# **Введение**

Как зайти на сервер?

Как выгрузить код на сервер?

Как запустить сайт?

Как перемещаться на сервере?

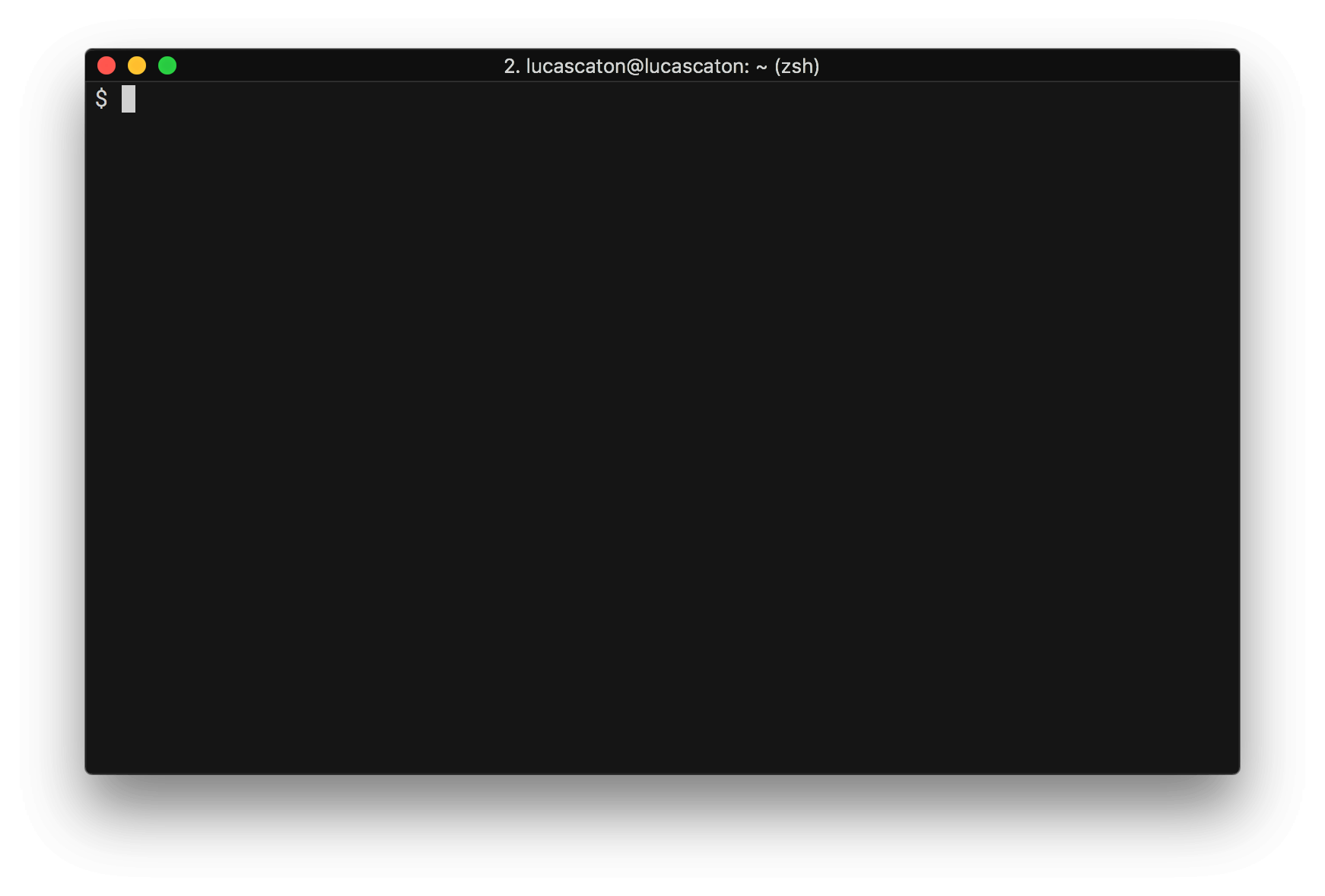
Как редактировать файлы на сервере?

Как анализировать проблемы в процессе работы сайта?

Как анализировать систему при различных неполадках?

Эти вопросы становятся особенно интересными, если учесть, что на большинстве серверов установлен не привычный Windows, а Linux без графической оболочки.

Подключившись к удаленной машине, вы увидите **терминал** — специальную программу, через которую идет подключение:



Это командная строка, в которой можно выполнять все привычные операции на домашнем компьютере:

* Вводить разные команды
* Запускать программы
* Анализировать систему

Кому-то такой способ взаимодействия с системой покажется архаичным, но это только на первый взгляд. Текстовый режим дает множество преимуществ:

* Значительно экономит ресурсы, что особенно важно для серверного окружения
* Позволяет автоматизировать любые действия — команды можно не только выполнять вручную, но и записать в файл и выполнять все вместе
* Дает практически неограниченный доступ к возможностям системы
* Открывает больше возможностей во многих программах

Умение работать с командной строкой помогает проще решать самые разные задачи и повышает ваш уровень как разработчика. Вы легко сможете запустить и отладить сайт на удаленной машине, а также оптимизировать ежедневную работу на своем компьютере. Командная строка нужна:

* Для запуска кода на любом языке
* Для совместной разработки
* Для работы с самыми распространенными инструментами фронтенд-разработчика (Gulp, Webpack, Babel и так далее)

# **Операционные системы**

Все началось в 1970-х годах. В те времена не существовало ни привычной Windows, ни персональных компьютеров. Компьютерами называли **мейнфреймы** — большие машины, занимающие целые этажи зданий:



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. Сам терминал выглядел так:



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. Другими словами, разработчикам приходилось писать отдельную программу под каждую операционную систему.

Такая ситуация усложняла жизнь всем, 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.

## **Операционная система Linux**

В те же времена начали активно развиваться сети. Сначала появились прообразы интернета, а затем и сам интернет. Компьютеры становились меньше, а серверов — все больше. В 1991 году студент Линус Торвальдс решил написать собственную операционную систему, которая совместима со стандартом POSIX.

Благодаря интернету, Линус собрал целое сообщество разработчиков, которые довольно быстро создали рабочую операционную систему под названием **Linux** и выложили ее исходный код в открытый доступ. С тех пор популярность Linux для серверного окружения быстро росла.

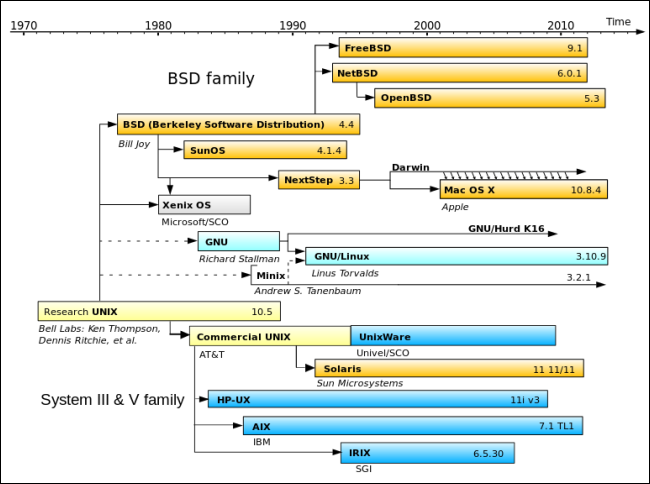
Но сам Linux — это только ядро операционной системы, которое не получится использовать без сопутствующих программ.

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**

Все эти программы были написаны для операционных систем, похожих на Unix. Поэтому в них использовались стандартные для UNIX системные вызовы — POSIX. Благодаря этому, Linux сразу начал поставляться с утилитами — например, архиваторами или командной оболочкой bash, которую мы начнем изучать в этом курсе.

Удивительный факт для тех, кто знаком только с Windows — Linux не имеет своей графической оболочки. Именно поэтому Linux может выглядеть очень по-разному, ведь на него можно поставить множество разных графических оболочек:

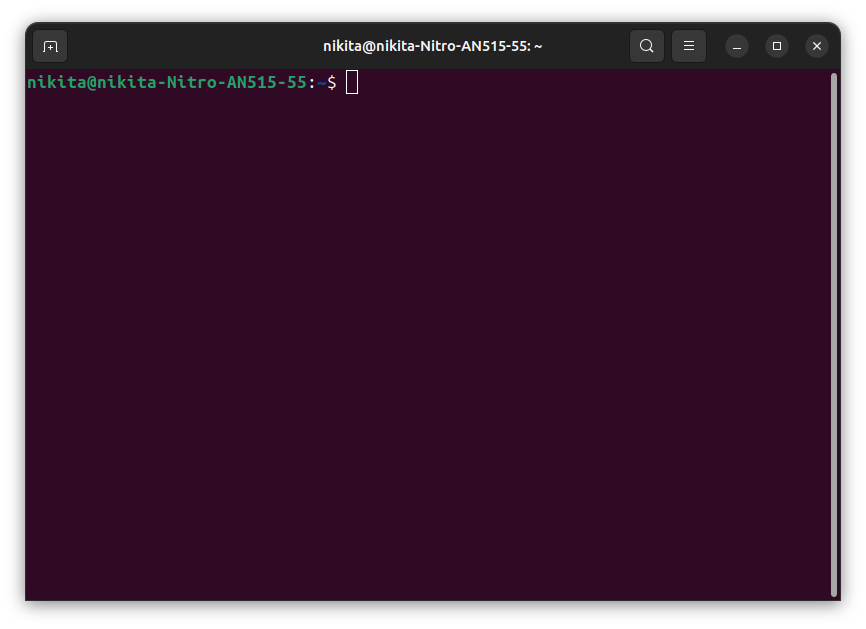


Сейчас Linux не используется в чистом виде. Вместо этого конечные пользователи имеют дело с **дистрибутивами**. Дистрибутив включает в себя:

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

## **Терминал**

Терминалом называют программу, которая эмулирует поведение железного терминала, состоящего из клавиатуры и монитора. В простейшем случае — это просто окошко с командной строкой внутри:

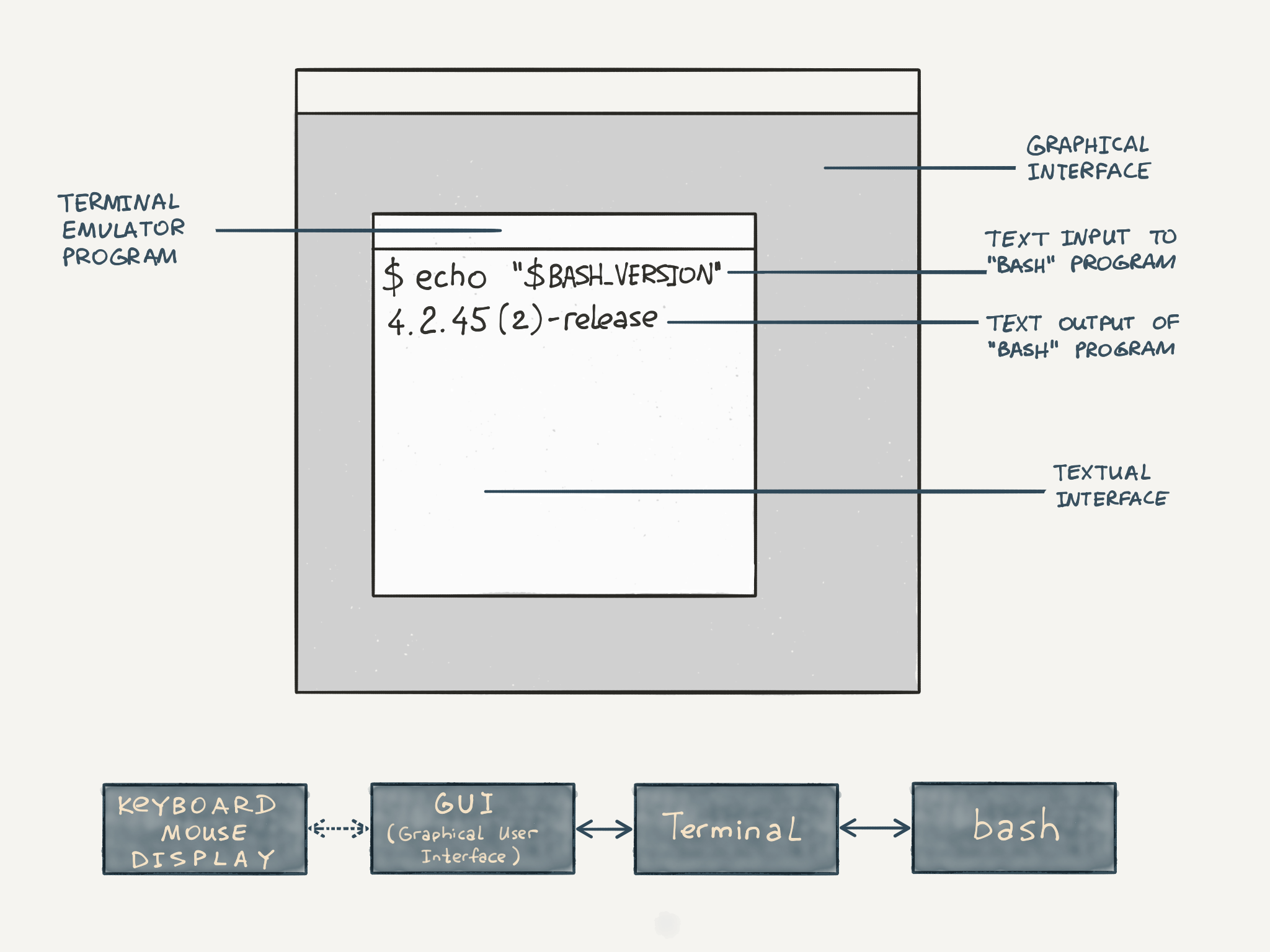


## **Командная оболочка**

**Командная оболочка** — программа, через которую пользователь или администратор управляет операционной системой и установленными программами, используя командную строку. У этого понятия множество синонимов — shell, командный интерпретатор и командный процессор.

Оболочка — это лишь средство для выполнения определенных задач, а не сама задача. Этим она отличается от многих других программ.

Внутри терминала пользователь передает системе строки:



Каждая строка — это команда, которую система должна выполнить. После ввода команды обязательно нажать Enter, только в этом случае команда отправится на исполнение. После нажатия Enter отредактировать команду уже нельзя.

Оболочка запускается внутри терминала и приглашает ко вводу команд:

~$ █

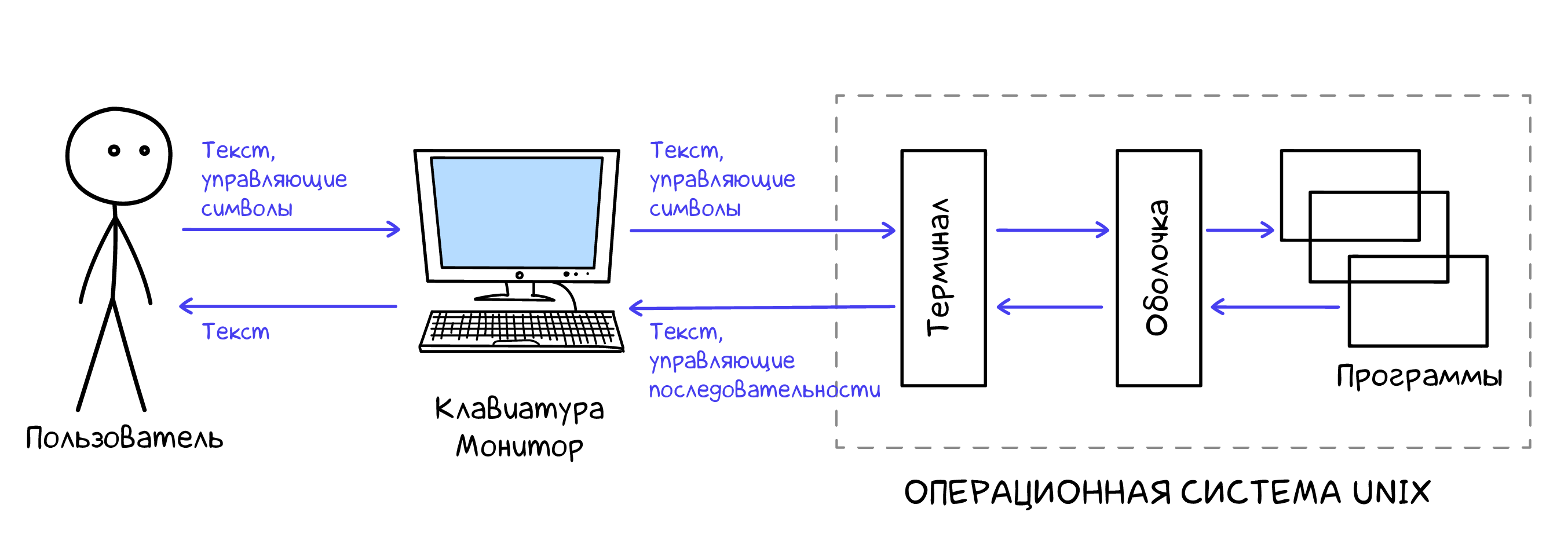
Все примеры в дальнейшем будут демонстрироваться без этого сообщения и символа. Рассмотрим для примера команду date, которая выводит текущую дату:

date

Sun Aug 26 14:02:59 CEST 2018 █

Другой пример команды — комментарии. Как и в любых языках программирования, они ни на что не влияют. С их помощью мы просто описываем происходящее:

*# Комментарий* █



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

## **Как проверить, в какой директории мы находимся**

Начнем с самого основного. Проверить, в какой директории мы сейчас находимся, можно командой pwd:

pwd

/Users/guest

Кстати, название команды pwd — это сокращение, которое расшифровывается как *print working directory*. Похожим образом устроены имена многих команд, что позволяет легче и быстрее их запомнить.

## **Как посмотреть список файлов**

Изучим команду ls (сокращение от *list*). Она выводит список файлов и директорий в текущей рабочей директории:

ls

Desktop Documents Downloads Library Movies Music Pictures Public

## **Как переместиться в другую директорию**

Еще одна полезная команда — cd (сокращение от *change directory*). С помощью нее мы перемещаемся по файловой структуре. Для этого ей нужно передать **аргумент** — директорию, в которую необходимо переместиться:

*# Входим в директорию*

cd

Music

*# Смотрим ее содержимое*

ls

iTunes

*# Смотрим текущую рабочую директорию*

pwd

/Users/guest/Music

*# Если имя директории содержит пробел, то его нужно экранировать с помощью `\`*

cd

Best\ music

Остановимся на этом моменте подробнее. Возможно, вы знаете, что есть два способа обозначить путь до файла:

* **Абсолютный путь** начинается от корня
* **Относительный путь** начинающийся от текущей рабочей директории

Теперь рассмотрим другую задачу. Предположим, что мы находимся в директории */Users/guest/Music*. Как выйти из нее и попасть снова в */Users/guest*? Мы уже знаем один способ — указать абсолютный путь и сделать cd:

cd

/Users/guest

Но есть путь проще. Можно указать специальное значение .. и перейти на директорию уровнем выше:

*# В директории /Users/guest/Music*

cd ..

pwd

/Users/guest

Более того, с помощью этого заполнителя можно выходить на любое количество уровней, указывая .. через разделитель:

*# В директории /Users/guest/Music*

*# Выходим на два уровня вверх*

cd ../..

pwd

/Users

Иногда в пути используется **одинарная точка**, которая означает текущую директорию. Например, вместо cd Music можно писать cd ./Music — разницы между этими выражениями нет.

Есть и третий вариант возврата в */Users/guest* из директории */Users/guest/Music*. Можно выполнить команду cd без аргументов, тогда мы перейдем в домашнюю директорию текущего пользователя:

*# Из любого места*

cd

pwd

/Users/guest

Ну и, наконец, четвертый вариант. Домашняя директория пользователя имеет специальное обозначение — ~ (тильда). В момент выполнения команды тильда заменяется на абсолютный путь. Поэтому из любого места можно напрямую перейти в любую поддиректорию домашней директории:

*# Из любого места*

cd ~/Music

pwd

/Users/guest/Music

Допустим, вы находитесь в домашней директории и хотите посмотреть файлы в поддиректории *Music*. Один способ вы уже знаете — для этого нужно перейти в директорию *Music* и выполнить программу ls.

Как обычно, есть другой способ. Команда ls также может принимать на вход **аргумент** — директорию, которую нужно проанализировать:

ls Music

iTunes

Со временем набирать пути становится все более лениво. Тогда можно нажать Tab и воспользоваться автокомплитом — функцией, которая автоматически завершает имена.

## **Аргументы и опции**

Чем чаще вы будете использовать командную строку, тем больше различных программ вам встретится. Многие из них станут повседневными инструментами. Например, вы часто будете пользоваться программой ls, которая выводит на экран список файлов и директорий.

Здесь все просто. Достаточно набрать название программы и нажать Enter:

ls

Desktop Documents Downloads Library Movies Music Pictures Public

Еще мы можем посмотреть скрытые файлы и директории. В \*nix-системах они начинаются с точки: .profile.

Тогда необходимо набрать ls -a:

ls -a

. .CFUserTextEncoding Desktop Downloads Movies Pictures

.. .localized Documents Library Music Public

А если захотим посмотреть содержимое каталога Public? Тогда мы воспользуемся командой ls с аргументом:

ls Public

Drop Box

Некоторые программы сложно конфигурируются, поэтому их бывает трудно использовать. Посмотрим на такой неочевидный пример:

ffmpeg -i input.mp4 -vcodec libx264 -crf 30 output.mp4

Практически любую команду можно дополнить двумя способами:

Способ 1 — это **аргументы**. Для примера рассмотрим команду ls Music, которая содержит аргумент Music

Способ 2 — это **опции**, еще их иногда называют флагами. Например, команда ls -a содержит в себе опцию -a

## **Опции**

Поговорим подробнее об опциях. Они всегда начинаются с одного или двух дефисов. Одна из часто используемых опций для просмотра списка файлов — -l. Она выводит дополнительную информацию по каждому файлу:

ls -l

total 0

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Desktop

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Documents

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Downloads

drwx------+ 26 Guest \_guest 832 Nov 21 2017 Library

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Movies

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Music

drwx------+ 3 Guest \_guest 96 Nov 21 2017 Pictures

drwxr-xr-x+ 4 Guest \_guest 128 Nov 21 2017 Public

Опции можно комбинировать. Представим, что мы хотим увидеть список всех файлов, включая скрытые, причем с подробным описанием. В таком случае нужно набрать команду ls -a -l. Можно объединить эти опции и записать ту же команду вот так:

* ls -al
* ls -la

При работе с опциями не забывайте ставить -. Без него вы получите команду ls la в которой la — это аргумент, а не опция. В таком случае командная оболочка покажет содержимое директории la.

Еще мы можем использовать опции и аргументы одновременно, хотя все зависит от программы. В случае с ls можно использовать одновременно и то, и другое. Чтобы просмотреть полное содержимое директории Music с информацией о каждом файле, можно набрать команду ls -la Music:

ls -la Music

total 0

drwx------+ 4 Guest \_guest 128 Nov 21 2017 .

drwxr-xr-x+ 89 Guest \_guest 2848 Aug 24 14:06 ..

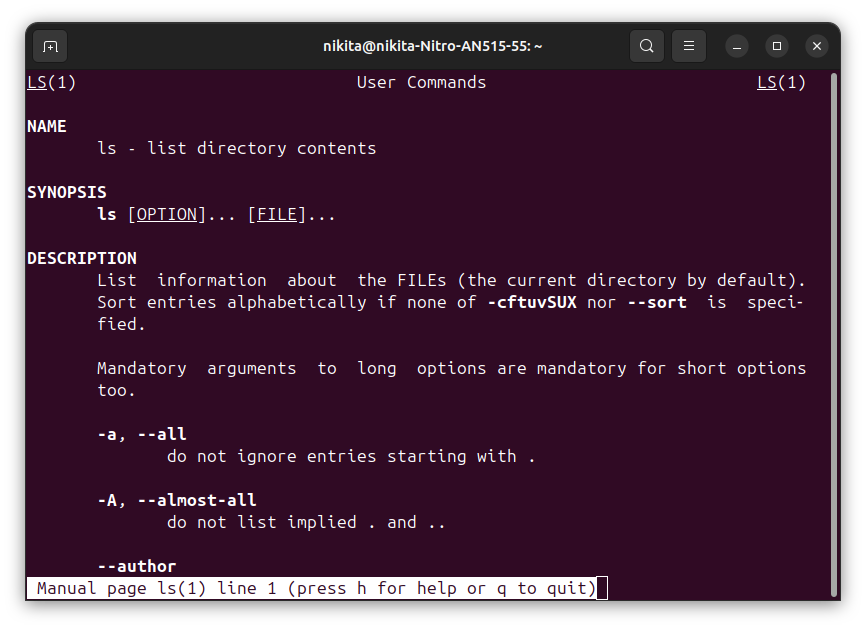
-rw-r--r-- 1 Guest \_guest 0 Nov 21 2017 .localized

drwxr-xr-x 9 Guest \_guest 288 Aug 26 17:25 iTunes

Иногда сложно понять подобные записи: -lupa. Не совсем понятно, что это:

* Одна опция lupa
* Четыре опции l, u, p и a, объединенные в одну цепочку

В таких ситуациях нужно смотреть документацию соответствующей программы. Это можно сделать с помощью команды man (сокращение от manual). Достаточно набрать man <имя команды> — и мы попадем в режим чтения документации.



Перемещаться внутри мануала можно так:

* Промотать вперед — f (forward)
* Промотать назад — b (backward)
* Выход из режима просмотра — q (quit)

## **Варианты опций**

У большинства утилит есть два варианта одной и той же опции — длинная и короткая версия. Например, в PHP есть -v и --version:

php -v

PHP 7.2.7 **(**cli**)** **(**built: Jun 22 2018 06:27:50**)** **(** NTS **)**

Copyright **(**c**)** 1997-2018 The PHP Group

Zend Engine v3.2.0, Copyright **(**c**)** 1998-2018 Zend Technologies

php --version

PHP 7.2.7 **(**cli**)** **(**built: Jun 22 2018 06:27:50**)** **(** NTS **)**

Copyright **(**c**)** 1997-2018 The PHP Group

Zend Engine v3.2.0, Copyright **(**c**)** 1998-2018 Zend Technologies

Длинные и короткие версии опций используются в разных ситуациях:

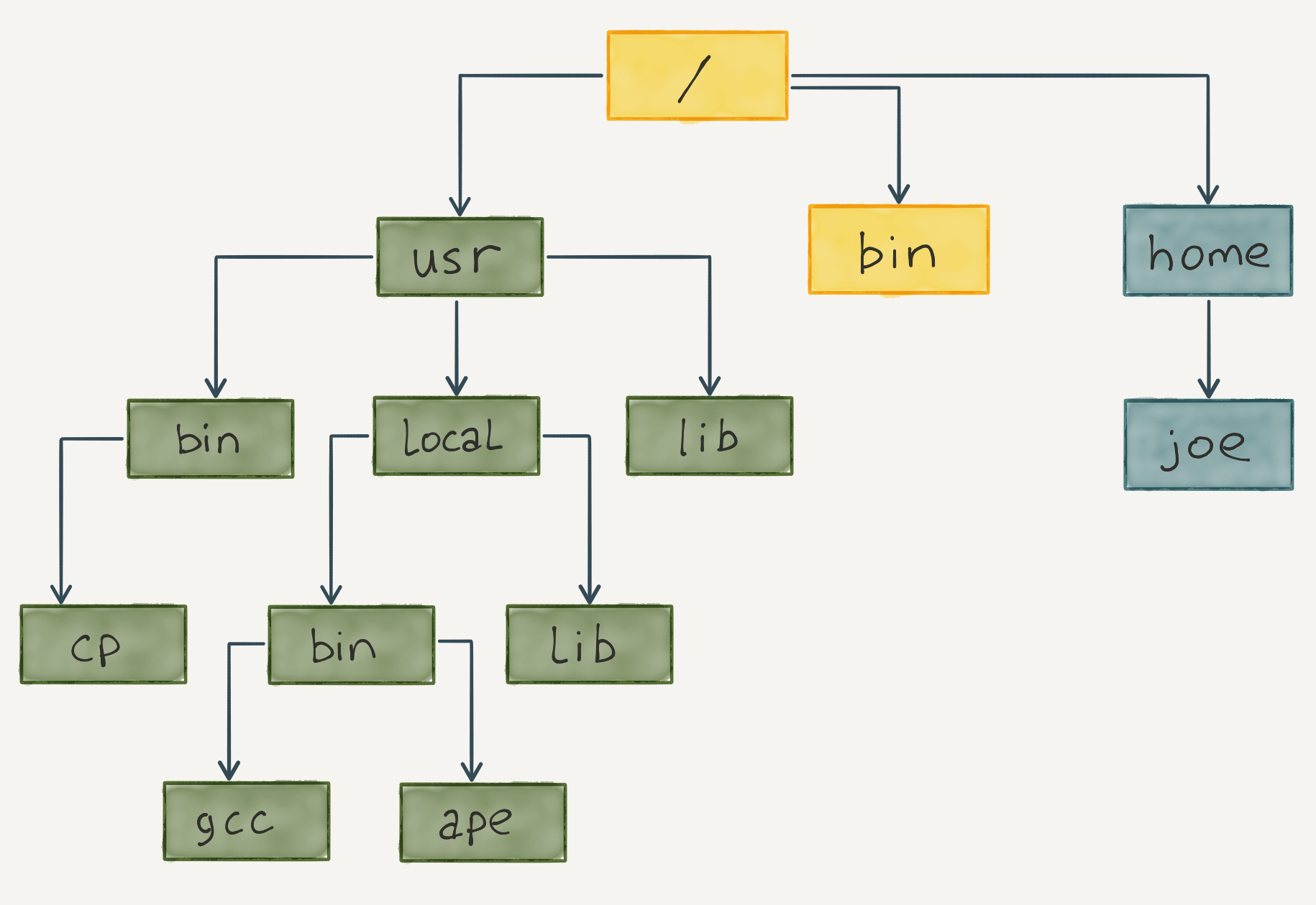
* Когда мы работаем в терминале, важно набирать быстро — там удобны короткие опции
* Когда мы пишем скрипт из разных команд, важно писать понятно — лучше использовать длинные опции. Так с первого взгляда очевидно, что означает каждая опция

# **Файловая структура**

Начнем с базовых понятий:

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

Схематично такое дерево можно показать так:



## **Директории**

Понятие «папка» в \*nix-системах не используется. Папки здесь называются **директориями** или **каталогами**, хотя по существу эти термины означают одно и то же.

Информация о любом файле или директории доступна по команде stat (сокращение от *file system status*):

*# Не обращайте внимание на непонятные для вас данные*

*# Об их значении поговорим позже*

stat .bashrc

File: '.bashrc'

Size: 3771

Blocks: 8

IO Block: 4096 regular file

Device: ca01h/51713d

Inode: 259234

Links: 1

Access: **(**0644/-rw-r--r--**)**

Uid: **(** 1002/kirill.m**)**

Gid: **(** 1002/kirill.m**)**

Access: 2018-08-27 17:24:11.237498138 +0000

Modify: 2015-08-31 23:27:45.000000000 +0000

Change: 2018-01-22 08:13:27.611966864 +0000

Birth: -

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

Но в \*nix-системах ситуация другая — здесь регистр имеет значение. Файлы index.html, Index.html, INDEX.HTML и index.HTML — это разные файлы. Всегда обращайте внимание на регистр, потому что ошибиться довольно легко.

Говорят, что в \*nix «все является файлом». На нижнем уровне так и есть. Директория — это специальный файл, который содержит список файлов. Любое подключаемое устройство становится файлом или директорией, если это накопитель.

## **Скрытые файлы**

Есть еще одна существенная разница между Windows и \*nix-системами. В обоих случаях скрытые файлы есть, но работают они по-разному:

* В Windows видимость файла — это его свойство, которое можно включить или выключить в настройках (при выключении файл станет скрытым)
* В \*nix-системах файл считается скрытым, если его имя начинается с точки. Чтобы скрыть файл, ничего не нужно делать в настройках — достаточно просто добавить точку в начале

Вывести все файлы, включая скрытые, можно командой ls -a:

ls -a

. .. .bash\_history .bash\_logout .bashrc .cache .profile .ssh

Обратите внимание на две особые директории:

* Обозначенные точкой (.) — это текущая директория
* Обозначенные двумя точками (..) — это директория верхнего уровня

Именно благодаря этой схеме работает команда cd .., которая перемещает нас на уровень выше.

# **Чтение файлов**

## **Команда cat**

Самый простой способ прочитать файл — команда cat:

cat .bash\_logout

*# ~/.bash\_logout: executed by bash(1) when login shell exits.*

*# when leaving the console clear the screen to increase privacy*

**if** **[** "$SHLVL" **=** 1 **]**; **then**

**[** -x /usr/bin/clear\_console **]** **&&** /usr/bin/clear\_console -q

**fi**

## **Команды head и tail**

Иногда нужно посмотреть только начало файла или его конец, в такой ситуации помогают команды head и tail. Они тоже принимают на вход путь до файла, только head показывает первые 10 строк, а tail — последние 10 строк файла.

Это поведение можно менять, задавая желаемое количество строк через опцию -n:

Посмотрим, как работает head:

head -n 2 .bashrc

*# ~/.bashrc: executed by bash(1) for non-login shells.*

*# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)*

Похожим образом работает tail:

tail -n 2 .bashrc

**fi**

**fi**

# **Grep**

Слово «грепать» входит в топ самых популярных терминов, используемых разработчиками. Оно происходит от одноименной консольной утилиты grep (сокращение от *global regular expression print*). Эта утилита выполняет поиск определенного текста по файлу или файлам.

Для разработчиков «грепать» — то же самое, что гуглить для активных пользователей интернета. Как правило, грепают файлы с исходным кодом или логи во время отладки:

man grep

SYNOPSIS

grep **[**OPTIONS] PATTERN **[**FILE...]

grep **[**OPTIONS] **[**-e PATTERN]... **[**-f FILE]... **[**FILE...]

Рассмотрим этот пример подробнее:

* PATTERN — это то, что мы хотим найти. Это может быть конкретная строчка или определенный шаблон с регулярными выражениями
* FILE — путь до файла, в котором нужно искать

Посмотрите на еще один пример:

*# Поиск всех строк в файле .bashrc, в которых встречается слово aliases*

grep aliases .bashrc

*# enable color support of ls and also add handy aliases*

*# some more ls aliases*

*# ~/.bash\_aliases, instead of adding them here directly.*

**if** **[** -f ~/.bash\_aliases **]**; **then**

. ~/.bash\_aliases

В примере выше утилита grep нашла пять строк. Найденные строчки выводятся на экран в том же порядке, в котором они встречаются в исходном файле.

В некоторых ситуациях нам нужно увидеть не только саму строку, но и текст вокруг нее. Количество выводимых соседних строк регулируется тремя опциями:

* Количество отображаемых строк до искомой строки — -B или --before-context
* Количество отображаемых строк после искомой — -A или --after-context
* Количество отображаемых строк до и после искомой строки — -C или --context

Изучим пример использования -C со значением 1. Это значит, что для каждой найденной строки будет выведена одна строка выше и одна строка ниже:

grep -C 1 aliases .bashrc

*# enable color support of ls and also add handy aliases*

**if** **[** -x /usr/bin/dircolors **]**; **then**

--

*# some more ls aliases*

alias ll**=**'ls -alF'

--

*# You may want to put all your additions into a separate file like*

*# ~/.bash\_aliases, instead of adding them here directly.*

*# See /usr/share/doc/bash-doc/examples in the bash-doc package.*

**if** **[** -f ~/.bash\_aliases **]**; **then**

. ~/.bash\_aliases

**fi**

Иногда мы не знаем, в каком файле находится то, что мы ищем. При этом мы можем знать директорию, в которой лежит этот файл.

В такой ситуации нужно сделать два изменения:

1. Добавить опцию -r — она указывает, что надо искать внутри директории. Обратите внимание, что поиск идет рекурсивно, то есть с включением всех поддиректорий
2. Указать путь до директории, а не файла

Попробуем применить утилиту grep с опцией -r:

grep -r bashrc .

./.profile: *# include .bashrc if it exists*

./.profile: **if** **[** -f "$HOME/.bashrc" **]**; **then**

./.profile: . "$HOME/.bashrc" ./.bash\_history:du -sh .bashrc

./.bash\_history:stat .bashrc

./.bash\_history:stat -h .bashrc

./.bash\_history:file .bashrc

./.bash\_history:stat .bashrc

./.bash\_history:cat .bashrc

./.bashrc:# ~/.bashrc: executed by bash**(**1**)** **for** non-login shells.

./.bashrc:# this, **if** it's already enabled in /etc/bash.bashrc and /etc/profile

./.bashrc:# sources /etc/bash.bashrc).

При таком поиске в выводе указывается файл, в котором была найдена строка. Если добавить опцию n, то дополнительно отобразится номер строки:

grep -rn bashrc .

./.profile:13: *# include .bashrc if it exists*

./.profile:14: **if** **[** -f "$HOME/.bashrc" **]**; **then**

./.profile:15: . "$HOME/.bashrc"

./.bash\_history:56:du -sh .bashrc

./.bash\_history:57:stat .bashrc

./.bash\_history:58:stat -h .bashrc

./.bash\_history:60:file .bashrc

./.bash\_history:61:stat .bashrc

./.bash\_history:63:cat .bashrc

./.bashrc:1:# ~/.bashrc: executed by bash**(**1**)** **for** non-login shells.

./.bashrc:109:# this, **if** it's already enabled in /etc/bash.bashrc and /etc/profile

./.bashrc:110:# sources /etc/bash.bashrc).

# **Редактирование файлов**

## **Nano**

Nano — очень простенький текстовый редактор, похожий на Блокнот в Windows, хотя по возможностям он богаче и даже способен подсвечивать содержимое файлов:



После запуска редактора внизу открывается панель с подсказками, какие горячие клавиши доступны для использования. Символ ^ означает клавишу Ctrl.

Например, нажав Ctrl + x, вы можете выйти из редактора. При выходе нужно уточнить, хотите ли вы сохранить изменения:

* Сохранить — y (yes)
* Не сохранять — n (no)