Введение в Linux, часть 2

Юрий Литвинов
y.litvinov@spbu.ru

1. Файловая система

Файловая система в Linux отличается от Windows довольно радикально, поэтому о ней следует поговорить отдельно и ещё до обсуждения консольных команд для работы с ней.

Файловая система — суть набор inode (индексных дескрипторов) — записей в файловой таблице, которые содержат метаинформацию о файлах (в частности, где на диске они физически лежат, права доступа). В первых файловых системах inode-ы физически лежали в одном массиве в начале диска, и номер inode-а был его индексом в массиве (может, для большинства файловых систем и сейчас так, не знаю). Важно то, что у каждого inode-а есть номер, его можно посмотреть командой `ls -i`, и каждый inode — это и есть файл, а всякие символьные имена файлов в директориях — это просто ссылки на inode. То есть можно понимать имя файла как указатель, а inode — структуру данных в памяти, на которую указатель ссылается. Тогда будет неудивительна концепция жёсткой ссылки — два файла в одной файловой системе, но в совершенно разных её местах, могут физически одним и тем же файлом. Windows тоже так умеет, но про это никто не знает и никто не пользуется, тогда как в Linux такие вещи распространены. Поскольку на один файл могут ссылаться несколько «имён файлов», то удаление файла чем-то напоминает сборку мусора — файл удаляется, только если на него не указывает больше никакая жёсткая ссылка. Жестких ссылок на директории не бывает, потому что они другое, и физически на диске не лежат.

Бывают ещё символьные ссылки — это прямой аналог ярлыков в Windows, файл, который ссылается на другой файл по пути до него (абсолютному или относительному). Символьные ссылки используются гораздо чаще, чем жёсткие ссылки, потому что могут действовать через границы файловых систем (с диска можно ссылаться на директорию на флешке), могут ссылаться на директории и переживают копирование (если путь в ссылке относительный). Для всех программ и утилит символьная ссылка — файл или директория, на которую она ссылается. Символьные ссылки, например, используются для разрешения альтернативных зависимостей, например, директория lib может быть символьной ссылкой на директорию lib32 или lib64, файл sh может быть ссылкой на оболочку командной строки bash, так что всем, кому не важно, какая именно оболочка нужна, могут просто sh вызывать. И потом одной командой пользователь может перекинуть sh на zsh, и тем сменить оболочку.

У каждого файла есть режим доступа, кодирующийся тремя группами по три бита. Три бита — это Read, Write, Execute, почему трёхбитовых прав у каждого файла три — потому что права выставляются отдельно для владельца, группы владельца и всех пользователей

системы. Кодируются права обычно трёхзначным числом в восьмеричной системе, например, 777 — разрешено всё всем, или 700 — владельцу можно всё, остальным ничего. Чаще обозначаются буквами rwx, например, в выводе команды `ls -l` может быть `rw-r--r--` (владелец может писать и читать, остальные только читать). Права можно поменять командой chmod (если у вас есть права на запись в файл), например, `chmod +x имя-файла.txt` сделает его доступным для исполнения. Без атрибута Execute запустить файл, даже если это скрипт или бинарник, нельзя — специально, чтобы пользователь не запустил случайно какой-то файл, который ему прислали во вложении к письму с темой «Вы выиграли миллион!». Права у директорий тоже есть, но несколько менее интуитивны — например, можно иметь права редактировать файлы в директории, если вы знаете имя файла, но одновременно с этим не иметь прав на просмотр списка файлов в этой же директории.

Ещё совершенно неожиданным для пользователей Windows может оказаться тот факт, что далеко не каждый файл в Linux лежит на диске. Вообще философия Linux состоит в том, что всё, что поддерживает потоковый ввод-вывод — файл. Это позволяет использовать стандартные утилиты для работы с файлами для работы с такими штуками, а стандартных утилит много и они умеют очень многое. Поэтому принято даже интерфейсы устройств, которые под Windows приходится программировать используя странные вызовы ядра, в Linux выводить в файловую систему, куда можно записать или прочитать строки определённого вида.

Простой пример — файл `/dev/null`, который содержит сразу символ EOF, и куда можно записать что угодно, и оно навеки исчезнет (этот факт часто используется, чтобы перенаправлять вывод гадящей на консоль программы). Или `/dev/random` — писать туда можно с тем же эффектом, что и в `/dev/null`, зато читать оттуда можно бесконечную последовательность «хороших» (то есть достаточно случайных, генерируемых посредством непрерывно накапливаемой системой энтропии, типа движений мышки) случайных чисел. Есть даже директория `/proc/`, где для каждого запущенного в системе процесса есть директория, где лежат файлы (ни в коем разе не на диске), из которых можно читать некоторую информацию о процессе или управлять им. Так же в `/proc/` есть и файлы с общей информацией о системе, например, `/proc/cpuinfo`, их можно открыть блокнотиком (лучше просто напечатать командой cat). Это настолько хорошо работает, что если вы делаете под Linux что-то такое, что долгое время запущено и может управляться извне или выдавать какую-то информацию/статус, это принято выводить как файл в файловой системе (например, датчики в роботе).

1.1. Filesystem Hierarchy Standard

Хорошая новость в том, что файловая система Linux и в целом всех Unix-подобных систем в целом стандартизована и многие директории там имеют стандартное назначение, узнав про которое можно будет довольно легко ориентироваться. Стандарт называется FHS (Filesystem Hierarchy Standard), он уже лет 10 не обновлялся, поэтому не все дистрибутивы ему строго следуют, но в целом всё так.

Стандарт классифицирует файлы по двум независимым признакам — изменяемые и статичные файлы, и разделяемые и неразделяемые. Статичные файлы — которые не меняются, если их не трогать (то есть при обновлении программы или изменении настроек — сколько угодно, но для этого требуется некая воля администратора). Разделяемые файлы — это те, которые можно выставить в сеть и пользоваться ими с других машин (например,

пользовательские данные). Разные сорта файлов могут иметь разные настройки бэкапов, разные режимы доступа (например, статически файлы для обычного пользователя могут быть подмонтированы только для чтения), поэтому должны лежать в разных директориях.

Корневая файловая система, на которой находится директория «/» и образ ядра системы, содержит только самое необходимое для загрузки и восстановления системы в случае, если что-то сломалось настолько, что другие файловые системы уже не монтируются (например, утилиту mount, которая собственно за монтирование и отвечает). Внутрь директории «/», в её разные поддиректории, монтируются другие файловые системы. Сделано так по историческим причинам, когда диски были маленькими, ну и чтобы было удобно работать на встроенных устройствах, где корневая система могла жить на отдельном очень маленьком, дешёвом и неэнергоёмком носителе, остальное монтировалось по необходимости. Сейчас, однако, диски большие и дешёвые, поэтому нередко вообще вся система ставится на один раздел диска, и никто с монтированием и минимизацией стартового набора корневой файловой системы не заморачивается. Поэтому современные дистрибутивы используют концепцию «joined root», которая прямо нарушает стандарт FHS — вовсе не отделять критичные для работы системы утилиты от всех остальных программ.

Всё не критичное для загрузки и аварийного администрирования системы лежит в директории `/usr`. Она по стандарту содержит в себе статичные файлы, поэтому может быть подключена в режиме «только для чтения», никто не мешает иметь её на съёмном диске или вообще монтировать по сети.

Вообще, все поддиректории корня (директории «/») фиксированы стандартом и авторам дистрибутивов и тем более разработчикам прикладных программ создавать дополнительные директории в «/» запрещено. В Windows последних версий в целом тоже более-менее так, там запись в корень диска C требует административных прав, но это не останавливает инсталляторы (Python по умолчанию на полном серьёзе создаёт в корне по папке для *каждой* своей версии), да и сама система имеет в корне файл подкачки и ещё какое-то барахло. В Linux подкачка живёт на отдельном разделе, который даже никуда не монтируется, так что пользователь его и не увидит (справедливости ради, оперативная память сейчас тоже большая и дешёвая, так что подкачку часто отключают вовсе).

Стандартные директории в FHS таковы:

- /bin — нужные при запуске и поломке системы программы, которыми может пользоваться не только системный администратор, но и простые смертные, физически лежит в корневой файловой системе;
- /boot — файлы загрузчика: конфигурация, часто образ ядра (образ на небольших системах может лежать в корне);
- /dev — файлы устройств (это те самые «псевдофайлы», которые не лежат на диске, и нужны для взаимодействия с устройствами);
- /etc — локальная (то есть не разделяемая между машинами) конфигурация, например, тут находятся зарегистрированные в системе пользователи со своими паролями (в зашифрованном виде, конечно), конфигурация оконной системы;
- /lib — библиотеки и модули ядра, разделяемые остальными программами и необходимые при старте системы;

- /media — сюда монтируются внешние носители типа флешек;
- /mnt — директория для временного монтирования, изначально пустая, но командой mount её содержимое заменяется на другую файловую систему; если хочется монтировать что-то постоянно, то уже в media (и настроив автоматическое монтирование в `/etc/fstab`);
- /opt — директория для дополнительных пакетов (обычно проприетарных);
- /sbin — нужные при запуске и поломке системы программы, которыми предполагается что будет пользоваться системный администратор (и имеющие соответствующие права), «system binaries»;
- /tmp — временные файлы, создаваемые приложениями и системой в ходе работы, чистятся при каждой загрузке;
- /usr — вторичная иерархия (примерно та же иерархия, но из некритичных файлов);
  - сейчас популярен «joined root», когда `/bin` — это символическая ссылка на `/usr/bin`, а `/sbin` — на `/usr/sbin`;
- /var — изменяемые данные, которые появляются по ходу работы приложений и системы — прежде всего, всякие логи;
- /home — директория для домашних директорий пользователей (то есть директорий, где хранятся всякие специфичные для пользователей файлы, типа локальной конфигурации или сохранений игр);
- /root — как /home, но для администратора, а поскольку на аккаунте администратора не должно быть сохранений игр, обычно пустая или не создаётся вовсе;
- /lost+found — результат деятельности fsck, inode-ы, которые есть на диске, но на которые не указывают никакие имена — так может случиться при повреждении файловой системы, например, при выключении электричества прямо в процессе записи файла: fsck при следующей загрузки такие inode-ы найдёт и сложит в `/lost+found`.

2. Продвинутые возможности консоли

Тут поговорим о том, что ещё может консоль, помимо исполнения команд.

Во-первых, есть вайлдкарты (wildcards), они же глобы (globs) — шаблоны имён файлов или директорий. В примере выше `rm -rf *.txt` вот это `*.txt` — это вайлдкарт, сопоставляющийся каждому файлу с расширением .txt в текущей директории. Вайлдкарты раскрываются *до* вызова команды, если хоть один файл подходит под вайлдкарт, так что если в директории есть два файла 1.txt и 2.txt, то команда выше будет исполнена как `rm -rf 1.txt 2.txt`. Если ни одного файла .txt нет, параметр передаётся в команду как есть, то есть rm увидит `*.txt` на входе. Вот ещё пара примеров синтаксиса вайлдкартов: `file[1-8]` (файл, заканчивающийся на цифру от 1 до 8), `file*` (файл, имя которого начинается с file, а дальше неважно). Символ `?` соответствует одному любому символу (`*` — сколько угодно любых символов), символ `!` внутри квадратных скобок означает «не» — например, `file[!a-z]`

соответствует всем файлам, имя которых начинается с `file`, но дальше не строчная буква. И да, это те самые шаблоны, которые надо писать в .gitignore, чтобы git не пытался коммитить всякий мусор в репозиторий — git создавался как набор консольных скриптов под Linux, в конце концов.

Во-вторых, есть переменные окружения и квотирование. Переменные окружения в Linux работают так же, как в Windows, разве что оператор подстановки — это `$`, а не `% %`. И пути в `PATH` разделяются двоеточиями, а не точками с запятой. Переменная выставляется обычной командой `=`, например, `PATH=$PATH:/home/user/.dotnet`, но это действует на одну строчку. Чтобы новую переменную видели все запускающиеся из текущей консоли программы, надо воспользоваться командой `export`, например, `export PATH=$PATH:/home/user/.dotnet`, чтобы видели все консоли, вписать этот export в файл .bashrc в домашней директории (и перезапустить консоли), чтобы все пользователи — в /etc/profile.

Квотирование — это заключение строки в кавычки. Сильное квотирование — это одинарные кавычки, внутри них текст интерпретируется шеллом и передаётся в команду как есть, например, `echo '$PATH'` выведет на экран `$PATH`. Внутри одинарных скобок вайлдкарты не раскрываются и подстановка переменных окружения не выполняется. Слабое квотирование — это двойные кавычки, внутри них выполняется подстановка, но не раскрываются вайлдкарты. Так что `echo "$PATH"` выведет таки текущее содержимое `PATH`, но `echo "*.tex"` выведет `*.tex`, даже если в директории есть теховские файлы. Кстати, `echo *` в силу правил раскрытия вайлдкартов работает как `ls` (и да, не выведет скрытые файлы).

Вот ещё четыре полезные команды, которые помогают найти файл или нужную команду.

- which — найти программу в путях из PATH (так можно узнать, что именно запустится, когда мы наберём имя команды).
- whereis — вывести все места, где находится нужная программа (например, если .NET SDK после ряда неудачных попыток его правильно поставить оказался в трёх разных директориях, она их покажет).
- find — найти файл, по имени, содержимому и много чему ещё: например, `find /usr/share/doc -name README` найдёт все файлы с именем `README` в `/usr/share/doc`. Имеет ключ `-exec`, где можно написать команду, которую надо выполнить с каждым файлом (то есть, например, `find . -name "*.csproj" -exec "dotnet build {} \;"` сами догадайтесь, что сделает). Вообще, find имеет много полезных ключей, возможность комбинировать условия поиска и чего только не делать.
- locate — ищет по именам, но не только в именах файлов, но и в путях. Например, `locate /usr/share/*/README`.

2.1. Управление процессами

Консоль вообще нужна прежде всего для управления программами, а запущенная программа в операционной системе называется «процесс». Первое, что надо уметь делать с процессами — это их останавливать, это делается, как и в Windows, сочетанием клавиш Ctrl-C. Например, `cat /dev/random` может поставить вас в неловкую ситуацию, выйти из которой можно, как раз нажав Ctrl-C. На самом деле, Ctrl-C посылает сигнал процессу

— команду в его очередь сообщений (как и в Windows), конкретно Ctrl-C шлёт процессу SIGINT. Поэтому, кстати, скопировать текст из консоли надо не Ctrl-C, а Shift-Ctrl-C.

Ещё бывает Ctrl-Z, он шлёт текущему процессу сигнал SIGTSTP — Temporary Stop, поставить процесс на паузу. Его можно будет перезапустить командой `fg` (от «foreground») или отправить работать в фоне командой `bg` (от «background»). В фоне процесс будет делать своё дело, выводить на консоль, если вывод не перенаправлен, но в консоль в это время можно вводить новые команды. Можно прямо при запуске указать процессу запускаться в фоне, оператором `&`, например, `okular 06-intro-to-linux-text.pdf &`.

Есть отдельные команды kill и killall, которые, несмотря на агрессивное название, посылают указанному процессу (процессам) сигнал. Например, `killall -2 okular` отправит сигнал SIGINT всем процессам с именем okular, так, будто вы нажали Ctrl-C каждому (даже если они были запущены не из консоли, так что жать Ctrl-C негде). SIGINT, однако, процесс может проигнорировать (почему, вы думаете, bash не закрывается, если вы случайно Ctrl-C нажали). Более надёжно послать процессу сигнал SIGKILL (`killall -9`), его игнорировать нельзя (это скорее сигнал операционной системе, что всё). kill отличается от killall тем, что принимает идентификатор конкретного процесса (pid, от process identifier). Полезно, когда вы хотите убить не все экземпляры программы N, а только один, который вышел из-под контроля. Напомним, что kill/killall позволяют слать любые сигналы, не только SIGKILL.

Чтобы узнать pid процесса, можно посмотреть список процессов, запущенных в ОС. Может, вы помните из текста выше, что можно глянуть содержимое директории `/proc` и вручную разобраться, кто там кто, но гораздо проще использовать команду `ps`. Без параметров она показывает список процессов текущего пользователя, запущенных из текущего шелла, что обычно не очень интересно. `ps -e` покажет все процессы в системе, но их может быть многовато, чтобы это было полезно. Есть команда `top`, которая показывает самые ресурсоёмкие процессы в системе, обновляя данные по ним в реальном времени. Выход — клавишей `q`.

Ещё одна очень важная возможность консоли — перенаправление ввода-вывода. Оператор перенаправления `>` перенаправляет вывод в файл, например, `echo "Hello, world!" > out.txt` создаст файл `out.txt` со строкой `Hello, world!` внутри. Есть в каком-то смысле парная команда `>>`, она не перезаписывает файл целиком, а добавляет вывод в конец. Однако самая интересная команда перенаправления в Linux — это пайп: `|`. Она перенаправляет стандартный вывод одного процесса на стандартный вход другого, так что, например, `echo lol | wc` выведет результат стандартной программы wc (word count) на строке «lol». Из пайпов можно собирать целые конвейеры обработки, например, `cat file.txt | sort | uniq | wc -l` возьмёт содержимое файла file.txt, отсортирует строки в нём в лексикографическом порядке, удалит повторяющиеся и посчитает, сколько строк осталось. Это, собственно, и есть Linux way — иметь в системе кучу мелких программок и удобный шелл с пайпами, который позволяет собирать сколь угодно сложные сценарии обработки данных прямо в консоли. Вообще, любой нормальный шелл в Linux — это полноценный язык программирования (хоть и такой себе в плане современной языковой науки, но, например, рисование .png-шек псевдографикой чисто на шелле — это домашка первого курса некоторых вузов). Мы тут не будем углубляться в детали, но если кому интересно, есть куча документации, и это всё равно будет крайне полезно, если планируете свою карьеру в любом embedded или DevOps.

2.2. Команды для работы с текстом

Есть набор команд для чтения и обработки текстовых файлов, которые крайне полезны, поскольку в Linux все настройки — это текстовые файлы, логи системы и программ — текстовые файлы, даже устройства и метрики работы системы — текстовые файлы. Некоторые команды мы уже видели выше:

- sort — отсортировать файл в том или ином порядке, умеет сортировать лексикографически, по датам и версиям, с разными другими настройками;
- uniq — работает только для отсортированных файлов (или входного потока, перенаправленного из sort), убирает повторения;
- wc — считает количество символов, слов и строк.
- head, tail — как cat, только вывести первые n строк из файла (по умолчанию 10). `tail -f` выводит последние n строк файла непрерывно, так можно наблюдать за обновляющимися логами.
- more, less — интерактивные читалки, less более новая и умеет скроллить файл вперёд-назад (поддерживая Page Up/Page Down). Выход по клавише q, как обычно. Есть ещё most, но ни разу не пользовался.
- sed, awk — потоковые текстовые редакторы, как бы странно это ни звучало. Умеют найти и заменить шаблон (обычно регулярное выражение), при этом awk считается более продвинутым, но sed более удобным для командной строки и скриптов. Не то чтобы часто приходится пользоваться, но sed позволяет прямо из коробки в одну строчку писать скрипты, которые на Windows потребовали бы Python или чего-то такого. Считается приличным знать и уметь любому опытному пользователю Linux.
- vim — полноценный интерактивный текстовый редактор в консоли. Очень старый, так что совершенно не уважает привычный интерфейс пользовательских редакторов, но достаточно удобный, чтобы многие его использовали как основной редактор и даже полноценную IDE, поскольку он очень хорошо интегрирован в консоль и имеет развитую систему плагинов. Обязательно хотя бы базовое редактирование на нём освоить, потому что если вдруг графическая оболочка с любимым VS Code или чем-то таким недоступна, то vim и его аналоги могут быть единственным способом редактировать конфиги, которые позволят эту самую графическую оболочку починить (ну или по ssh удалённо работать на сервере, где графической подсистемы нет вообще, потому что незачем). У него два режима — в котором он пищит в в котором всё портит, переключение — клавишей «i» и «Esc». Выход — из режима, когда он пищит, клавишами «:», затем «wq» (или «!q», если без сохранения).

2.3. Ещё полезные штуки консоли

Вот список случайных фактов о линуксовой консоли, которые могут понадобиться:

- Табуляция — ваш лучший друг. Она автодополняет текущую команду, причём контекстно. Например, если вы пытаетесь открыть .pdf-файл командой okular, пишете

«ok», жмёте Tab, оно дополняется до okular, жмёте Tab ещё раз — оно выводит список всего, что okular может открыть (и поддиректорий), начинаете вводить нужную, жмёте Tab, оно дописывает. Экономит кучу времени.

- .bashrc — скрипт, который исполняется каждый раз, когда вы запускаете bash вручную. .bash_profile — скрипт, запускающийся при логине в систему с помощью bash. Оба лежат в домашней директории пользователя, оба скрытые (что понятно, потому что они начинаются с точки). Туда можно писать что угодно на языке шелла, но особенно полезна команда `alias`, которая позволяет задать синоним для произвольной строки и использовать её как команду. Например, `alias gcc="gcc -lm -g -Wall -Wextra"` автор использует для компилирования домашки по Си, вызывая просто gcc и получая сразу линковку с математической библиотекой, отладочную информацию (на случай, если надо будет потыкать программу консольным отладчиком gdb) и все разумные предупреждения.

- Кстати, вайлдкарт ~ раскрывается в домашнюю директорию текущего пользователя, так что `cat ~\.bashrc` позволит вам посмотреть на свой .bashrc из любого места файловой системы.

- Midnight Commander — консольный файловый менеджер наподобие far под Windows (но несколько менее функциональный и удобный).

- reverse-i-search — поиск по истории вводимых в консоли команд. Так-то историю команд можно скроллить стрелками «вверх»/«вниз», что очень удобно, чтобы не вводить похожие команды дважды. Но если нужная команда вводилась когда-то давно, можно нажать Ctrl-R и начать её вводить, консоль её найдёт и по нажатию Esc или стрелки «вправо» оставит в командной строке для редактирования. Ctrl-S прокручивает историю команд вперёд, но есть нюанс (https://stackoverflow.com/questions/17760474/reverse-intelligent-search-reverse-i-search-how-to-get-previous-result?rq=1 (дата обращения: 26.02.2024)).

- Ctrl-W удаляет слово под курсором (влево до пробела), Ctrl-U — до начала строки, Ctrl-K наоборот, до конца строки. Ctrl-стрелки «влево»/«вправо» позволяют прыгать по словам (в отличие от Ctrl-W считают разделителем и «-» и т.п.). Ctrl-A и Ctrl-E — ставят курсор в начало или конец строки.

- Любой текст в консоли можно выделить мышью, потом просто кликнуть средней кнопкой мыши в нужном месте — и он вставится (не только в консоли это работает). И это делается *не* через буфер обмена, так что информация в буфере обмена при этом не стирается. Сравните это с адом, через который надо пройти, чтобы в Windows что-то копировать из консоли, особенно если это не PowerShell.

- Для ввода специальных символов (типа тире) используется клавиша Compose. Она по умолчанию может быть не настроена вовсе, но можно поковыряться в настройках своей оболочки рабочего стола и повесить её на какую-нибудь не очень нужную клавишу, типа правого Ctrl (хотя тут такое, он удобно расположен близко к стрелкам, а Ctrl-стрелки часто нужны). Дальше жмём Compose, жмём трижды на минус — получаем тире.

- Ctrl-Alt-F1, Ctrl-Alt-F2 и т.д. позволяют переключаться (прямо на рабочей системе) между графической подсистемой и чисто текстовым терминалом. Нужно, ~~если~~ когда графическая подсистема помрёт из-за ошибки в конфигурировании и надо будет её оживлять из консоли. Текстовые терминалы работают всегда, в чём их важнейшее преимущество. Ctrl-Alt-Backspace (дважды) позволяет перезапустить графический сервер и все процессы, им порождённые, без перезапуска всей ОС.

- Всегда имеет смысл ставить проприетарные драйвера для видеокарты, и они почти никогда не ставятся по умолчанию. Как именно ставить, может зависеть от дистрибутива — в паре российских дистрибутивов, которыми автор пользовался, для этого была целая специальная программа, которую можно было запустить (из консоли), и она сама ставила нужные драйвера nvidia. Linux по политическим мотивам не включает по умолчанию проприетарное ПО, потому что видение сообщества в том, что ПО должно быть свободным и поставляться только вместе с исходниками, а жадные корпорации, которые не хотят раскрывать свои исходники, могут идти под Windows. Но пока коммунизм не наступил, приходится ставить проприетарные драйвера видеокарты, принтера (скорее всего, хотя в Linux очень крутая система управления печатью cups, которая может и сама справиться, но если вдруг нужен ещё и сканер...), и даже кодек для MP3-файлов, если они нужны.

3. Где брать документацию

В Linux, в отличие от Windows, к документации по традиции относятся очень серьёзно. Документация стандартизована, включена как составная часть во многие пакеты (можно сказать, в почти все, если автор не выделил сознательно пакет с документацией отдельно в силу огромного её объёма), хорошо поддерживается, есть системные утилиты для поиска и просмотра, которые, конечно, прекрасно работают в консоли.

Первая такая утилита — `man` (от слова «manual»). Это та самая man, к которой отсылает знаменитое восклицание «RTFM» — краткая документация, доступная для всех стандартных утилит Linux, многих файлов конфигурации, форматов и т.п. Состоит из девяти тематических разделов, где первый — это пользовательские программы, второй — системные вызовы и т.д. Так что одна и та же штука может встречаться в нескольких разделах сразу, если доступна, например, и как консольная команда, и как функция API системы. И поэтому при ссылке на man-документацию обязательно указывают раздел, например, `man top` выведет `top(1)` — пользовательская программа. man открывает документацию в системном *пейджере*, обычно это less (так что выход по q, как обычно).

Команда whatis — типа оглавления man-документации, выводит однострочную справку по каждой man-странице, найденной по указанной строке. Например, `whatis top` выдаст `top (1)  - display Linux processes.`

Команда apropos позволяет искать по man-документации, не только в заголовках, но и по телу справки. Например, `apropos top` выдаст кучу всего, потому что строка запроса коротковата.

Второй формат справки — info-страницы и команда info для их просмотра. Формат info считается более прогрессивным, в частности, поддерживает гиперссылки (в консоли прямо). Одно может вроде как генериться из другого, так что для основных системных утилит есть и info-страницы, можно смотреть их.

Ещё очень важный источник документации — документация вашего дистрибутива. Она как повезёт, бывает хорошей, бывает так себе, бывает просто набором бессвязных статей на вики и постов на форуме (напомним, что в Linux-сообществе никто никому ничего не должен). Тем не менее, очень многие вопросы специфичны для дистрибутива, и никакие man-ы тут не помогут.

Ещё есть списки рассылки по электронной почте, настоящие линуксоиды по традиции используют их, но как-то прошлый век.