Хранение информации об учетной записи пользователя

`/etc/passwd`

`username:password:uid:gid:comment:home_dir:login_shell`

`/etc/shadow` детали шифрования пароля и его устаревание

`username:password:lastchg:min:max:inactive:expire`

↑ шифрованный пароль

или LK – недоступная запись, NP – нет пароля

`/etc/group` информация о группах

`group_name:group_password:gid:user_list`

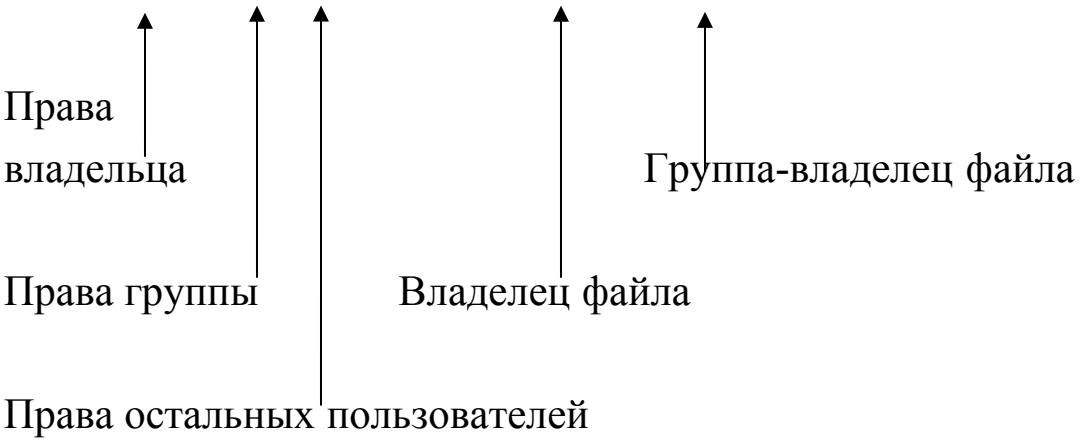
Права доступа к файлам

Категории пользователей

user	владелец файла
group	члены группы
other	все другие пользователи (не root, не владелец, не член группы)

```
$ ls -l
```

-	rw-	r--	r--	1	user3	class	37	Jul 24	11:06	f1
-	rwx	r-x	r-x	1	user3	class	52	Jul 24	11:08	f2
d	rwx	r-x	r-x	2	user3	class	1024	Jul 24	12:03	memo



Задание прав доступа в числовом формате

Каждое право доступа обозначается числом:
read=4
write=2
execute=1
Числа складываются для каждой категории пользователей

Примеры

1.

user	group	others
rw-	r--	---
4+2+0	4+0+0	0+0+0
6	4	0

```
chmod 640 имя_файла
```

2. `chmod 000 имя_файла` Отмена всех прав доступа к файлу

Маскирование прав доступа. Команда umask

Синтаксис:

umask [-S] [*режим_доступа*] ...

Примеры

1. `umask 022`

   Значение, заданное с помощью `umask`, вычитается из значения по умолчанию

   (умолчание - 666 для файла, 777 для каталога)

2. `umask -S u=rwx,g=rw-,o=---`

Команда umask

	Символьный вид	Числовой вид
Значение umask	**--- -w- -wx**	**023**
Маска по умолчанию для файла	**rw- rw- rw-**	**666**
Маска по умолчанию для каталога	**rwx rwx rwx**	**777**
Права на вновь созданный файл	**rw- r-- r--**	**644**
Права на вновь созданный каталог	**Rwx r-x r--**	**754**

Изменение прав доступа. Команда chmod

Синтаксис:

chmod *режим_доступа файл ...*	Изменение прав доступа к файлу(ам)
режим_доступа	[кто[оператор]право][...]
кто	владелец (**u**ser), группа (**g**roup), остальные (**o**ther) или все (**a**ll)
оператор	+ (добавить), - (отнять), = (присвоить)
право	чтение (**r**ead), запись (**w**rite), выполнение (e**x**ecute)

Команда chmod

Примеры

Исходные права:	режим	владелец	группа	остальные
	rw-r--r--	rw-	r--	r--

$ **`chmod u+x,g+x,o+x file`**

Новые права:	режим	владелец	группа	остальные
	rwxr-xr-x	rwx	r-x	r-x

$ **`chmod +x file`**

$ **chmod = имя_файла** Удаление всех прав доступа к файлу

$ **`chmod o= file`**

Новые права:	режим	владелец	группа	остальные
	rwxr-x---	rwx	r-x	---

Команда chmod

Примеры

/dev/tty0p1 - файл устройства, отвечающий за связь терминала пользователя с компьютером

$ **ls -l /dev/tty0**

crw--w--w- 1 bin bin 58 ox000003 Feb 15 11:34 /dev/tty0

$ **mesg n**

$ **ls -l /dev/tty0**

crw------- 1 bin bin 58 ox000003 Feb 15 11:34 /dev/tty0

Команда chown

Синтаксис:

chown *владелец [:группа] имя_файла* Изменение владельца файла и, дополнительно, группы

Пример:

```
$ id
uid=303 (user3), gid=300 (class)
$ cp fl /home/user2/fl
$ ls -1 /home/user2/fl
-rw-r----- 1 user3 class 3967 Jan 24 13:13 fl
$ chown user2 /home/user2/fl
$ ls -1 /home/user2/fl
-rw-r----- 1 user2 class 3967 Jan 24 13:13 fl
```

!Только владелец файла (или суперпользователь) может изменить владельца файла

Команда chgrp

Синтаксис:

chgrp *новая_группа имя_файла* — Изменяет права доступа группы к файлу

!Только владелец файла (или суперпользователь) может изменить группу файла

Пример:

```
$ id
uid=303 (user3), gid=300 (class)
$ ls -1 f3
-rw-r-––– 1 user3 class 3967 Jan 24 13:13 f3
$ chgrp class2 f3
$ ls -l f3
-rw-r----- 1 user3 class2 3967 Jan 24 13:13  f3
$ chown user2 f3
$ ls -l f3
-rw-r----- 1 user2 class2 3967 Jan 24 13:13  f3
```

Команда su

Синтаксис:

su *[-] [имя_пользователя]...*

Примеры

```
$ ls -1 /usr/local/bin/class_setup
-rwxr-x--- 1 class_admin teacher 3967 Jan    24      13:13   f3
$ id
uid=303 (user3), gid=300 (class)
$ su class_admin
Password:
$ id
uid=400 (class_admin), gid=300 (class)
$ /usr/local/bin/class_setup
$ <Ctrl>+<d><Enter>
log out of su session
$
```

Команда newgrp

Синтаксис:

newgrp *[имя_группы]*...

Примеры

```
$ ls -1 /usr/ local/bin/class_setup
-rwxr-x--- 1 class_admin teacher 3967 Jan 24 13:13    f3
$ id
uid=303 (user3), gid=300 (class)
$ newgrp teacher
$ id
uid=303 (user3), gid=33 (teacher)
$ /usr/local/bin/class_setup
$ newgrp
Enter to login group status
$ newgrp other
Sorry
$
```

Дополнительные атрибуты защиты файла

Атрибут	Описание
`setuid bit`	Бит смены эффективного идентификатора пользователя. При запуске исполняемого файла с установленным `setuid bit` программа будет выполняться с правами того пользователя, которому принадлежит файл
`setgid bit`	Бит смены эффективного идентификатора группы. При запуске исполняемого файла с установленным `setgid bit` программа будет выполняться с правами той группы, которой принадлежит файл
`sticky bit`	“Липкий” бит. Если на каталог установлен `sticky bit`, то файлы в каталоге может удалить или переименовать только тот пользователь, кто имеет право на запись в каталог и является (одним из): – владельцем файла, – владельцем каталога, – суперпользователем. Если значение `sticky-bit` равно `t`, то бит исполнения для всех пользователей включен, если значение `sticky-bit` равно T, то бит исполнения для всех пользователей выключен.

Дополнительные атрибуты защиты файла

Дополнительные атрибуты устанавливаются с помощью команды chmod

Пример

```
$ chmod +t mydirdoc

$ ls –l mydirdoc

drw------t 1 vova other 109056 Aug 03 15:27 mydirdoc
```

Списки управления доступом ACL

С помощью ACL можно устанавливать права для конкретных групп и пользователей.

Формат элемента списка ACL

```
тип_элемента:uid|gid:права
```

тип_элемента -- user, group,other,mask

uid -- имя пользователя или его ид.номер

gid -- имя группы или ее ид.номер

права -- аналогично chmod

Просмотр списка ACL

```
getfacl файл
```

Пример

```
getfacl /export/home/vova
#file: /export/home/vova
#owner: vova
#group: staff
user::rwx
user:peter:rwx
group::r--
mask:rwx
other:---
```

Установка списка ACL

`setfacl [-m|-d|-s] список файл`

`-m` – модифицирует существующий ACL

`-d` – удаляет ACL

`-s` – полностью заменяет ACL

ACL. Примеры

```
setfacl -s user::rw-,group::r--,other:---,
          mask:rw-,user:peter:rw- myfile

setfacl -m u:vasya:rw- myfile

setfacl -d u:peter  file1  file2
```

Использоваине шаблона

```
getfacl file1 > fff

setfacl -f fff  file2
```

или

```
getfacl file1 | setfacl -f file2
```

Процессы

ПРОЦЕСС -- это совокупность данных ядра системы, необходимых для описания образа программы в памяти и управления ее выполнением.

Процесс состоит из

- адресного пространства
- инструкций, выполняемых процессором
- набора структур данных в ядре ОС для отслеживания этого процесса (размещенная память, открытые файлы и статус процесса).

Типы процессов

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

Примеры системных процессов:

`shed` – диспетчер своппинга

`vhand` – диспетчер страничного замещения

`bdfflush` – диспетчер буферного кэша

`kmadaemon` – диспетчер памяти ядра

! К системным процессам также относят `init`, хотя у него есть исполняемый файл (`/etc/init`).

Типы процессов

2. Демоны (daemons)

Неинтерактивные процессы, которые запускаются обычным образом – путем загрузки в память соответствующих программ, и выполняются в фоновом режиме.

3. Прикладные (пользовательские) процессы

Как правило, запускаются в рамках пользовательского сеанса. Важнейший пользовательский процесс – `shell`.

Атрибуты процесса

PID	Идентификатор процесса
PPID	Идентификатор родительского процесса
UID	Идентификатор пользователя (владельца)
EUID	Действующий (эффективный) идентификатор владельца
GID	Идентификатор группы процессов
EGID	Действующий (эффективный) идентификатор группы
Priority	Приоритет

Иерархия процессов

Потомок наследует:

- идентификаторы пользователя и группы
- переменные окружения
- диспозицию сигналов (действие при получении сигнала) и их обработчики
- ограничения, накладываемые на процесс
- текущий каталог
- маску создания файлов
- управляющий терминал

.

Иерархия процессов

***Потомок не наследует*:**

- PID
- PPID
- сигналы, ожидающие доставки
- значение, возвращаемое `fork()`, разное
  (у родителя равно `pid` дочернего процесса
  или 1 при ошибке, у потомка - 0)

.

Наблюдение за процессами. Команда ps

Синтаксис:

ps [-efl] Выводит информацию о состоянии процесса

Пример:

$ ps

PID	TTY	TIME	COMMAND
1324	ttyp2	0:00	sh
1387	ttyp2	0:00	ps

$ ps -ef

UID	PID	PPID	C	STIME	TTY	TIME	COMMAND
root	0	0	0	Jun 1	?	0:20	swapper
root	1	0	0	Jun 23	?	0:00	init
root	2	0	0	Jun 23	?	0:16	vhand
root	3	0	0	Jun23	?	12:14	statdeamon
user3	1324	1	3	18:03:21	ttyp2	0:00	-sh
user3	1390	1324	22	18:30:23	ttyp2	0:00	ps-ef

Опции команды ps

Без опций	Краткая информация о процессах, связанных только с сеансом работы с системой запустившего ее пользователя.
-e	Выдача списка **всех** процессов, развивающихся в системе
-f	Получение **полной** информации о состоянии процессов
-l	Получение информации о процессах в т.н. **длинном** формате

Наблюдение за процессами

ps [-f] выводит на экран информацию о выполняемых (или ожидающих) процессах в системе

UID идентификатор владельца процесса

PID идентификатор процесса

PPID идентификатор родителя процесса

STIME начальное время процесса

TTY идентификатор терминала, с которого запущен процесс

TIME совокупное время выполнения процесса

COMMAND команда, соответствующая данному процессу

Наблюдение за процессами

```
$ ps -f
UID    PID   PPID       C          STIME  TTY  TIME   COMMAND
user3 4702  1          1          08:46:40 ttyp4  0:00      -sh
user3 4895   4702       18         09:55:10 ttyp4  0:00      ps -f

$ ksh ….программа

$ ps -f
UID PID     PPID       C          STIME   TTY  TIME   COMMAND
user3 4702  1          0          08:46:40 ttyp4  0:00        -sh
user3 4896  4702       1          09:57:20 ttyp4  0:00        ksh
user3 4898  4896       18         09:57:26 ttyp4  0:00        ps -f
```

Наблюдение за процессами

prstat – статистика по активным процессам

Пример

```
 $prstat   -u   ovs
  PID USERNAME SIZE    RSS  STATE PRI NICE TIME  CPU  PROCESS/NLWP
  2289 ovs          5600K 2932K cpu1    59    0   0:00:00 0,0% prstat/1
  2220 ovs          8232K 3624K sleep   59    0   0:00:00 0,0% sshd/1
  2226 ovs          4760K 1724K sleep   59    0   0:00:00 0,0% bash/1

Total: 3 processes, 3 lwps, load averages: 0,00, 0,00, 0,00
```

Наблюдение за процессами

ptree - просмотр дерева процессов

ptree [pid | user]

Пример

```
$ ps
   PID TTY          TIME CMD
  2226 pts/1    0:00 bash
  2255 pts/1    0:00 ps
```

```
$ ptree 2226
478   /usr/lib/ssh/sshd
  2219  /usr/lib/ssh/sshd
    2220  /usr/lib/ssh/sshd
      2226  -bash
        2256  ptree 2226
```

```
$ ptree ovs
478   /usr/lib/ssh/sshd
  2219  /usr/lib/ssh/sshd
    2220  /usr/lib/ssh/sshd
      2226  -bash
        2277  ptree ovs
```

Способы управления процессами

- Задание среды выполнения процесса
- Управление посредством сигналов
- Запуск процесса (задания) в фоновом режиме
- Изменение приоритета фонового процесса
- Планирование работ

Сигналы

Сигнал - некоторая переменная, передаваемая процессу, когда наступает определенное событие

Сигнал	**номер**	**Событие**
SIGHUP	1	Разрыв связи с терминалом
SIGINT	2	<Ctrl>+<c> - прерывание
SIGQUIT	3	<Ctrl>+<\> - прекращение работы (генерирует файл core)
SIGKILL	9	Уничтожение процесса
SIGTERM	15	Прекращение работы программы
SIGSTOP	23	Приостановка выполнения процесса
SIGCONT	24	Возобновление выполнения приостановленного процесса.

Команда kill

Синтаксис:

kill [-s *имя_сигнала*] PID [PID…] Посылает сигнал указанному процессу.

•Прекращает (kill -9 …) выполнение любой команды, включая запущенные с префикс-командой nohup.

•Команда kill не может использоваться по отношению к процессам других пользователей (исключение - суперпользователь).

•По умолчанию процессу посылается сигнал TERM. Другие сигналы задаются с помощью опции -s. Наибольшую гарантию, что процесс будет действительно уничтожен, обеспечивает сигнал KILL.

•Для уничтожения процесс должен быть специфицирован указанием идентификатора процесса, либо номера задания (перед числом следует поставить символ %).

•Если процесс указан цифрой 0, kill уничтожает все процессы, связанные текущим **shell'ом, включая и его самого**.

Команда kill

Примеры

```
$ cat /usr/share/man/cat1/* > bigfile1 &
[1] 995
$ cat /usr/share/man/cat2/* > bigfile2 &
[2] 996
$ kill 995
[1] - Terminated  cat /usr/share/man/cat1/* > bigfile1
&
$ kill -s INT %2
[2] - Interrupt cat /usr/share/man/cat2/* > bigfile2 &
```

Команда kill

`$ kill -l` вывод всех сигналов, допустимых в соотв.реализации системы

```
1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
9) SIGKILL      10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGUSR1
17) SIGUSR2     18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG      22) SIGIO       23) SIGSTOP     24) SIGTSTP
25) SIGCONT     26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM
29) SIGPROF     30) SIGXCPU     31) SIGXFSZ     32) SIGWAITING
33) SIGLWP      34) SIGFREEZE   35) SIGTHAW     36) SIGCANCEL
37) SIGLOST     41) SIGRTMIN    42) SIGRTMIN+1  43) SIGRTMIN+2
44) SIGRTMIN+3  45) SIGRTMAX-3  46) SIGRTMAX-2  47) SIGRTMAX-1
48) SIGRTMAX
```

Команда trap

Синтаксис :

`trap 'команды' сигнал [ сигнал... ]`

выполняются *команды* по получении *сигнала*

• Команда trap может быть использована в shell-программах с целью перехвата сигнала до того, как он удалит(уничтожит) процесс и выполнения некоторого дополнительного или альтернативного действия.

• Команду trap следует применить для указания в явном виде сигналов и связанных с ними соответствующх действий, которые должны быть выполнены.

• Команды могут представлять собой список команд системы UNIX, отделяемых «;». Можно также создать shell-программу, которая выполнит требуемые действия и указать имя этой программы в качестве аргумента.

• Обычно команды trap помещаются в начале shell-программы. Сигналы о прерывании определются shell'ом , когда он читает команду trap. Прерывания активизируются по получении соответствующего сигнала.

Команда trap

Пример:

```
$ cat > mytrap
trap ‘echo bye; exit’ INT QUIT TERM
while true
do
      echo hello
done

$ mytrap
hello
hello
Hello
нажать <Ctrl>+<c>
bye   ---------------------------------- $
```

Команда trap

Виды действий с сигналами:

• **Перехват сигнала.** Вместо прекращения выполнения процесса будут выполнены указанные специальные команды.

• **Игнорирование сигнала.** Используется если shell-программа работает с важными файлами и важно сохранить их содержимое, несмотря на прерывание.

• **Сбрасывание сигнала.** После захвата или игнорирования команда trap умолчанию.Обычно такое действие связано с завершением процесса. Это не вызовет сброса прерываний по отношению к действиям, которые были заданы раньше, чем задано игнорирование прерываний.

Сигнал NULL посылается процессом при нормальном завершении. Он используется для того, чтобы задать действие, которое должна выполнить shell-программа после нормального завершения процесса.

Игнорирование сигналов

Синтаксис :

trap ‘ ’ *сигнал* [*сигнал*...]

Пример:

```
$ cat > mytrap2
trap ‘  ’ INT
while true
do
       echo hello
done

$ mytrap2
hello
hello------------------------------------ нажать
<Ctrl>+<c> (проигнорирован)
hello
hello ------------------------------------ нажать
<Ctrl>+<\>
$
```

Сигнал KILL не может быть проигнорирован!

Размещение команды trap в программе

Помещайте команду в начале программы для управления удалением временных файлов, если выполнение программы прекращается:

```
trap 'rm /tmp/tempfile;exit' INT QUIT TERM
```

Помещайте команду перед критическим участком программы для того, чтобы игнорировать возможные сигналы прерывания:

```
trap ' ' INT QUIT TERM
```

С помощью команды trap восстанавливайте действие системы, принятое по умолчанию, когда выполняется участок программы, требующий нормального завершения:

```
trap INT QUIT TERM
```

Фоновые процессы

Синтаксис:

командная строка **> cmd.out &**

•Планирует *командную строку* как задание для фонового режима.

•Возвращает приглашение на ввод как только задание запущено.

•Перенаправляет вывод фонового процесса с тем, чтобы его вывод не смешивался с интерактивными командами.

•Выход из системы приведет к завершению процессов, выполняемых в фоновом режиме. Пользователь получит предупреждение в тот момент, когда попытается выйти из системы,следующего содержания: “There are running jobs”, и поэтому он снова должен ввести команду exit или нажать <Ctrl>+<d>.

•Любая из выполняемых в фоновом режиме команд, требующая стандартного ввода, должна читать его из файла за счет перенаправления ввода.

Фоновые процессы

Пример:

```
$ grep user * > grep.out &
[1] 194
$ ps
PID             TTY             TIME            COMMAND
164             ttyp2           0:00            sh
194             ttyp2           0:00            grep
195             ttyp2           0:00            ps
```

•Shell сообщает номер задания и идентификатор процесса при переводе команды в фоновый режим в том случае, если указана опция monitor в команде set (set -o monitor).При указании этой опции по завершении фонового процесса будет выдано на экран соответствующее сообщение.

•Команду, выполняемую в фоновом режиме,нельзя прервать нажатием <Ctrl>+<c>. Такого рода команды могут быть прекращены с помощью команды kill.

Весь ввод и вывод фонового процесса должен быть безусловно перенаправлен

Запуск заданий в фоновом/оперативном режиме

jobs	Выводит список выполняемых заданий
<Ctrl>+<z>	Приостанавливает оперативный процесс (Символ приостановки определяется во время входа в систему в файле .profile с помощью команды stty susp ^z), выдает приглашение shell'a
fg [%*номер*]	Переводит задание с *номером* в оперативный режим
fg [%*строк*а]	Переводит любое задание, запущенное командной строкой, начинающееся со *строки* в оперативный режим
bg [%*номер*]	Переводит задание с *номером* в фоновый режим
bg [%*строка*]	Переводит любое задание, запущенное командной строкой, начинающееся со *строки* в фоновый режим

Команда nohup

Синтаксис:

nohup ***команда*** **&** — Делает *команду* защищенной от выхода пользователя из системы

Пример:

```
$ nohup cat * > bigfile &
[1] 972
$ <Ctrl>+<d> <Enter>
login: user3
Password:
...
$ ps -ef | grep cat
UID          PID          PPID          ...      COMMAND
user3        972          1             ...      cat * > bigfile
&
```

Если файл вывода не указан, команда nohup сама обеспечит перенаправление вывода в файл **`nohup.out`**

Управление приоритетом. Команда nice

Синтаксис:

`nice [-N]` ***`командная_строка`*** Запускает процесс на выполнение с пониженным приоритетом. N - число в диапазоне от 0 до 19.

Пример:

```
$ nice  -10  cc  myprog.c  -o  myprog &

$ nice  -5  sort *  >  sort.out &
```

•По умолчанию базовый приоритет равен 10. Большее значение числа соответствует меньшему приоритету.

•Приоритеты процессов можно определить с помощью команды ps -l.

PRI – действующий приоритет

NI -- базовое значение приоритета

Изменение приоритета выполняющегося процесса

Команда renice

renice -n значение -p PID

Команда priocntl (более гибкая)

priocntl -s -p <new_priority> -i pid <process_id>

Примеры

`priocntl -s -p -5 -i pid 8200`	для процесса 8200
`priocntl -s -p -5 -i ppid 8200`	для всех потомков 8200

Планирование работ. Демон `cron`

Демон ***cron*** планирует системные события в соответствии с командами, заданными в файле ***crontab (***каталог ***/var/spool/cron/crontabs)***

Формат файла ***crontab*** (разделитель – пробел)

минута час день_месяца месяц день_нед команда

Пример

```
0 12  5,20  *  *  echo  "Получите зарплату" > /dev/console
```

Файлы, управляющие доступом пользователей к файлам cron: ***cron.deny, cron.allow***

Планирование одиночных событий. Команда `at`

`at [-m ] время дата команда`

`-m` – сообщает по почте о выполнении

Пример

```
at  07:45am today who>/tmp/log
```

Файл, управляющий доступом к at – **at.deny**