ТЕМА 4.4. ОПЕРАЦИОННАЯ СИСТЕМА LINUX

В данной теме рассматриваются следующие вопросы:

- Основные принципы функционирования Linux.
- Основные компоненты Linux.
- Дистрибутивы Linux.
- Файловая система Linux.
- Командная строка Linux.
- Работа с операционной системой Linux.
- Загрузка ОС Linux в текстовом режиме (консоль администратора).
- Подготовка жесткого диска к работе.
- Работа с файловой системой и командами консоли администратора.
- Изучение типов файлов в Linux.
- Поиск системных журналов.
- Архивирование и разархивирование файлов и директорий.
- Создание новых текстовых файлов.
- Разрезание и склеивание файлов.
- Быстрый анализ текстов.
- Администрирование.
- Получение сведений о системе.
- Управление процессами.
- Выполнение задач в фоновом режиме.
- Изменение приоритетов выполняющихся программ.
- Изучение базовых прав доступа.
- Добавление и удаление пользователей.
- Создание и выполнение shell-программ.

Лекции – 2 часа, лабораторные занятия – 2 часа, самостоятельная работа – 2 часа.

Экзаменационные вопросы по теме:

- Основные принципы функционирования Linux. Основные компоненты Linux. Дистрибутивы Linux. Файловая система Linux.
- ОС Linux: управление процессами, выполнение задач в фоновом режиме, изменение приоритетов выполняющихся программ.

4.4.1. Основные принципы функционирования Linux

Аппаратные требования у Linux в (текстовом режиме) достаточно скромные. Так, например, машина с 486 процессором и 16 MB RAM под Linux представляет собой мощную рабочую станцию или многопользовательский сетевой сервер. Организация программно-аппаратных средств во всех UNIX-совместимых системах организована по принципу клиент-сервер. С точки зрения распределения функций, возложенных на систему, все компьютеры в сети работают как один большой компьютер, который может быть легко дополнен аппаратными ресурсами, когда к сети подключается новый компьютер [1].

Каждый пользователь работает с системой через виртуальный терминал, которых может быть до 12-ти в зависимости от версии Linux (на экране обозначается как «tty1…tty12»). Переключение между ними осуществляется клавишами <Alt> + <F1…F12>. Один и тот же компьютер может одновременно работать и сервером сети, и рабочей станцией.

В ОС Linux существует возможность изменять существующие интерфейсы и создавать свои собственные. Для этой цели служат специальные библиотеки, в которых хранятся заготовки интерфейсов.

В ОС Linux все файлы организованы в непрерывный поток байтов. Данные, вводимые с клавиатуры, представляют собой входной поток данных, а отображаемые данные – выходной поток. Поскольку процедуры ввода и вывода организованы также, как и файлы, то они могут свободно взаимодействовать с файлами. В данной ОС широко используется переадресация (**cat**, **>**, **>>**, **<**) которая позволяет перемещать данные в файлы и из файлов. Таким образом, монитор и клавиатура рассматриваются системой как файлы.

Иногда возникают ситуации, когда нужно передать данные из одной команды в другую, а не в файл. Например, нужно послать список имен файлов на принтер. Для этого нужны две команды: **ls** и **lpr**, первая из них создает список, а вторая посылает его на принтер. Т.е. нужно направить вывод команды **ls** на ввод команды **lpr**. Для такого соединения в Linux используется оператор конвейера **|**, который помещается между двумя командами и связывает их стандартные потоки.

Пример:

```
$ ls | lpr          # (список каталогов передается на принтер)
```

С помощью каналов можно строить сложные длинные конструкции, называемые конвейерами.

В ОС Linux имена файлов могут содержать любые буквы, знаки подчеркивания и цифры. Но не должно начинаться с цифры, точки (кроме скрытых системных файлов) или содержать знаки `/`, `\`,`?`, `*`. Максимальная длина имени – 256 символов (как и в FAT32). Расширение рассматривается как часть имени, и оно может быть полезно для сортировки файлов по категориям. Пользователь может назначить любое удобное для себя расширение, что никак не повлияет на свойства файла.

Файл может быть каталогом или исполняемой программой (командой). Команда **file** помогает определить, для чего используется данный файл (например, текстовый файл или каталог).

Файловая система в ОС Linux как и в большинстве других систем имеет иерархическую (древовидную) структуру. Вверху дерева всегда находится корневой каталог ROOT. В этой операционной системе также справедливо понятие текущего каталога. Каждый файл имеет относительное имя пути, которое определяет его принадлежность к какому-либо каталогу, и абсолютное имя пути, которое показывает весь путь файла, начиная от корневого каталога.

Например, `/home/user_name/MyDocument/doc1` – абсолютное имя, а `MyDocument/doc1` – относительное имя.

4.4.2. Основные компоненты Linux

При загрузке компьютера операционная система Linux перехватывает управление компьютером и управляет следующими его компонентами (рис. 4.4.1) [1].

Рис. 4.4.1. Структура GNU/Linux

Linux - это ядро и ядро каждого дистрибутива Linux.

Программное обеспечение ядра Linux поддерживается группой людей, возглавляемой Линусом Торвальдсом. Торвальдс работает в отраслевом консорциуме The Linux Foundation для работы над ядром Linux.

В ядре собрана основная функциональность для работы с памятью, управления процессами и т.д.

Когда ядру требуется дополнительные функции, он обращается к модулям. В модулях, например, может содержаться код для работы с оборудованием.

Файлы ядра Linux находятся в `/boot` (рис. 4.4.2).

```
[user@centos8 ~]$ ls /boot/
config-4.18.0-147.8.1.el8_1.x86_64
config-4.18.0-147.el8.x86_64
efi
grub2
initramfs-0-rescue-91222dde95634287a2336070235dc625.img
initramfs-4.18.0-147.8.1.el8_1.x86_64.img
initramfs-4.18.0-147.8.1.el8_1.x86_64kdump.img
initramfs-4.18.0-147.el8.x86_64.img
initramfs-4.18.0-147.el8.x86_64kdump.img
loader
lost+found
System.map-4.18.0-147.8.1.el8_1.x86_64
System.map-4.18.0-147.el8.x86_64
vmlinuz-0-rescue-91222dde95634287a2336070235dc625
vmlinuz-4.18.0-147.8.1.el8_1.x86_64
vmlinuz-4.18.0-147.el8.x86_64
```

Рис. 4.4.2. Красными стрелками отмечены файлы ядра Linux

Версию используемого ядра Linux можно узнать с помощью команды **uname** (рис. 4.2.3).

```
[user@centos8 ~]$ uname -r
4.18.0-147.8.1.el8_1.x86_64
```

Рис. 4.4.3. Получение информации о версии ядра Linux

Планировщик

Так как ядро Linux обеспечивает одновременную работу нескольких процессов от нескольких пользователей (с поддержкой нескольких процессоров), операционная система нуждается в средствах управления многопоточностью.

Планировщик Linux назначает процессам приоритеты и определяет, какой процесс выполняется на конкретном процессоре (если в системе установлено несколько процессоров).

Планировщик можно настроить для работы в системах различного типа. При правильной настройке более важные процессы получают более быструю реакцию процессора. Например, планировщик Linux на настольном компьютере предоставляет больший приоритет задаче перемещения окна и меньший — задаче фонового копирования файлов.

Файл подкачки

Ядро Linux старается держать работающие в данный момент процессы в оперативной памяти. Простаивающие процессы перемещаются в файл подкачки, представляющий собой выделенную область на жестком диске, которая используется для хранения не перемещающихся в оперативную память данных и процессов.

При переполнении оперативной памяти процессы выносятся в файл подкачки.

При переполнении файла подкачки (но этого допускать нельзя) новые процессы не запускаются.

Модули

Ядро Linux поддерживает тысячи аппаратных устройств. При этом за счет включения в работающее ядро только актуальных драйверов размер ядра удается сохранять на приемлемом уровне.

Использование загружаемых модулей позволяет добавить в ядро поддержку дополнительных устройств.

Модули можно загружать и выгружать по запросу в результате подключения или отключения устройства.

Файловые системы

Файловые системы предоставляют структуры, в которых файлы хранятся на компакт-дисках, жестких дисках, гибких дисках, DVD и на других носителях.

Ядро Linux поддерживает множество типов файловых систем (например, файловые системы Linux: ext3 и ReiserFS, а также файловые системы VFAT и NTFS из операционной системы Windows).

Механизмы защиты

Как и UNIX, операционная система Linux изначально создавалась для обеспечения, одновременного многопользовательского доступа. Для защиты пользовательских ресурсов каждому файлу назначаются наборы разрешений на чтение, запись и выполнение, которые определяют права доступа.

В стандартной системе Linux пользователь root имеет доступ ко всей системе без ограничений, специальные регистрационные записи могут управлять определенными службами (например, службами Web-сервера Apache), а пользователям могут присваиваться разрешения по отдельности или в составе групп.

Последние нововведения, например, Security-Enhanced Linux (SELinux), поддерживают более тонкую настройку и защиту безопасных сред обработки информации.

Инструменты администрирования

Инструменты администрирования включают в себя сотни (а возможно и тысячи) команд и графических утилит, которые позволяют добавлять пользователей, управлять дисками, следить за состоянием сети, устанавливать программное обеспечение, а также гарантировать безопасность и управлять ресурсами компьютера.

Серверные возможности

Серверные возможности позволяют компьютеру под управлением Linux предоставлять службы для клиентов в сети. Иными словами, кроме установки Web-обозревателей для просмотра Web-страниц, компьютер можно превратить в сервер, который предоставляет Web-страницы другим компьютерам. При этом среди популярных серверных функций можно назвать Web-серверы, серверы электронной почты, баз данных, печати, файловые серверы, серверы DNS и DHCP.

Графический интерфейс пользователя

Графический интерфейс пользователя состоит из графической инфраструктуры (обычно это X WindowSystem), оконных менеджеров, панелей, пиктограмм и меню.

Графический интерфейс пользователя позволяет применять комбинацию мыши и клавиатуры вместо простого ввода команд с клавиатуры.

Система управления пакетами

Система управления пакетами - это набор инструментов, предназначенных для автоматизации процессов установки, обновления, конфигурирования и удаления пакетов программного обеспечения определенного формата.

Наиболее известными (или распространенными) системами управления пакетами являются:

- RPM/YUM — менеджер пакетов Red Hat
- dpkg/APT — система управления пакетами *.deb дистрибутива **Deb**ian
- tgz или tar.gz — стандартный набор из двух программ **t**ar + **gz**ip
- система портежей дистрибутива Gentoo
- YaST - утилита, разработанная Novell и используемая в дистрибутиве SuSE

4.4.3. Дистрибутивы Linux

Мы можем распределить дистрибутивы Linux на три группы:

- Enterprise Grade Linux Distributions
  - Red Hat Enterprise Linux
  - CentOS
  - SUSE Linux Enterprise Server
  - Debian GNU/Linux
  - Ubuntu LTS
- Consumer Grade Linux Distributions
  - Fedora
  - Ubuntu non-LTS
  - openSUSE
- Experimental and Hacker Linux Distributions
  - Arch
  - Gentoo

Enterprise Grade Linux

Дистрибутивы этой группы предназначены для развертывания в крупных организациях с использованием оборудования предприятия.

Чтобы обеспечить доступность своих услуг, корпоративные пользователи предъявляют более высокие требования к стабильности своего аппаратного и программного обеспечения.

Корпоративные дистрибутивы Linux обычно включают более старые выпуски ядра и другое программное обеспечение, которое, как известно, работает надежно. Часто дистрибутивы портируют важные обновления, такие как исправления безопасности, на эти стабильные версии.

В корпоративных дистрибутивах Linux может отсутствовать поддержка самого последнего потребительского оборудования, и они могут предоставлять более старые версии пакетов программного обеспечения.

Предприятия, как правило, также выбирают зрелые аппаратные компоненты и строят свои сервисы на стабильных версиях программного обеспечения.

Consumer Grade Linux

Дистрибутивы этой группы больше ориентированы на малый бизнес или домашних пользователей и любителей.

Подготовлены для использования новейшего оборудования, установленного в системах потребительского уровня. Этим системам потребуются новейшие драйверы, чтобы максимально использовать новое оборудование.

Зрелость как оборудования, так и драйверов вряд ли удовлетворит потребности крупных предприятий.

Для потребительского рынка новейшее ядро — это именно то, что нужно для большинства современных драйверов, даже если они мало проверены.

Более новые ядра Linux будут иметь новейшие драйверы для поддержки самого современного оборудования, которое, вероятно, будет использоваться.

Чрезвычайно важно, чтобы последние версии драйверов были доступны для пользователей. Особенно с развитием с Linux на игровом рынке.

Experimental and Hacker Linux

Дистрибутивы этой группы используют самые современные технологии.

Содержат самые последние версии программного обеспечения, даже если эти версии все еще содержат ошибки и непроверенные функции.

В этих дистрибутивах используется скользящая модель выпуска, которая позволяет им доставлять обновления в любое время.

Используются опытными пользователями, которые хотят всегда получать самые последние версии программного обеспечения и знают, что функциональность может быть нарушена в любое время, и в таких случаях могут восстановить свои системы.

Используются в качестве испытательного стенда для Enterprise Grade Linux, где потенциальные возможности будущих версий могут быть изучены до их доступности в корпоративном выпуске.

Жизненный цикл поддержки Linux

Корпоративные дистрибутивы Linux имеют более длительный срок поддержки, чем потребительские или общественные выпуски Linux

Red Hat Enterprise Linux имеет поддержку в течение 10 лет.

CentOS 7 релиз 7 июля 2014, полная поддержка до 4го квартала 2020, критические обновления до 30 июня 2024.

Потребительские выпуски часто получают поддержку сообщества только через форумы.

Ubuntu также поставляет выпуски с долгосрочной поддержкой, известные как выпуски LTS, которые имеют 5-летнюю поддержку от первоначального выпуска.

Ubuntu 18.04 LTS имеет поддержку до апреля 2023

Ubuntu 20.04 LTS имеет поддержку до апреля 2025

Debian 10 релиз Июля 2022, поддержка до Июля 2024

Дистрибутивы Linux

Дистрибутивы состоят из отдельных пакетов, каждый из которых содержит какое-то приложение, утилиту или сервис. Отдельный пакет может содержать, например, веб-браузер, библиотеку для работы с графическими файлами в формате PNG, набор шрифтов и так далее.

Программное обеспечение, содержащееся в пакете, поставляется в одном из двух основных видов:

- в виде **бинарных файлов**, которые предназначены для непосредственной установки в вашу систему, без какой-либо дополнительной обработки (например, компиляции).
- в виде **исходных текстов**, которые обычно содержат текст на каком-то языке программирования, заархивированный в формате tar и упакованный программой gzip, а также вспомогательные файлы, необходимые для компиляции приложения из файлов пакета.

4.4.4. Файловая система Linux

На заре компьютерной техники файлом называлась именованная область данных на диске.

В операционной системе GNU/Linux все объекты являются файлами, в том числе и директории, которые служат для организации доступа к файлам.

Существуют следующие типы файлов: обычные файлы, каталоги, символьные ссылки, блочные устройства, символьные устройства, сокеты, каналы.

Каждый тип имеет собственное обозначение одним символом. Этот тип выводится как первый символ в выводе команды **ls -l**

```
[devops@localhost demo]$ ls -l
total 8
-rw-rw-r--. 1 devops devops 46 Dec 15 08:32 file1
lrwxrwxrwx. 1 devops devops  5 Dec 25 22:19 file1a -> file1
-rw-rw-r--. 1 devops devops 46 Dec 15 08:33 file2
drwxrwxr-x. 2 devops devops  6 Dec 15 09:05 oldfiles
```

Рис. 4.4.4. Пример вывода команды ls -l

Ниже приведены значения этого первого символа:

- **-** (обычные файлы), для хранения символьных и двоичных данных, например, `text.txt`, `/etc/hosts`
- **d** (директории), служат для организации доступа к файлам, например, `/usr/bin`, `/root`, `/home`
- **l** (символьные ссылки), для организации доступа к файлу по альтернативному имени и/или пути
- **c** (символьные устройства) например, `/dev/tty1`
- **b** (блочные устройства) например, `/dev/sdb`, `/dev/sdb1`
- **p** (каналы)
- **s** (сокеты)

Структура файловой системы

В любой файловой системе Linux всегда есть только один корневой каталог, который называется **/**

Пользователь Linux всегда работает с единым деревом каталогов, даже если данные расположены на разных носителях: жёстких или съёмных дисках, CD-ROM, сетевых дисках и т. д.

Положение любого каталога в дереве каталогов точно и однозначно описывается при помощи полного пути. Полный путь всегда начинается от корневого каталога и состоит из перечисления всех вершин, встретившихся при движении по рёбрам дерева до искомого каталога включительно. Названия соседних вершин разделяются символом наклонной черты **/** (слэш).

Например, файл конфигурации Apache находится в файле

```
/etc/httpd/conf/httpd.conf
```

Абсолютные и относительные пути

Для каждого процесса Linux определена текущая директория, с которого система начинает относительный путь при выполнении файловых операций.

Между полным путём и относительным есть только одно существенное различие: **относительный** путь начинается от текущей директории, в то время как **полный** путь всегда начинается от корневой директории. То есть, первый символ полного пути всегда будет **/**

Текущая директория, каков бы ни был полный путь к ней, всегда имеет ещё одно обозначение в виде точки **.** Родительская директория обозначается двумя точками **..**

```
[devops@localhost ~]$ cat live/sample
Sample text
[devops@localhost ~]$ cat /home/devops/live/sample
Sample text
[devops@localhost ~]$ cd Music
[devops@localhost Music]$ cat ../live/sample
Sample text
```

Рис. 4.4.5. Обращение к файлу по относительному и абсолютному пути

Допустимые символы в имени файла

Имя может содержать любые символы (в том числе и кириллицу), кроме / ? < > * " |

Максимальная длина имени файла — 254 символа.

Linux не использует расширение имени файла для определения его типа или поиска ассоциированного с ним приложения. Поэтому точка внутри имени считается обычным символом. Но традиционно для некоторых типы файлов используют расширения для наглядности, например: `.tar` для архивов, `.rpm` для пакетов программного обеспечения, `.conf` – для файлов конфигурации.

Имена файлов в Linux чувствительны к регистру: `FILE.TXT` и `file.txt` — это два разных файла.

Разделение элементов пути осуществляется символом наклонной черты / (или слэш), в отличие от Windows, где используется обратная наклонная черта \ (обратный слэш, бэкслэш).

Скрытые файлы и директории

Точка **.** в начале имени файла делает его скрытым, то есть, он не показывается в выводе команды **ls**.

По умолчанию домашний каталог пользователя будет содержать много скрытых файлов. Они часто используются для установки пользовательских настроек конфигурации и должны быть изменены только опытным пользователем.

Для отображения скрытых файлов в команде **ls** предназначена опция **-a** или **–all**.

```
[devops@nnk4 ~]$ ls
Desktop    Downloads  live    newdir    Public  Templates
Documents  final1     Music   Pictures  task6   Videos
[devops@nnk4 ~]$ ls -a
.              .dbus           live       task6
..             Desktop         .local     Templates
.bash_history  Documents       .mozilla   .vboxclient-clipboard.pid
.bash_logout   Downloads       Music      .vboxclient-display-svga-x11.pid
.bash_profile  .esd_auth       newdir     .vboxclient-draganddrop.pid
.bashrc        final1          Pictures   .vboxclient-seamless.pid
.cache         .ICEauthority   .pki       Videos
.config        .lesshst        Public     .viminfo
```

Рис. 4.4.6. Домашний каталог содержит много скрытых файлов

Навигация по файловой системе

Команда **pwd** (Print Working Directory) возвращает полный путь к текущей директории командной оболочки (хранится в переменной PWD)

Для смены текущей директории используется команда **cd** (Change Directory)

Команда **cd** в качестве параметра получает путь к директории, которая должна стать текущей (при условии её существования и при наличии необходимых разрешений).

Если был указан относительный путь, поиск происходит в соответствии со значением переменной **CDPATH**. Если указан полный путь, переменная **CDPATH** не используется

По умолчанию (если параметр не указан) текущей директорией становится домашняя (в соответствии со значением переменной **HOME**)

Параметр **–** используется для того, чтобы вернуть текущую директорию в место, где она была до последней команды **cd** (было сохранено в переменной **OLDPWD**)

```
[devops@nnk4 ~]$ cd Music/
[devops@nnk4 Music]$ pwd
/home/devops/Music
[devops@nnk4 Music]$ echo PWD=$PWD OLDPWD=$OLDPWD HOME=$HOME
PWD=/home/devops/Music OLDPWD=/home/devops HOME=/home/devops
[devops@nnk4 Music]$ cd
[devops@nnk4 ~]$ pwd
/home/devops
[devops@nnk4 ~]$ cd -
/home/devops/Music
[devops@nnk4 Music]$ cd /root
bash: cd: /root: Permission denied
[devops@nnk4 Music]$ cd /etc/cron.d/
[devops@nnk4 cron.d]$ pwd
/etc/cron.d
[devops@nnk4 cron.d]$ cd
[devops@nnk4 ~]$
[devops@nnk4 ~]$ cd /var/log
[devops@nnk4 log]$ cd $HOME
[devops@nnk4 ~]$
```

Рис. 4.4.7. Демонстрация команд pwd и cd

Понятие домашнего каталога

Домашний каталог (домашняя папка, домашняя директория) – предназначен для хранения собственных данных пользователя Linux и личных настроек для программ.

Как правило, становится текущим непосредственно после регистрации пользователя в системе.

Полный путь к домашнему каталогу хранится в переменной окружения **HOME**, в полном пути к файлу можно заменять на знак тильды **~**, то есть, /home/user1/file1, ~/file1 и $HOME/file1 эквивалентны

Для обычных пользователей домашний каталог находится в директории **/home**.

FHS (Filesystem Hierarchy Standard)

FHS (Filesystem Hierarchy Standard) – стандарт файловой системы, специфичной для GNU/Linux, создан в 1994-1996 гг. Текущая версия стандарта FHS 3.0 выпущена 03.06.2015 г.

Некоторые Linux-системы отвергают FHS и следуют своему собственному стандарту.

В FHS все файлы и каталоги находятся внутри корневого каталога, даже если они расположены на различных физических носителях. В корневой директории должны быть следующие директории: bin, boot, dev, etc, lib, media, mnt, opt, run, sbin, srv, tmp, usr, var. При наличии соответствующих подсистем добавляются папки home, root и lib64.

Определенные в стандарте FHS директории:

/ Корневой каталог, содержащий всю файловую иерархию

/bin Основные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям

/boot Файлы ядра

/dev Файлы устройств

/etc Общесистемные конфигурационные файлы

/home Содержит домашние папки пользователей. Часто размещается на отдельном разделе

/lib Основные библиотеки, необходимые для работы программ из /bin и /sbin

/media Точки монтирования для сменных носителей, таких как CD-ROM, DVD-ROM

/mnt Содержит временно монтируемые файловые системы

/opt Дополнительное прикладное программное обеспечение

/proc Виртуальная файловая система, представляющая состояние ядра операционной системы и запущенных процессов в виде файлов

/root Домашняя папка пользователя root

/run Временные данные выполняющихся процессов

/sbin Основные системные программы для администрирования и настройки системы

/srv Данные для сервисов, предоставляемых системой

/tmp Временные файлы

/usr Вторичная иерархия для данных пользователя, содержит большинство пользовательских приложений и утилит, используемых в многопользовательском режиме.

/var Изменяемые файлы, такие как файлы регистрации, временные почтовые файлы, файлы спулеров

4.4.5. Командная строка Linux

Существует два вида интерфейсов: графический интерфейс пользователя и интерфейс командной строки.

Графический интерфейс пользователя (Graphical user interface, GUI) – для управления программами и отображения информации используются графические элементы: кнопки, меню, окна и т. д. Действия выполняются с помощью клавиатуры и мыши.

Преимущества: визуальное отображение программ и их содержимого, не требуется изучение документации перед началом работы с программой.

Интерфейс командной строки (Command Line Interface, CLI) - управление выполняется с помощью текстовых команд, вводимых по строкам. Основной инструмент ввода — клавиатура.

Данный интерфейс встроен в ядро системы, он будет доступен, даже если графический интерфейс не запустится.

Преимущества: небольшой расход ресурсов, возможность использовать копирование и вставку команд, возможность составлять сложные последовательности действий в текстовом редакторе (скрипты), передача результатов выполнения одной программы другой (конвейер), поддержка автоматического выполнения команд.

Получить доступ к командной строке можно двумя способами: через консоль или терминал. Открыть терминал Linux можно в любом графическом окружении, одновременно нажав клавиши **Ctrl+Alt+T**. Открыть консоль можно, нажав комбинацию клавиш Ctrl+Alt+F и номер консоли, например, **Ctrl+Alt+F4**. После нажатия этого сочетания графическое окружение исчезнет, а вместо него появится черный экран с предложением ввода логина и пароля.

Командные оболочки

В терминале запускается командная оболочка (shell).

Командная оболочка — это интерпретатор командной строки, который выполняет команды, вводимые пользователем. Он интерпретирует команду, выполняет её, а затем отображает результат ее выполнения. Также в оболочку встроен язык программирования (командный язык), который позволяет писать сложные командные сценарии (скрипты).

Существует большое количество командных оболочек, одной из первых была Bourne Shell, одной из самых популярных является **Bash** (Bourne-again shell).

Подсказка командной строки (prompt) выполняет функцию приглашения к вводу пользователем команды и предоставляет некоторую информацию о текущей ситуации.

Обычно указывается имя пользователя, имя хоста, текущий каталог и тип пользователя.

```
[devops@localhost ~]$
[devops@localhost ~]$ su
Password:
[root@localhost devops]#
[root@localhost devops]#
```

Рис. 4.4.8. Последний символ подсказки указывает вид пользователя: **#** для суперпользователя (root), **$** для обычного пользователя.

В разных дистрибутивах Linux подсказка может выглядеть иначе, её вид задаётся в переменной **PS1**. Например, можно задавать цвет и включать дополнительную информацию.

```
[root@localhost devops]# echo $PS1
[\u@\h \W]\$
```

Рис. 4.4.9. Формат стандартной подсказки для Ubuntu

Оболочка поддерживает два типа команд:

- **Internal**. Эти команды являются частью самой оболочки и не являются отдельными программами. Есть около 30 таких команд. Их основное назначение - выполнение задач внутри оболочки (например, cd, set, export).
- **External**. Эти команды находятся в отдельных файлах. Эти файлы обычно представляют собой двоичные программы или сценарии. Когда запускается команда, которая не является встроенной в оболочку, оболочка использует переменную **PATH** для поиска исполняемого файла с тем же именем, что и

команда. В дополнение к программам, которые устанавливаются вместе с менеджером пакетов дистрибутива, пользователи также могут создавать свои собственные внешние команды.

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

Рис. 4.4.10. Команда **type** показывает тип конкретной команды

Постраничный вывод

Когда вывод команды не помещается на один экран, для постраничного вывода можно использовать команды **more** и **less**.

Команда **more** делает задержку вывода после заполнения экрана и позволяет просматривать текст только в одном направлении.

Команда **less** имеет больше функций и позволяет перемещаться по тексту в обоих направлениях, выполнять поиск в тексте. Имеет преимущество при просмотре больших по размеру файлов: данные с диска считываются по мере необходимости (команда **more** считывает сразу весь файл) .

В обеих программах по умолчанию клавиша **Space** (пробел) пролистывает вниз на один экран, клавиша **Enter** – на одну строку.

В команде **less** поиск вперед **/**, поиск назад **?,** для перемещения к следующему результату поиска используется **n**, к предыдущему **N**.

Полный список команд можно увидеть в справочной документации.

4.4.6. Работа с операционной системой Linux

TBD.

4.4.7. Загрузка ОС Linux в текстовом режиме (консоль администратора)

TBD.

4.4.8. Подготовка жесткого диска к работе

TBD.

4.4.9. Работа с файловой системой и командами консоли администратора

Для создания директорий служит команда **mkdir**. Так как все директории (кроме корневой **/**) являются дочерними к какой-либо другой, то этот процесс можно еще называть созданием поддиректорий.

```
[devops@nnk4 ~]$ ls
Desktop    Downloads  List-Arguments.sh  Pictures  task6      Videos
Documents  final1     Music              Public    Templates
[devops@nnk4 ~]$ mkdir newdir
[devops@nnk4 ~]$ ls
Desktop    Downloads  List-Arguments.sh  newdir    Public  Templates
Documents  final1     Music              Pictures  task6   Videos
[devops@nnk4 ~]$ mkdir live/birds
mkdir: cannot create directory ‘live/birds’: No such file or directory
[devops@nnk4 ~]$ mkdir -p live/birds
[devops@nnk4 ~]$ ls
Desktop    Downloads  List-Arguments.sh  Music   Pictures  task6      Videos
Documents  final1     live               newdir  Public    Templates
[devops@nnk4 ~]$ ls live/
birds
```

Рис. 4.4.11. Опция **-p** создает также промежуточные папки в иерархии, если они не существуют.

Некоторые опции - команды ls

-a, --all

отображать файлы с именами, начинающимися с точки **.**

-l

использовать длинный формат вывода

-h, --human-readable

вместе с **-l** and **-s** отображает размеры в виде 1K 234M 2G

```
[devops@nnk4 live]$ ls
birds  elephant  hippo  hippopotamus
[devops@nnk4 live]$ ls -la
total 172052
drwxrwxr-x.  3 devops devops        83 Dec 30 10:59 .
drwx------. 20 devops devops      4096 Dec 30 08:18 ..
drwxrwxr-x.  2 devops devops        63 Dec 29 21:38 birds
-rw-rw-r--.  1 devops devops 117440512 Jun 23  2020 elephant
-rw-rw-r--.  1 devops devops     15360 Dec 30 10:59 .gopher
lrwxrwxrwx.  1 devops devops        12 Dec 30 10:56 hippo -> hippopotamus
-rw-rw-r--.  1 devops devops  58720256 Apr 11  2020 hippopotamus
[devops@nnk4 live]$ ls -lh elephant
-rw-rw-r--. 1 devops devops 112M Jun 23  2020 elephant
```

Рис. 4.4.12. Использование опций -a, -l, -h команды ls

-R, --recursive

рекурсивно выводит подкаталоги

```
[devops@nnk4 live]$ ls -lR
.:
total 172032
drwxrwxr-x. 2 devops devops        63 Dec 29 21:38 birds
-rw-rw-r--. 1 devops devops 117440512 Jun 23  2020 elephant
drwxrwxr-x. 2 devops devops        45 Dec 30 11:10 fish
lrwxrwxrwx. 1 devops devops        12 Dec 30 10:56 hippo -> hippopotamus
-rw-rw-r--. 1 devops devops  58720256 Apr 11  2020 hippopotamus

./birds:
total 4008
-rw-rw-r--. 1 devops devops      12 Dec 29 21:37 Crow
-rw-rw-r--. 1 devops devops       0 May  1  2020 nightingale
-rw-rw-r--. 1 devops devops 2000000 Nov 30 21:00 Swan
-rw-rw-r--. 1 devops devops 2097152 Dec 29 21:38 Turkey

./fish:
total 172
-rw-rw-r--. 1 devops devops 92725 Dec 30 11:09 eel
-rw-rw-r--. 1 devops devops 56370 Dec 30 11:09 flounder
-rw-rw-r--. 1 devops devops 23232 Dec 30 11:08 pike
```

Рис. 4.4.13. Использование опции -R команды ls

-S

сортировка по размеру файла, начиная с самых больших

-r, --reverse

сортировка в обратном порядке

```
[devops@nnk4 live]$ ls -lS
total 172032
-rw-rw-r--. 1 devops devops 117440512 Jun 23  2020 elephant
-rw-rw-r--. 1 devops devops  58720256 Apr 11  2020 hippopotamus
drwxrwxr-x. 2 devops devops        63 Dec 29 21:38 birds
drwxrwxr-x. 2 devops devops        45 Dec 30 11:10 fish
lrwxrwxrwx. 1 devops devops        12 Dec 30 10:56 hippo -> hippopotamus
[devops@nnk4 live]$
[devops@nnk4 live]$ ls -lSr
total 172032
lrwxrwxrwx. 1 devops devops        12 Dec 30 10:56 hippo -> hippopotamus
drwxrwxr-x. 2 devops devops        45 Dec 30 11:10 fish
drwxrwxr-x. 2 devops devops        63 Dec 29 21:38 birds
-rw-rw-r--. 1 devops devops  58720256 Apr 11  2020 hippopotamus
-rw-rw-r--. 1 devops devops 117440512 Jun 23  2020 elephant
```

Рис. 4.4.14. Использование сортировки в команде ls

Команда touch

Основное назначение команды **touch** – изменить отметку времени о последнем доступе к файлу.

```
[devops@localhost ~]$ date
Sat Dec 12 08:41:41 +03 2020
[devops@localhost ~]$ ls -l test
-rw-rw-r--. 1 devops devops 345 Oct  4 21:12 test
[devops@localhost ~]$ touch test
[devops@localhost ~]$ ls -l test
-rw-rw-r--. 1 devops devops 345 Dec 12 08:42 test
[devops@localhost ~]$
```

Рис. 4.4.15. Выполнение команды touch изменило отметку времени

Но если такого файла не существует, команда **touch** его создает. Поэтому она часто используется для создания файлов в демонстрациях и упражнениях по файловой системе Linux.

```
[devops@localhost ~]$ touch project
[devops@localhost ~]$ touch project{1..3}
[devops@localhost ~]$ ls -l project*
-rw-rw-r--. 1 devops devops 0 Dec 12 08:45 project
-rw-rw-r--. 1 devops devops 0 Dec 12 08:45 project1
-rw-rw-r--. 1 devops devops 0 Dec 12 08:45 project2
-rw-rw-r--. 1 devops devops 0 Dec 12 08:45 project3
[devops@localhost ~]$
```

Рис. 4.4.16. Использование команды touch для создания пустых файлов

Опция **-t** [[CC]YY]MMDDhhmm.ss команды **touch** позволяет использовать произвольное время вместо текущего, опция -d позволяет указать дату в более привычном виде, опция **-a** изменяет только время последнего доступа к файлу, **-m** изменяет только время изменения содержимого файла.

Опция **-l** или **--length** команды **fallocate** задает размер файла в байтах (не забываем, что 1 килобайт = 1000 байт, а 1024 байта содержится в одном кибибайте). **KB**, **MB** и т.д. обозначают десятичные единицы, а **K** или **KiB**, **M** или **MiB**, … обозначают двоичные единицы.

Команды вывода

Команда **cat** выводит один или несколько файлов в стандартный поток вывода (по умолчанию – на консоль). Без опций и параметров команда просто передает текст из стандартного потока ввода в стандартный поток вывода.

Команду **cat** можно использовать для создания многострочных файлов, используя конструкцию heredoc:

```
cat << маркер > файл
. . . . . .
. . . . . .
маркер
```

```
[devops@localhost demo]$ cat << EOF > file1
> === This is file 1 ===
> 11111111111111111111
> EOF
[devops@localhost demo]$ cat << EOF > file2
> === This is file 2 ===
> 222222222222222222222
> EOF
[devops@localhost demo]$
[devops@localhost demo]$ cat file*
=== This is file 1 ===
11111111111111111111
=== This is file 2 ===
22222222222222222222
[devops@localhost demo]$
```

Рис. 4.4.17. Создание файла с помощью команды cat

Команды **head** и **tail** служат для вывода заданного количества первых или последних строк файла соответственно (по умолчанию 10).

Переименование файлов

Команда **mv** используется и для перемещения, и для переименования файлов и директорий.

Синтаксис команды **mv**:

mv [опции...] исходный_файл целевой_файл

mv [опции...] исходный_файл... каталог

Переименование файла происходит, когда оба аргумента являются файлами, причем, если целевой файл существует, он будет перезаписан

Переименование директории происходит, когда первый аргумент является директорией, а второй аргумент указывает на несуществующую директорию. Иначе (если директория из второго аргумента существует) произойдет перемещение первой директории и всего ее содержимого во вторую)

Перемещение файлов

Если последний аргумент указывает на существующую директорию или указана опция **-t** *target*, происходит перемещение файлов и директорий, обозначенных остальными аргументами, в целевую папку

Допускается при задании источника использовать символы подстановки, например, *.jpg, 2020-12-* и т. п.

Некоторые опции:

-i, **--interactive** — выводить запросы на подтверждение операции перезаписи файла

-f, **--force** — перезаписывать существующие файлы без запроса на подтверждение операции

-n, **--no-clobber** — не перезаписывать существующий файл

-v, **--verbose** — выводить на экран сообщение о каждой выполняемой операции

Удаление файлов и директорий

Для удаления файлов предназначена утилита **rm** (remove).

По умолчанию, она не удаляет директории. Для удаления директорий нужно указывать опцию **-r**, **-R** или **--recursive**.

С опцией **–i** запрашивает подтверждение на удаление каждого файла, более мягкая опция **-I** запрашивает подтверждение однократно.

Опция **–f** или **--force** отключает запрос подтверждения на удаление.

Новичкам на форумах часто советуют команду **rm –rf /** Никогда не используйте!

Когда нужно предотвратить последующее восстановление файла, используется команда **shred**.

Для удаления пустых директорий служит команда **rmdir**. С опцией **–p** удалит и пустые родительские директории

Копирование файлов и директорий

Для создания копии файла используется команда **cp** (copy)

cp [OPTION] … [-T] SOURCE DEST

cp [OPTION] … SOURCE… DEST

cp [OPTION] … -t DIRECTORY SOURCE…

В именах файлов и директорий можно использовать полные и относительные пути

Как и в команде **mv**, в случае попытки перезаписи файла опция -**i** запрашивает подтверждение, опция **-n** запрещает перезапись файла, опция **-f** перезаписывает файл без подтверждения

Опция **-t** или **--target-directory** указывает целевую директорию для копирования.

С опцией **-T** или **--no-target-directory** параметр DEST считается обычным файлом.

Когда последний параметр файл, может быть только один SOURCE

Подстановка (globbing)

Признаком подстановки является наличие символов **? * [** в строке

? (не в квадратных скобках) – любой символ

***** (не в квадратных скобках) – любая строка, включая пустую

Выражение **[...]** соответствует любому символу между скобками

Выражение **[!...]** соответствует любому символу, кроме перечисленных между скобками (первый **!** не считается)

Помещенные в квадратные скобки два символа, разделенные дефисом, обозначают диапазон, например, **A-Z** обозначает все прописные латинские буквы, **A-Za-z** – все буквы

Чтобы лишить символы **? * [** их специального значения в командной строке, их нужно предварить обратной наклонной чертой или взять в кавычки

Точка **.** в начале файла не соответствует никакому шаблону, ее нужно указывать явно

Создание объектов с именами по шаблону

Для быстрого создания группы однотипных объектов можно использовать группировку аргументов. Например, команда

```
mkdir -p /games/game{1,2,3}
```

создаст три директории game1, game2 и game3 в директории games и на самом деле эквивалентна команде

```
mkdir -p /games/game1 /games/game2 /games/game3
```

или

```
mkdir -p /games/game{1..3}
```

```
[devops@nnk4 live]$ ls
birds  sample
[devops@nnk4 live]$ touch file{01..10}
[devops@nnk4 live]$ mkdir dir{A..E}
[devops@nnk4 live]$ ls
birds  dirB  dirD  file01  file03  file05  file07  file09  sample
dirA   dirC  dirE  file02  file04  file06  file08  file10
[devops@nnk4 live]$
```

Рис. 4.4.18. Использование globbing для создания однотипных файлов

Классы символов

Диапазоны вида **[A-Z]** хорошо работают для английского языка, но для большинства национальных языков они могут быть непригодны

В этом случае лучше использовать именованные классы символов:

[:alnum:] буквы и цифры

[:alpha:] буквы

[:blank:] пробел или табуляция

[:digit:] цифры

[:lower:] строчные буквы

[:punct:] знаки пунктуации

[:space:] пробел, табуляция, CR, LF, VT и FF

[:upper:] прописные буквы

[:xdigit:] шестнадцатеричные цифры

4.4.10. Изучение типов файлов в Linux

В подразделе 4.4.4 уже было сказано, что существуют следующие типы файлов: обычные файлы, каталоги, символьные ссылки, блочные устройства, символьные устройства, сокеты, каналы. Первый символ в выводе команды ls -l показывает тип файла:

- **-** (обычные файлы), для хранения символьных и двоичных данных, например, `text.txt`, `/etc/hosts`
- **d** (директории), служат для организации доступа к файлам, например, `/usr/bin`, `/root`, `/home`
- **l** (символьные ссылки), для организации доступа к файлу по альтернативному имени и/или пути
- **c** (символьные устройства) например, `/dev/tty1`
- **b** (блочные устройства) например, `/dev/sdb`, `/dev/sdb1`
- **p** (каналы)
- **s** (сокеты)

Расширение в имени служит только для удобства пользователей и ничего не говорит о содержимом файла операционной системе.

Команда **`file`** определяет тип файла, основываясь на его внутренней структуре (расширения файла не используются).

Необходимая информация хранится в файле `/usr/share/misc/magic.mgc`.

```
[devops@localhost ~]$ file /usr/bin/cat
/usr/bin/cat: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically l
inked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=bb18abe41992c607c9
d4ffd60c1070f1b9582c3f, stripped
[devops@localhost ~]$
[devops@localhost ~]$ file test.zip
test.zip: Zip archive data, at least v1.0 to extract
[devops@localhost ~]$
[devops@localhost ~]$ file tt
tt: ASCII text
[devops@localhost ~]$
```

Рис. 4.4.19. Использование globbing для создания однотипных файлов

4.4.11. Поиск системных журналов

Информация о работе системы и программ сохраняется в файлах журналов. Большинство журналов находятся в каталоге `/var/log` (рис. 4.4.20). Они имеют следующий формат:

отметка времени, имя хоста, имя программы или службы, PID, описание события (рис. 4.4.21).

```
[root@nnk2 ~]# ls /var/log
anaconda            dmesg.old           ntpstats            tallylog
audit               firewalld           pluto               tomcat
boot.log            gdm                 ppp                 tuned
boot.log-20210206   glusterfs           qemu-ga             vboxadd-install.log
boot.log-20210214   grubby              rhsm                vboxadd-setup.log
boot.log-20210222   grubby_prune_debug  sa                  vboxadd-setup.log.1
```

Рис. 4.4.20. Содержимое каталога /var/log

```
[root@nnk2 log]# cat messages | tail -3
Aug 22 17:20:13 nnk2 systemd: Failed to start Zabbix Agent.
Aug 22 17:20:13 nnk2 systemd: Unit zabbix-agent.service entered failed state.
Aug 22 17:20:13 nnk2 systemd: zabbix-agent.service failed.
```

Рис. 4.4.21. Фрагмент системного журнала

Файлы журналов имеют большой размер, поэтому для их просмотра лучше использовать команды **less** и **tail** (рис. 4.4.22).

```
[root@nnk2 log]# ls -lS
total 8084
-rw-------. 1 root   root   2427473 Feb 14  2021 messages-20210214
-rw-------. 1 root   root   2395204 Aug 22 16:20 messages
-rw-------. 1 root   root   1061508 Aug 20 12:35 messages-20210820
-rw-------. 1 root   root    698102 Feb 22 12:42 messages-20210222
-rw-r--r--. 1 root   root    351568 Aug 22 16:09 lastlog
```

Рис. 4.4.22. Журналы могут иметь размер более 1 мегабайта

Команда **tail -f** отображает новую информацию по мере ее поступления в журнал. Для поиска нужной информации лучше использовать **grep** или **awk** (рис. 4.4.23).

```
[root@nnk2 log]# cat dmesg | grep [Ee]rror
[    0.762339] BERT: Boot Error Record Table support is disabled. Enable it by u
sing bert_enable as kernel parameter.
```

Рис. 4.4.23. Использование регулярных выражений для поиска строк по шаблону

Для чтения бинарных журналов (utmp, wtmp, btmp) используются команды **w**, **who**, **last**, **lastb**, **utmpdump**.

Файлы журналов могут сильно увеличиваться в течение нескольких недель или месяцев и занимать все свободное место на диске. Для решения этой проблемы используется утилита **logrotate**.

Ротация или циклическое ведение журнала подразумевает:

- перемещение файлов журнала на новое имя,
- их архивирование и/или сжатие,
- [иногда] отправку их по электронной почте системному администратору,
- их удаление по мере устаревания

Часто к имени журнала добавляется суффикс с целым числом (рис. 4.4.24).

```
[root@nnk2 log]# ls -l vbox*
-rw-r--r--. 1 root root 475 Sep 30  2020 vboxadd-install.log
-rw-r--r--. 1 root root 168 Dec 28  2020 vboxadd-setup.log
-rw-r--r--. 1 root root 167 Dec 12  2020 vboxadd-setup.log.1
-rw-r--r--. 1 root root  26 Dec 12  2020 vboxadd-setup.log.2
-rw-r--r--. 1 root root 168 Sep 30  2020 vboxadd-setup.log.3
```

Рис. 4.4.24. Пример ротации журналов

Кольцевой буфер ядра

Кольцевой буфер ядра - это структура данных фиксированного размера, которая записывает загрузочные сообщения ядра. В нем регистрируются всех сообщений ядра, которые генерируются при загрузке (syslog в это время еще не доступен).

Хранится в файле `/var/log/dmesg`. По мере добавления новых сообщений самые старые удаляются.

Для просмотра сообщений из кольцевого буфера ядра служит команда **dmesg** (рис. 4.4.25).

```
[root@nnk2 log]# dmesg | grep boot
[    0.000000] smpboot: Allowing 1 CPUs, 0 hotplug CPUs
[    0.173929] smpboot: CPU0: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz (fam: 06,
model: 5e, stepping: 03)
[    0.269463] smpboot: Max logical packages: 1
[    0.269465] smpboot: Total of 1 processors activated (6384.00 BogoMIPS)
```

Рис. 4.4.25. Фрагмент кольцевого буфера ядра

Системный журнал: systemd-journald

С 2015 года **systemd** заменил SysV Init как де-факто системного и сервис-менеджера в большинстве основных дистрибутивов Linux. Демон журнала - **journald** - стал стандартным компонентом журналирования, заменяя системный журнал в большинстве случаев.

Данные больше не хранятся в виде простого текста, но в двоичном виде. Поэтому для чтения логов необходима утилита **journalctl** (рис. 4.4.26).

```
[root@nnk2 log]# journalctl | tail -5
Aug 22 18:02:04 nnk2 systemd[1]: Can't open PID file /run/zabbix/zabbix_agentd.p
id (yet?) after start: No such file or directory
Aug 22 18:02:04 nnk2 systemd[1]: Daemon never wrote its PID file. Failing.
Aug 22 18:02:04 nnk2 systemd[1]: Failed to start Zabbix Agent.
Aug 22 18:02:04 nnk2 systemd[1]: Unit zabbix-agent.service entered failed state.
Aug 22 18:02:04 nnk2 systemd[1]: zabbix-agent.service failed.
```

Рис. 4.4.26. Без параметров journalctl выводит все сообщения

Основные опции команды **journalctl**:

-n, --lines=
Показывает лишь указанное количество последних записей (по умолчанию 10), аналогичен команде **tail**

-k, --dmesg
Показывает лишь сообщения ядра, аналогичен команде **dmesg**

-u, --unit=*UNIT|PATTERN*
Показывает сообщения для указанной службы

-s, --since=, -U, --until=
Позволяет задать промежуток времени, для которого нужно вывести сообщения

-b *[ID],[±offset]*, **--boot=***[ID],[±offset]*
Показывает сообщения для указанной загрузки, по умолчанию для текущей

4.4.12. Архивирование и разархивирование файлов и директорий

Архивирование используются для объединения файлов и каталогов в один файл. Обычно используется для резервного копирования, формирования пакетов исходного кода программного обеспечения и хранения данных.

Сжатие используется для уменьшения объема пространства, используемого конкретным набором данных, для последующего хранения на диске или передачи по сети.

Сжатие бывает двух видов: без потерь и с потерями. Алгоритмы с потерями часто используются для изображений, видео и аудио, когда потеря качества незаметна для людей (например, JPEG, MPEG-4, H.263).

Архивирование и сжатие обычно используются вместе. Некоторые инструменты архивации даже сжимают свое содержимое по умолчанию. Другие могут использовать сжатие как опцию.

Наиболее распространенным инструментом для архивирования файлов в системах Linux является **tar**. Большинство дистрибутивов Linux поставляются с GNU-версией **tar**. Не все версии tar полностью совместимы между собой. Например, формат SUN Solaris tar немного отличается от GNU tar.

Первоначально программа **tar** использовалась для создания архивов на магнитной ленте (**t**ape **ar**chiver), а в настоящее время **tar** используется для хранения нескольких файлов внутри одного файла, для распространения программного обеспечения, а также по прямому назначению — для создания архива файловой системы. **tar** не создаёт сжатых архивов, а использует для сжатия внешние утилиты, такие, как gzip, XZ и bzip2

В архив записывается информация о структуре каталогов, о владельце и группе отдельных файлов, а также временны́е метки файлов, что можно увидеть на рис. 4.4.27.

```
[me@VM2 ~]$ echo "Canada USA Mexico" > NAmerica
[me@VM2 ~]$ echo "France Germany Spain Italy" > Europe
[me@VM2 ~]$ echo "Angola Egypt Lybia Niger" > Africa
[me@VM2 ~]$ echo "Urartu Media Edom Byzantium" > old/ancient
[me@VM2 ~]$ tar -cf world.tar NAmerica Europe Africa old/ancient
[me@VM2 ~]$ tar -tf world.tar
NAmerica
Europe
Africa
old/ancient
[me@VM2 ~]$ cat world.tar
NAmerica0000664000175100017510000000000221377527754701027 2 0ustar  memeCanada USA
 Mexico
Europe0000664000175100017510000000000331377527763701005 4 0ustar  memeFrance Germa
ny Spain Italy
Africa0000664000175100017510000000000311377530016600776 0ustar  memeAngola Egypt
 Lybia Niger
old/ancient0000664000175100017510000000000341377530223201077 1 0ustar  memeUrartu
Media Edom Byzantium
```

Рис. 4.4.27. Содержимое архива tar без сжатия

Основные опции tar:

-c, --create
: Создать новый архив

-t, --list
: Отобразить список файлов и директорий в архиве.

-x, --extract, --get
: Извлечь файлы из архива.

-A, --catenate, --concatenate
: Добавить один архив к концу другого.

-f, --file=ARCHIVE
: Использовать файл или устройство ARCHIVE, если не указан, анализируется содержимое переменной TAPE, если и оно не установлено, используется заданное в коде программы значение

-v, --verbose
: Выводить список обработанных файлов

Опции в **tar** можно задавать в трех стилях, совместное их использование поддерживается, но не рекомендуется

При **традиционном** стиле первым аргументом записывается группа опций, записанная слитно:

```
tar cvf etc.tar /etc
```

В стиле **UNIX** перед каждой опцией записывается дефис, опции без параметров можно объединять. Параметр должен следовать сразу за опцией, к которой он относится, и записываться слитно (пробел допускается только для обязательных параметров, например, для –f):

```
tar -cvf etc.tar /etc
tar -c -v –f etc.tar /etc
```

В стиле **GNU** перед каждой опцией записывается два дефиса, опции имеют осмысленный вид, их можно сокращать, если нет неоднозначности:

```
tar --create --file etc.tar --verbose /etc
tar --cre --file=etc.tar --verb /etc
```

Все примеры выше эквивалентны.

Данные в tar-архиве сохраняют свой размер

Для уменьшения размера архива (сжатия) используется внешняя программа, например, gzip. Опция **–z** на самом деле использует gzip как фильтр для выходных данных

Опции сжатия:

-a, --auto-compress
: Определить программу сжатия по суффиксу архива

-j, --bzip2
: Использовать для сжатия программу bzip2

-J, --xz
: Использовать для сжатия программу xz

--no-auto-compress
: Не определять программу сжатия по суффиксу архива

-z, --gzip, --gunzip, --ungzip
: Использовать для сжатия программу gzip

-Z, --compress, --uncompress
: Использовать для сжатия программу compress

Программа gzip

gzip, **gunzip**, **zcat** - сжать или распаковать файлы

gzip [-acdfhlLnNrtvV19] [-S суффикс] [файл ...]

gunzip [-acfhlLnNrtvV] [-S суффикс] [файл ...]

zcat [-fhLV] [файл ...]

gzip уменьшает размер перечисленных файлов, используя кодирование Лемпеля-Зива (LZ77). Там, где это возможно, каждый файл заменяется архивом с расширением .gz, с таким же владельцем, временем доступа и модификации

Сжатые файлы могут быть восстановлены в первоначальное состояние командой **gzip -d** или **gunzip** или **zcat**. Если первоначальное имя файла, сохранённое в архиве, не допускается файловой системой, в которой он находится, то ему создаётся новое имя из первоначального по правилам этой системы.

gunzip также распознаёт специальные расширения .tgz и .taz как сокращения от .tar.gz и .tar.Z соответственно.

Программа **gzip** используется для сжатия одного или нескольких файлов. Сжатая версия замещает оригинальный файл.

4.4.13. Создание новых текстовых файлов

Для создания новых текстовых файлов можно использовать несколько способов.

1. Команда **cat** с перенаправлением вывода в файл.

2. Команда **touch** создает указанный файл, если он не существует.

3. Символ перенаправления **>** перед именем несуществующего файла создаст пустой файл в локальном каталоге.

Примечание. Примеры из п. 1 и п.3 можно обобщить — вывод любой команды можно перенаправить в новый файл.

4. Текстовые редакторы: **vi**, **vim**, **nano** и другие.

5. Команда **fallocate** позволяет пользователям создавать файлы заданного размера.

Существуют и другие способы создания файлов.

4.4.14. Разрезание и склеивание файлов

Команда **split** позволяет разбивать один большой файл на несколько меньших частей. Данная операция полезна, например, при копировании файла с размером более 4 Гб на файловую систему FAT32 на переносном накопителе.

Следующая команда выполняет разбиение на две равных части:

```
$ split -n 2 -d bigfile
```

Разбиение файла с указанием размера части:

```
$ split -b 100M файл
```

Единицы изменения: K, M, G, T, P, E, Z, Y (или KB, MB и т.д.).

Собрать разбитый на части файл можно с помощью команды **cat**.

```
$ cat x* > restoredbigfile
```

Команда **wc** выводит количество строк (символов новой строки), слов и байтов в каждом файле (рис. 4.4.28).

```
[me@localhost g]$ wc text*
   4   14   80 text1
   2   14   70 text10
   1    4   19 text2
   5    5   39 text3
   7   67  399 text4
   3   12  136 text5
   3   21  111 text6
   2   11   56 text7
   3   16  125 text8
   9   54  330 text9
  39  218 1365 total
```

Рис. 4.4.28 Команда wc

С помощью опций можно ограничить вывод информации (рис. 4.4.29):

-c, --bytes
напечатать количество байтов

-m, --chars
напечатать количество символов

-l, --lines
напечатать количество символов перевода строки (newline, \n), в некотором смысле, количество строк

Здесь специально оговаривается использование символа перевода строки. Очень длинные строки, которые не вмещаются по ширине в окно терминала, только выглядят многострочными

-w, --words
напечатать количество word counts

```
[me@localhost g]$ wc -l text1
4 text1
[me@localhost g]$ wc -c text1
80 text1
```

Рис. 4.4.29 Использование опций в команде wc

Команда **tr** выполняет преобразование, подстановку (замену), сокращение и/или удаление символов, поступающих со стандартного ввода, записывая результат на стандартное устройство вывода (рис. 4.4.30).

Она часто применяется для удаления управляющих символов из файла или преобразования регистра символов.

tr [OPTION]… SET1 [SET2]

Как правило, команде **tr** передаются две строки (набора) символов: первый набор SET1 содержит искомые символы, а второй SET2 - те, на которые их следует заменить.

```
[me@nnk4 g]$ cat text8
Ingredients: horse radish, water, vinegar, sugar, salt,
stabilizers (E415, E412, E410), acidifier E330, pickling
agent E211.
[me@nnk4 g]$ cat text8 | tr a-z A-Z
INGREDIENTS: HORSE RADISH, WATER, VINEGAR, SUGAR, SALT,
STABILIZERS (E415, E412, E410), ACIDIFIER E330, PICKLING
AGENT E211.
```

Рис. 4.4.30. Использование команды tr для перевода всех символов в верхний регистр

Команда **cut** извлекает фрагменты из каждой строки файла и записывает в стандартный вывод:

```
cut [OPTION]… [FILE]…
```

Текст выбирается либо по смещению в символах или байтах (разница существенна только для некоторых языков), либо по разделителям

Режим задается одной из опций (использовать можно только одну из них):

-b, --bytes=LIST
выбрать только эти байты

-c, --characters=LIST
выбрать только эти символы

-f, --fields=LIST
выбрать только эти поля

Режим работы cut по символам, как правило, используется при извлечении фрагментов из текста, созданного компьютерными программами.

```
[me@nnk4 g]$ cat text11
12345678901234567890123456789012345678901234567890
 Planet    Semi-major  Radius    Orbital Satell-
           axis                  period  ites
 Mercury    57909050    2439.7    87.97    0
 Venus     108208000    6051.8   224.70    0
 Earth     149598023    6371.0   365.26    1
 Mars      227939200    3389.5   779.96    2
[me@nnk4 g]$ tail -6 text11 | cut -c2-9,33-40
Planet   Orbital
         period
Mercury   87.97
Venus    224.70
Earth    365.26
Mars     779.96
```

Рис. 4.4.31 Пример извлечения только первой и пятой колонки из текста, где табличный вид был достигнут с помощью пробелов

Поведение команды в режиме работы cut по полям зависит от дополнительных опций:

-d, --delimiter=DELIM
: в качестве разделителя использовать *DELIM* вместо TAB

-s, --only-delimited
: не выводить строки без разделителей

--output-delimiter=STRING
: заменять оригинальные разделители на STRING

```
[me@nnk4 g]$ tail -2 /etc/passwd
zabbix:x:985:979:Zabbix Monitoring System:/var/lib/zabbix:/sbin/nologin
me:x:1201:1201:Myself:/home/me:/bin/bash
[me@nnk4 g]$ tail -2 /etc/passwd | cut -d: -f1,3
zabbix:985
me:1201
[me@nnk4 g]$ tail -2 /etc/passwd | cut -d: -f1,5 --output-delimiter='='
zabbix=Zabbix Monitoring System
me=Myself
```

Рис. 4.4.32. Режим работы cut по полям

Команда **sort** упорядочивает строки в текстовом файле:

sort [OPTION]... [FILE]...

Результат записывается в стандартный вывод.

Если параметр *FILE* не указан или указано –, исходные данные берутся из стандартного ввода.

```
[me@nnk4 g]$ tail -4 text11
 Mercury    57909050    2439.7    87.97    0
 Venus     108208000    6051.8   224.70    0
 Earth     149598023    6371.0   365.26    1
 Mars      227939200    3389.5   779.96    2
[me@nnk4 g]$ tail -4 text11 | sort
 Earth     149598023    6371.0   365.26    1
 Mars      227939200    3389.5   779.96    2
 Mercury    57909050    2439.7    87.97    0
 Venus     108208000    6051.8   224.70    0
```

Рис. 4.4.33. Команда sort без опций в качестве ключа сортировки использует всю строку

Опции сортировки команды **sort**:

-b, --ignore-leading-blanks
: игнорировать пробелы в начале строки

-f, --ignore-case
не учитывать регистр символов

-g, --general-numeric-sort
проводить сравнение строк как числовых значений общего вида

-h, --human-numeric-sort
проводить сравнение чисел вида 2K 1G по их значениям

-n, --numeric-sort
проводить сравнение строк как числовых значений

-r, --reverse
сортировка по убыванию

Команда **uniq** удаляет повторяющиеся строки и выводит результат в стандартный вывод.

Повторяющиеся строки удаляются только в том случае, если они следуют непосредственно друг за другом, поэтому обычно данные сначала сортируют (команда **sort -u** тоже удаляет дубликаты).

```
[me@nnk4 u]$ cat forest.txt
pine
birch
aspen
pine
aspen
pine
[me@nnk4 u]$ cat forest.txt | uniq
pine
birch
aspen
pine
aspen
pine
[me@nnk4 u]$ cat forest.txt | sort | uniq
aspen
birch
pine
```

Рис. 4.4.34. В первом случае дубликаты не удалены, потому что они не смежны

Опции команды **uniq**:

-c Вывести список повторяющихся строк с указанием количества

-d Вывести только повторяющиеся (без уникальных) строки

-u Вывести только уникальные строки (режим по умолчанию)

-i Игнорировать регистр символов при сравнении строк

-f *n* Пропустить *n* полей от начала строки

-s *n* Пропустить *n* символов от начала строки

-w, --check-chars=*N*

сравнивать не более *N* символов в каждой строке

Команда **head** выводит несколько первых строк файла, команда **tail** – несколько последних строк, по умолчанию – десять.

С помощью опции **-n** можно задать нужное количество строк.

```
[me@nnk4 u]$ ls -Sl /usr/bin | wc -l
1760
[me@nnk4 u]$ ls -Sl /usr/bin | head -n 4
total 152960
-rwxr-xr-x. 1 root root    7565240 Nov 16 19:18 crash
-rwxr-xr-x. 1 root root    6826488 Sep 30 19:18 gdb
-rwxr-xr-x. 1 root root    5354072 Oct  1 19:37 ld.gold
```

Рис. 4.4.35. Вывод первых четырех строк из файла, содержащего 1760 строк

Команда **tail** с опцией **-f** переводит команду в режим слежения за файлом – добавляемые в конец файла данные тут же отображаются; чаще всего этой опцией пользуются для слежения за файлами журналов, например:

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

или

```
tail -f /var/log/nginx/error.log
```

4.4.15. Быстрый анализ текстов

Регулярные выражения – это символическая форма записи для шаблонов в тексте.

Есть много диалектов регулярных выражений. В этом курсе будем рассматривать только регулярные выражения, которые определены в стандарте POSIX и поддерживаются большинством инструментов командной строки.

Некоторые символы в регулярных выражениях получают специальное значение. Такие символы называют метасимволами:

**^ $. [ ] { } - ? * + ( ) | **

Все остальные символы называются литеральными или литералами.

Можно заметить, что все алфавитно-цифровые символы являются литералами.

Стандарт POSIX описывает два вида регулярных выражений:

- **Простые регулярные выражения** (Basic Regular Expressions, **BRE**)
- **Расширенные регулярные выражения** (Extended Regular Expressions, **ERE**)

В простых регулярных выражениях только следующие символы считаются метасимволами:

^ $. [ ] *

В расширенных регулярных выражениях метасимволами также считаются следующие символы :

( ) { } ? + |

Чтобы в BRE использовать метасимволы из ERE, нужно перед ними поставить символ экранирования ****

BRE: **`[0-9]\{1,3\}`** Эквивалент в ERE: **`[0-9]{1,3}`**

Команда grep

Команды **grep**, **egrep**, **fgrep** выводят строки, соответствующие образцу:

grep [OPTIONS] PATTERN [FILE...]

grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

Образец (*PATTERN*) может интерпретироваться по-разному:

-G, --basic-regexp
как простое регулярное выражение (BRE), режим по умолчанию

-E, --extended-regexp
как расширенное регулярное выражение (ERE)

-F, --fixed-strings
как список строк, разделенных символом новой строки

Команда **egrep** соответствует **grep -E**

Команда **fgrep** соответствует **grep -F**

Некоторые опции команды grep:

-i, --ignore-case
игнорировать различия в регистре символов

-v, --invert-match
выбирать только несовпадающие строки

-w, --word-regexp

Выбирать только строки, содержащие слово целиком

-c, --count

Вместо обычного вывода печатает только количество совпадающих строк

-r, --recursive

Просматривает рекурсивно все файлы в директории и поддиректориях. Без параметра *FILE* работает с текущей директорией

Символ **^** обозначает начало строки, символ **$** - конец строки. Такие символы называют якорями (anchors) (рис. 4.4.36):

```
[me@nnk4 g]$ cat text3
ant
anteater
antelope
elephant
panther
[me@nnk4 g]$ grep ant text3
ant
anteater
antelope
elephant
panther
[me@nnk4 g]$ grep ^ant text3
ant
anteater
antelope
[me@nnk4 g]$ grep ant$ text3
ant
elephant
[me@nnk4 g]$ grep ^ant$ text3
ant
```

Рис. 4.4.36 Использование якорей в регулярных выражениях (^ant возвращает только строки, начинающиеся на ant, ant$ - заканчивающиеся; использование обоих якорей возвращает только строки ant)

Метасимвол точка . соответствует любому символу в данной позиции/

Выражение в квадратных скобках означает, что в данной позиции должен быть один из перечисленных символов

Если первый символ после открывающей скобки ^, то остальные символы между скобок интерпретируются как недопустимые. Можно задать диапазон символов с помощью символа -.

Часто используют диапазон **[A-Z]** для прописной буквы, **[a-z]** – для строчной буквы, **[A-Za-z]** для любой буквы, **[0-9]** – для любой цифры.

Квантификаторы

Квантификатор позволяет указать количество повторений предыдущего символа, символьного класса или группы в шаблоне

Самый простой квантификатор – число в фигурных скобках: *{n}*

A{4} AAAA

[0-9]{3}-[0-9]{4} номер телефона в формате 123-4567

[0-9]{7}[АВСЕНКМ][0-9]{3}РВ[0-9] идентификационный номер гражданина Республики Беларусь

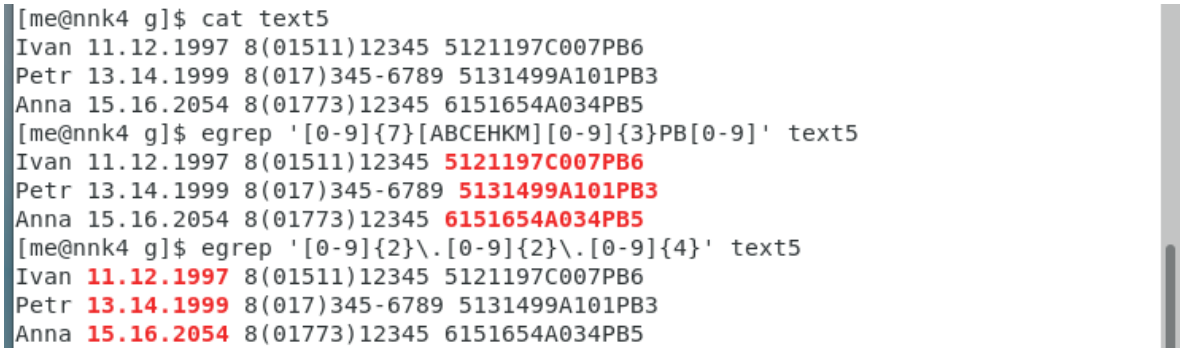

Рис. 4.4.37. Пример нахождения идентификационных номеров и дат в текстовых файлах

Более общее выражение для квантификаторов *{n,m}*

Цифры в скобках означают, что в искомом тексте может стоять подряд не менее n (или ноль, если n не задано), но и не более m (или любое количество, если m не задано) выражений, записанных непосредственно перед квантификатором:

{,5} – не более пяти

{2,} – не менее двух

{2,3} – от двух до пяти

Звездочка ***** обозначает необязательный элемент, это сокращенная запись для **{0,}**. Элемент может встретиться любое количество раз или отсутствовать совсем

Метасимвол **+** требует совпадения с предыдущим элементом хотя бы один раз, то есть, элемент обязательный, но может повторяться. Соответствует квантификатору **{1,}**.

Единичный необязательный символ обозначается метасимволом **?**. Соответствует квантификатору **{0,1}**.

4.4.16. Администрирование

TBD.

4.4.17. Получение сведений о системе

Каждый процесс в системе потенциально может потреблять системные ресурсы. Так называемая загрузка системы пытается объединить общую загрузку системы в один числовой индикатор. Вы можете увидеть текущую загрузку с помощью команды **uptime**: (рис. 4.4.38).

Рис. 4.4.38. Текущая загрузка системы, три последних числа обозначают среднюю загрузку системы: за последнюю минуту (0.38), за последние пять минут (0.23) и за последние пятнадцать минут (0.15) соответственно.

Каждое из этих чисел указывает, сколько процессов ожидали завершения ресурсов процессора или операций ввода / вывода. Это означает, что эти процессы были готовы к запуску, если бы они получили соответствующие ресурсы.

Каталог **/proc** – виртуальная файловая система, его содержимое не записывается на диск, а находится в памяти

Динамически заполняется каждый раз при загрузке компьютера и постоянно отражает текущее состояние системы

Каталог **/proc** содержит информацию о:

- выполняющихся процессах

- конфигурации ядра
- аппаратном обеспечении

Файл **/proc/cpuinfo** «хранит» информацию о процессоре системы.

Файл **/proc/cmdline** хранит строки, переданные ядру при загрузке.

Файл **/proc/modules** показывает список модулей, загруженных в ядро.

Каталог **/proc/sys** содержит параметры конфигурации ядра.

Файлы распределены по категориям, для каждой категории создан свой подкаталог.

Большинство этих параметров имеют только два значения:

- **0** – выключено
- **1** – включено

В файле **/proc/meminfo** можно найти всю информацию об использовании памяти.

Команда **free** отображает количество свободной и использованной памяти (рис. 4.4.39).

```
[root@nnk2 boot]# free
              total        used        free      shared  buff/cache   available
Mem:        3880404      884660     1431020       40664     1564724     2701908
Swap:       4833272           0     4833272
```

Рис. 4.4.39. Вывод команды free

Значение столбов в выводе команды **free**:

- **total** – общий объем установленной памяти
- **used** – объем используемой памяти (**total** – **free** – **buff/cache**)
- **free** – объем неиспользуемой памяти
- **shared** – эта память используется в основном tmpfs
- **buff/cache** – объем физической памяти, используемой в настоящее время буферами ядра, кэшем страниц и слэбами
- **available** – объем физической памяти, доступной для новых процессов

4.4.18. Управление процессами

Процессы существуют в иерархии: после загрузки ядра в память запускается первый процесс (init или systemd), который, в свою очередь, запускает другие процессы, которые, опять же, могут запускать другие процессы.

Каждый раз, когда пользователь вводит команду, запускается программа и генерируется один или несколько процессов.

Каждый процесс имеет уникальный идентификатор (PID) и идентификатор родительского процесса (PPID). Это положительные целые числа, которые назначаются в последовательном порядке.

```
[devops@nnk2 ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
devops    6811  3309  0 12:11 pts/1    00:00:00 bash
devops   32679  6811  0 20:55 pts/1    00:00:00 ps -f
```

Рис. 4.4.40 Список процессов выведен командой ps в режиме полного вывода

Команда **top** динамически отображает все запущенные процессы:

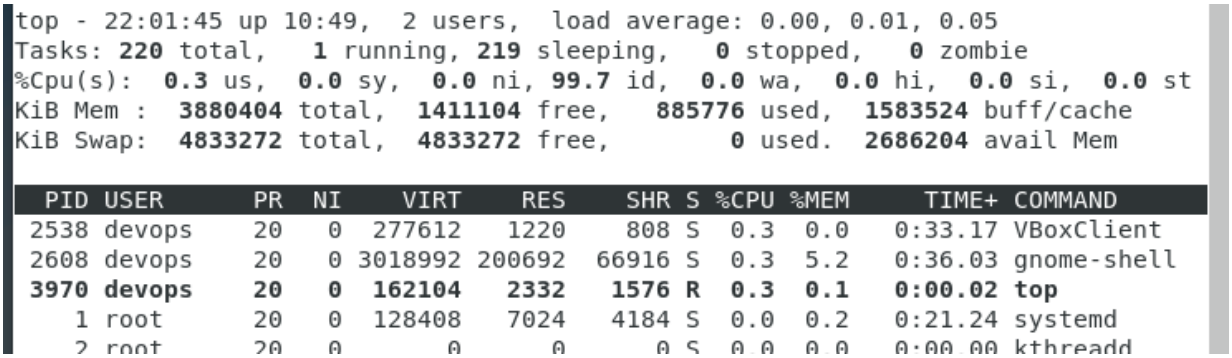

Рис. 4.4.41 Список процессов выведен командой top

Команды **l t m** управляют отображением строк статистики. Для сортировки служат команды **M** (память), **N** (ID процесса), **T** (время выполнения), **P** (процент загрузки процессора). Изменение порядка сортировки – **R**.

Альтернативные команды – **htop** и **atop**.

Команда ps выводит статическую информацию о процессах. Без опций выводит только процессы, относящиеся к текущей оболочке.

Команда поддерживает три стиля параметров, причем они не совпадают.

Например, **ps aux** не эквивалентна **ps –aux**, но **ps -e** эквивалентна **ps ax**.

Иерархию процессов можно увидеть с помощью команды **pstree**.

Завершить процесс можно с помощью команды **kill**.

4.4.19. Выполнение задач в фоновом режиме

Для выполнения команды в фоновом режиме достаточно добавить в конце символ амперсанда **&**.

В выводе терминала будут отображены порядковый номер задачи (в квадратных скобках) и идентификатор процесса.

Работая в фоновом режиме, команда все равно продолжает выводить сообщения в терминал, из которого была запущена. Для этого она использует потоки stdout и stderr, которые можно закрыть при помощи следующего синтаксиса:

```
$ command > /dev/null 2>&1 &
```

Узнать состояние всех остановленных и выполняемых в фоновом режиме задач в рамках текущей сессии терминала можно при помощи утилиты **jobs** с использованием опции **-l**. Вывод содержит порядковый номер задачи, идентификатор фонового процесса, состояние задачи и название команды, которая запустила задание.

В любое время можно вернуть процесс из фонового режима на передний план. Для этого служит команда **fg**. Если в фоновом режиме выполняется несколько программ, следует также указывать номер. Например:

```
$ fg %1
```

Для завершения фонового процесса применяют команду **kill** с номером программы.

Если изначально процесс был запущен обычным способом, его можно перевести в фоновый режим, выполнив следующие действия:

- Остановить выполнение команды, нажав комбинацию клавиш **Ctrl+Z**.
- Перевести процесс в фоновый режим при помощи команды **bg**.

4.4.20. Изменение приоритетов выполняющихся программ

Утилита **nice** — программа, предназначенная для запуска процессов с изменённым приоритетом nice. Приоритет nice (целое число) процесса используется планировщиком процессов ядра ОС при распределении процессорного времени между процессами

Приоритет nice — число, указывающее планировщику процессов ядра ОС приоритет, который пользователь хотел бы назначить процессу.

Утилита **nice**, запущенная без аргументов, выводит приоритет nice, унаследованный от родительского процесса. nice принимает аргумент «смещение» в диапазоне от -20 (наивысший приоритет) до +19 (низший приоритет). Если указать смещение и путь к исполняемому файлу, утилита **nice** получит приоритет своего процесса, изменит его на указанное смещение и использует системный вызов семейства **exec**() для замещения кода своего процесса кодом из указанного исполняемого файла. Команда nice сделает то же, но сначала выполнит системный вызов семейства **fork**() для запуска дочернего процесса (sub-shell). Если смещение не указано, будет использовано смещение +10. Привилегированный пользователь (root) может указать отрицательное смещение.

Планировщик процессов ядра ОС Linux поддерживает приоритеты от 0 (реальное время) до 139 включительно. Приоритеты -20…+19 утилиты или команды **nice** соответствуют приоритетам 100…139 планировщика процессов. Другие приоритеты планировщика процессов можно установить командой **chrt** из пакета util-linux.

Посмотреть приоритет процессов можно например с помощью утилиты **top**.

Для того, чтобы изменить приоритет у существующего процесса (т.е. такого процесса, который ранее был уже запущен), необходимо воспользоваться командой:

```
$ renice [значение приоритета] -p [id процесса]
```

У запущенной программы с помощью команды renice можно изменить назначенный приоритет. Предположим, что есть работающая программа yes со значением nice 10. Чтобы изменить его значение, можно использовать команду renice со значением nice и PID процесса. Изменим значение nice на 15:

```
$ renice -n 15 -p 21349
21349 (process ID) old priority 0, new priority 15
```

Согласно правилам, обычный пользователь может только увеличивать значение nice (уменьшать приоритет) любого процесса.

4.4.21. Изучение базовых прав доступа

Перенесено в тему 5.3.

4.4.22. Добавление и удаление пользователей

Информация о пользователях и группах хранится в текстовых файлах `/etc/passwd`, `/etc/shadow`, `/etc/group` и `/etc/gshadow`. Каждый пользователь имеет уникальный числовой идентификатор (**UID** – User Identification Number). Каждая группа также имеет уникальный числовой идентификатор (**GID** – Group Identification Number). Каждый пользователь имеет основной (primary) GID, а также может быть членом других групп.

Для имени пользователя рекомендуется использовать строчные буквы, цифры и символы подчеркивания и дефиса (первый символ – строчная буква или подчеркивание), в качестве завершающего символа можно использовать доллар. На языке регулярных выражений: `[a-z_][a-z0-9_-]*[$]?`. Максимальная длина имени 32 символа.

Пользователь root имеет UID 0. Диапазон 1..99 используется для предопределенных UID. Файл `/etc/login.defs` задает диапазон системных и пользовательских UID (рис. 4.4.42).

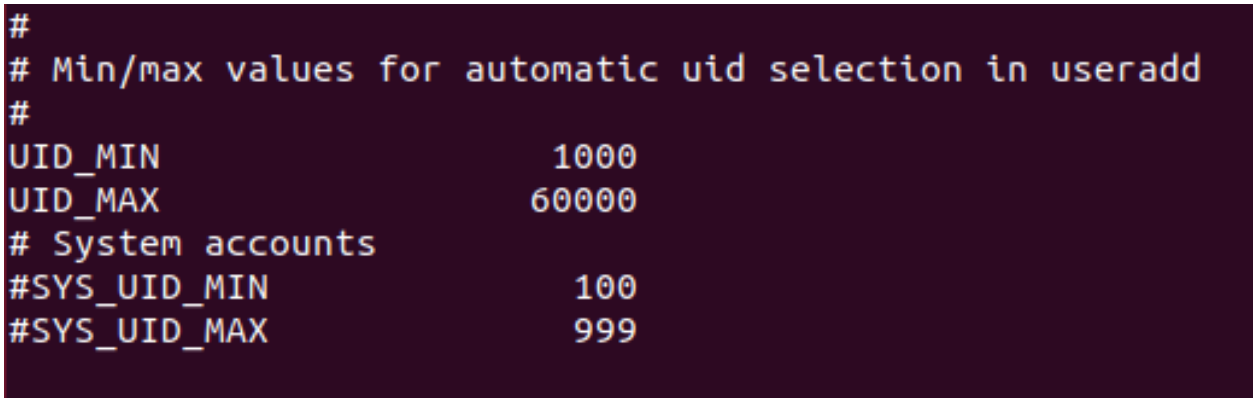

Рис. 4.4.42. Фрагмент файла /etc/login.defs

UID≥1000 соответствуют обычным (непривилегированным) пользователям (в некоторых устаревших системах ≥500).

В Linux учетной записью **суперпользователя** является root, UID которой всегда 0. Иногда суперпользователя называют системным администратором. Он имеет неограниченный доступ и контроль над системой, включая других пользователей. Группа по умолчанию для суперпользователя имеет GID 0 и также называется root. Домашний каталог суперпользователя - это выделенный каталог верхнего уровня `/root`, доступный только самому пользователю root.

Системные учетные записи (System accounts) обычно предварительно создаются во время установки системы. Они предназначены для объектов, программ и служб, которые не должны работать от имени суперпользователя. Как правило, это средства операционной системы. Они имеют UID в диапазоне или 1..99, или 500..999. Большинство системных учетных записей в Linux никогда не входят в систему, в качестве оболочки для входа установлена `/sbin/nologin`. Домашний каталог не назначен или находится не в папке `/home`. Эти учетные записи обычно имеют ограниченные или, чаще всего, не имеют никаких привилегий.

Учетные записи служб обычно создаются во время установки и настройки служб. Подобно системным учетным записям, они предназначены для средств, программ и служб, которые не должны работать от имени суперпользователя. В большинстве случаев учетные записи служб имеют UID≥1000. Но некоторые службы используют предопределенные UID (рис. 4.4.43).

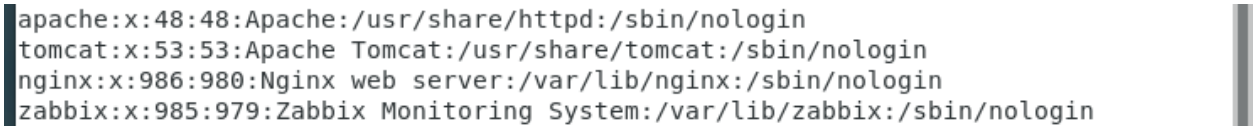

Рис. 4.4.43. Фрагмент файла /etc/passwd с учетными записями служб

В каталоге `/etc` хранятся файлы конфигурации операционной системы, включая информацию о пользователях и группах

`/etc/passwd`

`/etc/group`

`/etc/shadow`

`/etc/gshadow`

`/etc/sudoers`

`/etc/sudoers.d`

Файл «паролей» /etc/passwd

Каждая строка содержит несколько полей, всегда разделенных двоеточием.

Несмотря на название, хэш пароля в настоящее время в нём не хранится (он хранится в файле `/etc/shadow`).

Типичный синтаксис строки в этом файле:

`USERNAME:PASSWORD:UID:GID:GECOS:HOMEDIR:SHELL`

где:

USERNAME	имя пользователя (логин)
PASSWORD	Устаревшее расположение хэша пароля. Почти всегда **x** указывает на то, что пароль хранится в файле /etc/shadow
UID	UID пользователя
GID	GID основной группы пользователя
GECOS	CSV-список информации о пользователе, включая имя, адрес, номер телефона. например: Ivan Ivanov,Lenina 1-23,(17)1234567
HOMEDIR	Путь к домашнему каталогу пользователя
SHELL	Shell по умолчанию для этого пользователя

Файл /etc/shadow

Файл /etc/shadow содержит одну запись в каждой строке, каждая из которых представляет собой учетную запись пользователя [3]. Вы можете просмотреть содержимое файла с помощью текстового редактора или команды, такой как **`cat`**:

```
sudo cat /etc/shadow
```

Как правило, первая строка описывает пользователя root, затем системные и обычные учетные записи пользователей. Новые записи добавляются в конец файла.

Каждая строка файла `/etc/shadow` содержит девять полей, разделенных запятыми:

```
mark:$6$.n.:17736:0:99999:7:::
[--] [----] [---] - [---] ----
|       |       |   |   |   |||+-----------> 9. Неиспользованный
|       |       |   |   |   ||+------------> 8. Срок годности
|       |       |   |   |   |+-------------> 7. Период бездействия
|       |       |   |   |   +--------------> 6. Период предупреждения
|       |       |   |   +------------------> 5. Максимальный возраст пароля
|       |       |   +----------------------> 4. Минимальный возраст пароля
|       |       +--------------------------> 3. Последнее изменение пароля
|       +----------------------------------> 2. Зашифрованный пароль
+------------------------------------------> 1. Имя пользователя
```

1. **Имя пользователя**. Строка, которую вы вводите при входе в систему. Учетная запись пользователя, которая существует в системе.
2. **Зашифрованный пароль**. Пароль использует формат $type$salt$hashed. $type является методом криптографического алгоритма хеширования и может иметь следующие значения:
   - 1 — MD5
   - $2a$ — Blowfish
   - $2y$ — Eksblowfish
   - 5 — SHA-256
   - 6 — SHA-512

   Если поле пароля содержит звездочку (*) или восклицательный знак (!), пользователь не сможет войти в систему с использованием аутентификации по паролю. Другие методы входа, такие как аутентификация на основе ключей или переключение на пользователя, по-прежнему разрешены.

В старых системах Linux зашифрованный пароль пользователя хранился в файле `/etc/passwd`.

3. **Последнее изменения пароля**. Это дата последнего изменения пароля. Количество дней исчисляется с 1 января 1970 года (дата эпохи).
4. **Минимальный срок действия пароля**. Количество дней, которое должно пройти, прежде чем пароль пользователя может быть изменен. Как правило, он установлен на ноль, что означает отсутствие минимального срока действия пароля.
5. **Максимальный срок действия пароля**. Количество дней после смены пароля пользователя. По умолчанию этот номер установлен на 99999.
6. **Период предупреждения**. Количество дней до истечения срока действия пароля, в течение которого пользователь получает предупреждение о необходимости изменения пароля.
7. **Период бездействия**. Количество дней после истечения срока действия пароля пользователя до отключения учетной записи пользователя. Обычно это поле пустое.
8. **Срок хранения**. Дата, когда учетная запись была отключена. Это представляется как дата эпохи.
9. **Неиспользованный**. Это поле игнорируется. Оно зарезервированно для будущего использования.

Файл `/etc/shadow` не стоит редактировать вручную, если вы не знаете, что вы делаете. Всегда используйте команду, которая предназначена для этой цели. Например, чтобы изменить пароль пользователя, используйте команду **passwd**, а для изменения информации об устаревании пароля используйте команду **chage**.

Для получения информации о пользователях можно использовать большое количество команд. Текущую информацию о пользователе можно получить с помощью команды **id**. Список удачных входов пользователей отображает команда **last**, неудачных – **lastb**. Команды **who** и **w** отображают только активные в настоящий момент имена входа в систему и способ входа. Для переключения сеанса на другого пользователя (чаще всего суперпользователя root) служит команда **su**. Для выполнения только одной команды от имени суперпользователя можно применить команду **sudo**.

Команда **id** выводит идентификаторы пользователя (UID), основной группы (GID), а также других групп, в которых состоит пользователь USER:

id [OPTION]... [USER]...

Если аргумент USER не задан, будет выведена информация о текущем пользователе. Команда не требует привилегий

Команда **w** выводит следующую информацию:

- текущее время и как долго система работала
- сколько пользователей подключено
- средняя нагрузка (load averages) за последние 1, 5 и 15 минут

А также дополнительную информацию для каждого активного сеанса пользователя:

- время загрузки процессора (IDLE, JCPU и PCPU)
- текущий процесс (например, -bash). Общее время загрузки CPU этого процесса - последний элемент (PCPU)

Создание пользователей

Для создания учетной записи пользователя используется команда **useradd** (на Debian обычно используется **adduser**):

useradd [options] LOGIN

В зависимости от опций, может быть создана домашняя папка, установлены разрешения, скопированы файлы, также может быть создана группа для пользователя.

По умолчанию новый пользователь получает первый свободный UID из автоматического диапазона, то есть, 1000, 1001, 1002 и т. д.

Далее перечислены некоторые опции команды **useradd**.

-u, --uid *UID*
С помощью опции **-u** команды **useradd** можно явно задать UID

По умолчанию для пользователя также создается основная (primary) группа с таким же именем и новым GID.

-g, --gid *GROUP*
С помощью опции **-g** команды useradd можно явно задать GID для создаваемой основной группы

-N, --no-user-group
Не создавать группу с именем пользователя, использовать группу из опции **-g**.

Пользователь имеет основную группу, она указывается при создании, а также несколько дополнительных. Основная группа отличается от обычных тем, что все файлы в домашнем каталоге пользователя имеют эту группу в качестве группы-владельца, и при ее смене группа для этих каталогов тоже поменяется. Также именно эту группу получают все файлы, созданные пользователем. Дополнительные группы нужны, чтобы мы могли разрешить пользователям доступ к разным ресурсам, добавив его в эти группы в linux.

-G, --groups *GRP1[,GRP2,...[,GRPN]]]*
Список дополнительных групп, в которые нужно включить пользователя, задается через запятую, без пробелов

По умолчанию, создается папка /home/LOGIN в соответствии со значением переменной HOME в файле /etc/default/useradd

-b, --base-dir BASE_DIR
в качестве домашней будет создана папка BASE_DIR/LOGIN

-d, --home-dir HOME_DIR
в качестве домашней будет использоваться папка HOME_DIR, эту папку нужно создавать самостоятельно

-m, --create-home

-M, --no-create-home

-c, --comment COMMENT
Можно указать любую текстовую строку, обычно используется для полного имени пользователя. Пример:

useradd -c "Mark Twain" mark

Для изменения этой информации используется команда **chfn** (**ch**ange **f**i**n**ger information)

Учетной записи должен быть назначен пароль

passwd LOGIN

Любой пользователь может изменить свой пароль, но только root может изменить пароль любого пользователя

Удаление учетных записей

Для удаления учетной записи пользователя можно использовать команду **userdel**.

Эта команда обновляет информацию, хранящуюся в базах данных учетной записи, удаляя все записи, относящиеся к указанному пользователю.

Параметр **-r** также удаляет домашний каталог пользователя и все его содержимое, а также почтовую папку пользователя.

Другие файлы, расположенные в других местах, необходимо искать и удалять вручную.

Создание групп

Для создания группы используется команда **groupadd**:

```
groupadd [options] group
```

Некоторые опции:

-g, --gid GID
числовое значение идентификатора группы (GID)

-r, --system
создать системную группу

Информация о группах хранится в файлах `/etc/group` и `/etc/gshadow`.

4.4.23. Создание и выполнение shell-программ

Скрипт (сценарий) – это текстовый файл, который содержит команды для оболочки. Его можно запустить на выполнение, при этом по очереди будут выполняться содержащиеся в нем команды.

Скрипт не компилируется, а интерпретируется. То есть каждая команда анализируется перед каждым выполнением. Это ускоряет разработку скрипта, но делает его работу медленнее (по сравнению с скомпилированной программой.

Скрипты помогают автоматизировать выполнение сложных рутинных задач, при этом уменьшается вероятность ошибок. Их можно сохранять в системе управления версиями для документирования сделанных изменений

Традиционно скрипты имеют расширение вида .sh, .bash и т.п., а первая строка содержит строку, начинающуюся с сочетания символов **#!** (shebang, шебанг) и указывающую путь к программе, например:

```
#!/bin/bash
```

Чтобы сделать скрипт исполняемым, нужно или установить специальный флаг в разрешениях файла

```
$ sudo chmod +x скрипт.sh
```

или поместить его в папку, путь к которой находится в переменной **PATH**.

В первом случае при выполнении скрипта нужно указывать путь к файлу скрипта, даже если он находится в текущей директории.

В Bash все после символа хеш **#** (решетка) и до конца строки считается комментарием. Комментарии должны быть краткими и точными. Не нужно комментировать очевидное. Но, например, рекомендуется пояснить регулярные выражения. Комментарии также полезны при отладке скрипта: можно временно закомментировать некоторые строки или блоки кода.

Короткий скрипт можно создать и с помощью команд **cat** и **echo**. Но сложные скрипты удобнее разрабатывать в редакторах. Большинство редакторов подсвечивают элементы синтаксиса скрипта, что позволяет избежать многих ошибок.

В этом курсе рассмотрим два редактора: **nano** и **vim**:

- **nano** – более простой и легкий в использовании редактор, в нижней части экрана располагается меню с основными командами
- **vim** имеет несколько режимов работы и большое количество команд, но и возможностей у него намного больше

На некоторых системах есть только более старая версия этого редактора, называемая **vi**.

При наличии полной установки **vim**, **vi** будет просто алиасом для **vim**. При попытке запустить **vi** запустится **vim** в режиме совместимости.

В коде скрипта можно обращаться с помощью специальных переменных **$1**, **$2**, … **$9**, **${10}**, **${11}**, …

Количество аргументов доступно в специальной переменной **$#**

Команда **shift** сдвигает аргументы на один влево, то есть, первый становится недоступным, второй становится первым и т. д. Таким образом можно легко организовать перебор всех аргументов

Если аргумент содержит пробелы, его нужно заключать в кавычки или использовать символ экранирования ****

Оператор **if / then** проверяет, равен ли нулю код завершения списка команд после **if**, и если это так, то выполняет одну, или более, команд, следующих за словом **then**

```
if команды; then
    команды
[elif команды; then
    команды ]…
[else
    команды ]
fi
```

Дополнительные проверки можно выполнить с помощью **elif**. Если ни одно из условий в **if** / **elif** не выполнено – выполняются команды, следующие за **else**

Команда **test** может выполнять множество различных сравнений

```
test выражение
```

Чаще всего используется псевдоним **[**

[выражение]

Команда **test** возвращает код завершения 0, если *выражение* истинно, и код завершения 1, если *выражение* ложно

Существуют несколько классов выражений:

- выражения для проверки файлов
- выражения для проверки строк
- выражения для проверки чисел

В последних версиях bash есть дополнительные выражения для проверки условий:

- *[[выражение]]* улучшенная замена test
- *((выражение))* для работы с целыми числами

Выражения для проверки файлов:

-e файл	файл существует
-d файл	файл существует и является директорией
-f файл	файл существует и является обычным файлом
-s файл	файл существует и имеет размер больше нуля
-r файл	файл существует и доступен для чтения
-w файл	файл существует и доступен для записи
-x файл	файл существует и доступен для чтения

Выражения для проверки строк:

строка	строка не пустая
-n строка	Длина строки больше нуля
-z строка	длина строки равна нулю
строка1 = строка2	строки равны
строка1 == строка2	
строка1 != строка2	строки не равны
строка1 > строка2	строка1 в алфавитном порядке стоит позже строки2
строка1 < строка2	строка1 в алфавитном порядке стоит раньше строки2

Выражения для проверки чисел:

число1 -eq число2	число1 и число2 равны
число1 -ne число2	число1 и число2 не равны
число1 -lt число2	число1 меньше, чем число2
число1 -gt число2	число1 больше, чем число2
число1 -le число2	число1 меньше, чем число2 или равно
число1 -ge число2	число1 больше, чем число2 или равно

Другие методы проверки условий

Для проверки условий можно использовать улучшенную команду test

[[выражение]]

Эта команда поддерживает те же выражения, что и test, но дополнительно можно использовать следующее выражение для проверки строк:

строка =~ регулярное_выражение

Вторая команда используется для проверки арифметических выражений на истинность:

((выражение))

Например, можно проверить, представляет ли строка целое число

```
[[ "$X" =~ ^-?[[:digit:]]+$ ]]
```

или является ли число четным

```
(( (( X % 2 )) == 0 ))
```

Коды завершения

Команды, а также скрипты и функции, при завершении работы возвращают оболочке целочисленное значение, называемое кодом завершения

Код завершения может принимать значение от 0 до 255

Значение **0** означает нормальное завершение, остальные – ошибку

Код завершения последней выполненной команды можно получить с помощью переменной **$?**

Команда **true** всегда возвращает код завершения **0**, команда **false** – **1**

Стандартные коды завершения

- **0** – успешное завершение
- **1** – остальные ошибки
- **2** – неправильное использование команды или ошибка ввода-вывода
- **126** – команда найдена, но не является исполняемой
- **127** – команда не найдена
- **128** – неверный код выхода
- **128+n** – фатальная ошибка по сигналу n (например, 137 после **kill -9**)
- **130** – команда завершена по Ctrl-C (128 + 2)
- **255** – out of range (код завершения вне диапазона 0..255)

Цикл for

Цикл **for** позволяет выполнять последовательность команд многократно для каждого элемента списка

В современных версиях bash реализованы две формы команды **for**:

традиционная:

```
for переменная [in список]; do
    команды
done
```

и в стиле языка C:

```
for (( выражение1; выражение2; выражение3 )) ; do
 команды
done
```

Для преждевременного выхода из цикла применяют оператор **break**

Для пропуска оставшейся части тела цикла и перехода на новую итерацию применяют оператор **continue**

Цикл **for** в традиционной форме имеет следующий синтаксис

```
for переменная [in список]; do
    команды
done
```

Здесь *переменная* - это имя переменной, которая будет изменяться при выполнении цикла, *список* – необязательный список элементов, которые будут последовательно присваиваться переменной *переменная*, а *команды* - это команды, выполняемые в каждой итерации

Команду **for** в стиле языка C удобно использовать при работе с числовыми последовательностями

```
for (( initializer; condition; step )) ; do
 команды
done
```

Можно реализовать бесконечный цикл

```
for (( ; ; )) ...
```

В качестве альтернативы можно использовать традиционную форму **for** совместно с командой **seq** или использовать новый синтаксис.

Список использованных источников

1. Основные принципы функционирования ОС Linux
https://studopedia.ru/3_40867_osnovnie-printsipi-funktsionirovaniya-os-Linux.html

2. Краткое описание Linux и его компонентов
https://linux-user.ru/distributivy-linux/kratkoe-opisanie-linux-i-ego-komponentov/

3. Понимание файла /etc/shadow
https://andreyex.ru/linux/ponimanie-fajla-etc-shadow/