**Оглавление**

[История 2](#_Toc210315932)

[Базовые команды 5](#_Toc210315933)

[Command Line Interface CLI 8](#_Toc210315934)

[Файловая структура 10](#_Toc210315935)

[Работа с файлами 12](#_Toc210315936)

[Чтение файлов 14](#_Toc210315937)

[Команды 14](#_Toc210315938)

[Пейджеры 15](#_Toc210315939)

[Grep 16](#_Toc210315940)

[Редактирование файлов 17](#_Toc210315941)

[Nano 17](#_Toc210315942)

[Vim 17](#_Toc210315943)

[Потоки 19](#_Toc210315944)

[STDOUT 19](#_Toc210315945)

[STDIN 20](#_Toc210315946)

[STDERR 20](#_Toc210315947)

[Пайплайны 22](#_Toc210315948)

# История

В 1970-х годах появились большие **многопльзовательские** компьютеры – мейнфреймы.

В то же время на свет появилась Unix — операционная система, написанная на языке Си и предназначенная для работы на мейнфреймах. Ключевая ее особенность — это многопользовательский режим. Дело в том, что тогда компьютеры были редкостью, поэтому у каждого мейнфрейма было множество пользователей одновременно.

Unix позволяла подключаться к мейнфрейму через терминалы — специальные устройства, которые служили двум задачам:

* Ввод информации (клавиатура)
* Вывод информации (монитор)

При этом вся обработка выполнялась на отдельном сервере.

Unix оказалась очень удачной операционной системой, но была платной и **закрытой**. Другими словами, никто не мог посмотреть ее исходный код.

Со временем на основе Unix создавались новые системы, которые теряли обратную совместимость.

Поэтому был принят **стандарт** **POSIX** (Portable Operating System Interface **for UNIX**). Сейчас это не один стандарт, а целое семейство стандартов. Оно описывает работу ключевых компонентов операционной системы и способов взаимодействия с ними из программ, написанных на Си.

В 1991 году студент Линус Торвальдс решил написать собственную операционную систему, которая совместима со стандартом POSIX. Линус собрал целое сообщество разработчиков, которые довольно быстро создали рабочую операционную систему под названием **Linux** и выложили ее исходный код в открытый доступ.

Но сам **Linux** — это только ядро операционной системы, которое не получится использовать без сопутствующих программ.

Когда Линус начал свою разработку, уже существовал проект GNU, основанный Ричардом Столлманом. В рамках этого проекта уже были разработаны свободные программы, составляющие основной инструментарий разработчика программ на языке Си:

* Текстовый редактор Emacs
* Компилятор языка Си gcc
* Командная оболочка bash
* Библиотека важнейших функций для программ на Си libc

Linux сразу начал поставляться с утилитами.

**Linux не имеет своей графической оболочки.** Именно поэтому Linux может выглядеть очень по-разному, ведь на него можно поставить множество разных графических оболочек:

Сейчас Linux не используется в чистом виде. Вместо этого конечные пользователи имеют дело с дистрибутивами. Дистрибутив включает в себя:

* Сам Linux
* GNU-программы
* Утилиты для установки и настройки Linux
* Поддержку регулярного обновления самого ядра и его окружения

То есть **POSIX** – это стандарт совместимости с **UNIX-системами**.

**Linux** стремится быть совместимым с POSIX, чтобы поддерживать переносимость Unix-приложений.

Существует очень много дистрибутивов, но некоторые из них особо популярны. Самый популярный дистрибутив — это полностью бесплатная операционная система **Ubuntu**.

В отличие от Unix-подобных операционных систем, Windows изначально создавалась для однопользовательского режима и никогда не пыталась быть совместимой с POSIX. Ей удалось захватить рынок персональных компьютеров, но на серверах она почти не встречается, несмотря на существование Windows Server.

Windows редко ставят на серверы, потому что она платная, и совсем не совместима с POSIX. При этом у Windows есть командная строка cmd.exe, но ее функциональность отличается от командной строки Linux, в ней используются другие команды. Опыт работы с ней не сильно поможет при работе в Linux-окружении, с которым все равно придется столкнуться.

**MacOS** основана на **Unix** и считается даже **более POSIX-совместимой**, чем Linux. У нее прекрасная командная строка и инструментарий для работы.

# Терминал и командная строка

**Терминал (terminal)** — это программа или устройство, через которое вы доступ к командной строке.

Терминал запускает внутри себя оболчку (shell).

**Коамндная облочка (shell)** – интерпретирует команды (например, Bash, Zsh, Fish).

Командная оболочка – это полноценная среда программирования.

Командную оболочку нередко называют реплом от английского **REPL** (*Read*–*Eval*–*Print*–*Loop*). Это сокращение отражает способ взаимодействия командной оболочки с пользователем:

* Read — оболочка ждет ввода команды от пользователя
* Eval — исполняет введенную команду
* Print — выводит результат
* Loop — возвращаемся к первому пункту

Этот процесс называется интерактивной сессией: после загрузки командная оболочка ждет ввода команды, исполняет ее, выводит результат и снова ждет ввода команды.

Одна из распространенных оболочек **Bash** – *«Born again» shell* – «возрождённый» shell – усовершенствованная и модернизированная вариация командной оболочки Bourne shell.

Для взаимодействия с пользователем оболчкаа использует командную строку.

**Командная строка (command line)** – это интерфейс ввода команд – это текстовый способ взаимодействия с операционной системой: вы вводите команды (например, ls, cd, grep), система их выполняет и выводит результат.

**Вывод:**

Терминал – это программа, которая эмулирует физический терминал для ввода данных и вывода результатов на экран.

Внутри себя терминал заускает командную оболочку (shell), например, bash.

Облочка (shell) для взаимодетсвия с пользователем использует командную строку.

# Базовые команды

Если команда не выполняется по причине отсутствия доступа, то надо перед командой указать написать **sudo** – запуск от имени суперпользователя

<https://explainshell.com/> - документация по многим командам.

|  |  |  |
| --- | --- | --- |
| **echo** | Вывести переданный текст или значение переменной в стандартный поток вывода STDOUT (обычно — терминал, но можно перенаправить в файл). | echo [*опции*] [*строка*]  $ echo Привет, мир!  Привет, мир!  $ NAME="Алиса"  $ echo "Привет, $NAME!"  Привет, Алиса! |
| **echo $SHELL** | Выводит путь к оболочке (shell), которая назначена как ваша стандартная (login) оболочка в системе. | **$SHELL** — это переменная окружения, которая устанавливается при входе в систему (login).  echo $SHELL  /bin/bash |
| **echo $PWD** | Выводит текущий рабочий каталог (текущую рабочую директорию), в которой вы находитесь в терминале. | echo $PWD  /usr/src/app |
| **pwd** | Выводит текущий рабочий каталог. Аналогична **echo $PWD.** Обе эти команды выводят путь с учетом символических ссылок  С флагом **-P** выводит физический путь | cd /test  pwd  *# /test*  ln - s / test / symlink - example  cd / symlink - example  pwd  *# /symlink-example*  pwd - P  *# /test* |
| **ls**  сокр. от list  **ls -a**  Показать в том числе скрыте файлы  **ls -l**  Вывод информации по каждому файлу | Выводит список файлов и директорий в текущей рабочей директории  Вторым аргументов принимает как абсолютные, так и отнсительные пути (см. команду cd) | Без аргументов – список файлов и папко в текущей директории  ls путь – список файлов и папко в указанной директории |
| **cd** путь  сокр. от change diractory | Работает как с относительными, так и с абсолютными путями.  **Абсолютный** — первым символом в пути идет **/**  **Относительный** — во всех остальных случая | Абсолютный путь  *cd /root/test/inner-dir*  Относительный путь (когда уже находимся в /test)  *cd inner-dir*  или  *cd ./inner-dir* |
| **cd**  Без аругментов | Перейдем в домашнюю директорию текущего пользователя | Если домашняя директория /Users/guest  cd  pwd  *# /Users/guest* |
| **cd ..** | Переход на уровень вверх | Переход на три уровня вверх  cd ../../.. |
| **cd ~/имя\_папки** | ~ заменяется на абсолютный путь к домашней дирекории пользователя | Если домашняя директория /Users/guest  cd ~/Music  pwd  *#/Users/guest/Music* |
| **stat** | Информация о файле или директории | *# Информация о файле .profile в домашней директории пользователя*  stat ~/.profile |
| **env** | Вывести все установленные переменные окружения | env |
| **ps**  сокр. от process status | Покажет только процессы, запущенные в **текущем терминале текущим** пользователем. | ps  *# PID TTY          TIME CMD*  *# 118 pts/3    00:00:00 bash*  *# 131 pts/3    00:00:00 ps* |
| **ps aux** | **a** — показать процессы всех пользователей  **u** — отобразить информацию в удобочитаемом формате (с именем пользователя, CPU, памятью и т.д.)  **x** — включить процессы без управляющего терминала (например, демоны) | ps aux  *# root 22  0.0  0.0  12940  2244 pts/1    Ss   14:11   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri*  *# tirion  23  0.0  0.0   8632  5248 pts/1    S+   14:11   0:00 bash*  *# root 115  0.0  0.0  12940  6016 pts/2    Ss+  14:15   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri*  *# root 117  0.0  0.0  12940  1988 pts/3    Ss   14:15   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri* |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Command Line Interface CLI

Интерфейс командной строки

**Опции**

Всегда начинаются с одного – илли двух – дефисов.

Опции можно комбинировать между собой:

*# Вывод списка файлов и папок, в том числе скрытые (-a),*

*# с подробной информацией (-l)*

ls -la

И с аргументами. При этом опции обычно (*но не всегда*) указываются **слева** от аргументов:

*# Вывод списка файлов и папок, в том числе скрытые (-a),*

*# с подробной информацией (-l)*

*# из вложенной папки Music (относительный путь ./Music)*

ls - la Music

Часто опции имеют два варианта написания: корткий и длинный:

-v

*--*version

* Когда мы работаем в терминале, важно набирать быстро — там удобны короткие опции
* Когда мы пишем скрипт из разных команд, важно писать понятно — лучше использовать длинные опции. Так с первого взгляда очевидно, что означает каждая опция

Надо отметить, что обычно *длинные опции предваряются двумя дефисами*, но некоторые программы нарушают это правило и используют один, что вносит путаницу.

Опции могут принимаеть параметры.

В macOS есть утилита *say*. Если просто передать ей какой-то текст, то она его произнесет.

Можно передать ей текст, который она запишет в файл:

*# Вместо -o можно написать --output-file*

say -o hi.aac 'Hello, World.'

*# Запишет в файл i.aac*

Или так:

*# Можно имя файла обернуть в кавычки (если в нем есть спец символы или пробелы)*

say -o 'hi.aac' 'Hello, World.'

Значение опции обычно указывается через пробел в самой опции. Некоторые программы допускают использование =

say -o=hi.aac 'Hello, World.'

Кроме того, say допускает указать входный файл с текстом, который нужно прочитать.

*# -v - голос Алекса*

*# -f - файл, текст из которого надо прочитать*

say -v Alex -o hi.txt -f hello\_world.txt

Если открыть документацию по man, то увидим:

say [-v voice] [-r rate] [-o outfile [audio format options] | -n name:port | -a device] [-f file | string ...]

* [ ] – обозначают необязательные опции
* |– исключающее или: [-f file | string ...] – опция -f принимает или путь к файлу, или строку, которую надо прочитать.

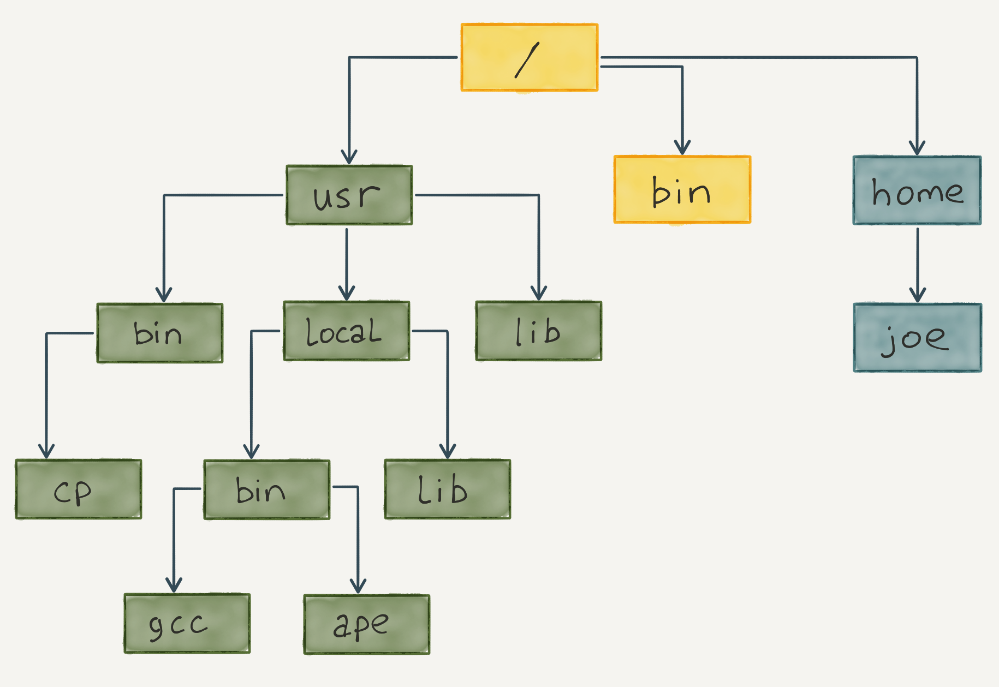
# Файловая структура

Понятие «папка» в \*nix-системах не используется. Папки здесь называются директориями или каталогами.

**Директория** — это специальный файл, который содержит список файлов. Любое подключаемое устройство становится файлом или директорией, если это накопитель.

Файловая структура \*nix-систем серьезно отличается от структуры в Windows и заслуживает отдельного внимания:

* Файловая структура представляет собой дерево
* В узлах этого дерева находятся директории — файлы специального типа
* В листьях дерева расположены файлы



В Windows файловая структура представлена не одним, а несколькими деревьями, так как каждая структура находится на своем диске. В \*nix-системах немного по-другому: есть **единственное дерево с корнем в /**. Все устройства, физические и логические диски находятся внутри этого дерева в виде директорий и файлов.

В \*nix-системах есть **FHS** — базовый стандартизированный набор каталогов.

* **/etc** содержит конфигурацию программ в обычных текстовых файлах. В \*nix-системах нет реестра, вся конфигурация лежит в обычных файлах
* **/home** содержит домашние директории пользователей системы, за исключением суперпользователя root
* **/root** домашний каталог суперпользователя root

**Особенности:**

* **Регистрозависимые имена**

В Windows и macOS мы привыкли, что имя файла может быть набрано в разных регистрах, и это всегда один и тот же файл.

Но в \*nix-системах ситуация другая — здесь регистр имеет значение. Файлы *index.html, Index.html, INDEX.HTML* и *index.HTML* — это разные файлы.

* **Скрытые файлы и папки**

Файл является скрытым, если его имя начинается с точки .

* **Директории (.) и (..)**

Если вызывать **ls -a**, то среди файлов будут файлы с именами (.) и (..)

* + (.) – одна точка – текущая директория
  + (..) – две точки – директория верхнего уровня

Благодаря этой схеме работает команда cd ..

* **Дополнительные типы файлов**
  + **Hard Link** — дополнительное имя для уже существующего файла
  + **Symbolic link** — символическая ссылка. Такой файл похож на ярлык в Windows.
  + **Socket** — специальный файл, через который происходит взаимодействие между разными процессами операционной системы.

# Работа с файлами

Файловую структуру можно не только просматривать, но и всячески модифицировать.

Возможность модифицировать файловую структуру завязана **на правах пользователя**. Если у вас нет соответствующих прав, вы получите ошибку доступа. Место, где вы гарантированно можете экспериментировать — ваша домашняя директория. Внутри нее все доступно на запись.

Последующие утилиты работают с файлами и директориями, расположенными в любом месте файловой системы. Поэтому вы всегда можете передать *абсолютный или относительный путь.*

|  |  |  |
| --- | --- | --- |
| **Работа с файлами** | | |
| **touch** | Создать файл.  Основное назначение – изменить время последнего доступа к файлу, но она обладает побочным эффектом:  Если файла нет, то он будет создан. | *# В текущей директории создается пустой файл*  touch empty-file |
| **rm**  сокр. от **remove** | Удалить файл | rm empty-file |
| **Mv**  сокр. от **move** | Перемещение  В \*nix-системах не существует понятия «переименовать файл».  **Переименование** всегда **равносильно перемещению** | touch file  mv file renamed-file |
| **cp**  сокр. от **copy** | Копирование | cp renamed-file renamed-file-copy |
| **Работа с директориями** | | |
| **mkdir**  сокр. от **make directory** | Создать директорию | mkdir my-dir |
| **mkdir -p**  **mkdir --parents** | Cоздаnm директории рекурсивно | mkdir -p one/two/three |
| **rm -r** | Удалить директорию | rm -r my-dir |
| **rm -rf**  **--force** | Удалить директорию, игнорируя несуществующие файлы и не запрашивать подтверждение на удаление |  |
| **cp -r** | Копирование директории (-r – рекурсивное) | cp -r dir dir-copy |

# Чтение файлов

## Команды

|  |  |  |
| --- | --- | --- |
| **cat** | Прочитать файл целиком | cat .profile  *# if [ -n "$BASH\_VERSION" ]; then*  *#     # include .bashrc if it exists*  *#     if [ -f "$HOME/.bashrc" ]; then*  *#     . "$HOME/.bashrc"*  *#     fi*  *# fi* |
| **Head**  **head -n 3** | Чтение первых 10 строк с начала файла  -n 3  Чтение 3 первых строк | head -n 2 .profile |
| **tail**  **tail -n 3** | Чтение последних 10 строк в файле  -n 3  Чтение 3 последних строк | tail -n 2 .profile |
| **tail -f** | Выводит последние строчки и ждет появления новых.  Захватывает управление | *# Чтение системных логов*  tail - f /var/etc/syslog |

## Пейджеры

Пейджеры – особая категория программ для просмотра содержимого файла. Пейджер похож на текстовый редактор, открытый только в *режиме чтения*.

Самый распространенный пейджер **less**

less /var/etc/syslog

Пейджер не пытается загрузить в память весь файл до его отображения. Он грузит только *ту часть, которая помещается на экран* и при перемещении подгружает остальное.

Пейджер позволяет производить поиск и перемещаться по файлу вперед и назад.

Основные команды для **less**:

* **q** – выход
* **f** – вперед на страницу
* **b** – назад на страницу
* Поиск введенного текста — вводим **/**, печатаем текст и нажимаем Enter
  + **n** – переход к следующему совпадению
  + **N** – переход к предыдущему совпадению

# Grep

grep (сокр. от global regular expression print) – утилита для поиска определенного текста по файлу или файлам.

|  |  |  |
| --- | --- | --- |
| **grep СТРОКА ФАЙЛ** | Поиск строки или регулярного выражения в файле  Выводит список строк, в которых встречается искомый текст. | *# Поиск всех строк в файле .profile, в которых встречается слово PATH*  grep PATH .profile  *# Вывод:*  *# set PATH so it includes user's private bin if it exists*      PATH="$HOME/bin:$PATH"  *# set PATH so it includes user's private bin if it exists*      PATH="$HOME/.local/bin:$PATH" |
| **grep -B ЧИСЛО**  **grep --before-context ЧИСЛО** | Количество отображаемых строк до искомой строки в результатах поиска. | grep -B 3 PATH .profile |
| **grep -A ЧИСЛО**  **grep --after-context ЧИСЛО** | Количество отображаемых строк после искомой строки в результатах поиска. | grep -A 3 PATH .profile |
| **grep -C ЧИСЛО**  **grep --context ЧИСЛО** | Количество отображаемых строк до и после искомой строки в результатах поиска. | grep -C 3 PATH .profile |
| **grep -r СТРОКА .**  Вроде бы, можно дажебез точки  **grep -r СТРОКА** | Не знаем, какой файл. Указываем искать во всех файлах текущей директории | grep -r PATH . |
| **grep -rn СТРОКА .** | Выводить имя файла напротив найденного совпадения | grep -rn PATH . |

# Редактирование файлов

Для редактирования файлов в командной строке используют редакторы Nano, Vim и Emacs.

В большинстве случаев в системе установлен только Vim.

Эти редакторы запускаются прямо в терминале, хотя Vim и Emacs имеют режим, в котором они запускаются как отдельные приложения.

## Nano

Запуск командой

nano ФАЙЛ

Например:

nano .profile

Это обычный редактор, который подходит для несложных изменений в файлах, но им совсем неудобно пользоваться для работы над проектами с множеством файлов и сложным кодом.

## Vim

**Vim** (сокр. от Vi Improved) – свободный текстовый редактор, созданный на основе более старого vi (visual editor).

Ключевое отличие Vim от остальных редакторов — наличие режимов. В обычных редакторах мы открываем файл и сразу можем приступать к редактированию. В отличие от них, Vim запускается в режиме COMMAND — это режим команд, при котором нажатие на любую клавишу воспринимается как сигнал к какому-то действию.

Если вы не знаете команд, то лучше ничего не трогать, иначе на экране начнется хаос. Обычное редактирование текста выполняется в режиме ввода — INSERT. Попасть в него можно с помощью клавиши i. Для выхода в режим команд достаточно нажать Esc или комбинацию Ctrl + [.

Всего у Vim есть четыре основных режима работы:

* **Командный режим** — это режим по умолчанию. В нем можно перемещаться по файлу, вносить изменения с помощью комбинаций клавиш, удалять и искать текст
* **Режим вставки** — это режим ввода текста с клавиатуры. Здесь все работает так, как вы привыкли это делать в обычных редакторах
* **Визуальный режим** — выделение произвольных участков текста
* **Режим командной строки** — собственная командная строка для редактора текста, из которой можно выполнять множество разнообразных команд

Vim использует широкую систему плагинов, которые делают из него полноцценный редактор кода.

Чтобы выйти из Vim:

1. Убедитесь, что вы находитесь в командном режиме. На всякий случай нажмите Esc или комбинацию Ctrl + [
2. Затем наберите : (символ «двоеточие») — эта команда переведет вас в режим командной строки
3. Введите:
   * q! – выйти без сохранений
   * :wq – сохранить все изменения перед выходом
   * :w – сохранить изменения без выхода

и нажмите Enter

# Потоки

Несмотря на разнообразие языков и способов печати, с точки зрения операционной системы все программы работают абсолютно идентично. При старте любой программы операционная система связывает с ней три потока:

* **STDIN** (Standard Input) – файловый дескриптор 0
* **STDOUT** (Standard Output) – файловый дескриптор 1
* **STDERR** (Standard Error) – файловый дескриптор 2

Для языка программирования эти потоки выглядят как файлы, и взаимодействие с ними происходит как с файлами.

За каждым потоком закреплен номер – файловый дескриптор

## STDOUT

**STDOUT** отвечает за вывод на экран. Во время каждой печати на экран в программе на любом языке происходят следующие два шага:

1. Функция печати записывает данные в поток STDOUT с помощью функции write
2. Операционная система решает, куда вывести результат. По умолчанию вывод происходит на экран терминала

В двух словах, языки программирования **не взаимодействуют с монитором и железом** в целом. За взаимодействие с железом целиком и **полностью отвечает операционная система**. Программы могут только лишь попросить операционную систему выполнить ту или иную задачу.

При таком разделении реализация языков программирования сильно упрощается. Достаточно знать про существование STDOUT и уметь писать в этот поток, а дальше все сделает операционная система.

Именно благодаря такому разделению и потокам, можно написать программу на одном компьютере и без проблем запустить ее на другом компьютере с другой конфигурацией и монитором.

Операционные системы позволяют подменять потоки при старте системы.

Например, вывод любой команды, выводящий на экран что-либо, можно перенаправить в файл:

ls -la > output

Перенаправит вывод команды (или правиьнее «программы») ls в файл output. В результате в output будет содержаться список файлов текущей директории

* Оператор > – файл вывода будет перезаписан (например, > output)
* Оператор >> – информация будет доабавлена в конец файла вывода (например, >> output)

## STDIN

**STDIN** отвечает за стандартный ввод: через него программа может получать данные на вход.

В командах надо сипользовать стрелку, указывающую в обратном направлении (если сравнивать со STDOUT)

Пример:

Утилита wc считает количество слов, строк и символов в файле

-l – считать количество строк

*# Посчитать количество строк в файле result и вывести на экран*

wc -l < result

Можно *объединять* перенаправления STDIN и STDOUT:

*# Содержимое файла result отправляем в стандартный ввод команды wc, а вывод направляем*

*# в файл output*

wc -l < result > output

Приблизительно таким образом устроен вывод на печать в компьюетере.

## STDERR

**STDERR** позволяет отделить нормальный вывод программы от возникающих ошибок.

Как и STDOUT, он по умолчанию выводит информацию на экран.

**Важно!** Перенаправление вывода в файл перенаправляет только STDOUT. Поток STDERR будет продолжать выводиться на экран.

Решить эту проблему можно:

* перенаправлением потока STDERR в STDOUT

*# Перенаправить поток вывода в файл output*

*# 2 поток (STDERR) перенаправить в 1 (STDOUT)*

ls lala > output 2>&1

Символ & после перенаправления > важен – такой синтаксис.

Также важно вначале указать файл, а потом перенаправить STDERR в SDTOUT. Если сделать наоборот, то работать не будет.

* перенаправлением STDERR в файл

cd lala 2> output

* перенаправлением обоих поток в файл

cd lala &> output

# Пайплайны

Благодаря пайплайну можно соединять программы и протаскивать данные сквозь них. Получается цепочка функций, в которой каждое звено выступает в роли преобразователя или фильтра.

Символ **пайп |** в пайплайне означает командная оболочка должна взять поток STDOUT одного процесса и соединить его с потоком STDIN другого процесса.

Все утилиты, которые читают файлы, могут принимать данные через поток STDIN.

*# В файле .profile найти все строки, содержащие текст PATH*

*# затем найти в результатах всех строки, содержащие текст local*

grep PATH .profile | grep local

Эту же команду можно сделать более читаемой:

cat .profile | grep PATH | grep local

Или более сложная команда

*# Прочитать source*

*# Найти все строки с текстом Dog*

*# Удалить дубли*

*# Отсортировать и вывести на экран*

cat source | grep Dog | uniq | sort

Пайплайн стал основой **философии Unix**:

* Пишите программы, которые делают что-то одно и делают это хорошо
* Пишите программы, которые бы работали вместе
* Пишите программы, которые бы поддерживали текстовые потоки, поскольку это универсальный интерфейс

Именно поэтому большинство утилит работают с сырым текстом — принимают его на вход и возвращают в поток STDOUT.

Такой подход позволяет получать сложное поведение из крайне простых составных блоков. Такая концепция называется **стандартные интерфейсы**.

# Переменные окружения

**Переменные окружения** похожи на переменные в языках программирования. Они существуют в рамках запущенной сессии командной оболочки, то есть, *переменные пропадут, когда терминал закроется*. Они подгружаются туда во время его инициализации, хотя это не единственный путь их появления.

Посмотреть установленные переменные

env

*# TERM=xterm-256color*

*# SHELL=/bin/bash*

*# LC\_ALL=en\_US.UTF-8*

*# USER=kirill.m*

*# HEXLET\_VERSION=v2711*

*# PATH=/home/kirill.m/bin:/home/kirill.m/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin*

*# PWD=/home/kirill.m*

*# LANG=en\_US.UTF-8*

*# SHLVL=1*

*# HOME=/home/kirill.m*

*# LOGNAME=kirill.m*

Или по отдельности:

echo *$*HOME

*# /home/kirill.m*

В разных ОС состав переменных может немного отличаться.

При выозве *cd* осуществляется переход в домашнюю директорию, которая определена в HOME

echo *$*HOME

*# /home/kirill.m*

cd

pwd

*# /home/kirill.m*

**Два способа переопределить переменные окружения:**

1. **Локально** для конкретной команды

*# Достаточно выполнить присваивание слева от запускаемой команды*

HOME=/tmp cd

*# /tmp*

pwd

*# /tmp*

1. **Глобально**

export HOME=/tmp

cd

pwd

*# /tmp*

echo *$*HOME

*# /tmp*

Обратите внимание, что изменение происходит в рамках текущей сессии. Другими словами, если у нас открыто две вкладки терминала, то изменение произойдет только в той вкладке, в которой мы написали команду.

# История

История команд bash хранится в специальном файле **.bash\_history**, который лежит в домашней директории пользователя.

Запись происходит при завершении сеанса.

**HISTFILESIZE** – переменная, в которой указано, какое количество команд хранится в истории, отвечает переменная окружения. Если переменной нет, история не обрезается и файл *.bash\_history* растет бесконечно.

**Посмотреть историю:**

history

*#  1  docke rps*

*#  2  free -m*

*#  3  docker ps*

*#  4  docker exec -it 8678a6520641 bash*

*#  5  ls*

*#  6  exit*

*#  7  docker ps*

*#  8  docker exec -it 1209b6e5ce6b bash*

*# Выведет 5 послдених команд*

history 5

Чтобы повторно не вводить команду с клавиатуры, можно вввести в командной строке !НОМЕР\_КОМАНДЫ

*# Вызовет команду под номером 7 из файла history*

*!*7

**Поиск по истории:**

history | grep export

*#  174  export HOME=/tmp*

*#  183  history | grep export*

**Реверсивный поиск по истории:**

1. Нажимаем Crtl + r
2. Появляется надпись:



1. Начинаем вводить, например, MAIL
2. Если найденная команда устроила



То нажимаем Enter

1. Если нет, нажатие еще раз Crtl + r переведет нас к следующему совпадению.

# Пользователи и группы

Выяснить имя пользователя

whoami

*# ivan.ivanov*

*Абсолютно любой процесс*, запускаемый в операционной системе*, стартует от имени некоторого пользователя*. Соответственно, его возможности по влиянию на файловую систему ограничены правами пользователя, от имени которого процесс запущен.

|  |  |  |
| --- | --- | --- |
| **ps**  сокр. от process status | Покажет только процессы, запущенные в **текущем терминале текущим** пользователем. | ps  *# PID TTY          TIME CMD*  *# 118 pts/3    00:00:00 bash*  *# 131 pts/3    00:00:00 ps* |
| **ps aux** | **a** — показать процессы всех пользователей  **u** — отобразить информацию в удобочитаемом формате (с именем пользователя, CPU, памятью и т.д.)  **x** — включить процессы без управляющего терминала (например, демоны) | ps aux  *# root 22  0.0  0.0  12940  2244 pts/1    Ss   14:11   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri*  *# tirion  23  0.0  0.0   8632  5248 pts/1    S+   14:11   0:00 bash*  *# root 115  0.0  0.0  12940  6016 pts/2    Ss+  14:15   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri*  *# root 117  0.0  0.0  12940  1988 pts/3    Ss   14:15   0:00 sudo -H PATH=/usr/local/lib/hexlet-javascri* |
| **ps -u $USER** | Показать все процессы текущего пользователя |  |
| **ps axjf** | **Показать процессы в формате дерева (иерархия)** |  |

Например, мы запускаем утилиту touch. Мы от своего имени стартуем процесс, внутри которого запускается программа touch. Эта программа создает файл и делает вас владельцем нового файла.

Каждый каталог в директории /home — это домашний каталог конкретного пользователя. Поэтому они все имеют разных владельцев, как правило, совпадающих с именем директории.

Каждому пользователю соответствуют имя и UID. Если изменить имя (так можно), но сохранить UID, то доступы сохраняться.

*# Посмотреть UID*

id

*# uid=1000(tirion) gid=1000(tirion) groups=1000(tirion),100(users)*

Записи о пользователях хранятся в файле **/etc/passwd**

cat / etc / passwd

*# root:x:0:0:root:/root:/bin/bash*

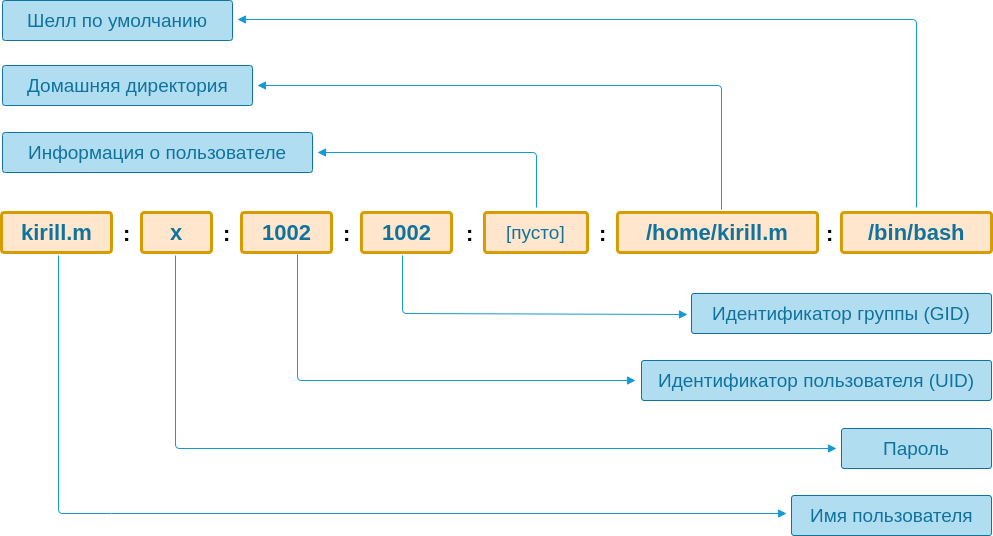
*# daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin*

*# proxy:x:13:13:proxy:/bin:/usr/sbin/nologin*

*# ...*

*# kirill.m:x:1002:1002::/home/kirill.m:/bin/bash*

Каждая строка расшифровывается по схеме:



Запись */usr/sbin/nologin* у некоторых пользователей говорит о том, что пользователь не может входить в систему. Такие пользователи нужны для запуска программ, у которых ограниченные права — естественно, входить в систему им не нужно.

**Группы**

Пользователей можно объединять в группы и назначать права сразу целой группе.

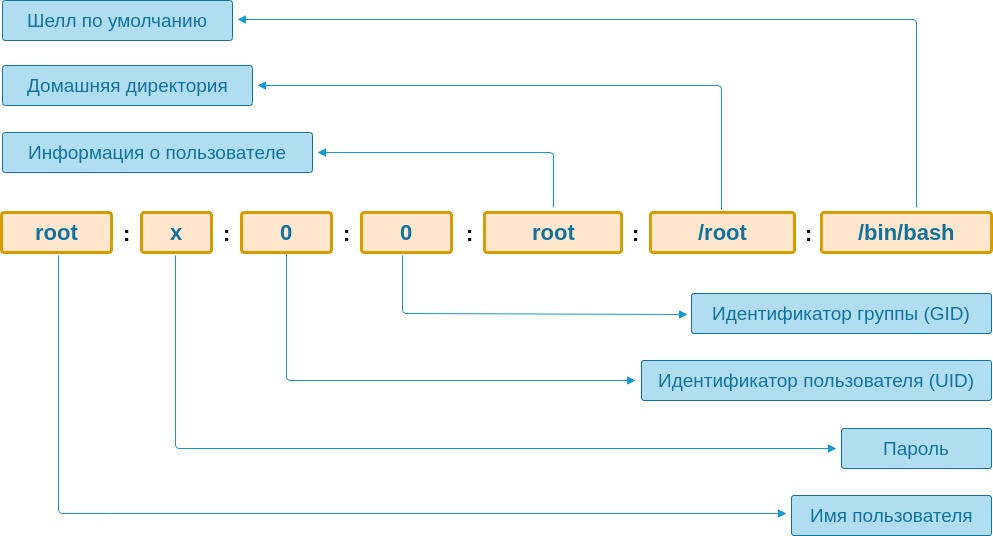
id

*# uid=1000(tirion) gid=1000(tirion) groups=1000(tirion),100(users)*

groups содержит группы, к которым относится пользователь

**Суперпользоватеь root**

В любой \*nix-системе присутствует специальный пользователь — root или **суперпользователь**. Главная его особенность — это **идентификатор** со значением **0**. Этот пользователь имеет особое значение и может выполнять абсолютно любые действия в системе. У пользователя root в файле */etc/passwd* будет вот такая запись:



# Sudo

Ряд действий и программ могут потребовать права суперпользователя:

* Установка новых программ
* Навигация по чужим директориям
* Изменение прав доступа
* Изменение содержимого в файлах, не принадлежащих текущему пользователю
* Создание, редактирование и удаление файлов, когда не хватает прав текущего пользователя
* Запуск программ, требующих повышенных привилегий

Устаревший способо – утилита **su**

Современный способ – утилита **sudo** (*substitute user and do* — дословно «подменить пользователя и выполнить»).

**sudo** пытается повысить привилегии до суперпользователя

touch /etc/myfile

*# touch: cannot touch '/etc/myfile': Permission denied*

*# С sudo все работает*

sudo touch /etc/myfile

В зависимости от настроек, утилита sudo попросит ваш пароль для входа или вообще откажется работать, сказав, что у вас нет права ее использовать. Как правило, в Ubuntu sudo спрашивает пароль и запоминает его на пять минут. На протяжении этого времени вы можете использовать sudo, не вводя пароль каждый раз.

Чтобы выполнить sudo от имени другого пользователя, используется флаг **-u**:

sudo -u nobody mkdir /tmp/test

*# Директория создана от имени пользователя nobody*

Чтобы не вводить паролькаждый раз, можно открытьсессию от имени суперпользователя:

*# Команда `sudo` запросит пароль текущего пользователя*

sudo -i

*# Проверяем, что сессия запущена от суперпользователя*

id

*# uid=0(root) gid=0(root) groups=0(root)*

*# ...*

*# После завершения работы выйти из сессии:*

exit

# Права доступа

Кроме имени пользователя и группы, с каждым файлом ассоциированы **права доступа**:

* **r** — чтение (read)
* **w** — запись (write)
* **x** — исполнение (execute)

Права доступа к файлам и каталогам могут изменять *только* *определённые пользователи*, *в зависимости от контекста.*

* **Владелец** файла/каталога

Может изменять права доступа к *своему* файлу или каталогу с помощью команды *chmod*.

* **root суперпользователь**

Может изменять права *любого* файла или каталога в системе

* **Обычный пользователь**, не являющийся владельцем

Не может изменять права доступа к чужому файлу — даже если у него есть права на запись в родительский каталог.

Права доступа представлены строкой

**-rw-r--r--** *(пример)*

Первые три символа – права для владельца

Следующий три символа – права пользователей, входящих в ту же группу (Group), что и владелец

Последние три символа – права остальных пользователей

Прочерк (-) означает отсутствие соответствующих прав.

То есть в пример -*rw-r--r—*владелец имеет права на чтение и запись, его группа и остальные – тоько на чтение.

Не важно, какой файл или директорию мы смотрим — порядок прав в этой группе всегда один и тот же — **чтение-запись-исполнение**, а прочерк означает отсутствие этого права.

Домашняя директория имеет другие права:

* В самом начале вместо - стоит d, которая обозначает директорию
* Права для владельца — **rwx**, а для всех остальных — **r-x**. Из этого описания видно, что **только пользователь может писать внутрь своей домашней директории**

Что такое **x** в отношении директорий? Это право позволяет перемещаться в директорию и обращаться ко всем расположенным в ней файлам и каталогам. Обращаться можно при условии, что эти файлы доступны на чтение, запись или выполнение. Например, если положить доступный на чтение файл в директорию с правом x, то вы сможете прочитать этот файл. Если убрать с этой директории право x, то вы лишитесь доступа к файлу.

Иногда для опсиания прав используют цифры вместо букв

| **A** | **B** | **C** | **D** |
| --- | --- | --- | --- |
| # | Permission | rwx | Binary |
| 7 | read, write and execute | rwx | 111 |
| 6 | read and write | rw- | 110 |
| 5 | read and execute | r-x | 101 |
| 4 | read only | r-- | 100 |
| 3 | write and execute | -wx | 011 |
| 2 | write only | -w- | 010 |
| 1 | execute only | --x | 001 |

Примеры:

**drwxr-xr-x** соответствует **755**

**-rw-r--r--** соответствует **644**

**Сменить владельца:**

* Смена **владельца** (chown)
  + Только root может менять владельца файла.
  + Даже владелец файла не может передать его другому пользователю через chown (в целях безопасности).
* Смена **группы** (chgrp)
  + Владелец может изменить группу файла только на ту, в которой он сам состоит.
  + Root может назначить любую группу.

|  |  |  |
| --- | --- | --- |
| **chown НОВЫЙ\_ВЛАДЕЛЕЦ ФАЙЛ**  Сокр. от change owner | Сменить владельца файла | *# Сменить владельца файла .profile на another-user*  chown another-user .profile |
| **chown НОВЫЕ\_ПРАВА ФАЙЛ**  Сокр. от change mode | Сменить права у файла  С помощью знаков + и - права добавляются или удаляются соответственно | chmod +r file.txt  chmod 754 file.txt |

# Пакетны менеджер

Для устанвоки программ используются пакетные менеджеры.

Пакет – это упаковка для программ. Чтобы пользователь мог установить программу через пакетный менеджер, ее упаковывают в пакет.

В Ubuntu используется специальный формат пакетов, называемый deb. У этого формата есть подробная документация о том, как производить упаковку.

Почти всегда пакетные менеджеры требуют запуска от суперпользователя root, потому что установка программ происходит в системные директории, недоступные для записи обычному пользователю.

В **MacOS** используется пакетный менеджер **brew**

Ubuntu поставляется вместе с пакетным менеджером **apt**

|  |  |  |
| --- | --- | --- |
| sudo apt install | Установить пакет | sudo apt install php |
| apt remove | Удалить пакет | apt remove php |
| sudo apt update | Обновить список пакетов в индексе (для ситуация, когда пакет в реальности существует, но apt говорит, что нет).  Скачивает свежие списки пакетов из настроенных репозиториев и обновляет локальный кэш информации о пакетах.  Она не устанавливает и не обновляет программы — только обновляет метаданные. | |

# Выполнение программ

## Общий порядок

Программа – это набор файлов. Чтобы программа выполнилась, один из ее файлов должен быть **исполняемым** (традиционно их называют *бинарники*).

При установки программы пакетный менеджер переносит ее исполняемые файлы в одну из системных папок /**bin.** Список этих системных папок /bin содержится в переменной PATH

echo *$*PATH

*# /usr/local/lib/hexlet-javascript/node\_modules/.bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin*

В этой строке директории отделены отдна от другой символом двоеточия :

Когда мы зпускам программу, например, утилиту **ls**

ls

система ничаниет искать ее в этих папках слева направо. Если файл с одним и темже именем находится в нескольких папках, то будет заущен первый найденый.

|  |  |
| --- | --- |
| **type**  type ls | Узнать, где находится исполняемый файл |
| **whereis**  whereis ls | Узнать, где находится исполняемый файл |
| **which**  which ls | Узнать, где находится исполняемый файл |

Кроме того, сиполняемый файл можно запустить из любой директории, если на этот файл установлены права на исполнение (x). Но **важно** указывать пусть к этому файлу через слэши, даже если он лижет в текущий папке:

*# Примеры*

./executable

*# или*

/home/hex/temp/executable

## Изменение PATH

При правильной установке большинство программ самостоятельно добавляют свои исполняемые файлы в нужные места. В некоторых случаях они просят об этом пользователя — дают подсказки, какие файлы и куда нужно добавить после установки.

Иногда возникает необходимость расширить PATH (чтобы вызов установленной программы стал доступен глоавльно из любой директории ОС).

Особенно часто так происходит, когда мы устанавливаем программу через пакетные менеджеры языков программирования. Делать это надо осторожно — убедившись в том, что по-другому нельзя.

Чтобы вручную изменить переменную PATH, нужно поправить один из конфигурационных файлов Bash. Эти файлы лежат в домашней директории пользователя и могут называться так:

* .bashrc.
* .bash\_profile.
* .profile.

В зависимости от настроек терминала, Bash загружает либо одни файлы, либо другие. Если в вашей домашней директории есть файл .bashrc, то попробуйте использовать его. Если нет, пробуйте остальные файлы в том порядке, в котором они приведены выше. Добавьте в этот файл следующую строку:

export PATH=*$*PATH:/path/to/directory

Если вы используете оболочку Zsh (например, на MacOS), то вместо .bashrc используется файл .zshrc. Чтобы изменить значение переменной PATH в Zsh, откройте файл ~/.zshrc и добавьте ту же самую строку.

После этих манипуляций не забудьте перезапустить терминал.