

Unix

Алексей Зубаков
на основе лекций Д. Халанского

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

Контакты

- tg: @neprav
- e-mail: aleks.zubakov [at] gmail.com
- группа в tg (пришлем инвайт)

Про коммуникацию

- на Вы; Алексей,
- глупых вопросов не бывает,
- преподаватель тоже человек,
- предполагайте добрые намерения.

О чём этот курс?

Курс про программы для программистов: командную строку, Docker, языковые инструменты, git.

Домашние задания будут, но несложные и с автоматической проверкой.

Пять недель занятий: четыре в сентябре, одно в октябре.

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

Минута истории



Источник: <https://www.flickr.com/photos/dancentury/3916969051/>

Рис. 1: Телетайп (teletype)

Когда-то люди взаимодействовали с компьютером, вбивая команды на клавиатуре и получая напечатанный ответ...

Минута истории



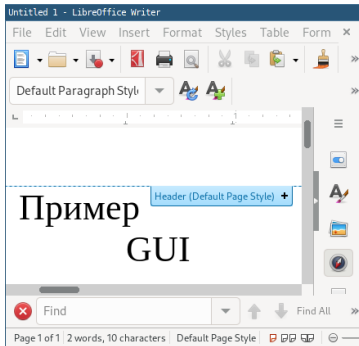
Источник: <https://www.flickr.com/photos/dancentury/3916969051/>

Рис. 1: Телетайп (teletype)

Когда-то люди взаимодействовали с компьютером, вбивая команды на клавиатуре и получая напечатанный ответ... Это когда-то — сегодня.

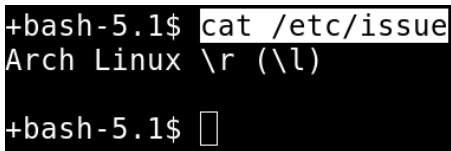
Взаимодействие с компьютером: GUI

GUI (Graphical User Interface, *графический пользовательский интерфейс*, «гуй») — интерфейс, способный использовать графические иконки, а не только текст. Обычно управляется мышью. Знаком всем.



Взаимодействие с компьютером: CLI

CLI (Command Line Interface, “командная строка”) — интерфейс, который мог бы работать и на телетайпе.

A screenshot of a terminal window with a black background and white text. The prompt is '+bash-5.1\$'. The command 'cat /etc/issue' has been entered and is highlighted with a white background. The output of the command is 'Arch Linux \r (\l)'. Below the output, the prompt '+bash-5.1\$' is followed by a white cursor box.

```
+bash-5.1$ cat /etc/issue
Arch Linux \r (\l)
+bash-5.1$
```

Рис. 2: CLI. Белым выделено то, что ввёл пользователь.

Взаимодействие с компьютером: TUI

TUI (Terminal User Interface) — интерфейс, который выглядит как графический, но оформлен текстом.

```
q:Quit d:Del u:Undel s:Save i:Mail r:Reply R:Group <F1>:Help
5065 + Aug 26 Ivan Oleynikov (7.6K)
5066 + Aug 27 Ivan Oleynikov (3.2K)
5067 + Aug 26 Ivan Oleynikov (1.7K)
5068 + Aug 23 Steam ( 33K) Fling to the Finish is now a
5069 + Aug 25 Maria Kuklina (0.4K) Рецент
5070 + Aug 26 Магазины полезны (9.6K) Ваш заказ отправлен.
5071 + Aug 26 GOG.COM ( 31K) ▲ Myst is OUT NOW ▲
5072 + Aug 26 Госуслуги ( 27K) Выберите удобный избирательн

---NeoMutt: =general [Msgs:5072 515M]---(threads/date)-----(end)---
Key is not bound. Press '<F1>' for help.
```

Рис. 3: TUI

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

shell

shell (“оболочка”) — программа, являющаяся самым внешним слоем, через который пользователь взаимодействует с компьютером.

Например, Windows shell (1995-наши дни) — графическая оболочка, в которой есть, в частности, меню Пуск, рабочий стол и панель задач.



Рис. 4: Windows XP.

Unix shell

Unix shell (1970-наши дни) — текстовая программа для операционной системы Unix и подобных ей, предназначенная для запуска других программ. Unix shell может работать через телетайп. Unix shell можно найти на любой популярной операционной системе:

- Linux — Unix-подобная система, там есть Unix shell.
- MacOS — это наследник Unix.
- Windows, начиная с 10, предоставляет Unix shell через Windows
- Subsystem for Linux (WSL).

Unix shell полезно знать в следующих случаях:

- Вам нужно работать с Linux. Он есть повсюду: в микроэлектронике, в роутерах, на почти всех веб-серверах. В Linux редко можно обойтись без командной строки.
- Вам нужно делать что-то экзотическое на MacOS.
- Для автоматизации: во многих случаях Unix shell удобнее какого-нибудь Python.

Виды Unix shell

Графических оболочек для Windows довольно мало¹. Для Unix даже популярных реализаций shell довольно много (по убыванию популярности):

- bash
- zsh
- dash
- fish
- ksh

У них всех разные языки и разное поведение. Мы будем рассматривать не какой-то конкретный shell, а стандарт POSIX, который задаёт, какими свойствами должен обладать каждый корректный shell на Unix. Все перечисленные shell-ы, кроме fish, являются более-менее POSIX-совместимыми.

¹https://en.wikipedia.org/wiki/List_of_alternative_shells_for_Windows
Unix shell

Интерфейс Unix shell

Далее просто “shell”.

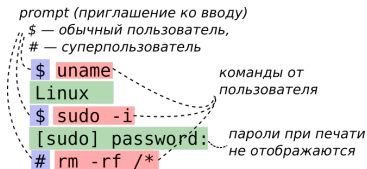


Рис. 5: Пример взаимодействия в Unix shell.

prompt настраивается, поэтому у всех выглядит по-разному;
\$ и # — это значения по умолчанию.

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

Код возврата

Код возврата (exit status) — число, которым программа сообщает, удачно ли завершилось исполнение.

По соглашению, 0 означает, что всё прошло хорошо; любое другое — что нет. У остальных чисел смысл разный для разных программ.

Пример для программы под названием ls:

Exit status:

- 0** if OK,
- 1** if minor problems (e.g., cannot access subdirectory),
- 2** if serious trouble (e.g., cannot access command-line argument).

Примеры программ

C, C++:

```
1  # include <stdlib.h>
2  # include <time.h>
3  int main ()
4  {
5      srand(time(NULL));
6      return rand() % 10; // <---
7  }
```

Python:

```
1  import random
2  exit(random.randint(0, 9)) # <----
```

Код возврата и shell

```
$ python -c 'exit(0)' && echo Success || echo Failure
Success
$ python -c 'exit(1)' && echo Success || echo Failure
Failure
$ if python -c "exit(0)"; then echo OK; else echo FAIL; fi
OK
$ python -c "exit(5)"
$ echo $?
5
```

- &&** Исполняет левую часть; если код возврата не 0, то не исполняет правую, а сразу возвращает этот код; иначе исполняет правую и возвращает её код.
- ||** Исполняет левую часть; если код возврата 0, то не исполняет правую, а сразу возвращает 0; иначе исполняет правую и возвращает её код.
- \$?** Код возврата последней исполненной команды.

Потоки ввода-вывода в shell

Все писали программу, которая считывает с консоли слово, а потом выводит в консоль “Hello, слово”. В Unix-подобных системах это реализовано через потоки ввода-вывода.

Есть три стандартные операции для ввода-вывода, реализованные через потоки:

- Прочитать данные с клавиатуры — через поток “stdin” (standard input, стандартный ввод);
- Вывести данные в консоль — через поток “stdout” (standard output, стандартный вывод);
- Вывести ошибки в консоль — через поток “stderr” (standard error, стандартный вывод ошибок).

Пример кода

Python:

```
1  import sys
2
3  print("Enter name: ", end="", file=sys.stderr) # stderr
4
5  name = input() # stdin
6
7  print("Hello, " + name ) # stdout
```

Потоки ввода-вывода в shell

В shell есть *перенаправления* (*redirections*) — конструкции, которые позволяют изменить указать какие-то другие места кроме экрана и клавиатуры, куда надо выводить результаты и ошибки и откуда можно читать данные.

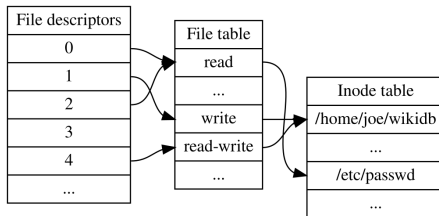
- `>f` означает, что вместо вывода результата на экран надо направить его в файл `f`. Файл `f` при этом полностью перетрётся. Например, `ls >f` запишет в `f` список файлов в текущей директории.
- `>>f` — почти то же самое, только `f` не перетрётся: новые данные будут записаны в его конец.
- `<f` означает, что надо читать данные не с клавиатуры, а из файла `f`.
- `2>f` означает, что ошибки надо выводить не на экран, а в файл `f`.
- `|` — очень мощная вещь, основа всего *nix.
`command1 | command2` означает, что вывод `command1` надо подать как ввод в `command2`.

Конвейеры

Набор разделённых | команд называется *конвейером* (pipeline).
Мы много раз с ними столкнёмся.

Файловые дескрипторы

Каждый процесс (экземпляр запущенной программы) обладает таблицей файловых дескрипторов. Файловый дескриптор открыт на чтение, запись или на чтение и запись сразу, а также связан с какой-то конкретной сущностью (например, файлом или сетевым соединением), с которой можно производить соответствующие операции.



Источник: https://en.wikipedia.org/wiki/File:File_table_and_inode_table.svg

Рис. 6: Таблица файловых дескрипторов процесса.

Примеры кода

C, C++:

```
1  # include <stdio.h>
2  # include <unistd.h>
3  int main()
4  {
5      const char string [] = "Hello, world!\n";
6      int result = write(14, string, sizeof(string));
7      if (result == sizeof(string)) return 0;
8      if (result >= 0) return 1;
9      perror("fileDes.c"); return 2;
10 }
```

Python:

```
1  fileDes = open(14, "w")
2  fileDes.write("Hello, world!\n")
```

Файловые дескрипторы и shell

Помимо `>`, `<` и иных перенаправлений, которые мы видели ранее, есть ещё такие (и некоторые другие):

- `x>f` Открыть поток `x` на запись в файл `f`. Например, `python fileDes.py 14>file`.
- `x<f` Открыть поток `x` на чтение из файла `f`.
- `x>>f` Открыть поток `x` на дописывание в файл `f`.
- `x>&y` Открыть поток `x` на запись и пересылать то, что в него попадает, туда, куда записывал бы поток `y`. Если `y` — не поток, а `-`, то поток `x` закрывается. Например, `python fileDes.py 14>&1`.

Таких команд можно писать несколько подряд. Например, если изначально открыты потоки 1 (на запись в `x`) и 2 (на запись в `y`), а поток 3 закрыт, то `ls 3>&2 2>&1 1>&3 3>&-` запустит `ls`, поменяв потоки 1 и 2 местами:

- Открывается поток 3, который записывает в `y`.
- Поток 2 переоткрывается и записывает в `x`.
- Поток 1 переоткрывается и записывает в `y`.
- Поток 3 закрывается.

Стандартные потоки ввода-вывода

По соглашению, все программы считают, что в их таблице файловых дескрипторов уже при запуске есть хотя бы такие три:

- 0 `stdin`
- 1 `stdout`
- 2 `stderr`

Из-за их особой роли shell вводит специальные сокращения, которые мы видели ранее.

Когда эмулятор терминала запускается, то перед тем, как открыть shell, он связывает эти файловые дескрипторы со своим механизмом вывода на экран и чтения с клавиатуры.

Аргументы командной строки

Основной способ настраивать поведение программы — через аргументы командной строки. Аргументы — набор строчек.

Примеры кода

C, C++

```
1  # include <stdio.h>
2  int main (int argc, char *argv[])
3  {
4      for (int i = 0; i < argc ; i++) {
5          puts(argv[i]);
6      }
7      return 0;
8  }
```

Python:

```
1  import sys
2  print(sys.argv)
```

Первый элемент этих массивов обычно не является аргументом: в нём записано само имя программы.

Аргументы командной строки в shell

Аргументы отделяются от команды и друг от друга пробельными символами. Например, `echo 1 2 3` запустит команду `echo` с аргументами 1, 2, 3.

Иногда хочется подать аргументы, в которых есть пробельные символы (или спецсимволы shell, например, `>`). Это можно сделать двумя способами:

- Заключить аргумент (или его проблемный кусок) в одинарные кавычки:

```
echo x'1 2\x$y#!'3
```

Тогда гарантируется, что аргумент будет подан как есть.

- Заключить аргумент (или кусок) в двойные кавычки:

```
echo "1\"2" 3
```

То, что в кавычках, будет считаться одним аргументом, но внутри кавычек какие-то символы (`$`, `\` и `'`) будут иметь особый смысл.

Флаги

Все аргументы командной строки — просто строки, но у аргументов, которые начинаются с одного или двух дефисов, по соглашению, есть особый смысл. Это флаги, и они используются для того, чтобы настраивать особенности работы программы.

Например, `cat file1 file2` передаст в стандартный вывод содержимое файлов `file1` и `file2` подряд без изменений. `cat -n file1 file2` также пронумерует строки в выводе.

Флаг может задавать особый смысл идущим за ним аргументам: `head 5 file1` передаст в стандартный вывод первые 10 строк файлов `5` и `file1`, но `head -n 5 file1` — пять строк файла `file1`. Тогда говорят, что у флага есть аргументы.

Длинные и короткие флаги

`cat -n file1` также можно записать как `cat --number file1`.

`head -n 10 file1` также можно записать как `head --lines=10 file1`.

Разница только в читаемости.

Короткие флаги обычно также можно записывать подряд: `rm -r -f *` можно записать как `rm -rf *`. С длинными флагами так нельзя.

Это всё соглашения, и есть очень много исключений.

Текущая директория

Каждый процесс привязан к какой-то директории — его *текущей директории* (working directory). Все операции, про которые не указано явно, к какой директории они относятся, будут исполняться в текущей директории.

При запуске программы из shell текущая директория наследуется.

Команда `pwd` печатает в стандартный вывод путь к своей текущей директории. Команда `cd D` делает D текущей директорией.

- Структура курса
- GUI vs CLI vs TUI
- shell
- Обмен данными с программами в *nix
 - Код возврата
 - Потоки ввода-вывода в shell
 - Файловые дескрипторы
 - Аргументы командной строки
 - Текущая директория
- Ориентирование

man

Unix разрабатывался, когда Интернета не было и в помине, и в его сообществе до сих пор сохранилась практика не обращаться к Stack Overflow, а искать ответ у себя на компьютере.

`man` — программа, которая выдаёт инструкции (manuals) к программам, функциям, файлам конфигурации и прочему. Например, `man ls` расскажет, что такое команда `ls`, а `man sudoers` — как оформлять файл конфигурации `/etc/sudoers`.

`apropos X` (или `man -k X`) ищет упоминания `X` в `man`-страницах.

Разделы man

Иногда бывает так, что разного рода вещи называются одинаково. `time` — это и функция в языке Си, и функция в языке `gawk`, и концепция операционной системы, и команда. Чтобы получить конкретную `man`-страницу, надо ещё указать секцию, в которой она находится. Например, секция 1 — команды; секция 3 — библиотечные функции. Например, `man 1 time` выдаст документацию именно на команду.

`whatis X` (или `man -f X`) перечисляет все разделы `man`, в которых есть `X`. `apropos` имеет флаг `-s S`, который означает, что искать нужно в секции `S`.

Когда хочется сообщить друзьям о какой-то `man`-странице, принято после её названия в скобках указывать номер секции. Например, `poll(3)` или `ls(1)`.

Case study: написание программы на Си

Вспомним программу, которая завершалась со случайным кодом возврата. Как её написать, если нет доступа в Интернет?

- `man man` утверждает, что функции Си перечислены в третьей секции: “3 – Library calls (functions within program libraries)”.
- Хотим, чтобы это работало не только на нашей ОС, но и на других. Добавим к названию секции суффикс `p` и поищем:

```
man -s 3 p -k random
```

- Пробуем тривиальную версию:

```
1 # include <stdio.h>
2 int main () { return rand (); }
```

Она не работает: нужно проинициализировать генератор чисел случайным зерном. Хорошее зерно — текущее время.

- `man -s 3 p -k time`

info-страницы

Помимо `man`, есть также система `info`. Она поддерживает гиперссылки между страницами и более мощный язык разметки; её предполагается использовать для подробной, развёрнутой документации с примерами и лирическими отступлениями. Она почти не используется.

Встроенная в программу документация

Иногда программа не предоставляет man-страницы. Скорее всего, у неё есть флаг `-h` (или `--help`, или `-? . . .`), который заставит её напечатать справку о себе.

Идущая с программой документация

В Linux (и, в меньшей мере, в MacOS) принято пользоваться пакетными менеджерами — программами, которые принимают команду “установи X” и сами выгружают X из репозитория и обеспечивают все условия для того, чтобы X работал. Эту систему напоминает App Store или Play Market.

Имеет смысл узнать, как данный пакетный менеджер перечисляет файлы, входящие в установленный пакет. Примеры таких команд — `rpm -ql` и `dpkg -l`. Часто с программой идёт документация отдельным файлом. Если нет, возможно, её можно установить отдельным пакетом.

which, type

`which` X позволяет узнать, где находится файл с программой под названием x. Это хорошо работает в тандеме с пакетным менеджером, у которого можно спросить, к какому пакету относится файл.

`type` X позволяет узнать, является ли вообще X отдельной программой (а не, например, функцией, написанной на языке shell).