Оглавление

О тексте

В данном материале используются следующие обозначения и шрифты:

- Основной текст набирается обычным шрифтом.
- Примеры команд и исходного кода набираются моноширинным шрифтом, например

  ```
  $ echo "Hello world"
  ```

  Символ "$" в примере выше (или весь текст до первого "$") – приветствие командной строки, текст после него является командой для командной оболочки, например Bash. Если вместо "$" указан символ "#", значит указанная команда (например, остановка системной службы) требует особых привилегий для выполнения.
- Верхним регистром в примерах команд обозначаются сами команды, их аргументы или опции, подставляемые пользователем, например путь к файлу вместо FILE в

  ```
  $ cat FILE
  ```

  и название любой команды в примере опции вывода справки, доступной для многих команд

  ```
  $ COMMAND --help
  ```
- В квадратные скобки в примерах команд заключаются необязательные аргументы, многоточие указывает на допустимость нескольких схожих аргументов, например

```
ls [OPTIONS]... [FILE]...
```

означает, что допустима конструкция с несколькими опциями (`-a`, `-l`) и аргументами (`dir1`, `dir2`)

```
$ ls -l -a dir1 dir2
```

- В угловые скобки "<" и ">" в примерах заключаются горячие клавиши, например при использовании Tab для автодополнения команды.
- Списки, разделённые символом "|", используются для перечисления нескольких вариантов использования (чаще всего короткая и подробная запись опций). Иногда такие списки заключаются в фигурные скобки "{" и "}". Например

```
rpm {-i|--install} [OPTIONS] PACKAGEFILE...
```

Замечания, найденные ошибки, предложения и вопросы можно направлять с пометкой в теме "[linux course]" по контактному email df6.626@gmail.com, Винокуров Дмитрий.

Введение в ОС Linux

О Linux[1]

Linux (произносится "ли́нукс") – общее название Unix-подобных операционных систем (ОС) на основе одноимённого ядра и собранных для него библиотек и системных программ, разработанных в рамках проекта GNU по созданию свободной ОС. Термин Linux для наименования всей системы (а не только ядра) более распространён и будет использован далее в этом руководстве, но более правильным наверное является название GNU/Linux, отражающее большую роль проекта GNU.

Linux работает на множестве архитектур процессора таких как Intel x86, x86-64, PowerPC, ARM, Alpha AXP, Sun SPARC, Motorola 68000, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, AXIS CRIS, Renesas M32R, Atmel AVR32, Renesas H8/300, NEC V850, Tensilica Xtensa и многих других.

Ядро Linux и большая часть программ, входящих в дистрибутив, является свободным (в соответствии с определением, данным "Фондом свободного программного обеспечения") программным обеспечением. Пользователям предоставляются 4 свободы: свобода использования ПО с любой целью, свободы изучения программы, свобода распространения, свобода улучшения программы и распространения новой версии.

В отличие от большинства других ОС, Linux не имеет единой «официальной» комплектации. Вместо этого Linux поставляется в большом количестве так называемых дистрибутивов, в которых ядро Linux соединяется с утилитами GNU и другими прикладными программами (например, X.org – свободная реализация оконной системы X Window), делающими её полноценной многофункциональной операционной средой. Существуют как свободно распространяемые дистрибутивы (например, Ubuntu), так и коммерческие (например, Red Hat Enterprise Linux, исходные коды открыты, но бинарные пакеты и поддержка платные)

На сегодняшний день существует множество различных поставок Linux, "дистрибутивов", которые можно разделить на дистрибутивы общего назначения и специализированные. Примерами специализированных дистрибутивов являются системы для сетевого оборудования, мобильных телефонов, банкоматов, беспилотных военных машин и т.д.

Наиболее известными дистрибутивами общего назначения Linux являются Arch Linux, CentOS, Debian, Fedora, Gentoo, Mandriva, Mint, openSUSE, Red Hat, Slackware, Ubuntu.

1 По материалам http://ru.wikipedia.org/wiki/Linux

Ubuntu – самый популярный из них.

Российские дистрибутивы — ALT Linux, ASPLinux, Calculate Linux, НауЛинукс, AgiliaLinux (ранее MOPSLinux), Runtu и Linux XP.

В мае 2010 года семейство ОС на базе ядра Linux — третье (после Windows и Mac) по популярности (1.13 %) в мире на рынке настольных компьютеров. На рынке веб-серверов доля Linux порядка 65 %. По данным TOP500, Linux используется на 91 % самых мощных суперкомпьютеров планеты. В 4-м квартале 2010 платформа Android, основанная на ядре Linux, стала самой популярной платформой для смартфонов с долей продаж в 33%.

Архитектура Linux, основные принципы

- Монолитное ядро с динамически подгружаемыми модулями.
- Всё – файл или процесс (директория — тоже файл). Устройства представляются в виде файлов.
- Виртуальная файловая системы (VFS), уровень абстракции поверх конкретной реализации файловой системы, может быть использована для единообразного и прозрачного доступа к локальным, сетевым и виртуальным файловым системам.
- UNIX-подход: совместное использование множества небольших программ, выполняющих отдельные функции.

CentOS[2]

CentOS (англ. Community ENTerprise Operating System) — дистрибутив Linux, основанный на коммерческом Red Hat Enterprise Linux компании Red Hat и совместимый с ним.

Red Hat Enterprise Linux состоит из свободного ПО с открытым кодом, но доступно в виде дисков с бинарными пакетами только для платных подписчиков. Как требуется в лицензии GPL и других, Red Hat предоставляет все исходные коды. Разработчики CentOS используют данный исходный код для создания окончательного продукта, очень близкого к Red Hat Enterprise Linux и доступного для скачивания. Существуют и другие клоны Red Hat Enterprise Linux, созданные на основе этого кода.

Нумерация версий CentOS соответствует версиям RHEL, на которых они основаны.

Основные команды и получение справки

Запуск команд, синтаксис

Общий формат запуска команды в командной оболочке следующий

```
$ command [OPTIONS]... [ARGUMENTS]...
```

Опции бывают двух форматов:

1. Короткий, чаще всего предваряются минусом, как опция вывода в расширенном формате "-l" в примере

   ```
   $ ls -l
   ```

   Несколько подряд идущих опций могут быть объединены, например

   ```
   $ ls -la
   ```

2 По материалам http://ru.wikipedia.org/wiki/Centos

2. Длинный, предваряются двумя минусами, как опция вывода подсказки в примере

```
$ ls --help
```

Аргументы – это чаще всего имя файла или другие входные данные, например IP адрес, название сетевого интерфейса и т.д.

Несколько простых команд

Вывод даты и времени

```
$ date
```

Календарь

```
$ cal
```

Получение справки

1. Вывод краткого описания команды

```
$ whatis COMMAND
```

2. Вывод краткой справки по команде

```
$ COMMAND --help
```

3. Более подробная справка (PAGE для утилит командной строки соответствует названию утилиты, также есть справка по библиотечным функциям, файлам конфигурации и т.д.)

```
$ man PAGE
$ info PAGE
```

4. Google :)

Файловая система. Поиск и обработка файлов. Управление файловой системой

Основные директории

Согласно Filesystem Hierarchy Standard (FHS) в Linux используются следующие директории:

- / Корневая директория, содержащая всю файловую иерархию.
  - /bin Основные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям (например: cat, ls, cp).
  - /boot Загрузочные файлы (в том числе файлы загрузчика, ядро, initrd, System.map)..
  - /dev/ Файлы устройств.
  - /etc/ Общесистемные конфигурационные файлы
  - /home/ Содержит домашние директории пользователей, которые в свою очередь содержат персональные настройки и данные пользователя.
  - /lib/ Основные библиотеки, необходимые для работы программ из /bin/ и /sbin/.
  - /media/ Точки монтирования для сменных носителей.
  - /mnt/ Содержит временно монтируемые файловые системы.
  - /opt/ Дополнительное программное обеспечение (в частности не придерживающееся стандартной).

- /proc/ Виртуальная файловая система, представляющая состояние ядра операционной системы и запущенных процессов в виде файлов.
- /root/ Домашняя директория пользователя root.
- /sbin/ Основные системные программы для администрирования и настройки системы, например, init, iptables, ifconfig.
- /srv/ Данные, специфичные для окружения системы; данные ftp-серверов и т.д.
- /tmp/ Временные файлы (см. также /var/tmp).
- /usr/ Вторичная иерархия для данных пользователя; содержит большинство пользовательских приложений и утилит, используемых в многопользовательском режиме.
  - /usr/bin/ Дополнительные программы для всех пользователей, не являющиеся необходимыми в однопользовательском режиме.
  - /usr/include/ Стандартные заголовочные файлы.
  - /usr/lib/ Библиотеки для программ, находящихся в /usr/bin/ и /usr/sbin/.
  - /usr/sbin/ Дополнительные системные программы (такие как демоны различных сетевых сервисов).
  - /usr/share/ Архитектурно-независимые общие данные.
  - /usr/src/ Исходные коды (например, здесь располагаются исходные коды ядра).
  - /usr/local/ Третичная иерархия для данных, специфичных для данного хоста. Обычно содержит такие поддиректории, как bin/, lib/, share/
  - ...
- /var/ Изменяемые файлы, такие как файлы регистрации (log-файлы), временные почтовые файлы, файлы спулеров.
  - /var/lib/ Информация о состоянии. Постоянные данные, изменяемые программами в процессе работы (например, базы данных, метаданные пакетного менеджера и др.).
  - /var/lock/ Файлы-блокировки, указывающие на занятость некоторого ресурса.
  - /var/log/ Различные файлы регистрации (log-файлы).
  - /var/run/ Информация о запущенных программах (в основном, о демонах).
  - /var/spool/ Задачи, ожидающие обработки (например, очереди печати, непрочитанные или неотправленные письма).
  - /var/www/ Файлы веб-сайтов (например, иерархия файлов виртуальных хостов).
  - /var/tmp/ Временные файлы, которые должны быть сохранены между перезагрузками.
  - ...

Рабочая директория

Рабочая директория (англ. working directory, также рабочий каталог, текущий каталог или текущая директория) процесса – директория файловой системы, который используется для нахождения файлов, указанных только по имени либо по относительному пути. Также используется термин текущая рабочая директория (англ. current working directory, CWD)

Команды для работы с рабочей директорией:

- Просмотр рабочей директории

```
$ pwd
```

- Смена рабочей директории

```
$ cd DIRECTORY
```

Имена файлов и директорий

Имена файлов имеют длину до 255 байт.

Все символы допустимы, кроме / – это разделитель пути

При использовании специальных символов имя следует брать в кавычки или использовать символ экранирования "\". Рассмотрим это на примере работы команды `touch`, создающей пустой файл или обновляющий дату изменения существующего, в зависимости от переданных аргументов.

- Команда создаст 2 файла с именами "foo" и "file"

```
$ touch foo file
```

- Команда создаст 1 файл с именем "foo file"

```
$ touch foo\ file
```

- Аналогичный результат

```
$ touch "foo file"
```

Регистр имеет значение при именовании файлов: `File`, `file`, `FILE` — три разных имени файлов.

Абсолютный и относительный пути

Абсолютный путь:

- начинается с "/";
- являет собой полную "карту" расположения файла;
- остается неизменным при смене рабочей директории;

Относительный путь:

- начинается с имени файла (особые имена "." и ".." указывают на текущую и родительскую директории соответственно)
- являет собой путь от рабочей директории (CWD);

Примеры:

- /home/student1/documents/workbook/doc1.txt — полный путь, не зависит от текущего каталога
- documents/workbook/doc1.txt — относительный путь к тому же файлу при рабочей директории /home/student1.

Для смены рабочей директории используется команда `cd`.

Примеры:

- Аргументом может быть указан абсолютный или относительный путь к новой директории:

```
$ cd /home/user1/documents
$ cd documents/workbook
```

- Директория уровнем выше:

```
$ cd ..
```

- Предыдущая рабочая директория:

```
$ cd -
```

- Ваш домашний каталог (~ означает домашнюю директорию пользователя):

```
$ cd
$ cd ~
```

Просмотр файлов в каталоге

ls выводит список файлов в директории. Общий формат:

```
ls [OPTION]... [FILE]...
```

Примеры:

- Просмотр файлов в текущей директории

```
$ ls
```

- В указанной

```
$ ls /home/
```

- С информацией о параметрах файлов

```
$ ls -l DIRECTORY
```

- Показывать скрытые файлы

```
$ ls -a DIRECTORY
```

Работа с файлами

Создать пустой файл или обновить дату изменения в существующем

```
$ touch FILE
```

Создать директорию

```
$ mkdir DIRECTORY
```

Копировать файл в указанное место

```
$ cp FILE DESTINATION
```

Переместить файл в указанное место

```
$ mv FILE DESTINATION
```

Удалить файл

```
$ rm FILE
```

Ключ `-r` указывает на рекурсивность операции для копирования/удаления директории со всем содержимым. Ключ `-f` для команды `rm` отключает запрос подтверждения удаления. Стоит быть осторожным с использованием команды

```
$ rm -rf DIRECTORY
```

так как восстановить удалённые файлы достаточно сложно, а в некоторых случаях уже невозможно. На некоторых системах может быть доступна команда `trash` для удаления файла в корзину, а в графических оболочках Linux в большинстве случаев файловый

менеджер позволяет переместить файл в корзину вместо удаления.

Работа с архивами

Для работы с архивами в Linux есть множество команд, основной является tar. Архив tar представляет собой файл, в который упаковываются файлы. Особенностью tar архива является то, что в нём, в отличие например от zip, сохраняются атрибуты файловой системы UNIX (пользователь, группа и т. д.). Стоит сказать, что tar архив — не сжатый архив, при необходимости сжатия можно воспользоваться например утилитами gzip и bzip2 и создать сжатые tar.gz или tar.bz2 архивы. Но можно и указать tar использовать эти утилиты и создать сразу сжатый архив.

Примеры:

- Создать tar архив из файлов, начинающихся на `log`

  ```
  $ tar -cf backuplogs.tar log*
  ```

- Создать сжатый архив tar.bz2 из файлов, начинающихся на `log`

  ```
  $ tar -cjf backuplogs.tar.bz2 log*
  ```

- Распаковать сжатый архив в текущую директорию

  ```
  $ tar -xf  backuplogs.tar.gz
  ```

- Распаковать сжатый архив в текущую директорию `logs`

  ```
  $ tar -xf  backuplogs.tar.gz -C logs
  ```

- Сжать файл `log.txt`, использую GNU zip (создаётся файл `log.txt.gz`, а исходный удаляется)

  ```
  $ gzip log.txt
  ```

- Разжать `log.txt.gz` (`log.txt` восстанавливается из сжатого, а сжатый файл удаляется)

  ```
  $ gzip -d log.txt.gz
  ```

Поиск файлов

Для поиска файлов по файловой системе используется команда find. Примеры поиска:

- Поиск файлов с именем `snow.png` в директории `/home`

  ```
  $ find /home -name snow.png
  ```

- Поиск файлов с именем `snow.png` в текущей директории без учёта регистра (будут найдены файлы `snow.png`, `Snow.png`, `SNOW.png` и т. д.)

  ```
  $ find -name snow.png
  ```

- Поиск файлов пользователя `student1` в директории `/tmp`, не углубляясь в поддиректории

  ```
  $ find /tmp -maxdepth 1 -user student1
  ```

- Поиск файлов пользователя `student1`, но не принадлежащих группе `students`, в текущей директории, не углубляясь в поддиректории

  ```
  $ find -user student1 -not -group students
  ```

Более быстрый поиск по индексированной базе выполняется с помощью `locate`.

Обновление базы регулярно (зачастую ежедневно) выполняется автоматически демоном cron. Обновить базу вручную можно командой `updatedb`.

Лабораторная работа 1

Зайдите в систему, авторизуйтесь как studentX (где X это ваш номер) с паролем linux. За подробностями обратитесь к тренеру. Проверьте, в какой директории вы оказались.

Выполните следующие действия:

1. Создайте директорию documents, в ней поддиректорию workbook.
2. Перейдите в директорию workbook. В ней создайте три файла doc1.txt, doc2.txt и doc3.txt
3. Создайте в домашнем каталоге директорию archive.
4. Поместите все три файла в архив docs.tar.gz
5. Переместите docs.tar.gz в archive.
6. Удалите файлы созданные ранее файлы в директории ~/documents/workbook.
7. Восстановите файлы из архива.
8. Проверьте дату последнего изменения всех трёх файлов.

Подсказка

```
login: studentX
password: linux
$ pwd
/home/studentX
$ mkdir documents
$ mkdir documents/workbook
$ cd documents/workbook
$ touch doc1.txt doc2.txt doc3.txt
$ ls
doc1.txt doc2.txt doc3.txt
$ mkdir ~/archive
$ tar -czf docs.tar.gz doc1.txt doc2.txt doc3.txt
$ mv docs.tar.gz ~/archive
$ rm doc1.txt doc2.txt doc3.txt
$ tar -xf ~/archive/docs.tar.gz
$ ls -l
total 0
-rw-r--r-- 1 student1 students 0 2011-02-14 17:03 doc1.txt
-rw-r--r-- 1 student1 students 0 2011-02-14 17:03 doc2.txt
-rw-r--r-- 1 student1 students 0 2011-02-14 17:03 doc3.txt
```

Пользователи, группы и разрешения. Администрирование пользователей

Пользователи

- Каждому пользователю присваивается уникальный номер UID (User ID).
- Пользовательская информация (имя; UID; используемая командная оболочка, например bash; домашний каталог и т.д.) хранятся в /etc/passwd.
- Пользователи не могут читать, писать и выполнять чужие файлы без разрешения.
- Пользователь root – суперпользователь системы, для него нет ограничений.

Группы

- Каждая группа обладает уникальным номером GID (Group ID).
- Информация о именах, GID и участниках групп хранится в /etc/group.
- Каждый пользователь участвует как минимум в одной (основной группе).
- Пользователь может открывать доступ к своим файлам для группы.

Модель безопасности файлов в Linux

Каждый файл принадлежит UID-у и GID-у.

К каждому файлу есть 3 категории доступа:

- владелец (по UID)
- группа (по GID)
- остальные

Для каждой категории есть три вида доступа:

- r – read (чтение)
- w – write (запись)
- e – eXecute (выполнение)

Для директории право на выполнение — это право заходить в неё.

Рассмотрим примеры.

```
$ ls -l documents/workbook/doc1.txt
-rw-r--r-- 1 student1 students 0 2011-02-14 17:03 doc1.txt
```

Здесь владелец файла `doc1.txt` – `student1`, группа – `students`, владелец имеет право на чтение и запись, группа и прочие – только на чтение.

```
$ ls -l documents
drwxr-xr-x 5 student1 students 170 2011-02-14 17:03 workbook
```

Здесь d – указывает на то, что этот объект – директория, владелец может читать, писать (создавать и удалять файлы) и заходить, а остальные – только читать (получать список файлов и их свойств) и заходить (делать текущей).

Смена владельца

- Только пользователь root может менять владельца файла
- Только root и владелец могут менять группу файла
- Владельца меняем с помощью команды (опция -R для изменения владельца в поддиректориях)

```
chown [OPTION]... USER FILE...
```

- Группу меняем командой (опция `-R` для изменения группы в поддиректориях)

```
chgrp [OPTION]... GROUP FILE...
```

Смена прав доступа

Для смены прав доступа используется команда

```
chmod [OPTION] MODE FILE
```

В этой команде для установки прав доступа можно использовать два режима – символьный и цифровой.

Символьный режим

- u - владелец, g - группа, o - все остальные
- +/- – разрешить/запретить
- r – чтение, w – запись, x – выполнение

Пример

```
$ chmod ug+w doc1.txt
```

устанавливает для владельца и группы право на запись в файл doc1.txt

Цифровой режим

- Права задаются тремя цифрами от 0 до 7, 0 означает отсутствие каких-либо прав, 7 — все права.
- Первая цифра задает права владельца.
- Вторая задает права группы.
- Третья – всех остальных.
- Права задаются суммированием трех чисел:
  - 4 – чтение
  - 2 – запись
  - 1 – выполнение

Пример:

```
$ chmod 740 doc2.txt
```

Задает права `rwxr-----` или все права для владельца, чтение для группы и нет прав для прочих.

Оболочка Bash. Настройка, использование, разработка сценариев

Спецсимволы

- * – ноль и более произвольных символов
- ? – ровно один произвольный символ
- [0-9] – символ из заданного диапазона
- [abc] – символ из списка
- [^abc] – символ НЕ из списка

Автодополнение

Используйте клавишу TAB для завершения ввода.

Примеры:

- Автодополнение в случае однозначности набираемой команды/опции/аргумента

```
$ tou<TAB>
```

- дополнится до

```
$ touch
```

- Аналогично для файлов

```
$ ls doc<TAB>
$ ls docracadabra.txt
```

- Автодополнение при неоднозначности

```
$ ls doc<TAB><TAB>
doc1.txt doc2.txt doc3.txt
```

Тильда ~

Указатель на домашний каталог текущего пользователя

```
$ cat ~/.bash_profile
```

Может указывать на домашний каталог любого пользователя

```
$ ls ~student2/archive
```

Командные расширения

`$(command)` или `` `command` `` подставляет вместо команды результат ее выполнения

Примеры:

- Вывод имени хоста

```
$ echo "This system's name is $(hostname)"
This system's name is server1.example.com
```

- Вывод некоторой информации о директории

```
$ echo `pwd` content: `ls`
doc1.txt doc2.txt doc3.txt
```

Расширение фигурных скобок {}

```
$ ls doc{1,2,3}.txt
doc1.txt doc2.txt doc3.txt
```

Переменные

Переменные в баше не типизированы.

Задаются переменные так:

```
$ VARIABLE=VALUE
```

Вывести можно например так:

```
$ echo $VARIABLE
```

Пример работы с переменной:

```
$ HI="Hello world"
$ echo $HI
Hello world
```

Переменные окружения

Обычные переменные локальны, существуют в контексте текущей оболочки. Переменные окружения наследуются дочерней оболочкой и задаются так:

```
$ export VARIABLE=VALUE
```

Например можно посмотреть переменную $PS1, содержащую формат приветствия командной строки (там скорее всего будет достаточно много спецсимволов типа "\u", означающего имя пользователя), и поменять её:

```
Luke@x-wing008[~]$ echo $PS1
\u@\h[\w]$
Luke@x-wing008[~]$ export PS1="Waiting for your orders, master
\u$ "
Waiting for your orders, master Luke$
```

Основы работы в текстовом редакторе nano

Nano – один из стандартных консольных текстовых редакторов в UNIX системах с подсветкой синтаксиса большинства языков программирования. Помимо него из консольных можно назвать vim и Emacs (каждый из них имеет ещё и графический интерфейс), но они достаточно сложны для освоения и не факт, что пользователю будут нужны все действительно большие возможности, предоставляемые ими. Nano проще.

Редактирование файлов

Для запуска редактора без открытия файла выполните:

```
$ nano
```

Сейчас редактор не связан с каким-либо файлом, но набранный текст можно будет сохранить в существующий или новый файл.

Чтобы открыть файл в nano из командной строки, используйте:

```
$ nano FILE
```

Если файл существует, он откроется для редактирования. Если файл не существует, то он будет создан, о чём сообщит соответствующая надпись внизу интерфейса.

Интерфейс nano состоит из следующих частей:

- верхняя строка состояния (в ней слева показывается версия редактора, в центре — имя редактируемого файла, а справа — информация о том, был ли изменён файл);
- главное окно редактора, в нём показывается содержимое редактируемого файла;
- нижняя строка состояния, показывает разные важные сообщения;
- контекстные подсказки по клавишам управления.

Управление nano осуществляется либо комбинациями клавиш с Ctrl (в nano обозначается "^") или Alt (в nano обозначаются "M", сокращение от "Meta", кроме Alt, "M" соответствует Esc), например "^X" для выхода из редактора. Если в контекстных подсказках у сочетания клавиш нет префикса "^" или "M-", значит используется просто указанная клавиша. H

Когда nano только запущен, внизу указаны следующие подсказки по следующим функциям (но все доступные функции ими не ограничиваются) в порядке слева направо, сверху вниз:

- ^G ("**G**et help") — контекстная помощь;
- ^O ("write **O**ut") — записать файл на диск (по текущему пути или по новому, работает автодополнение по TAB и вывод списка путей по TAB TAB);
- ^R ("**R**ead file") — вставить другой файл в редактируемый;
- ^Y — предыдущая страница (также можно использовать PageUp);
- ^K ("**C**ut text", "**K**ill text") — вырезать всю текущую строку;
- ^C ("**C**urrent position") — показать в нижней строке состояния данные о текущей позиции курсора (номер строки, номер ряда и т. д.)
- ^X ("e**X**it") — выход;
- ^J ("**J**ustify") — перенести длинную строку на следующую, судя по ширине экрана;
- ^W ("**W**here is") — поиск по точному совпадению или регулярному выражению;
- ^V — следующая страница (также можно использовать PageDown);
- ^U ("**U**ncut text", "**U**nkill text") — вернуть строку, вырезанную ^K;
- ^T ("**T**o spell") — проверка орфографии.

Это далеко не весь список доступных комбинаций клавиш, полный можно посмотреть в справке, нажав ^G.

Советы по эффективной работе в nano

1. По умолчанию nano показывает данные о текущей позиции курсора в тексте только по нажатию ^C. Чтобы эти данные всегда показывались, запускайте nano с аргументом "-c":

```
$ nano -c
```

2. Для перехода к определённой позиции в тексте по номеру строки и ряда, используйте M-G ("**G**o to position").
3. Для поиска с заменой используйте ^WR ("**W**here is", "**R**eplace").
4. Для перехода по словам вперёд используйте ^Space, назад — M-Space.
5. Для вывода количества слов, строк, символов - ^D ("**D**isplay statistics").
6. Традиционное и привычное в GUI редакторах выделение текста для копирования в nano делается не очень удобным способом. Для выделения текста M-A, для копирования — M-6, для вырезания — ^K, для вставки — ^U.

Написание скриптов

Скрипт это текстовый файл, содержащий последовательность команд для выполнения.

Скрипты могут быть предназначены для решения следующих задач:

- автоматизация частых действий;
- написание простых приложений;
- манипуляции с файлами и текстами;

Как написать скрипт?

1. С помощью редактора nano или любого другого (vim, emacs, gedit в GNOME, kate в KDE) создаём текстовый файл.
2. Первой строкой обязательно должен быть указатель на интерпретатор:

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

   Или более переносимый вариант

   ```
   #!/usr/bin/env bash
   ```
3. Не забываем комментировать скрипт, комментарии начинаются с символа "#".
4. После редактирования делаем скрипт выполняемым:

   ```
   $ chmod +x myscript.sh
   ```
5. Запускаем скрипт:

   ```
   $ ./myscript.sh
   ```

Стоит сказать, что по умолчанию bash не проверяет переменные при использовании, не инициализированные ранее переменные считаются имеющими пустое значение. За подобным следует внимательно следить, так как опечатка в имени переменной может привести к печальным последствиям. Например, если вместо инициализированной PATHTOREMOVE из-за опечатки использована PTHTOREMOVE, то выполнение следующего скрипта может привести к очень печальным последствиям (к каким именно, оставим на размышление читателю), особенно при выполнении от суперпользователя:

```
$!/usr/bin/env bash
PATHTOREMOVE=/tmp/stuff
rm -rf $PTHTOREMOVE/*
```

Чтобы избежать таких неприятностей, можно приказать bash сообщать о использовании неинициализированных переменных и прекращать выполнение скрипта. Для этого используется команда

```
set -u
```

и предыдущий пример можно переписать так, чтобы на строчке с командой удаления bash выдал предупреждения и не выполнял скрипт дальше

```
$!/usr/bin/env bash
set -u
PATHTOREMOVE=/tmp/stuff
rm -rf $PTHTOREMOVE/*
```

Операторы test и if

test проверяет истинность высказывания и в зависимости от результата возвращает одно из двух значения: 0 ("истина", "успех") или 1 ("ложь", "ошибка"). Отличие в соответствии чисел и смысла по сравнению с традиционной бинарной логикой связано с тем, что в UNIX процесс возвращает 0 в случае успешного завершения и ненулевой код ошибки иначе. Для простоты можно думать в терминах "успех" и "ошибка", а не 0 и 1, в рамках работы с командной оболочкой этого вполне достаточно.

Два вида синтаксиса:

```
test EXPRESSION
[ EXPRESSION ]
```

Примеры

```
$ test "windows" = "windows" && echo yes || echo no
yes
$ [ "linux" = "windows" ] && echo yes || echo no
no
```

В приведённом выше примере используются операторы AND (&&) и OR (||). Они работают по следующим правилам:

`command1 && command2`	command2 выполняется лишь в том случае, когда command1 вернула "успех"
`command1 || command2`	command2 выполняется лишь в том случае, когда command1 вернула "ложь"

Практический смысл применения этих операторов — выполнение определённых действий в случае успеха или ошибки в выполнении команды. Например, скрипт может завершаться командой exit, если одна из его команд завершилась с ошибкой. Для этого в скрипре можно написать:

```
COMMAND || exit
```

test часто используется для проверки существования файлов и директорий:

- файл существует и является обычным файлом (не директория, не устройство и т. д.)

  `[-f FILENAME]`

- файл существует и является директорией

  `[-d DIRNAME]`

Чаще всего применяется в операторе IF, например для запуска исполняемого файла с проверкой на существование может быть использован следующий код:

```
EXPATH=/bin/ls
if [ -x "$EXPATH" ]
then
    $EXPATH
else
    echo Executable file \"$EXPATH\" not found
fi
```

Стандартный ввод/вывод и каналы

Стандартные потоки ввода/вывода

Linux поддерживает три стандартных потока для каждой программы:

- 0 – стандартный ввод (STDIN), по умолчанию клавиатура
- 1 – стандартный вывод (STDOUT), по умолчанию экран терминала
- 2 – стандартный поток ошибок (STDERR), тоже экран терминала

Перенаправление стандартного вывода в файл

STDOUT и STDERR можно перенаправить в файл, STDIN — из файла. Поддерживаются следующие операторы перенаправления:

- > — перенаправляет поток вывода в файл
- 2> — перенаправляет поток ошибок в файл
- &> — перенаправляет оба потока в файл
- < — перенаправление ввода из файла

Содержимое файла при перенаправлении вывода будет переписано, для дописывания в конец файла используется >>.

Примеры перенаправления потоков

- Обычная команда без перенаправления, ищет в домашних директориях пользователей все файлы с расширением "txt":

  ```
  $ find /home -name *.txt
  ```

- Перенаправление вывода команды в файл `find.out`

  ```
  $ find /home -name *.txt > find.out
  ```

- Перенаправление ошибок программы в "никуда":

  ```
  $ find /home -name *.txt 2> /dev/null
  ```

- Объединение двух предыдущих вариантов:

  ```
  $ find /home -name *.txt > find.out 2>find.err
  ```

Перенаправление вывода программе (pipe)

Пайпы (неименованные каналы межпроцессного взаимодействия) позволяют передать STDOUT одной программы в STDIN другой (STDERR не передается).

```
$ COMMAND1 | COMMAND2
```

Используется для комбинирования функциональности разный утилит:

- Просмотр содержимого директории /etc с возможностью прокрутки

```
$ ls -l /etc | less
```

- Вывод имён всех пользователей, принадлежащих группе с идентификатором группы равным 100, это группа users:

```
$ cut -d ":" -f 1,4 /etc/passwd | grep ":100$" | cut -d ":" -f
1
```

Для разветвления потоков используется команда tee, она позволяет одновременно писать и в файл, и на экран терминала. Это бывает удобно если нужно одновременно и вести лог длительно работающей программы (например процесса компиляции) и наблюдать за ходом её работы. Рассмотрим два простых примера (правда достаточно надуманных):

- Результаты работы ls выводятся только в файл:

```
$ ls > listing
```

- Результаты работы ls выводятся и на экран и в файл

```
$ ls | tee listing
```

Лабораторная работа 2

Назначение прав

1. В директории ~/documents/workbook создайте пустой файл с именем mydoc.txt
2. Всем пользователям выставьте право на чтение файла
3. Теперь выставьте права rwxrw-r--
4. В директории documents создайте директорию test
5. Уберите право на выполнение для владельца
6. Создайте файл test внутри этой директории
7. Получилось? Почему?
8. Удалите директорию test

Написание скрипта

1. В своем домашнем каталоге создайте директорию bin
2. В директории создайте и отредактируйте с помощью nano скрипт с именем myscript
3. Скрипт при запуске должен выводить имя хоста и дату.
4. Запустите скрипт, проверьте как он работает.

Подсказка

Назначение прав

```
$ cd ~/documents/workbook
$ touch mydoc.txt
$ chmod a+r mydoc.txt
$ chmod 764 mydoc.txt
$ cd ../
$ mkdir test
$ chmod u-x test
$ touch test/test1
touch: creating 'test/test1': Permission denied
$ rmdir test
```

Написание скрипта

```
$ cd
$ mkdir bin
$ cd bin
$ nano myscript
```

Далее введите текст

```
#!/bin/bash
echo Hostname is $HOSTNAME
echo Current time is `date`
```

И нажмите ^O, Enter, ^X для сохранения и выхода из nano.

```
$ chmod a+x myscript
$ ./myscript
```

Изучение и управление процессами.

Что такое процесс?

Процесс — выполняемая программа вместе с ресурсами (адресное пространство с программой и данными, другие ресурсы). Программа — пассивная совокупность инструкций.

У каждого процесса есть цифровой идентификатор процесса PID, наследующиеся от запустившего пользователя идентификатор пользователя процесса UID и идентификатор пользователя процесса GID, SELinux контекст и др.

Просмотр списка процессов

Для просмотра списка процессов используется команда **ps**, вот некоторые опции:

a — показывает процессы во всех терминалах (не только в текущем);

x — показывает процессы, не привязанные к терминалам ;

u — указывает владельцев процессов;

f — вывод в виде дерева

-o *format* — вывод специфических колонок таблицы

Поиск нужных процессов

Поиск процессов, соответствующих некоторому шаблону

```
$ ps aux | grep PATTERN
```

Вывод идентификаторов процессов пользователя USER

```
$ pgrep -U USER
```

Вывод идентификаторов процессов пользователей из группы students

```
$ pgrep -G students
```

Вывод идентификаторов всех процессов команды bash

```
$ pidof bash
```

Сигналы

Сигналы – одно из фундаментальных средств межпроцессной комуникации. Сигналы посылаются непосредственно процессу, минуя пользовательский интерфейс. Для каждого сигнала есть действие по умолчанию, для большинства сигналов можно назначить собственный обработчик.

Сигналам идентифицируются по именам и номерам.

Примеры:

- Сигнал 15 TERM – сигнал по умолчанию, чистое завершение;
- Сигнал 9 KILL – немедленное завершение;
- Сигнал 1 HUP – выполнение некоторого предусмотренного действия (обычно – перечитывание конфигурации и переинициализация)

Подробная документация:

```
$ man 7 signal
```

Передача сигналов процессу

По PID:

```
$ kill [SIGNAL] PID
```

По имени:

```
$ killall [SIGNAL] COMMAND
```

По параметрам (параметры такие же как в pgrep):

```
$ pkill [signal] [PARAMETER]...
```

Инициализация системы и службы

Типичная последовательность процесса загрузки машины с GNU/Linux

BIOS (Basic Input-Output System)

MBR (Master boot record)

GRUB (GRand Unified Bootloader)

Ядро системы

init

init скрипты

login

Уровень загрузки

Определяется в файле /etc/inittab

```
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
id:5:initdefault:
```

Определяет набор запускаемых при старте системы и останавливаемых при завершении работы сервисов. Уровень запуска можно задать при запуске системы через параметры загрузчика.

Скрипты запуска сервисов

- Расположены в /etc/rc.d/init.d
- Служат для запуска, остановки и перезапуска сервисов
- Большинство принимает параметры: start, stop, restart, status, reload (перечитать конфигурацию)

Запуск сервисов

Обращение к ним идет через директории /etc/rc.d/rcX.d/, где X – уровень загрузки.

```
$ ls -1 /etc/rc5.d/
S20fancontrol
S20kerneloops
```

```
S20nfs-kernel-server
...
$ ls /etc/rc0.d/
K20xrdp
K74bluetooth
K80nfs-kernel-server
...
S90halt
...
```

Ссылки, начинающиеся с S, — запуск сервиса. Чем больше число после S (назовём его SN), тем позже запускается сервис. Ссылки начинающиеся с K — остановка сервиса. Первыми запускаются K скрипты, за ними S. Чем больше число после K (назовём его KN), тем позже останавливается сервис. Как правило, KN = 100 - SN. Основные наиболее важные системные службы имеют маленькие SN и большие KN.

Управление сервисами

Для управления сервисами можно использовать сами скрипты:

```
# /etc/init.d/httpd start
```

Существует специальная команда

```
# service httpd start
```

Управление сервисами при загрузке

```
# chkconfig
```

Настройка сети. Сетевые клиенты

Настройка сетевых интерфейсов

- Команда **ifconfig** позволяет узнать MAC адрес, IP адрес и прочее, а также настроить интерфейс.
- Команды **ifup**, **ifdown** используются соответственно для поднятия и отключения интерфейса.
- Системные конфигурационные файлы для используемых в системе интерфейсов расположены в /etc/sysconfig/network-scripts/, например для eth0 — /etc/sysconfig/network-scripts/ifcfg-eth0
- Глобальные настройки — /etc/sysconfig/network

Сетевые клиенты – web

- Firefox и другие X браузеры
- Консольные браузеры (links и другие)
- Загрузчик файлов wget

Сетевые клиенты – почта и месенджеры

- Thunderbird
- Evolution
- Консольные почтовые клиенты
- Pidgin (AIM, MSN, ICQ, Yahoo, Jabber, Gadu-Gadu, SILC, IRC)

OpenSSH – безопасный удаленный шелл

Удаленный шелл

```
$ ssh USER@HOST
```

Удаленное выполнение команды

```
$ ssh USER@HOST COMMAND
```

Защищенное копирование по сети

```
$ scp SRC DEST
```

Синхронизация файлов (для работы с удалёнными хостами может использовать ssh) – rsync.

Установка ПО. RPM

Управление пакетами

В дистрибутивах Linux есть разные способы управления пакетами, наиболее распространены форматы Deb (Debian) и RPM (RedHat Package Manager, RPM Package Manager) пакетов программного обеспечения. Deb используется в Debian, Ubuntu и др. RHEL, CentOS, Fedora Core, Alt Linux, SUSE Linux.

Весь дистрибутив — набор пакетов RPM, пакеты можно добавлять, удалять и обновлять пакеты.

RPM – RedHat Package Manager

RPM:

- формат пакетов программного обеспечения;
- утилита для установки пакетов и запросов к базе.

Операции с пакетами

- Установка пакета

```
rpm {-i|--install} PACKAGEFILE
```

- Удаление пакета

```
rpm {-e|--erase} PACKAGENAME
```

- Обновление пакета или установка при отсутствии

```
rpm {-U|--upgrade} PACKAGEFILE
```

- Обновление пакета, только если установлен

```
rpm {-F|--freshen} PACKAGEFILE
```

Примеры запросов к базе RPM пакетов

Список всех установленных пакетов

```
$ rpm -qa
```

Список файлов в пакете

```
$ rpm -ql PACKAGENAME
```

Описание пакета

```
$ rpm -qi PACKAGENAME
```

YUM – система управления пакетами

CentOS использует программу YUM для скачивания и установки пакетов с репозитория CentOS Mirror Network и сторонних репозиториев. YUM представляет собой надстройку над RPM и, помимо прочего, может автоматически разрешать зависимости между пакетами.

Настройки

- Основной файл настроек /etc/yum.conf
- Директория с настройками репозиториев пакетов /etc/yum.repos.d

Используется для установки, удаления и просмотра списков ПО:

- Установка пакета из репозитория

  ```
  # yum install PACKAGENAME
  ```

- Удаление пакета

  ```
  # yum remove PACKAGENAME
  ```

- Обновление пакета

  ```
  # yum update PACKAGENAME
  ```

- Вывод списка доступных для установки пакетов (не установленных в системе, но существующих в подключенных репозиториях)

  ```
  $ yum list available
  ```

- Вывод списка установленных в системе пакетов

  ```
  $ yum list installed
  ```

Система X Window[3]

X Window System — оконная система, обеспечивающая стандартные инструменты и протоколы для построения графического интерфейса пользователя. Используется в UNIX-подобных ОС.

X Window System обеспечивает базовые функции графической среды: отрисовку и перемещение окон на экране, взаимодействие с устройствами ввода, такими как, например, мышь и клавиатура. X Window System не определяет деталей интерфейса пользователя — этим занимаются менеджеры окон, которых разработано множество. По этой причине внешний вид программ в среде X Window System может очень сильно различаться в

3 По материалам http://ru.wikipedia.org/wiki/X_Window

зависимости от возможностей и настроек конкретного оконного менеджера.

X.Org Server — свободная реализация сервера X Window System с открытым кодом.

В X Window System предусмотрена сетевая прозрачность: графические приложения могут выполняться на другой машине в сети, а их интерфейс при этом будет передаваться по сети и отображаться на локальной машине пользователя (в случае если это разрешено в настройках). В контексте X Window System термины «клиент» и «сервер» имеют непривычное для многих пользователей значение: «сервер» означает локальный дисплей пользователя (дисплейный сервер), а «клиент» — программу, которая этот дисплей использует (она может выполняться на удалённом компьютере).

Два основных менеджера рабочего стола

GNOME – используется в CentOS по умолчанию.

KDE – альтернативный пакет.