**Работа в графическом и консольном режиме**

**Встроенная документация. Команды man, apropos**

На прошлом занятии мы уже касались поиска информации о нужной программе. Разберем более подробно возможности работы с документацией.

В Unix существует встроенная система документации, обратившись к которой, всегда можно найти название нужной утилиты, описание  параметра команды или пример ее использования. Например, если требуется  подробное описание команды cp, используем команду man (от MANual).

|  |
| --- |
| user@user-virtual-machine:~$ man cp |

Справка по командам обычно занимает несколько страниц, поэтому man использует для просмотра команду less. Для перехода к следующей странице используйте клавишу <PgDn>, обратно — <PgUp>. Если эти клавиши не срабатывают, используйте для перехода пробел и клавишу u соответственно. Для выхода из  man служит клавиша q. Есть еще пара полезных клавиш: g — переход в начало, G — в конец документа. Для поиска вперед используйте **/чтоискать**, для обратного поиска — **?что искать**, повторить поиск в том же направлении — n.

Теперь пара слов о стандартных разделах справочной страницы.

NAME — краткое описание команды.

Раздел SYNOPSYS описывает различные варианты синтаксиса. В этом разделе могут использоваться специальные обозначения. Например, если опции или параметры заключены в квадратные скобки, это означает, что они не обязательны для использования. Троеточие после параметра говорит, что он может повторяться многократно, как в случае, когда несколько файлов копируются в один каталог назначения.

В разделе DESCRIPTION описываются опции (иногда их называют ключами) программы. Бывает, что в описании через запятую перечислены 2 опции, например -v, --verbose. В таких случаях мы имеем дело с разными стилями одной опции. -v — классический вариант,  описанный в стандарте POSIX. --verbose — опция в стиле GNU, новомодный вариант. Можно использовать то, что больше нравится.

Еще один полезный раздел  обычно находится в конце справочной странице, он называется SEE ALSO.  Как правило, здесь приводят ссылки на другие утилиты с похожим функционалом.

Кроме man есть еще несколько полезных справочных утилит. Если мы не знаем точного имени утилиты, можно попробовать найти ее с помощью apropos, которая выполнит поиск по ключевым словам в описании утилит. Например найдем как называется утилита, которая меняет пароль пользователя:

|  |
| --- |
| user@user-virtual-machine:~$ apropos password  apg (1)          - generates several random passwords  chage (1)        - change user password expiry information  chgpasswd (8)    - update group passwords in batch mode  chpasswd (8)     - update passwords in batch mode  cpgr (8)         - copy with locking the given file to the password or g.. cppw (8)         - copy with locking the given file to the password or g.. cracklib-check (8)   - Check passwords using libcrack2  create-cracklib-dict (8) - Check passwords using libcrack2  crypt (3)        - password and data encryption [...] |

Мы получили нужную информацию, но слишком много вариантов — на 3 страницы текста.

Попробуем уточнить поиск, выбрав все строчки, где встречается слово change. Используем утилиту grep, которая ищет по заданной строке в выводе от предыдущей программы. Делается это так:

|  |
| --- |
| user@user-virtual-machine:~$ apropos password | grep change chage (1)        - change user password expiry information lppasswd (1)     - add, change, or delete digest passwords.  passwd (1)       - change user password smbpasswd (8)    - change a user's SMB password |

Вот она, passwd  — то, что нужно.  Обратите внимание на число в скобках после имени команды. Это глава или раздел справочного руководства man, в которых может встречаться описание утилиты. Разделов много, приведем самые полезные:

* 1 — команды пользователя,
* 2 — системные вызовы ядра (используется программистами),
* 5 — форматы файлов,
* 8 — команды администрирования.

Эти номера можно использовать в командах man и apropos. Например, passwd — не только команда, но и имя системного файла. Если нам интересен формат этого файла, надо набрать:

|  |
| --- |
| man 5 passwd |

Для apropos полезна опция -s. Например, если мы ищем программы для работы с паролями, набираем:

|  |
| --- |
| user@user-virtual-machine:~$ apropos -s 1 passwd  gpasswd (1)      - управление /etc/group и /etc/gshadow  passwd (1)       - изменяет пароль пользователя  grub-mkpasswd-pbkdf2 (1) - generate hashed password for GRUB  lppasswd (1)     - add, change, or delete digest passwords. passwd (1ssl)    - compute password hashes  user@user-virtual-machine:~$ |

Описание passwd — по-русски. Иногда это может создать проблемы для поиска, поэтому есть еще полезная опция, -L, она задает язык, на котором будет справка. Чтобы все искать и читать по-английски, задаем  apropos -s 1 -L C passwd.

|  |
| --- |
| user@user-virtual-machine:~$ apropos -s 1 -L C passwd  gpasswd (1)          - administer /etc/group and /etc/gshadow  grub-mkpasswd-pbkdf2 (1) - generate hashed password for GRUB  lppasswd (1)     - add, change, or delete digest passwords. passwd (1)       - change user password  passwd (1ssl)    - compute password hashes  user@user-virtual-machine:~$ |

Еще одна полезная справочная программа — whatis. От apropos отличается тем, что ищет по именам команд:

|  |
| --- |
| user@user-virtual-machine:~$ whatis passwd  passwd (5)       - файл паролей  passwd (1)       - изменяет пароль пользователя  passwd (1ssl)    - compute password hashes |

Иногда в разделе справочной страницы SEE ALSO встречается ссылка на более полное описание утилиты в системе info и проводится команда для получения полной справки. Пример для cp:

|  |
| --- |
| info coreutils 'cp invocation' |

Info — гипертекстовая система документации вроде текстового браузера, а команда в man содержит полную ссылку на информацию о вызове команды в info. Если запустить info без параметров, мы выйдем на главную страницу документации. Перемещение по страницам и поиск  как в man. Строчки, которые начинаются со звездочек, — гиперссылки, по которым можно пойти дальше. Чтобы вернуться из ссылки на уровень выше, используйте клавишу u как Back в браузере. Выход — по клавише q.

**Процессы, потоки, потоки ядра**

Современные операционные системы состоят из как минимум двух сильно отличающихся по своим возможностям частей — ядро, работающее в пространстве ядра, и на уровне режима работы процессора взаимодействующее с оборудованием и ресурсами, и пользовательское пространство, работающее в уровне ядра, и на уровне режима работы процессора  ограниченное в доступе к ресурсам. Попытки обратиться к не разрешенным ресурсам приводят к вызову процессором исключения и передачи управления ядру операционной системы.

Так как Linux — система многозадачная, существует три возможности реализации многозадачности, одна — для ядра, и два — для пользовательского пространства.

Для ядра это потоки (иногда говорят процессы) ядра.

Для пользовательского пространства это процесс и потоки. В Linux потоки также являются особого рода процессами (так они тоже имеют собственные PID), но при этом они делят одну и ту же память процесса, которому они принадлежат (первый созданный процесс, фактически первый поток процесса). При создании копии  процесса (fork) происходит копирование его ресурсов, так что это более безопасно, но менее эффективно.

Каждый процесс, кроме упомянутых UID и GID, также имеет идентификаторы PID (process ID) и PPID (parent process ID).

Перечень процессов можно посмотреть с помощью команды:

|  |
| --- |
| $ ps aux |

ps отображает процессы пользовательского пространства и потоки ядра (отображаются в квадратных скобках). Потоки не отображаются Чтобы посмотреть потоки, используйте ключ -T, если хотите посмотреть потоки конкретного процесса, используйте ключ -p

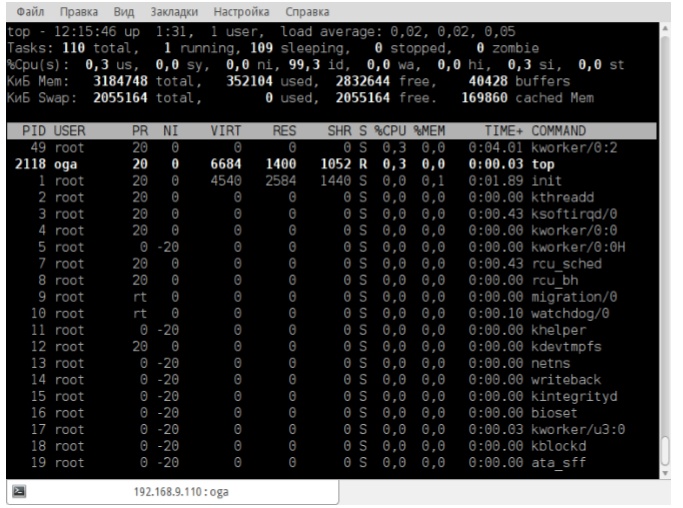
|  |
| --- |
| $ ps -T -p 4242 |

Чтобы посмотреть историю запущенных процессов, можно воспользоваться командой:

|  |
| --- |
| $ pstree |

Существуют и другие способы мониторинга процессов: top  и htop — обязательно ознакомьтесь с ними!

Консольная утилита top служит для динамического просмотра списка процессов. Утилита каждые несколько секунд выдает топ-20 процессов, по умолчанию отсортированный по потреблению процессорного времени. Можно сортировать список процессов по объёму занимаемой памяти, утилизации процессора, пользователям или номерам процессов.



top имеет горячие клавиши:

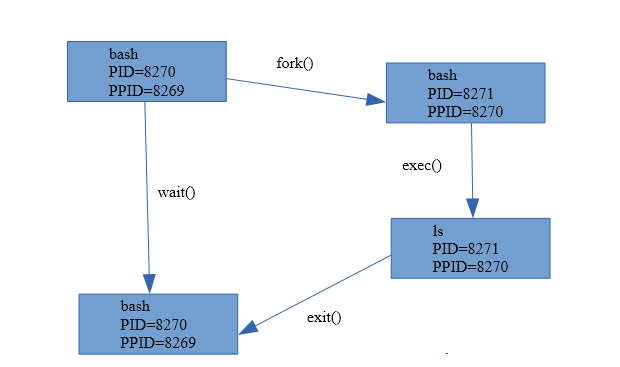
* h — вызов справки.  
  k — ввести PID для команды kill.
* r — указать PID для изменения параметра nice, влияющий на приоритет процесса ( -20 макс. приоритет, 19  мин. приоритет, 0 — по умолчанию). Обычный пользователь может только уменьшать приоритет своих процессов, назначая им nice от 0 до 19.
* q — завершить top.

Существует более продвинутая консольная утилита для мониторинга и управления процессами — htop. Она не установлена по умолчанию, но доступна из репозитария Ubuntu. Перечислим её возможности в сравнении с top:

* позволяет выполнять горизонтальный и вертикальный скроллинг, что позволяет видеть все процессы и параметры командной строки;
* htop запускается быстрее top, который предварительно собирает данные перед тем как показать список процессов;
* в htop не надо вводить PID для kill как в top, все через скроллинг и функциональные клавиши;
* также нет необходимости вводить PID и значение nice;
* в htop поддерживается мышь;
* top идет в системе по умолчанию и более распространен в мире UNIX.

## **Порядок запуска процесса**

Рассмотрим порядок запуска процесса. Например, работая в оболочке, мы запустили команду ls.



На первом этапе оболочка выполняет системный вызов fork(), в результате которого происходит ее клонирование: создается полная копия процесса оболочки, включая копию адресного пространства и контекста. Между клонами есть отличие: в процессе-родителе fork() возвращает PID потомка, а в потомке код возврата — 0. Каждый из клонов начинает выполнение с момента вызова fork, но уже самостоятельно и с учетом вышесказанного. Итак, потомок, выяснив, что он потомок (так как код возврата 0) с помощью системного вызова exec загружает в свое адресное пространство исполняемый код запускаемой программы, в нашем случае ls, и далее ее выполняет.

В это же время родительский процесс ожидает завершение дочернего процесса с помощью системного вызова wait(). Если мы запустили команду с & в конце, ожидания не будет.

Дочерний процесс завершается функцией exit(). В случае успешного завершения, exit возвращает 0, если произошла ошибка — код ошибки, отличный от нуля. После этого ядро освобождает ресурсы, которые занимал потомок, и передает код его завершения родителю в качестве кода возврата для wait().

Иногда бывает так, что процесс-родитель прекращает существование, не успев дождаться завершения wait(), то есть раньше завершения потомка. Процесс, у которого процесс-родитель завершился раньше) называют процессом-сиротой. Процессы-сироты усыновляет процесс init, который теперь считается для них родителем.

## **Сигналы процессам**

Допустим, программа зависла. Нажимаем Alt-F2 (или Ctrl-Alt-F2, если работали в X-сервер), смотрим id процесса.

|  |
| --- |
| $ ps ax | grep hello |

Процессу можно направить сигнал.

Например, 9 сигнал – принудительно снять.

|  |
| --- |
| $ kill -9 4242 |

В Linux существуют следующие сигналы:

* 1 (SIGHUP) — информирует программу о потери связи с терминалом. Ситуация часто возникала в прошлом, в режиме терминального доступа. В настоящее время может применяться для двух целей:
  + Информирование дочернего процесса о завершении родительского. При завершении родительского процесса ядро направит сигнал 1 дочерним процессам.
    - Если приложение запущено в фоновом режиме, с помощью указания & в конце команды или fg, при завершении консоли получит сигнал SIGHUP и программа, исполняющаяся в фоновом режиме, если была запущена таким образом. Это напоминает исходное значение SIGHUP.

|  |
| --- |
| $ someprogram& |

* + - либо

|  |
| --- |
| $ someprogram  ^Z  $ fg |

* + - Этому можно воспрепятствовать с указанием nohup.

|  |
| --- |
| $ nohup someprogram& |

* + Очевидно, что по умолчанию сигнал приводит к завершению программы. Но демоны перехватывают данный сигнал и используют для перечитывания файлов конфигурации.
* 2 (SIGINT) формируется при нажатии Ctrl-C. Завершает программу, но может перехватываться или блокироваться (например, в программах, в которых Ctrl-C используются для операции «копирование» или в принципе нет необходимости такого варианта остановки).
* 8 (SIGFPE) — ошибка операции с целочисленной арифметикой (переполнение, деление на ноль). Сохраняется дамп памяти.
* 9 (SIGKILL) — безусловное завершение программы. Сигнал не может быть перехвачен программой, потому позволяет её остановить в любом случае (но не позволит снять процесс-зомби).
* 11 (SIGSEGV) — формируется при попытке программы обратиться к не принадлежащей ей области памяти. Обычно выводится сообщение «Segmentation fault» и сохраняется дамп памяти. Как правило, такое случается в результате ошибок программиста при работе с указателями.
* 15 (SIGTERM) — вежливая просьба программе завершить работу. Программа может сохранить данные и т.д.
* Есть и другие варианты.

Для отправки сигнала из операционной системы можно использовать команду kill.

|  |
| --- |
| $ kill -15 4242 |