Основы DevOps

КОДЕБАЙ
АКАДЕМИЯ

РАБОТА С ОПЕРАЦИОННОЙ СИСТЕМОЙ LINUX

ОГЛАВЛЕНИЕ

ОСНОВЫ LINUX

ВВЕДЕНИЕ В LINUX

Linux (GNU/Linux) — семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты.

Как и ядро Linux, системы на его основе, как правило, создаются и распространяются в соответствии с моделью разработки свободного и открытого программного обеспечения.

Linux-системы распространяются в основном бесплатно в виде различных дистрибутивов — в форме, готовой для установки и удобной для сопровождения и обновлений, — и имеющих свой набор системных и прикладных компонентов, как свободных, так и проприетарных.

ИСТОРИЯ LINUX

В 1991 году студент *Линус Торвальдс* увлёкся идеей написать совместимое с UNIX ядро операционной системы для своего персонального компьютера с процессором архитектуры Intel 80386. Прототипом для ядра стала операционная система MINIX: совместимая с UNIX операционная система для персональных компьютеров, которая загружалась с дискет и умещалась в очень ограниченной в те времена памяти персонального компьютера. MINIX был создан в качестве учебной операционной системы, демонстрирующей архитектуру и возможности UNIX. Именно полноценное ядро для своего ПК и хотел сделать Линус Торвальдс.

GNU — проект по разработке свободного программного обеспечения, является результатом сотрудничества множества отдельных проектов. Проект был запущен известным программистом Ричардом Столлманом 27 сентября 1983 года в Массачусетском технологическом институте.

ХРОНОЛОГИЯ

1991: 25 августа анонсировано ядро Linux.

1992: Ядро Linux перелицензируется под лицензией GNU GPL.

1993: Более 100 разработчиков работают над ядром Linux. С их помощью ядро адаптируется к среде GNU. Выпуск дистрибутивов Slackware, Debian.

1994: Выпуск версии 1.0 Linux.

1998: Многие крупные компании, такие как IBM, Compaq и Oracle, объявляют о своей поддержке Linux.

1999: Группа разработчиков начинает работу над графической средой GNOME

2001: Выпущена версия 2.4 ядра Linux.

2003: Выпущена версия 2.6 ядра Linux.

2006: Oracle выпускает собственный дистрибутив Red Hat Enterprise Linux.

2011: Выпущена версия 3.0 ядра Linux.

2013: Android на базе Linux от Google претендует на 75% доли рынка смартфонов по количеству поставленных телефонов.

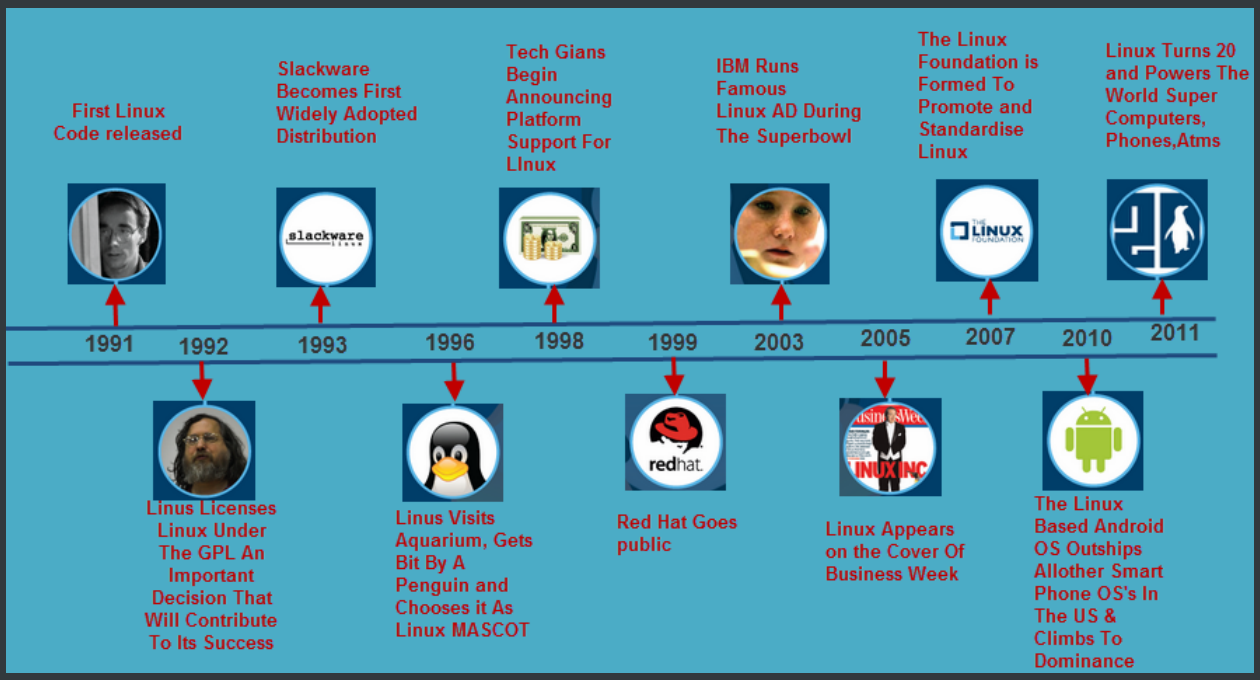
2014: Ubuntu насчитывает 22 000 000 пользователей.

2015: Выпущена версия 4.0 ядра Linux.

2017: Все суперкомпьютеры из списка Top500 работают под управлением Linux.

2019: Выпущена версия 5.0 ядра Linux.

2022 Выпущена версия 6.0 ядра Linux.



АРХИТЕКТУРА

Наиболее общим подходом к структуризации операционной системы является разделение всех её модулей на две группы:

- ядро – набор библиотек и модулей, выполняющих основные функции операционной системы, решающих внутрисистемные задачи организации вычислительного процесса, такие как переключение контекста, управление памятью, обработка прерываний, работа с внешними устройствами и т.п.
- компоненты, реализующие дополнительные функции операционной системы – всевозможные служебные программы, или утилиты.

Основные задачи ядра:

- предоставление доступа к железу через API, в виде функций, которые называются системными вызовами (syscall).
- распределение ресурсов (scheduling) между задачами.

Виды ядер:

- Монолитное (Linux, Cisco IOS)
- Микроядро (GNU Mach, QNX)
- Гибрид (Windows)
- Почти монолит (Mac, Xen)

Монолитные ядра дают лучший доступ к оборудованию и реализуют лучшую многозадачность.

Linux-системы реализуются на модульных принципах, стандартах и соглашениях, заложенных в Unix в течение 1970-х и 1980-х годов. Такая система использует монолитное ядро, которое управляет процессами, сетевыми функциями, периферией и доступом к файловой системе. Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы.

Отдельные программы, взаимодействуя с ядром, обеспечивают функции системы более высокого уровня. Например, пользовательские компоненты GNU являются важной частью большинства Линукс-систем, включающей в себя наиболее распространённые реализации библиотеки языка Си, популярных оболочек операционной системы, и многих других общих инструментов Unix, которые выполняют основные задачи операционной системы.

Ядро Linux позволяет драйверам оборудования, файловым системам, и некоторым другим компонентам быть скомпилированными отдельно - как модули, а не как часть самого ядра. Таким образом, вы можете обновлять драйвера не пересобирая ядро, а также динамически расширять его функциональность. Также вы можете включить в ядре только самое необходимое, а все остальное подключать с помощью модулей.

Утилиты для работы с модулями:

`lsmod` - посмотреть загруженные модули

`modinfo` - информация о модуле

`insmod` - загрузить модуль

`rmmod` - удалить модуль

Конфигурация модулей:

```
/sys/module
/etc/modprobe.d
/etc/modules-load.d
```

МНОГОЗАДАЧНОСТЬ

Свойство операционной системы обеспечивать возможность параллельной или псевдопараллельной обработки нескольких задач.

ВИДЫ МНОГОЗАДАЧНОСТИ:

Кооперативная - переключение контекста процессов инициирует приложение.

Плюсом является отсутствие необходимости синхронизации. Минусом – ненадёжность.

Вытесняющая - переключение контекста инициирует ОС по прерыванию таймера

Плюсом является надёжность, Минусом - необходимость в эффективных механизмах синхронизации.

Система Linux реализует вытесняющую многозадачность.

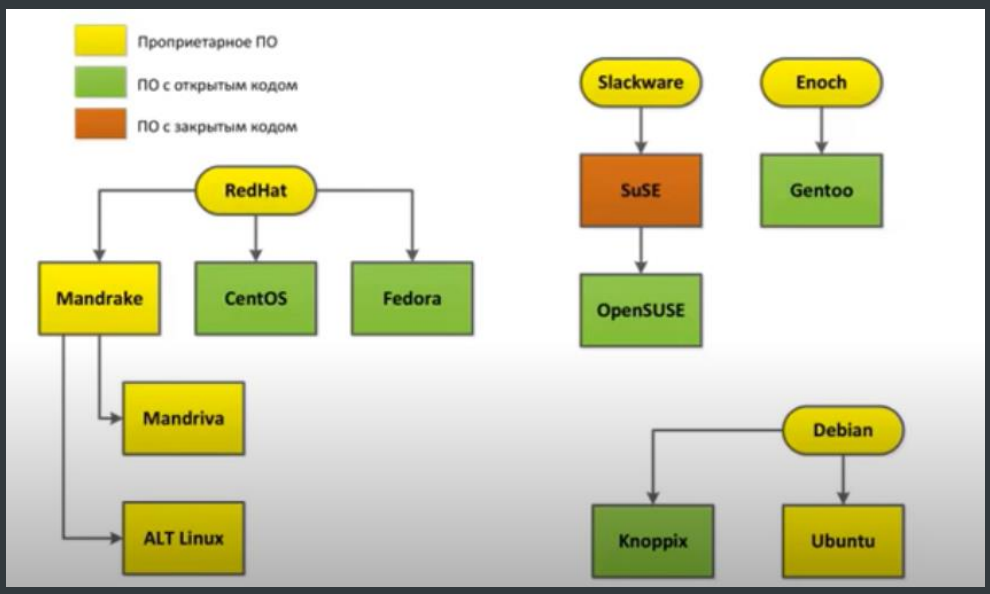
ДИСТРИБУТИВЫ LINUX

Дистрибутив ОС – набор программного обеспечения, готовый для конечной установки на пользовательское оборудование.

В СОСТАВ ДИСТРИБУТИВА ВХОДИТ:

- Ядро
- Файловая система
- Интерпретатор команд или оболочка – программа, организующая взаимодействие пользователя с ОС
- Служебные утилиты
- Прикладные программы

СЕМЕЙСТВА ОС НА ЯДРЕ LINUX



РАБОТА С КОМАНДНОЙ СТРОКОЙ

В операционной системе Linux существует два вида интерфейса: графический интерфейс пользователя и интерфейс командной строки.

ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ

Интерфейс командной строки (англ. Command Line Interface, CLI) - управление программами с помощью команд. Команды состоят из букв, цифр, символов, набираются построчно, выполняются после нажатия клавиши Enter. Этот интерфейс встроен в ядро системы и всегда доступен.

Преимущества: небольшой расход ресурсов, гибкость при составлении перечня действий из команд, возможность автоматического выполнения команд, возможность копировать и вставлять команды.

В Linux часто встречаются слова: консоль, терминал, командная строка, командная оболочка, tty, эмулятор терминала. Все они относятся к терминалу, но означают немного разные вещи.

Под *терминалом* принято понимать окружение, где можно вводить команды и получать на них ответ, это может быть физический терминал или терминал на компьютере.

Консоль — это физическое оборудование для управления сервером. Когда к серверу нет доступа из сети, для управления им можно использовать только консоль.

TTY — это файл устройства, который создается ядром и предоставляет доступ к терминалу для программ. Это могут быть файлы /dev/tty для постоянных текстовых терминалов и /dev/pts/* для эмуляторов терминалов. Вы можете выполнить команду или отправить сообщение просто записав данные в этот файл, и также получить результат, прочитав данные из этого файла.

Эмулятор терминала — это графическая программа, которая предоставляет вам доступ к tty или pts терминалу. Например, Gnome Terminal, Konsole, Terminix, Xterm и другие.

Командная оболочка — устройство tty занимается только передачей и приемом данных, но все эти данные должен еще кто-то обрабатывать, выполнять команды, интерпретировать их синтаксис. Командных оболочек достаточно много, это bash, sh, zsh, ksh и другие, но чаще всего применяется Bash.

Командная строка — это место, куда вводятся команды, приглашение терминала для ввода.

КАК ОТКРЫТЬ ТЕРМИНАЛ LINUX?

Система инициализации по умолчанию создает 12 виртуальных терминалов. В одном из них - обычно седьмом, запущена графическая оболочка, но все другие могут быть свободно использованы. Для переключения между терминалами можно использовать сочетания Ctrl+Alt+F1-F12. Для авторизации нужно будет ввести логин и пароль.

Второй способ позволяет открыть виртуальный терминал прямо в графическом интерфейсе с помощью эмулятора терминала. Эмулятор терминала linux работает с файлами в каталоге /dev/pts/* и еще называется псевдотерминалом, потому что не использует tty.

ВЫПОЛНЕНИЕ КОМАНД В ТЕРМИНАЛЕ

Терминал и файлы устройств tty отвечают только за передачу данных. За обработку команд отвечает командная оболочка, которой и передаются полученные данные.

Чтобы выполнить команду достаточно написать ее и нажать Enter.

Командная оболочка Bash поддерживает автодополнение, поэтому можно написать половину команды, нажать TAB и если на такие символы начинается только одна команда, то она будет автоматически дополнена, если же нет, то вы можете нажать два раза TAB, чтобы посмотреть возможные варианты.

Работа в командной строке может выполняться с помощью большого количества команд, многие из них поставляются вместе с системой.

Команды могут выполняться без параметров или с параметрами, которые позволяют указать данные, с которыми будет работать программа. Также есть опции, с помощью которых можно настроить поведение команды. Большинство стандартных утилит Linux придерживаются такого синтаксиса:

```
$ команда опции параметр1 параметр2...
```

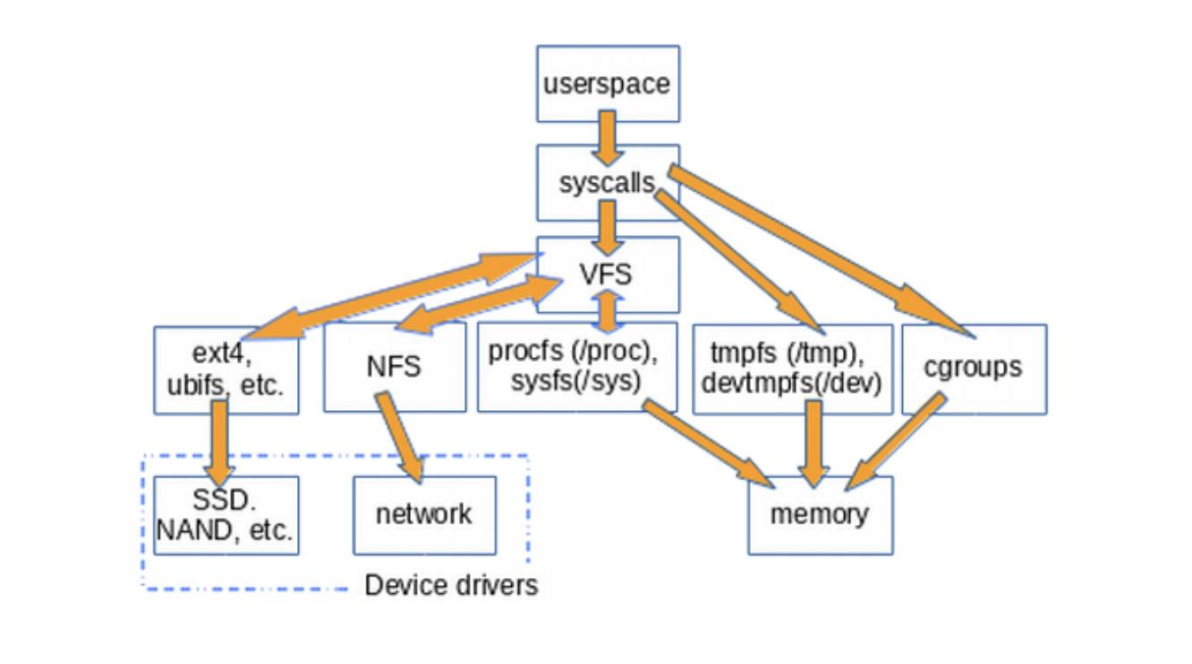
Опции как правило необязательны и записываются в форме черточка и символ или двойная черточка и слово. Например `-o` или `--output`.

ФАЙЛОВАЯ СИСТЕМА

VIRTUAL FILE SYSTEM

В ядре Linux существует специальная подсистема *Virtual File System (VFS)*, позволяющая работать с большим количеством различных файловых систем (ФС):

- с собственными ФС Linux (ext, ext2, ext3, ext4, ext5).
- с ФС Windows (ntfs),
- с ФС Mas OS (hfs),
- с ФС старых версий Unix (sysv),
- с ФС DOS (msdos),
- с ФС CD-, DVD- и flash-устройств
- с сетевой ФС NFS,
- с специальными ФС (ProcS и др.).



ext (англ. extended file system) — журналируемая файловая система, используемая преимущественно в операционных системах с ядром Linux

Журналируемая файловая система — файловая система, в которой осуществляется ведение журнала, хранящего список изменений и, в той или иной степени, помогающего сохранить целостность файловой системы при сбоях.

Основные изменения в *ext4* по сравнению с *ext3:*

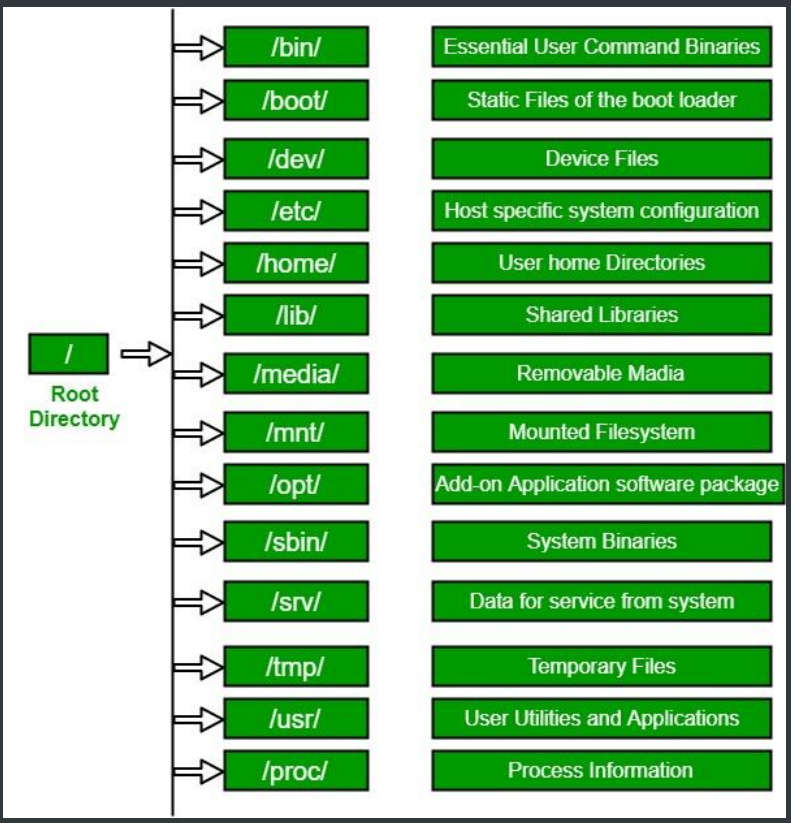
- увеличен максимальный объём одного раздела диска до 1 эксбибайта (260 байт) при размере блока 4 кибибайт;
- увеличен размера одного файла до 16 тебибайт (244 байт);
- введён механизм протяжённой (extent) записи файлов, уменьшающий фрагментацию и повышающий производительность (новая информация добавляется в конец области диска, выделенной заранее по соседству с областью, занятой файлом);
- поднято ограничение на число вложенных каталогов с 32 000 подкаталогов до 65 535 (при этом в некоторых случаях требуется изменить константы ядра).

СТРУКТУРА КАТАЛОГОВ

В Linux есть такое понятие как корневая файловая система. В качестве неё монтируется раздел жесткого диска, на котором установлен Linux. В различные подпапки подключаются другие реальные разделы жесткого диска, например, домашний раздел подключается в папку /home, а загрузочный – в папку /boot. Существуют также виртуальные файловые системы, созданные ядром. Например в папку /proc монтируется псевдо-файловая система procfs, которая позволяет получить доступ к параметрам ядра, а в папку /dev – devfs содержащая устройства, подключённые к компьютеру и тоже в виде файлов.

- /bin – необходимые исполняемые файлы
- /boot – статичные файлы системного загрузчика
- /dev – файлы устройств
- /etc – настройки системы данной машины
- /home – домашние каталоги пользователей
- /lib – необходимые общие библиотеки и модули ядра
- /media – содержит точки монтирования для съёмных носителей
- /mnt – точка монтирования для временно монтируемой файловой системы
- /proc – виртуальный каталог для системной информации
- /root – домашний каталог суперпользователя
- /run – изменяемые данные времени выполнения
- /sbin – необходимые системные исполняемые файлы
- /sys – виртуальный каталог для системной информации
- /tmp – временные файлы
- /usr – вторичная иерархия
- /var – изменяемые данные

- /srv – Данные сервисов, предоставляемых системой
- /opt – дополнительное программное обеспечение



ОСНОВНЫЕ КОМАНДЫ ДЛЯ РАБОТЫ С ФАЙЛОВЫМИ СИСТЕМАМИ

Для управления файловой системой *ext* в Linux используется целый набор команд из пакета *e2progs*:

- *badblocks* – помечает битые блоки жесткого диска, чтобы больше не использовать.
- *e2label* - изменяет метку раздела.
- *fsck* - проверяет файловую систему и исправляет найденные ошибки.
- *mkfs* - создает файловую систему.
- *resize2fs* - изменяет размер раздела с файловой системой.
- *tune2fs* - настраивает параметры файловой системы.

СОЗДАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

Создать файловую систему linux, семейства ext, на устройстве можно с помощью команды mkfs:

```
$ sudo mkfs -t тип устройство
```

Дополнительные параметры:

-c - проверить устройство на наличие битых секторов
-b - размер блока файловой системы
-j - использовать журналирование для ext3
-L - задать метку раздела
-v - показать подробную информацию о процессе работы
-V - версия программы

Создадим файловую систему:

```
sudo mkfs -t ext4 -L root /dev/sda6
Creating filesystem with 7847168 4k blocks and 1962240 inodes
Filesystem UUID: 3ba3f7f5-1fb2-47af-b22c-fa4ca227744a
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632,
2654208,
4096000
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

ИЗМЕНЕНИЕ РАЗМЕРА ФАЙЛОВОЙ СИСТЕМЫ

Для запуска утилиты требуется передать ей два параметра:

```
$ resize2fs [опции] устройство размер
```

Доступные опции:

-M уменьшить файловую систему до минимального размера
-f - принудительное изменение, не смотря на потерю данных
-F - очистить буфер файловой системы

Размер передается целым числом с указанием единиц измерения, например, 100M или 1G.

Уменьшим размер раздела до 400 Мегабайт:

```
sudo resize2fs /dev/sda6 400M
Resizing the filesystem on /dev/sda7 to 102400 (4k) blocks.
The filesystem on /dev/sda7 is now 102400 blocks long
```

ПРОВЕРКА ФАЙЛОВОЙ СИСТЕМЫ

При неправильном отключении носителей или неожиданном отключении питания, файловая система Linux может быть повреждена. Для выполнения проверки используется утилита fsck:

```
$ fsck [опции] устройство
```

Доступные опции:

-p - автоматическое восстановление
-n - только проверка, без восстановления
-y - ответить да на все запросы программы
-c - проверить на битые сектора (аналог badblocks
-f - принудительная проверка, даже если раздел помечен как чистый
-j - внешний журнал файловой системы

Проверим диск /dev/sda6. Диск должен быть не примонтирован:

```
sudo fsck -a /dev/sda6
root: clean, 11/32704 files, 37901/102400 blocks
```

НАСТРОЙКА ФАЙЛОВОЙ СИСТЕМЫ

Такие параметры файловой системы, как размер блока данных, иноды или зарезервированное место под данные пользователя root могут быть настроены. Для этого существует утилита tune2fs:

```
$ tune2fs опции устройство
```

Доступные опции:

-j - создать файл журнала. Позволяет превратить файловую систему ext2 в ext3.
-J - настроить параметры журнала
-l - получить содержимое суперблока
-L - изменить метку раздела
-m - изменить процент дискового пространства, зарезервированного для суперпользователя
-M - изменить последнюю папку монтирования
-U - задать UUID файловой системы
-C - изменить значение счетчика монтирования
-T - изменить последнюю дату проверки файловой системы
-c - изменить периодичность проверок файловой системы с помощью fsck
-O - изменить опции файловой системы.

Изменим счетчик количества монтирований:

```
tune2fs -C 0 /dev/sda6
Setting current mount count to 0
```

ВСЕ В LINUX – ФАЙЛЫ

Основная концепция Linux - всё есть файл. Эта концепция была перенята от Unix, чтобы предоставить простой доступ ко всем возможностям операционной системы, не разрабатывая специальных интерфейсов.

Ссылки удобны, когда мы имеем различные версии приложения, а хотим одной и той же командой запускать какую-то конкретную версию.

Типы файлов в Linux

Типы файлов		Назначение
Обычные файлы	–	Хранение символьных и двоичных данных
Каталоги	d	Организация доступа к файлам
Символьные ссылки	l	Предоставление доступа к файлам, расположенных на любых носителях
Блочные устройства	b	Предоставление интерфейса для взаимодействия с аппаратным обеспечением компьютера
Символьные устройства	c	
Каналы	p	Организация взаимодействия процессов в операционной системе
Сокеты	s	

http://younglinux.info

Блочные или символьные устройства - вид файла устройства, обеспечивающий интерфейс к устройству, реальному или виртуальному, с возможностью блочного или посимвольного обмена информацией.

Каналы – виртуальные файлы, которые помогают приложениям внутри ОС обмениваться данными.

Сокеты – файлы для передачи данных через сетевые устройства.

НЕКОТОРЫЕ КОМАНДЫ ДЛЯ РАБОТЫ С ФАЙЛАМИ И КАТАЛОГАМИ

- ✓ Команда `ls` выводит на экран содержимое заданного каталога.

```
$ ls [флаг] [каталог]
```

Если каталог не указан, то отображается содержимое текущего каталога.

Команда имеет множество ключей, которые определяют:

- в каком формате выводить информацию
- какую именно информацию об этих объектах нужно вывести
- как упорядочить выводимую информацию

- ✓ Команда `cd` изменяет текущий каталог (сокр. от change directory)

```
$ cd [каталог]
```

Примеры:

`$ cd /usr` – перейти в каталог `/usr`

`$ cd ..` – перейти в родительский каталог (находящийся в иерархии каталогов на один уровень выше)

`$ cd ~` – перейти в рабочий каталог пользователя

- ✓ Вывести на экран текущий каталог можно с помощью команды `pwd` (сокр. от print working directory). Команда не имеет параметров.

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

- ✓ Для создания одного или нескольких каталогов используется команда `mkdir`.

```
$ mkdir [ключи] имя_каталога …
$ mkdir dir_a dir_b dir_c
```

Используя ключ `-m` можно при создании каталога сразу задать ему права доступа.

```
$ mkdir -m 777 dir1
```

Если в команде mkdir задать ключ `-p`, то можно сразу создать цепочку вложенных друг в друга подкаталогов.

```
$ mkdir -p dir1/ dir2/ dir3
```

Для удаления пустых каталогов предназначена команда `rmdir`.

```
$ rmdir [ключи] каталог …
$ rmdir dir1 dir2
```

Если задать в команде ключ `-p`, то она позволяет удалять вложенные друг в друга пустые подкаталоги

```
$ rmdir -p dir1/ dir2/ dir3
```

- ✓ Очень полезной командой является команда `cat`.

```
$ cat [ключи] имя_файла …
```

Команду `cat` можно использовать для:

- ✓ Вывода файла на экран

```
$ cat file
```

- ✓ Вывод файла с клавиатуры. В конце ввода нужно нажать Ctrl+D

```
$ cat > file
```

- ✓ Объединения нескольких файлов в один

```
$ cat file1 file2 file3 > file123
```

- ✓ Для создания ссылок на файлы в Linux предназначена команда `ln`. Она может создавать как жесткие, так и символические ссылки.

```
$ ln [ключи] имя_файла [имя_ссылки]
$ ln [ключи] имя_файла … каталог
```

Первый вариант команды создает (по умолчанию – в текущем каталоге) ссылку с именем ссылка на задний файл с именем `имя_файла`.

Второй вариант команды создает в заданном каталоге ссылки на указанные файлы, причем имена файлов ссылок будут совпадать с именами исходных файлов.

По умолчанию создается жесткая ссылка. Если требуется создать символическую ссылку, то в команде нужно указать ключ `-s`

ПРОЦЕССЫ В LINUX

ПРОЦЕССЫ

Процесс — это программа, которая выполняется в отдельном виртуальном адресном пространстве.

Дочерний процесс (child) — процессы могут запускать другие процессы для выполнения параллельных задач или других целей такие процессы называются дочерними. Для них выделяется отдельная область в памяти.

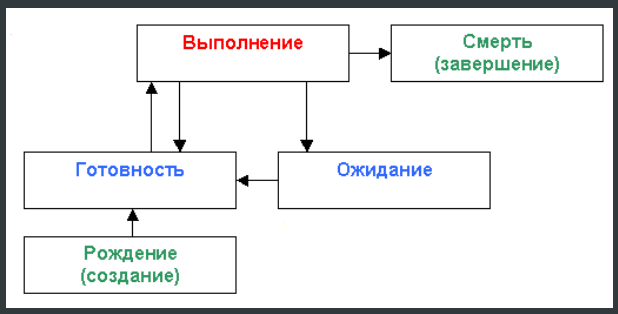
Поток отличается от процесса тем, что использует ту же память, данные и дескрипторы файлов, что и процесс, в котором он был создан.

Linux поддерживает параллельное (или квазипараллельного при наличии только одного процессора) выполнение процессов пользователя. Каждый процесс выполняется в собственном виртуальном адресном пространстве, т.е. процессы защищены друг от друга и крах одного процесса никак не повлияет на другие выполняющиеся процессы и на всю систему в целом. Один процесс не может прочитать что-либо из памяти (или записать в нее) другого процесса без "разрешения" на то другого процесса.

СОСТОЯНИЯ ПРОЦЕССА

Для описания состояний процессов использовать модель пяти состояний. В этой модели процесс может находится в 5 состояниях:

- рождение,
- готовности,
- выполнения,
- ожидания,
- смерть.



Ядро предоставляет системные вызовы для создания и управления процессами. Любая программа может начать выполняться только если другой процесс ее запустит или произойдет какое-то прерывание (например, прерывание внешнего устройства).

Рождение — это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса. Для создания процессов используются два системных вызова: fork() и exec. fork() создает новое адресное пространство, которое полностью идентично адресному пространству основного процесса. После выполнения этого

системного вызова мы получаем два абсолютно одинаковых процесса - основной и порожденный. Функция fork() возвращает 0 в порожденном процессе и PID (Process ID - идентификатор порожденного процесса) - в основном. PID (Process ID) — это целое число.

Родительский процесс может дожидаться окончания выполнения всех своих процессов-потомков с помощью системного вызова wait.

Готовность — это тоже пассивное состояние, процесс заблокирован по внешним, независящим от процесса, причинам. Из состояния готовности процесс может перейти только в состояние выполнения. В состоянии выполнения может находится только один процесс на один процессор. Если у вас n-процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов.

Выполнение — это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

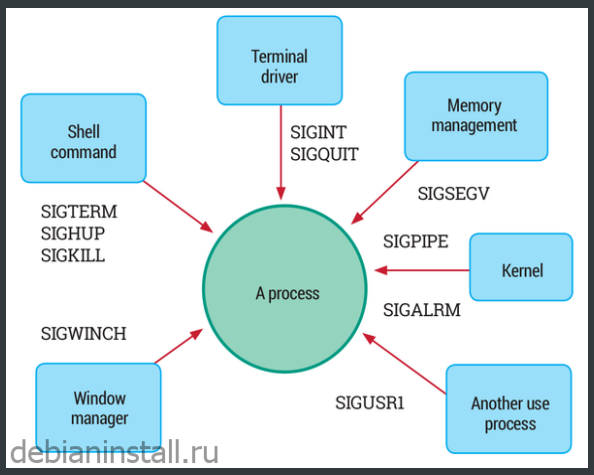
Из состояния выполнения процесс может перейти либо в состояние ожидания или состояние готовности. Процесс может оказаться в состоянии ожидания, если ему нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла.

В состояние готовности процесс может перейти, если во время его выполнения, квант времени выполнения "вышел". Другими словами, в операционной системе есть специальная программа - планировщик, которая следит за тем, чтобы все процессы выполнялись отведенное им время.

Ожидание — это пассивное состояние, во время которого процесс заблокирован, он не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

Смерть — это состояние, когда самого процесса уже нет, но его "место", то есть структура, осталась в списке процессов. Такие процессы называются *зобми*.

Сигнал — это короткое сообщение, посылаемое системой или процессом другому процессу. Обрабатывается асинхронно специальной подпрограммой-обработчиком. Если процесс не обрабатывает сигнал самостоятельно, это делает система. Каждый процесс реагирует на сигналы и может установить собственную реакцию на сигналы.



Для того, чтобы передать сигнал, процессу достаточно задействовать системный вызов kill().

АТРИБУТЫ ПРОЦЕССА

Каждый процесс в ОС Linux характеризуется набором атрибутов, который отличает данный процесс от всех остальных процессов. К таким атрибутам относятся:

Идентификатор процесса (PID). Каждый процесс в системе имеет уникальный идентификатор. Каждый новый запущенный процесс получает номер на единицу больше предыдущего.

Идентификатор родительского процесса (PPID). Данный атрибут процесс получает во время своего запуска и используется для получения статуса родительского процесса.

Реальный и эффективный идентификаторы пользователя (UID,EUID) и группы (GID, EGID). Данные атрибуты процесса говорят о его принадлежности к конкретному пользователю и группе. Реальные идентификаторы совпадают с идентификаторами пользователя, который запустил процесс, и группы, к которой он принадлежит. Эффективные - от чьего имени был запущен процесс. Права доступа процесса к ресурсам ОС Linux эффективными идентификаторами. Если на исполняемом файле программы установлен специальный бит SGID или SUID, то процесс данной программы будет обладать правами доступа владельца исполняемого файла. Все идентификаторы передаются от родительского процесса к дочернему.

Приоритет или динамический приоритет (priority) и относительный или статический (nice) приоритет процесса.

Статический приоритет (nice) лежит в диапазоне от -20 до 19, по умолчанию используется значение 0. Значение –20 соответствует наиболее высокому приоритету, nice-приоритет не изменяется планировщиком, он наследуется от родителя или его указывает

пользователь. Пользователь имеет возможность изменять только статический приоритет процесса. При этом повышать приоритет может только root.

Динамический приоритет (priority) используется планировщиком для планирования выполнения процессов. Динамический приоритет вычисляется исходя из значения параметра nice для данной задачи путем вычисления надбавки или штрафа, в зависимости от интерактивности задачи.

Состояние процесса, в котором находится процесс.

Для просмотра атрибутов процесса можно воспользоваться командой `ps`, задав желаемый формат отображения. Например, чтобы посмотреть все процессы, добавьте опцию -e, а для максимально подробной информации - опцию -F:

```
ps -eF
```

Чтобы посмотреть список процессов в виде дерева, и понимать какой процесс имеет какие дочерние процессы, выполните команду:

```
ps -efH
```

ТИПЫ ПРОЦЕССОВ

В Linux процессы делятся на три типа:

Системные процессы - являются частью ядра и всегда расположены в оперативной памяти. Системные процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются при инициализации ядра системы.

Демоны — это неинтерактивные процессы, которые запускаются обычным образом — путем загрузки в память соответствующих им программ (исполняемых файлов), и выполняются в фоновом режиме. Обычно демоны запускаются при инициализации системы (после инициализации ядра) и обеспечивают работу различных подсистем: системы терминального доступа, системы печати, системы сетевого доступа и т. п. Демоны не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть времени демоны ожидают пока тот или иной процесс запросит определенную услугу.

Прикладные (пользовательские) - все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порожденные в рамках пользовательского сеанса работы. Например, команда `ps` породит соответствующий процесс. Важнейшим прикладным процессом является командный интерпретатор (shell, bash), который обеспечивает вашу работу в LINUX. Он запускается сразу же после регистрации в системе. Прикладные

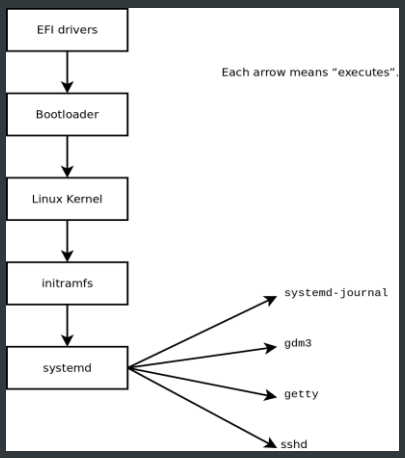
процессы linux могут выполняться как в интерактивном, так и в фоновом режиме, но в любом случае время их жизни (и выполнения) ограничено сеансом работы пользователя. При выходе из системы все прикладные процессы будут уничтожены.

SYSTEMD

Для управления службами нужна *система инициализации* — это основной сервис, который запускает другие сервисы в нужное время, следит за их нормальной работой, пишет логи и, самое главное, предоставляет пользователям интерфейс для управления сервисами. Система инициализации запускается ядром Linux.

Systemd — системный менеджер, демон инициализации других демонов в Linux. Его особенностью является интенсивное распараллеливание запуска служб в процессе загрузки системы, что позволяет существенно ускорить запуск операционной системы.

До появления Systemd все дистрибутивы Linux использовали *init-сценарии* инициализации для управления службами. Каждый сервис запускался последовательно один за другим, без возможности запуска параллельно. После скрипт инициализации брал идентификатор PID процесса сервисов и сохранял его. Этот идентификатор можно было использовать для проверки состояния службы и ее остановки при необходимости.



В Systemd порядок запуска сервисов определяется сложным деревом зависимостей. Когда служба запускается, Systemd собирает ее выходные данные в журнал и отслеживает ее состояние. Например, он может перезапустить службу в случае ее внезапного отказа.

В основе Systemd – *cgroups* – механизм контейнеризации для процессов. Решает задачу принадлежности процесса сервису, так как любой процесс сервиса будет запущен в той-же группе.

Каждая служба в Systemd должна быть описана в специальном *unit-файле*. Файл содержит информацию о том, как запустить службу и как ею управлять. Список типов юнитов:

- *service* – обычный сервис, программа;
- *target* – группа сервисов;

- *automount* — конфигурация точки монтирования, которая должна монтироваться автоматически;
- *device* — файл устройства, который генерируется при запуске системы;
- *mount* — конфигурация точки монтирования;
- *path* — путь к файлу или папке;
- *scope* — внешний процесс;
- *slice* — группа внутренних сервисов Systemd;
- *snapshot* — состояние запущенных сервисов;
- *socket* — позволяет активировать услугу, когда это необходимо.

Все службы Systemd можно разделить на две группы:

- *системные службы*, которыми могут управлять только пользователи с правами суперпользователя. Файлы конфигурации для этих служб находятся в папках `/lib/systemd/system` и `/usr/lib/systemd/system`.
- *пользовательские* — сервисы, которыми может управлять обычный пользователь. Их файлы конфигурации находятся в `/lib/systemd/user` или `/usr/lib/systemd/user`. Также пользовательские сервисы могут располагаться в домашнем каталоге `$USER/.config/systemd/user`.

Пример unit-файла:

```
[Unit]
Description=The Apache HTTP Server
After=network.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)

[Service]
Type=notify # simple или forking
EnvironmentFile=/etc/sysconfig/httpd
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/bin/kill -WINCH ${MAINPID}

[Install]
WantedBy=multi-user.target
```

Systemd имеет инструмент для управления службами в Linux - команда *systemctl*. Эта команда позволяет перезапускать службы, проверять их состояние и анализировать производительность запуска системы. Синтаксис команды:

```
$ systemctl <options> <command> <service_name_1> <service_name_2>…
```

Некоторые доступные команды:

- *list-units* — список всех юнитов, которые загружены в память;
- *start* — запустить службу;
- *stop* — остановить службу;
- *reload* — попросить службу обновить свою конфигурацию из файловой системы;
- *restart* — перезапустить службу
- *try-restart* — перезапустить службу, только если она запущена;
- *kill* — отправить сервису сигнал об уничтожении;
- *is-active* — проверить, запущена ли служба;
- *is-failed* — проверить, не произошел ли сбой службы
- *status* — просмотр состояния сервиса и его логов;
- *show* — просмотреть свойства сервиса;
- *cat* — просмотреть содержимое unit-файла для сервиса;
- *list-dependencies* — просмотр зависимостей для сервиса;
- *enable* — добавить службу в автозагрузку системы;
- *disable* — убрать службу из системного автозапуска;
- *is-enabled* — проверить, запускается ли служба автоматически;
- *daemon-reload* — перезагрузить специфичную для Systemd конфигурацию для всех служб;
- *mask* — сделать unit недоступным;
- *unmask* — восстановить замаскированный unit;
- *link* — добавить unit, который находится не в одной из папок по умолчанию для unit;
- *edit* — редактировать конфигурацию сервиса без изменения файла модуля.

НЕКОТОРЫЕ КОМАНДЫ УПРАВЛЕНИЯ ПРОЦЕССАМИ

Команда	Описание
&	заставит команду работать в фоновом режиме
bg	возобновляет исполнение остановленной задачи в фоновом режиме
fg	переводит фоновую задачу в обычный режим
lsof	выводит информацию об открытых файлах и сетевых сокетах
fuser	выводит список процессов, работающих с заданным файлом, сетевым портом или файловой системой и, в случае необходимости, позволяет автоматически завершать их работу
kill	отправляет системные сигналы определенным процессам
nohup	продолжить выполнение процесса в фоновом режиме после выхода пользователя
ps	выводит сведения о процессах в статическом виде
top	выводит список работающих в системе процессов и информацию о них
free	вывод информации об использовании оперативной памяти

ПОЛЬЗОВАТЕЛИ И ГРУППЫ В LINUX

Linux, как и любая unix-подобная система, является не только многозадачной, но и многопользовательской.

ПОЛЬЗОВАТЕЛИ

Учетная запись пользователя — это необходимая для системы информация о пользователе, хранящаяся в специальных файлах. Информация используется Linux для аутентификации пользователя и назначения ему прав доступа.

`/etc/passwd` - этот файл содержит информацию о пользователях. Запись для каждого пользователя занимает одну строку:

```
root:x:0:0:root:/root:/bin/bash
```

И содержит значения, разделенные двоеточием:

- *имя пользователя* - имя, используемое пользователем на все приглашения типа login при аутентификации в системе.
- *зашифрованный пароль* - обычно хешированный по необратимому алгоритму MD5 пароль пользователя (в современных версиях Linux пароль не хранится здесь).
- *UID* - числовой идентификатор пользователя. Система использует его для распределения прав файлам и процессам.
- *GID* - числовой идентификатор группы. Имена групп расположены в файле /etc/group. Система использует его для распределения прав файлам и процессам.
- *Настоящее имя пользователя* - используется в административных целях, а также командами типа finger (получение информации о пользователе через сеть).
- *Домашний каталог* - полный путь к домашнему каталогу пользователя.
- *Оболочка* - командная оболочка, которую использует пользователь при сеансе. Для нормальной работы она должна быть указана в файле регистрации оболочек /etc/shells.

В Linux, кроме обычных пользователей, существует один суперпользователь с неограниченными правами. Идентификаторы UID и GID такого пользователя всегда 0. Его имя, как правило, root. Для пользователя root права доступа к файлам и процессам не проверяются системой.

ГРУППЫ

Несколько пользователей могут быть объединены в *группу*, которой присваивается имя. При создании регистрационной записи пользователя автоматически создается группа, состоящая из одного члена (этого пользователя) и названная так же, как и этот пользователь.

`/etc/group` - этот файл содержит информацию о группах, к которым принадлежат пользователи:

```
project:x:100:root,bin,daemon
```

И содержит:

- *Имя группы* - имя, используемое для удобства использования.
- *Шифрованный пароль* - используется при смене группы командой newgrp. Пароль для групп может отсутствовать.
- *GID* - числовой идентификатор группы. Система использует его для распределения прав файлам и процессам.
- *Пользователи, включенные в несколько групп* - В этом поле через запятую отображаются те пользователи, у которых по умолчанию (в файле /etc/passwd) назначена другая группа.

Файл `/etc/group` может содержать также некоторые группы, созданные программами, для управления доступом этих программ к общим ресурсам:

- *daemon* - от имени этой группы и пользователя daemon запускаются сервисы, которым необходима возможность записи файлов на диск.
- *sys* - группа открывает доступ к исходникам ядра и файлам include сохраненным в системе
- *sync* - позволяет выполнять команду /bin/sync
- *games* - разрешает играм записывать свои файлы настроек и историю в определенную папку
- *man* - позволяет добавлять страницы в директорию /var/cache/man
- *lp* - позволяет использовать устройства параллельных портов
- *mail* - позволяет записывать данные в почтовые ящики /var/mail/
- *proxy* - используется прокси серверами, нет доступа записи файлов на диск
- *www-data* - с этой группой запускается веб-сервер, она дает доступ на запись /var/www, где находятся файлы веб-документов
- *list* - позволяет просматривать сообщения в /var/mail

- *nogroup* - используется для процессов, которые не могут создавать файлов на жестком диске, а только читать, обычно применяется вместе с пользователем nobody.
- *adm* - позволяет читать логи из директории /var/log
- *tty* - все устройства /dev/vca разрешают доступ на чтение и запись пользователям из этой группы
- *disk* - открывает доступ к жестким дискам /dev/sd* /dev/hd*, можно сказать, что это аналог рут доступа.
- *dialout* - полный доступ к серийному порту
- *cdrom* - доступ к CD-ROM
- *wheel* - позволяет запускать утилиту sudo для повышения привилегий
- *audio* - управление аудиодрайвером
- *src* - полный доступ к исходникам в каталоге /usr/src/
- *shadow* - разрешает чтение файла /etc/shadow
- *utmp* - разрешает запись в файлы /var/log/utmp /var/log/wtmp
- *video* - позволяет работать с видеодрайвером
- *plugdev* - позволяет монтировать внешние устройства USB, CD и т д
- *staff* - разрешает запись в папку /usr/local

Хранение паролей в файлах passwd и group считается ненадежным. В новых версиях Linux применяются теневые файлы паролей - `shadow` и `gshadow`. Права на них назначены таким образом, что даже чтение этих фалов без прав суперпользователя невозможно. Структура файла `shadow`:

```
cisco:$1$oAJZcVg0$EGORy8Mh3swT1RfJeX.UR0:13770:10:99999:7:30:99999:
```

Файл shadow хранит защищенную информацию о пользователях, а также обеспечивает механизмы устаревания паролей и учетных записей:

- имя пользователя
- шифрованный пароль - применяются алгоритмы хеширования, как правило MD5
- число дней последнего изменения пароля, начиная с 1 января 1970 года, последнего изменения пароля
- число дней, перед тем как пароль может быть изменён
- число дней, после которых пароль должен быть изменён
- число дней, за сколько пользователя начнут предупреждать, что пароль устаревает
- число дней, после устаревания пароля для блокировки учётной записи
- дней, отсчитывая с 1 января 1970 года, когда учётная запись будет заблокирована
- зарезервированное поле

Структура файла `gshadow`:

```
root:$1$QydTRu2w$Cm5gk.6w6nmNdUjerh5pu:root:cisco,oem
```

Файл `gshadow` так же накладывает дополнительную функциональность, вкупе с защищенным хранением паролей групп:

- Имя группы - имя, используемое для удобства использования таких программ, как newgrp.
- Шифрованный пароль - используется при смене группы командой newgrp. Пароль для групп может отсутствовать.
- Администратор группы - пользователь, имеющий право изменять пароль с помощью gpasswd.
- Список пользователей - В этом поле через запятую отображаются те пользователи, у которых по умолчанию (в файле /etc/passwd) назначена другая группа.

ВХОД И СМЕНА ПОЛЬЗОВАТЕЛЯ

Команда *login* используется при входе в систему. Она проверяет правильность ввода имени и пароля пользователя, меняет каталог на домашний, выстраивает окружение и запускает командный интерпретатор. Команду login нельзя запускать из командной строки.

Команда *su* позволяет сменить идентификатор пользователя уже в процессе сеанса. Синтаксис:

```
su username
```

где `username` - имя пользователя, которое будет использоваться.

После этого программа запросит пароль. При правильно введенном пароле, su запустит новый командный интерпретатор с правами указанного пользователя и присвоит сеансу его идентификаторы. Если имя пользователя опущено, то используется имя root.

Команда *newgrp* аналогична по своим возможностям su с той разницей, что происходит смена группы. Пользователь должен быть включен в группу, которая указывается в командной строке newgrp. Синтаксис команды:

```
newgrp groupname
```

где `groupname` - имя группы, на которую пользователь меняет текущую.

Команда *sudo* позволяет запускать программы от имени других пользователей, а также от имени суперпользователя. Правила, используемые sudo для принятия решения о предоставлении доступа, находятся в файле `/etc/sudoers`

Синтаксис:

```
$ sudo опции программа параметры
```

Например, для авторизоваться от имени другого пользователя используйте опцию `-i`:

`sudo -i` – авторизация как суперпользователь

`sudo -i -u username` – авторизация как username

Команда sudo отличается от команды su двумя моментами:

- она по умолчанию не переходит в отдельный режим ввода привилегированных команд, а выполняет только одну такую команду. И эта команда указывается в качестве параметра в команде $ sudo команда параметры
- при выполнении команды требуется ввести не пароль пользователя root, а свой собственный пароль. Команда sudo является более гибкой, чем команда su в том смысле, что можно разрешить пользователю выполнять не все, а только некоторые привилегированные команды.

НЕКОТОРЫЕ КОМАНДЫ ДЛЯ РАБОТЫ С ПОЛЬЗОВАТЕЛЯМИ И ГРУППАМИ

Команда	Описание
less /etc/passwd	Вывести список всех пользователей, в том числе системных
users	Вывести только учетные записи пользователей
id username	Проверить существование пользователя и вывести его UID
useradd <имя пользователя> [опции]	Создать пользователя
passwd <имя пользователя>	Задать пароль пользователя
usermod <имя пользователя> [опции]	Редактирование пользователя
usermod -a -G <группы через запятую> <пользователь>	Добавление пользователя в группу
userdel <имя пользователя> [опции]	Удаление пользователя
usermod -L <имя пользователя>	Блокировка пользователя
groupadd <группа> [опции]	Добавление группы
groupmod <группа> [опции]	Редактирование группы
groupdel <группа> [опции]	Удаление группы

ПРАВА ДОСТУПА

МОДЕЛЬ ПРЕДОСТАВЛЕНИЯ ПРАВ ДОСТУПА

Каждый файл имеет три параметра доступа:

- *Чтение* - разрешает получать содержимое файла. Для каталога позволяет получить список файлов и каталогов, расположенных в нем;
- *Запись* - разрешает записывать данные в файл, а также позволяет создавать и изменять файлы и каталоги;
- *Выполнение* – разрешает запускать файл, как программу.

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

- *Владелец* - набор прав для владельца файла, пользователя, который его создал или сейчас установлен его владельцем. Обычно владелец имеет все права, чтение, запись и выполнение.
- *Группа* - любая группа пользователей, существующая в системе и привязанная к файлу. Но это может быть только одна группа и обычно это группа владельца.
- *Остальные* - все пользователи, кроме владельца и пользователей, входящих в группу файла.

Чтобы позволить обычным пользователям выполнять программы от имени суперпользователя без знания его пароля, используются SUID и SGID биты:

- *SUID* - 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* - 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.

С помощью наборов этих полномочий устанавливаются права доступа к файлам. Только пользователь root может работать со всеми файлами независимо от их набора их полномочий.

ПРОСМОТР ПРАВА ДОСТУПА К ФАЙЛАМ

Для отображения подробной информации обо всех флагах, в том числе специальных, нужно использовать команду ls с параметром -l. Все файлы из каталога будут выведены в виде списка, и там будут показаны все атрибуты и биты.

```
ls -l
```

Условные значения флагов прав в выводе команды:

`---` - нет прав, совсем;

`--x` - разрешено только выполнение файла, как программы но не изменение и не чтение;

`-w-` - разрешена только запись и изменение файла;

`-wx` - разрешено изменение и выполнение, но 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.

ИЗМЕНЕНИЕ ПРАВ ДОСТУПА К ФАЙЛАМ

Для изменения прав доступа к файлу можно использовать утилиту chmod. Она позволяет менять все флаги, включая специальные. Синтаксис:

```
$ chmod опции категориядействиеправа файл
```

С помощью опции `-R` вы можно применять изменения ко всем файлам и каталогам рекурсивно.

`категория` указывает для какой группы пользователей нужно применять права:

- `u` - владелец файла;
- `g` - группа файла;
- `o` - другие пользователи.

`действие` может быть одно из двух:

- `+` - добавить;
- `-` - убрать.

`права`:

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

Например, предоставить всем пользователям полный доступ к файлу test:

```
chmod ugo+rwx test
```

Забрать все права у группы и остальных пользователей к файлу test:

```
chmod go-rwx test
```

Установить для файла test SUID:

```
chmod u+s test
```

Можно кроме буквенных выражений прав использовать цифровые:

```
chmod 644 test
```

Права	Цифровое представление
Read (Чтение)	4
Write (Запись)	2
E**x**ecute (Выполнение)	1

ИЗМЕНЕНИЕ ВЛАДЕЛЬЦА ИЛИ ГРУППЫ ФАЙЛА

Для изменения владельца и/или группы файлов используется команды *chown*. В качестве имени владельца/группы берётся первый аргумент, не являющийся опцией. Если задано только имя пользователя (или числовой идентификатор пользователя), то данный пользователь становится владельцем каждого из указанных файлов, а группа этих файлов не изменяется. Если за именем пользователя через двоеточие следует имя группы (или числовой идентификатор группы), без пробелов между ними, то изменяется также и группа файла.

Синтаксис:

```
chown опции пользователь[:группа] файл
```

Опция `-R` выполняет изменения рекурсивно, т.е. всех вложенных в каталог файлов.

Поменять владельца для файла test на 'user' и идентификатор группы в 'developers'.

```
chown user:developers test
```

РАБОТА С ПАКЕТНЫМИ МЕНЕДЖЕРАМИ

Большинство популярных дистрибутивов Linux уже оснащены пакетными менеджерами, способными устанавливать любое программное обеспечение. Будь то внешнее приложение или компоненты ОС. В этом заключается основное различие между пакетным менеджером и инсталлятором. Последний нужен для установки только одной специфической программы, тогда как система управления пакетами — универсальный установщик ПО.

Все пакетные менеджеры Linux имеют свой список репозиториев – серверов с базой пакетов. Во время установки алгоритм менеджера находит необходимый пакет в базе и производит автоматическое скачивание, установку и настройку.

Система управления пакетами — набор программного обеспечения, позволяющего управлять процессом установки, удаления, настройки и обновления различных компонентов программного обеспечения.

КАТЕГОРИИ ПАКЕТНЫХ МЕНЕДЖЕРОВ

Высокоуровневые менеджеры. Применяются для поиска и скачивания пакетов из репозиториев. В процессе работы могут задействовать низкоуровневые менеджеры для инсталляции загруженных программ.

Низкоуровневые менеджеры. Используются для установки локальных пакетов, загруженных вручную пользователем, или высокоуровневым пакетным менеджером.

РАСПРОСТРАНЕННЫЕ ФОРМАТЫ ПАКЕТОВ

DEB (.deb). Самый популярный формат пакетов дистрибутива Debian и его ближайших родственников (Ubuntu, Mint и других).

Представляет собой архивный файл специального формата ar, внутри которого содержатся три других файла:

- debian-binary – текстовый файл, содержащий номер версии формата deb
- control.tag.gz – архив, содержащий различную служебную информацию о пакете: описание пакета, версию пакета, сценарии, которые выполняются при установке и удалении пакета и др.

- data.tar – архив, содержащий устанавливаемые в системе файлы: выполняемые файлы, файлы настроек, страницы man/info и другие файлы. Данный файл чаще всего хранится в сжатом виде и в этом случае имеет имя data.tar.gz, data.tar.bz2 и т.п.

Рассмотренная структура файла соответствует версиям формата deb, начиная с версии 0,93. Она может меняться.

RPM (.rpm). Разработан компанией Red Hat и внедрен в дистрибутив RHEL. Также применяется в таких системах, как Fedora и CentOS.

Пакет представляет из себя архив определенного формата, в нем содержится:

- Метаинформация
- Скриптлеты
- Файлы

ПОПУЛЯРНЫЕ ПАКЕТНЫЕ МЕНЕДЖЕРЫ

DPKG (Debian Package) – система управления пакетами в Debian и дистрибутивах на его основе, например Ubuntu.

Утилита DPKG появилась в дистрибутиве Debian в 1995 году. Низкоуровневый пакетный менеджер создан только для работы с локальными DEB пакетами и не может самостоятельно разрешать зависимости, а также скачивать пакеты из репозиториев.

Особенности:

- Поддерживает добавление архитектур из других дистрибутивов Linux.
- DPKG выполняет работу только с локальными пакетами.
- Под архитектуру DEB выпущено более 55000 пакетов.

РАСПРОСТРАНЕНИЕ	ИНСТРУМЕНТ НИЗКОГО УРОВНЯ	ИНСТРУМЕНТ ВЫСОКОГО УРОВНЯ
Debian и производные	dpkg	apt-get / aptitude
CentOS	rpm	yum
openSUSE	rpm	zypper

APT (Advanced Packaging Tool) – консольная утилита, выполняющая роль «поисковика» и загрузчика пакетов из репозиториев. Установка скачанных пакетов производится утилитой DPKG. Благодаря эффективному разрешению зависимостей, пакетный менеджер APT используется по умолчанию в дистрибутивах с архитектурой Debian и поддерживает систему в актуальном состоянии.

Список репозиториев хранится в файле `/etc/apt/sources.list` и может быть изменён пользователем в любой момент для установки или обновления программы, не входящей в базу дистрибутива. Установка скачанных пакетов производится утилитой DPKG.

Изначально APT разрабатывался только для работы с пакетами DEB, использующихся в Debian и родственных ОС (Ubuntu, Linux Mint). Позже в него была добавлена поддержка rpm-файлов.

RPM (Red Hat Package Manager) – формат пакетов и низкоуровневый пакетный менеджер систем RED HAT (RHEL, CentOS, Fedora и др.) Как и DPKG, способен работать только с локальными файлами.

Пакетный менеджер выпущен в 1997 году. Он работает с пакетами RPM. В отличие от DEB, пакеты RPM архивируются утилитой cpio, сжимающий пакет алгоритмом gzip.

Особенности:

- Обновление программ производится в ускоренном режиме, благодаря замене только отредактированных разработчиком элементов пакета.
- Для скачивания, обновления пакетов, а также разрешения зависимостей придётся использовать пакетные менеджеры более высокого уровня (YUM, DNF).
- Начиная с 2010 года, пакеты подписываются с хешем MD5. Это исключает вероятность изменения файла RPM злоумышленником для внедрения вирусного кода.

YUM (Yellowdog Updater, Modified) – высокоуровневый пакетный менеджер, написанный на языке Python для систем RED HAT (RHEL, CentOS, Fedora). Программа представляет собой своеобразную оболочку для утилиты RPM.

В задачу YUM входит скачивание и обновление пакетов из репозиториев, а также удовлетворение зависимостей во время установки программы.

DNF (Dandified YUM) – модифицированная версия пакетного менеджера YUM на языке на Python. Разработка утилиты начата в 2011 году. В 2015 году DNF стал основным менеджером пакетов для системы Fedora 22. В DNF были исправлены такие недостатки

YUM, как некорректная установка зависимостей, низкая скорость работы, большое потребление оперативной памяти.

НЕКОТОРЫЕ КОМАНДЫ ДЛЯ РАБОТЫ С ПАКЕТАМИ

Использование инструментов низкого уровня

	dpkg	rpm
Установка пакета из скомпилированного файла (Недостатком этого метода установки является то, что не предусмотрено разрешение зависимости)	dpkg –i file.deb	rpm –i file.rpm
Обновление пакета	dpkg -i file.deb	rpm –U file.rpm
Список установленных пакетов	dpkg –l	rpm –qa
Информация о пакете	Dpkg -status package_name	rpm –q package_name
Проверить, какой пакет установил файл	dpkg –-search file_name	rpm –qf file_name
Удалить пакет	dpkg -r package_name	rpm –e package_name

Использование высокоуровневых инструментов

	apt	yum
Поиск пакета	apt-get update && apt-get search package_name	yum search package_name yum provides file_name
Установка пакета из репозитория. При установке пакета вам может быть предложено подтвердить установку после того, как менеджер пакетов разрешит все зависимости	apt-get update && apt-get install package_name	yum update && yum install package_name
Просмотр пакетов	apt-cache pkgnames apt list –installed	yum list available yum list installed yum list kernel
Удаление пакета	apt-get remove package_name	yum remove package_name

УСТАНОВКА ПРОГРАММ НЕ ИЗ РЕПОЗИТОРИЕВ

Когда и зачем?

- В репозиториях нет нужной версии
- В репозиториях нет нужной программы или утилиты
- Мы не доверяем репозиториям
- Репозитории недоступны

Какие варианты?

- Скачать пакет «откуда-то» и установить из локального файла (опасно и могут быть проблемы с зависимостями)
- Программа представлена просто бинарным (или исполняемым) файлом и мы можем его просто разместить в директории из PATH переменной окружения
- Сборка из исходного кода и установка

MAKE

make - инструмент контролирующий процесс сборки ПО из исходных файлов. Разработан в 1976 году в Bell Labs. Make ожидает инструкции из текстового файла Makefile. В 99% случаев файл находится рядом с исходниками программы.

Синтаксис:

```
make [ -f make-файл ] [ цель ] ...
```

Стандартные `цели` для сборки:

- `all` — выполнить сборку пакета;
- `install` — установить пакет из дистрибутива (производит копирование исполняемых файлов, библиотек и документации в системные каталоги);
- `uninstall` — удалить пакет (производит удаление исполняемых файлов и библиотек из системных каталогов);
- `clean` — очистить дистрибутив (удалить из дистрибутива объектные и исполняемые файлы, созданные в процессе компиляции);
- `distclean` — очистить все созданные при компиляции файлы и все вспомогательные файлы, созданные утилитой ./configure в процессе настройки параметров компиляции дистрибутива.

`./configure --help` - помощь в предварительной конфигурации сборки

ПРЕИМУЩЕСТВА MAKE

- Возможность использовать последнюю или, наоборот, устаревшую версию ПО, которой нет в репозиториях.
- Возможность собрать ПО с теми модулями/опциями, которые необходимы именно в вашем случае и для ваших целей.
- Отсутствие необходимости разбираться с созданием пакета и делать свои сложные «обертки» для сборки.

НЕДОСТАТКИ MAKE

- Требует наличия сборочного окружения в ОС
- В случае компрометации облегчает задачу по сборке эксплоита злоумышленнику
- Занимает много времени
- Ручное разрешение зависимостей
- Сложность тиражирования
- Возможны затруднения с unit’ами и init-скриптами
- Больше затрат на поддержание системы в целом за счет необходимости самостоятельно следить за обновлениями безопасности

МОНИТОРИНГ СИСТЕМЫ И ЛОГИ

В большинстве дистрибутивов Linux уже встроены разные инструменты мониторинга. Используя их, можно посмотреть метрики с информацией о системных процессах и найти возможные причины возникновения проблем с производительностью.

ИНСТРУМЕНТЫ МОНИТОРИНГА

TOP, HTOP, ATOP — МОНИТОРИНГ АКТИВНОСТИ ПРОЦЕССОВ

Команда *top* отображает процессы Linux. Она позволяет представить работающую систему в реальном времени, то есть в момент фактической активности процесса. По умолчанию она отображает наиболее ресурсоемкие задачи, выполняемые на сервере, и обновляет их список каждые пять секунд.

Нажав кнопку h можно открыть справку по утилите и по используемым горячим клавишам.

Существуют более продвинутые и информативные версии top, например htop, atop. Так команда *htop* строит псевдографические графики загрузки каждого ядра процессора, памяти и свопа. *atop* выводит только новые изменения об активных системных процессах. Позволяет контролировать нагрузку процессора, памяти, накопителя, сети, а также просматривать распределение нагрузок по работающим процессам.

IOTOP — МОНИТОРИНГ ВВОДА/ВЫВОДА

Утилита *iotop* позволяет определить процесс, узурпировавший всю подсистему ввода/вывода.

IFTOP, JNETTOP — МОНИТОРИНГ СЕТЕВОГО ИНТЕРФЕЙСА

Утилита *iftop* прослушивает сетевой трафик на выбранных интерфейсах и отображает таблицу текуще

Утилита *jnettop* показывает и URL, и передаваемый в данный момент файл. Можно увидеть, какие файлы передаются в данный момент клиентам.

NMON - МОНИТОРИНГ НА ВСЕ СЛУЧАИ

Утилита *nmon* позволяет выбрать объект и получить по нему статистику — процессор, память, диски, ядро, сеть, виртуальная память и т. д.

W —КТО ВОШЕЛ В СИСТЕМУ И КАКИЕ ДЕЙСТВИЯ ПРОИЗВОДИТ

Команда *who (w)* отображает информацию о пользователях, находящихся в данный момент в системе, и их процессах.

NETSTAT, SS — ИНСТРУМЕНТ МОНИТОРИНГА СЕТИ И СТАТИСТИКИ

Команда *netstat* отображает состояние TCP-соединений (как входящих, так и исходящих), таблицы маршрутизации, число сетевых интерфейсов и сетевую статистику по протоколам.

```
netstat -tulpn
netstat -nat
```

Команда *ss* используется для вывода статистики сокетов. Позволяет отображать информацию, аналогичную netstat. Обратите внимание, что команда netstat считается устаревшей. Лучше использовать команду ss. Чтобы посмотреть все сокеты TCP и UDP в Linux:

```
ss -t -a
ss -u -a
```

STRACE — ОТСЛЕЖИВАЕТ СИСТЕМНЫЕ ЗАПРОСЫ

Strace позволяет отслеживать системные запросы и сигналы Linux. Это пригодится при отладке веб-сервера и решении других проблем с ним.

ФАЙЛОВАЯ СИСТЕМА /PROC — СТАТИСТИКА ЯДРА

Файловая система */proc* предоставляет подробную информацию о различных аппаратных устройствах и другие данные об ядре Linux. Примеры:

```
cat /proc/cpuinfo
cat /proc/meminfo
cat /proc/zoneinfo
cat /proc/mounts
```

РАБОТА С ЛОГАМИ

Большинство файлов логов Linux находятся в папке /var/log Можно получить список файлов логов с помощью команды ls:

```
ls -l /var/log/
```

Пример лог-файлов приведены ниже. Некоторые из них специфичны только в определенных дистрибутивах.

/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/auth.log - содержит информацию об авторизации пользователей в системе, включая пользовательские логины и механизмы аутентификации, которые были использованы.

/var/log/boot.log - Содержит информацию, которая регистрируется при загрузке системы.

/var/log/daemon.log - Включает сообщения от различных фоновых демонов

/var/log/kern.log - Тоже содержит сообщения от ядра, полезны при устранении ошибок пользовательских модулей, встроенных в ядро.

/var/log/user.log - Информация из всех журналов на уровне пользователей.

/var/log/cron - Всякий раз когда демон Cron запускает выполнения программы, он записывает отчет и сообщения самой программы в этом файле.

/var/log/secure - содержит информацию, относящуюся к аутентификации и авторизации. Например, SSHd регистрирует здесь все, в том числе неудачные попытки входа в систему.

/var/log/audit/- Содержит информацию, созданную демоном аудита auditd.

Кроме того, у каждого сервиса есть свой лог файл, который можно посмотреть с помощью утилиты *journalctl*.

Чтобы посмотреть логи на Linux удобно использовать несколько утилит командной строки Linux. Это может быть любой текстовый редактор, или специальная утилита. Для просмотра логов понадобятся права суперпользователя. Команды, которые чаще всего используются для этих целей:

- less;
- more;
- cat;

- grep;
- tail.

Например, просмотреть лог /var/log/dmesg, с возможностью прокрутки

```
less /var/log/dmesg
```

Просмотр логов, в реальном времени:

```
tail -f /var/log/dmesg
```

Выводим только ошибки из /var/log/messages:

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