dist btrfs* xfs* zfs*
latencytop
schedtool
GPU
/proc/cpuinfo cpuid lscpu
lsof pcstat
VFS
Sockets
Scheduler
mpstat powertop /proc/stat
df -h
File Systems
TCP/UDP
top htop ps pidstat
turbostat rdmsr
funccount funclatency stackcount kprobe uprobe argdist trace profile
mdadm lvm dmsetup
Volume Manager
IP
Virtual Memory
vmstat slabtop free /proc/meminfo
CPU 1
CPU Interconnect
Linux Kernel
pidstat -d
Block Device Int.
Ethernet
Clocksource
memleak oomkill slabratetop
numactl
tiptop perf pcm
multipath
Device Drivers
Memory Bus
mdflush
Firmware
/sys/...
llcstat profile
DRAM
biotop biosnoop biolatency bitesize
ss
tcpdump
ip route iptables
netstat iptraf-ng
dmesg
perf tiptop
I/O Bus
numastat
lstopo
storcli MegaCli
hardirqs ttysnoop
Expander Interconnect
I/O Bridge
iostat iotop blktrace
I/O Controller
lspci lsusb
Interface Transports
Network Controller
nicstat netstat ip
Disk
Disk
Swap
Port
Port
FAN
Power Supply
lsblk lsscsi blockdev smartctl fdisk -l
swapon -s /proc/swaps
ethtool ifconfig
snmpget lldptool
sar -m FAN
ipmitool dmidecode
observability tools
static performance tools
perf-tools/bcc tracing tools
these can observe the state of the system at rest, without load
https://github.com/brendangregg/perf-tools https://github.com/iovisor/bcc
style inspired by reddit.com/u/redct
http://www.brendangregg.com/linuxperf.html 2017

14. Привилегии суперпользователя (su, sudo)

Немного истории

В большинстве систем Unix и похожих/наследованных от них (в том числе и Linux) существует учетная запись либо группа, которая дает пользователю полный контроль над системой - запись корневого пользователя, или суперпользователя, он же root. Аналог для Windows-систем - локальный администратор. Этот пользователь имеет права на выполнение любых действий, удаление любых файлов и изменение любых параметров. Если доступ к данной учетной записи дать злоумышленнику, либо неопытному пользователю - велика вероятность потерять данные, доступ к системе, или систему целиком. По этой причине непосредственно под данной учетной записью не рекомендуется работать на постоянной основе.

Команда su

Для работы с системой, используя привилегии суперпользователя (подразумевая, что в систему вы зашли под обычным пользователем), исторически сперва использовалась команда `su`, что означает "substitute user" - заменить пользователя.

Команда

```
su <логин пользователя>
```

запускает командную оболочку под пользователем, логин которого указан через пробел. При этом нужно будет по приглашению ввести пароль - ввести пароль именно того пользователя, логин которого указан через пробел:

```
inmd@ub-srv-1804-base:/$ su anton
Password:
anton@ub-srv-1804-base:/$
anton@ub-srv-1804-base:/$ exit
exit
inmd@ub-srv-1804-base:/$
```

Если перед именем пользователя через пробел поставить дефис - будет запущен login shell для этого пользователя, то есть сработают его логин-скрипты, и применятся параметры командной оболочки, специфичные именно для этого пользователя, если они для него указаны (например, домашняя директория, та же раскраска параметров):

```
inmd@ub-srv-1804-base:/$ su - john
Password:
No directory, logging in with HOME=/
john@ub-srv-1804-base:/$
```

Здесь видно, что для пользователя john не определена домашняя директория.

Если после su ничего не указано - будет запущена командная оболочка суперпользователя, что для командной оболочки Bash видно по изменению приглашения с $ на #. Соответственно, будет запрошен пароль суперпользователя:

```
inmd@ub-srv-1804-base:/$ su
Password:
root@ub-srv-1804-base:/#
root@ub-srv-1804-base:/# exit
```

```
exit
inmd@ub-srv-1804-base:/$
```

Обычно для администрирования раньше все использовали команду `su`, либо заходили в систему под пользователем root. Затем, после выполнения всех необходимых операций в системе - нужно было вернуться обратно в учетную запись обычного пользователя. В принципе, такая схема работает неплохо, однако у неё есть много существенных недостатков, в частности, невозможно никак (точнее, очень сложно) ограничивать административные привилегии только определённым кругом задач.

Поэтому в современных дистрибутивах Linux вместо root аккаунта для администрирования используется утилита `sudo`.

В Ubuntu, к примеру, по умолчанию root аккаунт вообще отключён. Если попытаться на свежей, немодифицированной системе, выполнить команду `su` без параметров, можно получить вот такое:

```
inmd@ub-srv-1804-base:/$ su
Password:
su: Authentication failure
```

Команда sudo

Команда sudo так же исполняет команды от имени другого пользователя, но так же применяет набор ограничений - какие пользователи могут выполнять какой набор команд от имени каких пользователей (обычно эти ограничения определяются в конфигурационном файле /etc/sudoers, который можно редактировать напрямую, но лучше пользоваться утилитой visudo).

В отличие от su, sudo спрашивает пароль только того пользователя, который выполняет команду sudo, делегируя выполнение команд другим пользователям, не передавая им пароли и тем самым не увеличивая риск компрометации сервера.

Давайте поясним на примере.

```
inmd@ub-srv-1804-base:/$ cp /etc/hosts /etc/hosts.bak
cp: cannot create regular file '/etc/hosts.bak': Permission denied
inmd@ub-srv-1804-base:/$ sudo -u john cp /etc/hosts /etc/hosts.bak
[sudo] password for inmd:
cp: cannot create regular file '/etc/hosts.bak': Permission denied
inmd@ub-srv-1804-base:/$ sudo cp /etc/hosts /etc/hosts.bak
```

В первой команде мы попробовали скопировать файл в системном каталоге /etc в тот же каталог, но под другим именем. Поскольку доступ на создание и изменение файлов в каталоге /etc есть по умолчанию только у суперпользователя, то система не дает нам это сделать.

Попробуем использовать параметр -u, по аналогии с дефисом в утилите su, выполнив эту команду от пользователя john, которые тоже не обладает правом выполнять команды от имени суперпользователя - и тоже получаем отказ.

После этого выполняем ту же команду, но без параметров. Пользователь inmd входит в группу, которая может выполнять команды от имени суперпользователя. Теперь все получилось.

А теперь попробуем пример посложнее.

```
inmd@ub-srv-1804-base:/$ sudo echo "127.0.0.1 nginx-test" >> /etc/hosts
-bash: /etc/hosts: Permission denied
```

Почему так происходит?

Потому, что команда sudo действует в примере выше только на команду echo. А оператор перенаправления уже выполняется интерпретатором bash, который НЕ выполняется с повышением привилегий. Почему так происходит - попытайтесь разобраться сами.

Как исправить команду, чтобы она выполнилась с нужным результатом? Вызвать команду sudo с сложным аргументом в виде вызова bash, и уже внутри bash - аргументом передать команду выше:

```
inmd@ub-srv-1804-base:/$ sudo bash -c "echo '127.0.0.1 nginx-tes
t' >> /etc/hosts"
```

Часто доступ по su и sudo в Linux описывают по аналогии с фильмами про Супермена. Обычная учетная запись, под которой работает администратор - это как Кларк Кент, обычный человек. Кларк Кент становится Суперменом только по острой необходимости, для спасения людей. Тем не менее, находясь в обличье Кларка Кента, Супермен может использовать свои суперспособности, не привлекая к себе внимание. Это аналогия использования утилиты sudo.

Итого

Итак, суммируем и закрепляем знания:

- `su` и `sudo` - это не логины и не учетные записи. Это утилиты, которые используются для административных привилегий.
- `su user` запускает командную оболочку от имени пользователя `user`. Если вы не работаете под root, то система спросит у вас пароль от учетной записи `user`.
- `su` (без аргументов) запускает командную оболочку для пользователя `root` (после того, как спросит и подтвердит пароль суперпользователя).
- `sudo` (без аргументов) спрашивает ВАШ пароль (подразумевая, что у вас есть нужные права и привилегии) и выполняет команду с привилегиями суперпользователя (`sudo reboot` например - спросит ваш пароль и перезагрузит компьютер).
- `sudo -u user` работает так же, как `su user`, накладывая дополнительные ограничения из `/etc/sudoers`
- `sudo su user` выполняет утилиту `su` с привилегиями суперпользователя. Таким образом система НЕ запрашивает пароль пользователя `user`. Однако, система запросит ВАШ пароль, чтобы подтвердить ваши права на sudo. После этого будет запущена оболочка от имени пользователя `user`. Это нужно, чтобы администратор мог выполнять команды ОТ ИМЕНИ других пользователей, НЕ зная их паролей.

Дополнительные материалы

https://en.wikipedia.org/wiki/Sudo (https://en.wikipedia.org/wiki/Sudo)

https://www.sudo.ws/ (https://www.sudo.ws/)

Статья на русском про su, sudo, /etc/sudoers
(https://help.ubuntu.ru/wiki/%D1%81%D1%83%D0%BF%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_%D0%B2_ubuntu)

man su

man sudo