# Консольные команды

Большинство консольных команд — это имена соответствующих программ-утилит (если таковые файлы программ найдены на путях поиска `$PATH`), но есть некоторая часть команд, которые являются внутренними командами интерпретатора `bash` (как и для других интерпретаторов). Полный их список команда, которые реализованы как внутренние, можно получить, указав в команде `man` **любую** из внутренних команд:

```
$ man set
BASH_BUILTINS(1)                                          BASH_BUILTINS(1)
NAME
   bash, :, ., [, alias, bg, bind, break, builtin, cd, command, compgen, complete, continue, declare, dirs, disown, echo, enable,
   eval, exec, exit, export, fc, fg, getopts, hash, help, history, jobs, kill, let, local, logout, popd, printf, pushd, pwd, read,
   readonly, return, set, shift, shopt, source, suspend, test, times, trap, type, typeset, ulimit, umask, unalias, unset, wait - bash
   built-in commands, see bash(1)
...
    cd [-L|-P] [dir]
           Change the current directory to dir.
...
```

Эта же справочная страница даст вам подробные сведения об использовании каждой из внутренних команд интерпретатора.

**Примечание:** Обратите внимание, что справка по самому интерпретатору даст вам совсем другую страницу, а не ту, которую мы здесь обсуждаем:

```
$ man bash
BASH(1)                                                          BASH(1)
NAME
       bash - GNU Bourne-Again SHell
SYNOPSIS
       bash [options] [file]
COPYRIGHT
       Bash is Copyright (C) 1989-2009 by the Free Software Foundation, Inc.
...
```

Все прочие, не являющиеся внутренними, как уже сказано, являются именем **файла** программы, реализующей эту команду, если этот файл находится в одном из каталогов, перечисленных списком в переменной `$PATH`:

```
$ which cp
/bin/cp
```

Если для всех внутренних команд мы видели единую справочную страницу, то для внешних команд, естественно, они будут индивидуальными:

```
$ man cp
CP(1)                                                              CP(1)
ИМЯ
       cp - копирование файлов и каталогов
ОБЗОР
       cp [опции] файл путь
       cp [опции] файл... каталог
      Опции POSIX: [-fiprR] [--]
      Дополнительные опции POSIX 1003.1-2003: [-HLP]
...
```

**Примечание:** Обратите внимание (когда-то это может стать сюрпризом), что имена некоторых внешних команд (файлов) может перекрываться такой же внутренней командой конкретного интерпретатора команд (а для другого интерпретатора такое перекрытие может и отсутствовать). Самый яркий тому пример:

```
$ which echo
/bin/echo
$ echo --help
```

```
--help
$ /bin/echo --help
Usage: /bin/echo [SHORT-OPTION]... [STRING]...
  or:  /bin/echo LONG-OPTION
Печатает СТРОКУ(СТРОКИ) на стандартный вывод.
...
ЗАМЕЧАНИЕ: ваша оболочка может предоставлять свою версию echo, которая
обычно перекрывает версию, описанную здесь.  Пожалуйста, обращайтесь к
документации по вашей оболочке, чтобы узнать, какие ключи она
поддерживает.
```

И ещё одно несоответствие в этом случае — очевидно, что это справочная страница «не той» команды, которая используется в системе по умолчанию:

```
$ which echo
ECHO(1)                          User Commands                          ECHO(1)
NAME
       echo - display a line of text
...
```

## *Формат командной строки*

Формат командной строки определяет как, с какими параметрами и опциями, может быть записан вызов программ утилит. Детально формат определяется используемым **командным интерпретатором**, и даже его версией, но общие правила сохраняются:

```
$ <[путь/]команда> [ключи] [параметры] [ключи] [параметры]...
```

Порядок следования [ключи] [параметры] чаще всего произвольный, но в некоторых shell может требоваться именно такой!

```
<ключи> := [<ключ>] [<ключи>]
<ключ> := { -p | -p[<пробелы>]<значение> | --plong }
<параметры> := [<параметр>] [<параметры>]
<параметры> := <значение>
```

Формат записи ключей и длинных ключей определяется POSIX программными вызовами `getopt()` и `getopt_long()` (настоятельно рекомендуется проработать). В «хорошем» (по реализации) командном интерпретаторе ключи (опции, опциональные параметры) и параметры в командной строке могут чередоваться произвольным образом, вот эти две команды должны быть эквивалентными:

```
$ gcc hello.c -l ld -o hello
$ gcc -o hello -l ld hello.c
```

Если опция (ключ), требует значения, то это значение может отделяться пробелом, а может записываться слитно с написанием ключа (одно символьным), вот ещё примеры эквивалентных записей:

```
$ gcc hello.c -o hello
$ gcc hello.c -ohello
```

Только простые ключи (односимвольные, начинающиеся с однократного '-') могут иметь значения, «длинные» опции (многосимвольные, начинающиеся с 2-х '-') значений не имеют:

```
$ gcc --help
```

Запись командной строки можно переносить на несколько строк обратным слэшем ('\') в конце каждой продолжаемой строки.

## Уровень диагностического вывода команд

Во многих командах-утилитах реализован ключ `-v` - «детализировать диагностический вывод», причём этот ключ может повторяться в командной строке несколько раз, и число повторений его определяет уровень детализации диагностики: чем больше повторений, тем выше уровень детализации.

На уровне кода (в своих собственных приложениях) это реализуется примерно так:

```
int main( int argc, char *argv[] ) {
   int c, debuglevel = 0;
   while( ( c = getopt( argc, argv, "v" ) ) != EOF )
      switch( c ) {
         case 'v': debuglevel++; break;
      }
   // к этому месту в коде сформирован уровень диагностики  debuglevel
   ...
}
```

## Фильтры, каналы, конвейеры

Большинство команд (утилит) UNIX (GNU, Linux) являются **фильтрами**:

1. они имеют неявный стандартный поток ввода (файловый дескриптор 0, `SYSIN`) и стандартный поток вывода (файловый дескриптор 1, `SYSOUT`) ... (и стандартный поток журнала ошибок, 2, `SYSERR`, который в этом рассмотрении нас будет меньше прочих интересовать);

2. часто (но не обязательно) `SYSIN` это клавиатура ввода, а `SYSOUT` это экран консоли или терминала, но, например, при запуске из-под суперсервера `inetd (xinetd)` `SYSIN` и `SYSOUT` это будут сетевые TCP/IP сокеты;

3. эти потоки ввода-вывода могут перенаправляться (`>, >>, <`), или через каналы (`|`) поток вывода одной утилиты направляется в поток ввода другой:

```
$ prog 2>/dev/null
```

- подавляется вывод ошибок (`SYSERR` направляется на `/dev/null` — псевдоустройство, которое поглощает любой вывод).

```
$ ps -Af | grep /usr/bin/mc | awk '{ print $2 }'
4920
4973
5013
5096
```

- выбираем только 2-е поле (`PID`) интересующих нас строк.

4. потоки могут сливаться (очень часто это характерно для потока ошибок 2):

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

- поток ошибок 2 направляется в поток вывода 1 (знак `&` отмечает, что это номер дескриптора потока, `SYSOUT`, а не новый файл с именем 1); изменение порядка записи операндов в этом примере изменит результат: поток ошибок будет выводиться;

5. чаще всего (но и это не обязательно) `SYSIN` и `SYSOUT` это символьные потоки, но это могут быть и потоки бинарных данных, например, аудиопотоки в утилитах пакетов: `sox, ogg, vorbis, speex` ...

```
$ speexdec -V male.spx - | sox -traw -u -sw -r8000 - -t alsa default
$ speexdec -V male.spx - | tee male3.raw | sox -traw -u -sw -r8000 - \
-t alsa default
```

Не представляет большого труда писать свои собственные консольные (текстовые) приложения так, чтобы они тоже соответствовали общим правилам утилит POSIX. К этому стоит стремиться.

## *Справочные системы*

Значение онлайновых справочных систем (их несколько) в Linux велико, его трудно переоценить:

- практически всю справочную информацию (команды системы, конфигурационные файлы, программные API) можно получить непосредственно из справочной системы, за экраном терминала;
- из-за «свободности» системы Linux и программного обеспечения GNU, по ним нет, и никогда не будет упорядоченной и исчерпывающей технической документации, такой полноты, скажем, как по проприетарным операционным системам Solaris или QNX; техническую информацию приходится черпать из справочных систем.

При работе со справочными системами Linux нужно учитывать одно обстоятельство и проявлять известную осторожность: справочные системы обновляются нерегулярно и неравномерно, одновременно в них могут находиться статьи совершенно различающихся лет написания, и даже относящиеся к различным версиям обсуждаемых понятий — нужно пытаться отслеживать степень свежести справочной информации!

Основная онлайн справочная система Linux, это так называемая man-справка (manual), например:

```
$ man ifconfig
IFCONFIG(8)                Linux Programmer's Manual               IFCONFIG(8)
NAME
       ifconfig - configure a network interface
SYNOPSIS
       ifconfig [interface]
       ifconfig interface [aftype] options | address ...
DESCRIPTION
...
```

**Выход** из страницы `man`: клавиша 'q' (quit)! Стандартное завершение по Ctrl-C для этой утилиты не срабатывает.

Вся справочная система разбита на **секции** по принадлежности справки. Если не возникает неоднозначности (термин не встречается в нескольких секциях), номер секции можно не указывать, иначе номер секции указывается как параметр (номер секции может указываться всегда, это не будет ошибкой):

```
$ man 1 man
...
       The standard sections of the manual include:
      1      User Commands
      2      System Calls
      3      C Library Functions
      4      Devices and Special Files
      5      File Formats and Conventions
      6      Games et. Al.
      7      Miscellanea
      8      System Administration tools and Deamons
```

Здесь мы видим тематическое разделение всей справочной системы по секциям.

Другая справочная система — `info`:

```
$ info ifconfig
...
```

```
$ info info
-----Info: (*manpages*)ifconfig, строк: 169 --Top-----------------------------------------
Добро пожаловать в Info версии 4.8. ? -- справка, m выбирает пункт меню.
File: info.info,  Node: Top,  Next: Getting Started,  Up: (dir)
Info: An Introduction
*********************
The GNU Project distributes most of its on-line manuals in the "Info
format", which you read using an "Info reader".  You are probably using
an Info reader to read this now.
...
```

Есть ещё база данных по **терминам** системы, и работающие с ней несколько команд:

```
$ whatis ifconfig
ifconfig             (8)  - configure a network interface
$ whatis whatis
whatis               (1)  - search the whatis database for complete words
$ apropos whatis
apropos              (1)  - search the whatis database for strings
makewhatis           (8)  - Create the whatis database
whatis               (1)  - search the whatis database for complete words
```

Базу данных для работы нужно предварительно сформировать :

```
# makewhatis
...
```

Это (форматирование базы данных) : а). делается с правами `root`, б). потребует существенно продолжительного времени, чтоб вас это не смущало (программа не зависла!), но потребует его один раз.

Разница между `whatis` и `apropos` :

```
$ whatis /dev
/dev: nothing appropriate
$ apropos /dev
MAKEDEV              (rpm) - Программа, используемая для создания файлов устройств в /dev.
swapoff [swapon]      (2)  - start/stop swapping to file/device
swapon                (2)  - start/stop swapping to file/device
```

Наконец, справочную информацию (подсказку) принято включать непосредственно в команды, и разработчики утилит часто следуют этой традиции:

```
$ rlogin --help
usage: rlogin host [-option] [-option...] [-k realm ] [-t ttytype] [-l username]
     where option is e, 7, 8, noflow, n, a, x, f, F, c, 4, PO, or PN
$ gcc --version
gcc (GCC) 4.1.2 20071124 (Red Hat 4.1.2-42)
Copyright (C) 2006 Free Software Foundation, Inc.
```

## *Пользователи и права*

Эта группа команд позволяет манипулировать с именами пользователей (добавлять, удалять, менять им права). Управление учётными записями пользователей — это целый раздел искусства системного администрирования. Мы же рассмотрим эту группу команд в минимальном объёме, достаточном для администрирования **своего** локального рабочего места.

Основная команда добавления нового имени пользователя:

```
# adduser
Usage: useradd [options] LOGIN
Options:
```

```
  -b, --base-dir BASE_DIR       base directory for the new user account
                                home directory
  -c, --comment COMMENT         set the GECOS field for the new user account
  -d, --home-dir HOME_DIR       home directory for the new user account
...
# which adduser
/usr/sbin/adduser
```

При создании нового имени пользователя для него будут определены (в диалоге) и значения основных параметров пользователя: пароль, домашний каталог и другие. Команды той же группы:

```
# ls /usr/sbin/user*
/usr/sbin/useradd  /usr/sbin/userhelper   /usr/sbin/usermod
/usr/sbin/userdel  /usr/sbin/userisdnctl  /usr/sbin/usernetctl
```

С этими командами достаточно ясно без объяснений. Вот как мы меняем домашний каталог для нового созданного пользователя:

```
# adduser kernel
# usermod -d /home/guest kernel
# cat /etc/passwd | grep kernel
kernel:x:503:100:kernel:/home/guest:/bin/bash
```

А вот так администратор может сменить пароль любого другого пользователя:

```
# passwd kernel
Смена пароля для пользователя kernel.
Новый пароль :
НЕУДАЧНЫЙ ПАРОЛЬ: основан на слове из словаря
НЕУДАЧНЫЙ ПАРОЛЬ: слишком простой
Повторите ввод нового пароля :
passwd: все токены проверки подлинности успешно обновлены.
```

Уже находясь в системе (под каким-то, естественно именем), мы можем всегда перерегистрироваться (в одном отдельном терминале) под именем любого известного системе пользователя:

```
$ su - kernel
Пароль:
$ whoami
kernel
$ pwd
/home/guest
```

Некоторые дистрибутивы (Ubuntu и производные) запрещают регистрацию локального сеанса под именем `root`, в таких системах административные (привилегированные) команды выполняются с префиксной командой `sudo`.

**Примечание:** Особенностью таких систем может быть то, что, если вы попытаетесь выполнить привилегированную команду без префикса sudo, то она выполнится без всякого вывода (результата) и установки кода ошибочного результата, что может вводить в недоумение, сравните:

```
$ fdisk -l
$ echo $?
0
$ sudo fdisk -l
[sudo] password for olej:
Диск /dev/sda: 30.0 ГБ, 30016659456 байт
...
```

Сложнее обстоят дела с действиями над паролем пользователя. Вот как посмотреть **состояние** пароля пользователя (выполняется только с правами `root`):

```
$ sudo passwd -S olej
olej PS 2010-03-13 0 99999 7 -1 (Пароль задан, шифр SHA512.)
```

Но узнать (восстановить) пароль любого ординарного пользователя администратор не может, он может только

сменить его на новый — это один из основных принципов UNIX. А как следствие этого принципа: если вы утеряли пароль пользователя `root`, то легальных способов исправить ситуацию не существует ... да и нелегальные вряд ли помогут[13] — система достаточно надёжно защищена.

Ещё один часто задаваемый вопрос с не очевидным ответом: если команда создания пользователя `adduser` всегда создаёт пользователя с паролем входа, то как создать пользователя с пустым паролем? Для этого нам нужно удалить пароль у уже существующего пользователя:

```
# adduser guest
...
# passwd -d guest
Удаляется пароль для пользователя guest..
passwd: Успех
# passwd -S guest
guest NP 2011-03-23 0 99999 7 -1 (Пустой пароль.)
```

Все регистрационные записи пользователей (как созданных администратором, так и создаваемых самой системой) хранятся в файле:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
...
olej:x:500:500:O.Tsiliuric:/home/olej:/bin/bash
olga:x:501:501:olga:/home/olga:/bin/bash
```

- 3-е поле каждой записи — это численный идентификатор пользователя (UID), а 4-е - численный идентификатор основной группы (GID) этого пользователя. А вот значение 'x' во 2-м поле говорит о том, что пароль для пользователя хранится в файле теневых паролей:

```
$ ls -l /etc/shad*
-r-------- 1 root root 1288 Янв 24  2010 /etc/shadow
-r-------- 1 root root 1288 Янв 24  2010 /etc/shadow-
```

Обратите внимание на права доступа к этому файлу: даже `root` не имеет права записи в этот файл.

По умолчанию, при создании командой `adduser` нового пользователя с именем `xxx` создаётся и новая группа с тем же именем `xxx`. Кроме того, пользователя дополнительно можно включить в любое число существующих групп. Смотрим состав групп, их (групп) численные идентификаторы (GID) и принадлежность пользователей к группам:

```
$ cat /etc/group
...
nobody:x:99:
users:x:100:olej,games,guest,olga,kernel
...
olej:x:500:
...
```

- во 2-й показанной строке здесь `olej` — это имя пользователя в группе `users`, а в последней — имя группы, только совпадающее **по написанию** с именем `olej` пользователя: это результат отмеченного умолчания при создании пользователя, но от него можно и отказаться, задавая группу вручную.

Пример того, как получить информацию (если забыли) кто, как и где зарегистрирован и работает в системе на

13 Сказанное относится только к системе Linux, в которой администратором приняты адекватные меры для защиты и предотвращения несанкционированной **смены** пароля `root` (иногда называемой: восстановление утерянного пароля `root`). Практически в любой инсталляции, установленной из дистрибутива («из коробки»), это не так и пароль `root` может быть сменён. Средства восстановления пароля `root` описаны в приложении в конце текста.

текущий момент времени:

```
$ who
root     tty2         2011-03-19 08:55
olej     tty3         2011-03-19 08:56
olej     :0           2011-03-19 08:22
olej     pts/1        2011-03-19 08:22 (:0)
olej     pts/0        2011-03-19 08:22 (:0)
olej     pts/2        2011-03-19 08:22 (:0)
olej     pts/3        2011-03-19 08:22 (:0)
olej     pts/4        2011-03-19 08:22 (:0)
olej     pts/5        2011-03-19 08:22 (:0)
olej     pts/6        2011-03-19 08:22 (:0)
olej     pts/9        2011-03-19 09:03 (notebook)
```

- здесь: а). 2 (стр.1,2) регистрации в **текстовых консолях** (# 2 и 3) под разными именами (`root` и `olej`); б). X11 (стр.3) регистрация (консоль #7, CentOS 5.2 ядро 2.6.18); в). 7 открытых графических терминалов в X11, дисплей :0; г). одна удалённая регистрация по SSH (последняя строка) с компьютера **с сетевым именем** `notebook`.

А вот так узнать имя, под которым зарегистрирован пользователь на текущем терминале:

```
$ whoami
olej
```

- и это совсем не пустая формальность при одновременно открытых в системе нескольких десятков терминалов.

А как получают права `root`? На то есть несколько возможностей:

1. Перерегистрация[14]:

```
$ su -
Пароль:
#
```

2. Выполнение единичной команды от имени `root`:

```
$ su -c ls
Пароль:
build.articles
$ su -c 'ls -l'
Пароль:
итого 520
drwxrwxr-x 4 olej olej   4096 Map 13 19:54 build.articles
...
```

3. Наилучший способ выполнения команды от имени `root`:

```
$ sudo makewhatis
...
```

Но иногда (в зависимости от дистрибутива), при первом употреблении команда `sudo` нещадно ругается... В этом случае нужно настроить поведение `sudo`:

```
cat /etc/sudoers
...
## Same thing without a password
# %wheel    ALL=(ALL)    NOPASSWD: ALL
%olej       ALL=(ALL)    NOPASSWD: ALL
```

14 Уже отмечалось, что такая возможность запрещена в Ubuntu и родственных дистрибутивах.

...
- здесь показана одна новая строка, добавленная по образцу закомментированной, разрешающая «беспарольный sudo» для пользователя с именем `olej`.

## *Файловая система: структура и команды*

В файловой системы UNIX сверх того, что уже было сказано о файловой системе ранее, работаю правила:

1. каждый объект имеет **имя**, которое может состоять из нескольких доменов-имён, разделённых символом '.' (точка), понятие «расширение» или «тип» (как в системе именования «8.3», идущей ещё от MS-DOS) не имеет такого жёсткого смысла (определяющего функциональное назначение файла):

```
$ touch start.start.start
$ ls
start.start.start
```

2. каждый объект имеет полное **путевое имя** (путь от корня файловой системы, абсолютное имя), которое составляется из имени включающего каталога и собственно имени объекта (файла):

```
$ pwd
/home/guest
```

В данном случае полное путевое имя только-что созданного выше файла будет выглядеть так:
`/home/guest/start.start.start`

В файловой системе не может быть двух элементов с полностью совпадающими путевыми именами.

Путевое имя может быть абсолютным (показано выше) и относительным: относительно текущего каталога (или/и каталогов промежуточного уровня): `/home/guest/start.start.start` и `./start.start.start` - это одно и то же имя (в условиях обсуждаемого примера). Когда мы находимся, например, в каталоге:

```
$ pwd
/var/cache/yum/updates
```

- то путевые имена: `../../../log/mail` и `/var/log/mail` — это одно и то же:

```
$ ls ../../../log/mail
statistics
$ ls /var/log/mail
statistics
```

В относительных именах часто используется знак '~' - **домашний** каталог текущего пользователя:

```
$ cd ~/Download/
$ pwd
/home/olej/Download
```

3. функциональное назначение имени (файла) может быть определено (не всегда точно![15]) командой `file` :

```
$ file start.start.start
start.start.start: empty
$ file KERNEL_11.odt
KERNEL_11.odt: OpenDocument Text
$ file /dev/hde
/dev/hde: block special (33/0)
$ file /dev/tty
/dev/tty: character special (5/0)
$ file mod_proc.ko
mod_proc.ko: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
```

15 Команда `file` привлекает для «угадывания» формата и назначения файла несколько разнородных механизмов, таких как: начиная от магических символов (заголовочных байт) в начале файла, и до просмотра формата и содержимого файла.

```
$ file mod_proc.c
mod_proc.c: UTF-8 Unicode C program text
$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9, dynamically
linked (uses shared libs), for GNU/Linux 2.6.9, not stripped
```

4. каждый объект (имя) файловой системы имеет **2-х** владельцев: пользователя и группу:

```
$ ls -l
-rw-rw-r-- 1 guest guest 0 Map 27 14:20 start.start.start
```

При создании файла обычно группой является **первичная** группа создающего пользователя (но в общем случае, элементы создаются процессами, которые могут накладывать свои правила).

## Владельцы и права

В смысле прав доступа к объекту файловой системы определены 3 уровня (группы): владелец, группа, остальные. В каждой группе права определены триадой (прав): `r` — чтение, `w` — запись, `x` — исполнение (для для каталогов есть отличия в толковании флагов: `w` — это право создания и удаления объектов в каталоге, `x` — это право вхождение в каталог).

```
$ sudo chown olej:guest start.start.start
$ ls -l
-rw-rw-r-- 1 olej guest 0 Map 27 15:00 start.start.start
$ sudo chgrp users start.start.start
$ ls -l
-rw-rw-r-- 1 olej users 0 Map 27 15:00 start.start.start
$ sudo chown root *
$ ls -l
-rw-rw-r-- 1 root users 0 Map 27 15:00 start.start.start
```

- пользователь и группа владения меняются, а установленное расположение флагов относительно владельца и группы — остаётся.

Изменение прав (`u` — владелец, `g` — группа владения, `o` — остальные, `a` — все):

```
$ sudo chmod a+x  start.start.start
$ ls -l start.start.start
-rwxrwxr-x 1 root users 0 Map 27 15:00 start.start.start
$ sudo chmod go-x  start.start.start
$ ls -l start.start.start
-rwxrw-r-- 1 root users 0 Map 27 15:00 start.start.start
$ sudo chmod go=r  start.start.start
$ ls -l start.start.start
-rwxr--r-- 1 root users 0 Map 27 15:00 start.start.start
$ sudo chmod 765  start.start.start
$ ls -l start.start.start
-rwxrw-r-x 1 root users 0 Map 27 15:00 start.start.start
```

Флаг `x` должен выставляться для любых файлов, подлежащих исполнению; его отсутствие — частая причина проблем с выполнением текстовых файлов содержащих скриптовые сценарии (на языках: `bash`, `perl`, `python`, ...).

### *Информация о файле*

Детальную информацию о файле (в том числе и атрибутах) даёт команда `stat`:

```
$ stat start.start.start
  File: `start.start.start'
  Size: 0               Blocks: 0          IO Block: 4096   пустой обычный файл
Device: 2146h/8518d     Inode: 1168895     Links: 1
```

```
Access: (0765/-rwxrw-r-x)  Uid: (    0/    root)   Gid: (  100/   users)
Access: 2011-03-27 15:00:35.000000000 +0000
Modify: 2011-03-27 15:00:35.000000000 +0000
Change: 2011-03-27 15:38:06.000000000 +0000
```

У команды есть множество опций, позволяющих определить формат вывода команды `stat`:

```
$ stat -c%A start.start.start
-rwxrw-r-x
$ stat -c%a start.start.start
765
```

### *Дополнительные атрибуты файла*

У файла могут устанавливаться дополнительные атрибуты, которые выставляются флагам. Такими реально употребляемыми атрибутами являются (для атрибута числом показано значение флага в 1-м байте атрибутов):

- установка идентификатора пользователя (`setuid`) - 4;

- установка идентификатора группы (`setgid`) - 2;

- установка `sticky`-бита - 1;

Последний атрибут устарел, и используется редко. А вот возможность установки `setuid` и/или `setgid` принципиально для UNIX и устанавливаются они для исполнимых файлов: при запуске файла программы с такими атрибутами, запущенная программа выполняется не с правами (от имени) запустившего её пользователя (зарегистрировавшегося в терминале, как обычно), а от имени того пользователя (группы) который установлен как владелец этого файла программы. Это обычная практика использования `setuid` , например, когда:

- владельцем файла программы является `root`;
- и такая программа должна иметь доступ к файлам данных, доступных только `root` (например `/etc/passwd`)...
- нужно дать возможность рядовому пользователю выполнять такую программу, но не давать пользователю прав `root`;
- элементарным примером такой ситуации является то, когда вы меняете **свой** пароль доступа к системе, выполняя от своего имени команду `passwd` (проанализируйте эту противоречивую ситуацию).

Принципиально важное значение имеет возможность установки `setuid` и/или `setgid` для исполнимых файлов, в числовой записи прав доступа это выглядит так:

```
$ stat -c%a start.start.start
765
$ sudo chmod 2765  start.start.start
$ stat -c%a start.start.start
2765
$ stat -c%A start.start.start
-rwxrwSr-x
$ sudo chmod 4765  start.start.start
$ stat -c%A start.start.start
-rwsrw-r-x
$ sudo chmod 1765  start.start.start
$ stat -c%A start.start.start
-rwxrw-r-t
```

В символической записи прав для `chmod` ключ `-s` устанавливает `setuid` и `setgid` (одновременно, нет возможности управлять ими раздельно):

```
$ stat -c%a start.start.start
765
```

```
$ sudo chmod a+s start.start.start
$ stat -c%a start.start.start
6765
$ stat -c%A start.start.start
-rwsrwSr-x
```

## Навигация в дереве имён

Здесь мы вспомним как перемещаться по каталогам дерева, ориентироваться где мы находимся, и искать нужные нам места файловой системы:

```
$ pwd
/home/olej/2011_WORK/Linux-kernel
$ echo $HOME
/home/olej
$ cd ~
$ pwd
/home/olej
```

Поиск бинарных файлов может осуществляться:

- только исполняемых файлов на путях из списка переменной `$PATH` для запуска приложений:

```
$ which java
/opt/java/jre1.6.0_18/bin/java
```

- поиск бинарных и некоторых других типов файлов (`man`) в списке каталогов их основного нахождения:

```
$ whereis java
java: /usr/bin/java /etc/java /usr/lib/java /usr/share/java
```

Можно видеть, что в 1-м случае найден файл из списка каталогов в переменной `$PATH`, который будет запускаться по имени без указания пути, а во 2-м случае — файлы, не лежащие в этих каталогах:

```
$ echo $PATH
/opt/java/jre1.6.0_18/bin:/usr/lib/qt-3.3/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin
```

Наконец, есть утилита, позволяющая находить файл в любом месте файловой системе по любым самым сложным и комбинированным критериям поиска:

```
$ find /etc -name passwd
/etc/passwd
find: /etc/libvirt: Отказано в доступе
/etc/pam.d/passwd
find: /etc/lvm/cache: Отказано в доступе
...
```

- в данном примере найдено 2 файла по критерию «искать файл с именем...».

## Основные операции

Основные операции над объектами файловой системы (чаще всего эти объекты — файлы, но это могут быть и файлы особого рода — каталоги, и даже вообще не файлы — псевдофайлы, устройства, путевые имена в файловой системе)...

Создание каталога:

```
$ mkdir newdir
$ cd newdir
```

Создание нового (пустого) файла, что бывает нужно достаточно часто:

```
$ touch cmd.txt
```

Другой способ создания нового файла — команда `cat`:

```
$ cat > new.file
123
^C
$ cat new.file
123
```

Копирование файла, или целога каталога файлов (для копирования всех файлов каталога указание ключа рекурсивности `-R` обязательно):

```
$ cp ../f1.txt cmd.txt
$ cp -R /etc ~
cp: невозможно открыть `/etc/at.deny' для чтения: Отказано в доступе
cp: невозможно открыть `/etc/shadow-' для чтения: Отказано в доступе
cp: невозможно открыть `/etc/gshadow-' для чтения: Отказано в доступе
...
```

Подсчитать суммарный объём, занимаемый всеми файлами в указанном каталоге:

```
$ du -hs ~/etc
89M     /home/olej/etc
```

Удалить **каталог**:

```
$ rmdir ~/etc
rmdir: /home/olej/etc: Каталог не пуст
```

Команда завершается ошибкой, так как в каталоге имеются файлы. Но команда рекурсивного удаления **файлов** (каталог — файл один из ...) справится с той же задачей:

```
$ rm -R ~/etc
rm: удалить защищенный от записи обычный файл `/home/olej/etc/sudoers'? Y
...
$ rm -Rf ~/etc
$ ls ~/etc
ls: /home/olej/etc: Нет такого файла или каталога
```

Перемещение или переименование файла (или целой иерархии файлов — каталога). Если перемещение происходит в пределах одного каталога, то это переименование (1-я команда), иначе — реальное перемещение (2-я команда):

```
$ mv cmd.txt list.txt
$ mv list.txt ../cmd.txt
```

Побайтовое сравнение файлов по содержимому:

```
$ cmp -s huck.tgz huck1.tgz
$ echo $?
0
```

Для сравнения текстовых файлов (файлов программного кода) с выделением различий для последующего применения команды `patch`, используется другая команда. Ниже показан короткий законченный пример создания такой заплатки, и последующего его наложения на файл — этого достаточно для понимания основ принципа:

```
$ diff --help
Использование: diff [КЛЮЧ]... ФАЙЛЫ
Построчно сравнивает два файла.
...
$ echo 1234 > f4
```

```
$ echo 12345 > f5
$ diff f4 f5
1c1
< 1234
---
> 12345
$ diff f4 f5 > 45.patch
$ patch -i 45.patch f4
patching file f4
$ cmp f4 f5
$ echo $?
0
$ ls -l
итого 12
-rw-rw-r-- 1 olej olej 23 Апр 14 07:12 45.patch
-rw-rw-r-- 1 olej olej  6 Апр 14 07:13 f4
-rw-rw-r-- 1 olej olej  6 Апр 14 07:10 f5
```

Команды с потоками и конвейерами:

```
$ cat url.txt >> cmd.txt
$ echo 111 > newfile
$ echo $?
0
$ ls -l newf*
-rw-rw-r-- 1 olej olej 4 Map 19 15:17 newfile
```

Команда «размножения» потока вывода на несколько потоков, с записью каждого такого экземпляра потока в свой отдельный файл:

```
$ pwd
/home/olej/TMP
$ ls
$ echo 12345 | tee 1 2 3 4 > 5
$ ls
1  2  3  4  5
```

Некоторые служебные операции над файловой системой:

- сбросить буфера файловой системы на диск:

```
$ sync
```

- проверка (и восстановление) структуры файловой системы на носителе:

```
# fsck -c
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
/dev/hdf6 is mounted.
WARNING!!!  Running e2fsck on a mounted filesystem may cause
SEVERE filesystem damage.
Do you really want to continue (y/n)? no
check aborted.
```

Этот пример говорит о том, что проверку дисковых носителей (файловых систем) нужно производить в размонтированном состоянии. Единая команда `fsck` будет вызывать другую программу, соответствующую тому типу файловой системы, который обнаружен на этом носителе, вот из этого числа:

```
$ ls /sbin/fsck*
fsck  fsck.cramfs  fsck.ext2  fsck.ext3  fsck.msdos  fsck.vfat
```

## Архивы

В Linux имеется множество программ архивирования и сжатия информации. Но почти на все случаи жизни достаточно средств архивирования, интегрированных в реализацию утилиты `tar`:

```
$ tar -zxvf abs-guide-flat.tar.gz
$ tar -jxvf ldd3_pdf.tar.bz2
```

- это показано создание разархивирование из форматов `.zip` и `.bz2`, соответственно.

А вот так сворачиваются в архив `.zip` файлы текущего каталога:

```
$ tar -zcvf new-arch.tgz *
```

А вообще, в Linux собраны архиваторы, работающие с форматами, накопившимися практически за всё время существования системы:

```
$ cd /usr/bin/
$ ls *zip*
bunzip2  bzip2recover  gpg-zip  gzip  p7zip     prezip      unzip     zip       zipgrep  zipnote
bzip2    funzip        gunzip   mzip  preunzip  prezip-bin  unzipsfx  zipcloak  zipinfo  zipsplit
```

## Устройства

Как сказано ранее, все имена в каталоге `/dev` соответствуют устройствам системы. Каждое устройство (кроме имени в `/dev`) однозначно характеризуется двумя номерами: старший, `major` — родовой номер класса устройств, младший, `minor` — индивидуальный номер устройства внутри класса. Именно через эти два номера происходит связь устройства с ядром Linux (или, если совсем точно, с модулем-драйвером этого устройства в составе ядра). Сами имена устройства системе не нужны — они нужны человеку-пользователю для его удобства. Номера не произвольные. Все известные номера устройств описаны в документе `devices.txt` (лежит в дереве исходных кодов ядра[16]):

```
$ cd /usr/src/linux/Documentation
$ ls -l devices.txt
-rw-rw-r-- 1 olej olej 118626 Map  8 01:05 devices.txt
$ cat devices.txt
                    LINUX ALLOCATED DEVICES (2.6+ version)
             Maintained by Alan Cox <device@lanana.org>
                      Last revised: 6th April 2009
...
```

Все устройства делятся на символьные и блочные (устройства прямого доступа, диски). Они различаются по первой литере в выводе содержимого каталога `/dev` :

```
$ ls -l /dev | grep ^c
crw-------  1 root video      10, 175 Июл 31 10:42 agpgart
crw-rw----  1 root root       10,  57 Июл 31 10:43 autofs
crw-------  1 root root        5,   1 Июл 31 10:42 console
...
$ ls -l /dev | grep ^b
...
brw-rw----  1 root disk        8,   0 Июл 31 10:42 sda
brw-rw----  1 root disk        8,   1 Июл 31 10:42 sda1
brw-rw----  1 root disk        8,   2 Июл 31 10:42 sda2
```

16 Когда говорят о исходных кодах Linux, нужно иметь в виду, что они не присутствуют в системе изначально: в некоторых дистрибутивах (Debian и др.), вы можете загрузить их из репозитариев вашего дистрибутива менеджером пакетов, в других (Fedora, CentOS и др.) это делается более «ручным» способом... это что касается «патченных» под дистрибутив ядер. Код официального вы можете загрузить самостоятельно с адреса http://www.kernel.org/ - для обсуждаемых нами целей (рассмотрение файлов) тонкие отличия не имеют значения: вы должны выбрать архив вашей версии ядра (архив вида `linux-2.6.37.3.tar.bz2`), разархивировать его в каталог `/usr/src` (это потребует около 500Mb) и, обычно, на полученный каталог устанавливают ссылку `/usr/src/linux` — это и есть дерево исходных кодов Linux ...

```
brw-rw----  1 root disk        8,   3 Июл 31 10:42 sda3
...
```

Старшие (major) номера символьных и блочных номеров могут совпадать: они принадлежат к разным пространствам номеров. Но вот внутри класса полной идентичности двух номеров (major+minor) не может быть.

Создание нового имени устройства в каталоге /dev:

```
# mknod -m 0777 /dev/hello c 200 0
```

- в команде указываются (кроме имени устройства): права доступа к имени, характер устройства (символьное или блочное), и два номера, характеризующих устройство.

Так же, как и любое путевое имя, имена устройств удаляются командной:

```
# rm /dev/hello
```

**Примечание:** в нарушение любых приличий, файл устройства можно создавать в произвольном месте, например, в текущем каталоге :

```
$ pwd
/home/olej
$ sudo mknod -m 0777 ./hello c 200 0
$ echo $?
0
$ ls -l hello
crwxrwxrwx 1 root root 200, 0 Map 19 16:27 hello
$ sudo rm ./hello
$ echo $?
0
```

Самые разнообразные устройства представляются в /dev. Часто задаваемый вопрос: как представлены, например, последовательные линии связи RS-232 (RS-485)? Вот они:

```
$ ls -l /dev/ttyS*
crw-rw---- 1 root uucp 4, 64 Апр 27 06:19 /dev/ttyS0
crw-rw---- 1 root uucp 4, 65 Апр 27 06:19 /dev/ttyS1
crw-rw---- 1 root uucp 4, 66 Апр 27 06:19 /dev/ttyS2
crw-rw---- 1 root uucp 4, 67 Апр 27 06:19 /dev/ttyS3
```

Причём, представлены как терминальные линии все 4 (максимально возможные) каналы RS-232, но откликаться на команды (например, конфигурироваться командой stty) будут только линии, реально представленные в аппаратуре компьютера (часто /dev/ttyS0 и /dev/ttyS1 — COM1 и COM2 в терминологии MS-DOS).

## Подсистема udev

udev - подсистема, которая заменяет devfs без потерь для функциональности системы. Более того, udev создаёт в /dev файлы только для тех устройств, которые присутствуют на данный момент в системе. Подсистема udev является надстройкой пространства пользователя над /sys. Задача ядра определять изменения в аппаратной конфигурации системы, регистрировать эти изменения, и вносить изменения в каталог /sys. Задача подсистемы udev выполнить дальнейшую интеграцию и настройку такого устройства в системе (отобразить его в каталоге /dev), и предоставить пользователю уже готовое к работе устройство.

Подсистема udev настраивает устройства в соответствии с заданными правилами. Правила содержаться в файлах каталога /etc/udev/rules.d/ (также файлы с правилами могут содержаться и в каталоге /etc/udev/). Все файлы правил просматриваются в алфавитном порядке.

```
$ ls /etc/udev/rules.d/
05-udev-early.rules  51-hotplug.rules  60-pcmcia.rules  61-uinput-stddev.rules  90-dm.rules          bluetooth.rules
```

```
40-multipath.rules   60-libsane.rules  60-raw.rules     61-uinput-wacom.rules   90-hal.rules
50-udev.rules        60-net.rules      60-wacom.rules   90-alsa.rules           95-pam-console.rules
$ cat 60-raw.rules
...
# An example would be:
#   ACTION=="add", KERNEL=="sda", RUN+="/bin/raw /dev/raw/raw1 %N"
...
```

Информация по `udev` :

```
$ man udev
UDEV(7)                                udev                                UDEV(7)
NAME
       udev - dynamic device management
...
```

Основной объём потребностей по работе с `udev` покрывает не очень широко известная команда `udevadm` с огромным множеством параметров и опций:

```
$ udevadm info -q path -n sda
/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda
$ udevadm info -a -p $(udevadm info -q path -n sda)
...
looking at device '/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda':
    KERNEL=="sda"
    SUBSYSTEM=="block"
...
$ udevadm info -h
Usage: udevadm info OPTIONS
  --query=<type>             query device information:
      name                     name of device node
      symlink                  pointing to node
      path                     sys device path
      property                 the device properties
      all                      all values
  --path=<syspath>           sys device path used for query or attribute walk
  --name=<name>              node or symlink name used for query or attribute walk
...
```

Разработчики прикладных систем часто сталкиваются с `udev` в разработке конфигурационных правил для своих систем (пример: системы SoftSwitch для VoIP PBX и их интерфейс к аппаратуре связи `zaptel/DAHDI`).

## *Команды диагностики оборудования*

Характерное отличие потребностей программиста-разработчика (как, собственно, и системного администратора) от потребностей пользователя Linux состоит в том, что разработчику часто нужны средства детальной диагностики установленного в системе периферийного оборудования (диагностики по типу, производителю, модели, по функционированию и другое). В отношении анализа всего установленного в системе оборудования, начиная с анализа производителя и BIOS — существует достаточно много команд «редкого применения», которые часто помнят только заматерелые системные администраторы, и которые не всегда попадают в справочные руководства. Все такие команды, в большинстве, требует прав `root`, кроме того, некоторые из них могут присутствовать в некоторых дистрибутивах Linux, но отсутствовать в других (устанавливаются в составе системных пакетов, если утилиты нет, то можно легко определить нужный пакет и установить его из репозитария). Информация от этих команд в какой-то мере дублирует друг друга (а в какой-то - дополняет). Но сбор такой информации об оборудовании может стать ключевой позицией при работе с периферийными устройствами.

Ниже приводится только краткое перечисление (в порядке справки-напоминания) некоторых подобных команд (и несколько начальных строк вывода, для идентификации того, что это именно та команда о которой мы

говорим) — более детальное обсуждение увело бы нас слишком далеко от наших целей. Вот некоторые такие команды:

```
$ lspci
...
00:1c.0 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 1 (rev 01)
00:1c.2 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 3 (rev 01)
00:1c.3 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 4 (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #3 (rev 01)
00:1d.3 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #4 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801G (ICH7 Family) USB2 EHCI Controller (rev 01)
...
$ lsusb
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 003: ID 0461:4d17 Primax Electronics, Ltd Optical Mouse
Bus 004 Device 002: ID 0458:0708 KYE Systems Corp. (Mouse Systems)
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 006: ID 08ff:2580 AuthenTec, Inc. AES2501 Fingerprint Sensor
Bus 001 Device 003: ID 046d:080f Logitech, Inc.
Bus 001 Device 002: ID 0424:2503 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
$ lshal
Dumping 162 device(s) from the Global Device List:
-------------------------------------------------
udi = '/org/freedesktop/Hal/devices/computer'
  info.addons = {'hald-addon-acpi'} (string list)
...
$ sudo lshw
notebook.localdomain
    description: Notebook
    product: HP Compaq nc6320 (ES527EA#ACB)
    vendor: Hewlett-Packard
    version: F.0E
    serial: CNU6250CFF
    width: 32 bits
    capabilities: smbios-2.4 dmi-2.4
...
```

Детальная информация, в том числе, по банках памяти, и какие модули памяти куда установлены:

```
$ sudo dmidecode
# dmidecode 2.10
SMBIOS 2.4 present.
23 structures occupying 1029 bytes.
Table at 0x000F38EB.
...
```

Пакет `smartctl` (предустановлен почти в любом дистрибутиве) - детальная информация по дисковому накопителю:

```
$ sudo smartctl -A /dev/sda
smartctl 5.39.1 2010-01-28 r3054 [i386-redhat-linux-gnu] (local build)
Copyright (C) 2002-10 by Bruce Allen, http://smartmontools.sourceforge.net

=== START OF READ SMART DATA SECTION ===
SMART Attributes Data Structure revision number: 16
```

```
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   100   100   046    Pre-fail  Always       -       49961
  2 Throughput_Performance  0x0005   100   100   030    Pre-fail  Offline      -       15335665
  3 Spin_Up_Time            0x0003   100   100   025    Pre-fail  Always       -       1
  4 Start_Stop_Count        0x0032   098   098   000    Old_age   Always       -       7320
...
```

Ещё один способ получения информации о дисковом накопителе:

```
$ sudo hdparm -i /dev/sda
/dev/sda:
 Model=WDC WD2500AAKX-001CA0, FwRev=15.01H15, SerialNo=WD-WMAYU0425651
 Config={ HardSect NotMFM HdSw>15uSec SpinMotCtl Fixed DTR>5Mbs FmtGapReq }
 RawCHS=16383/16/63, TrkSize=0, SectSize=0, ECCbytes=50
 BuffType=unknown, BuffSize=16384kB, MaxMultSect=16, MultSect=16
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=488397168
 IORDY=on/off, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
 PIO modes:  pio0 pio3 pio4
 DMA modes:  mdma0 mdma1 mdma2
 UDMA modes: udma0 udma1 udma2 udma3 udma4 udma5 *udma6
 AdvancedPM=no WriteCache=enabled
 Drive conforms to: Unspecified:  ATA/ATAPI-1,2,3,4,5,6,7
```

Все такие команды имеют разветвлённую систему опций, определяющих вид затребованной информации. Все они имеют онлайновую систему подсказок (ключи `-v`, `-h`, `--help`), позволяющую разобраться со всем этим множеством опций.