Введение в командную строку UNIX подобных систем

ЛКШ.Зима, Январь 2025

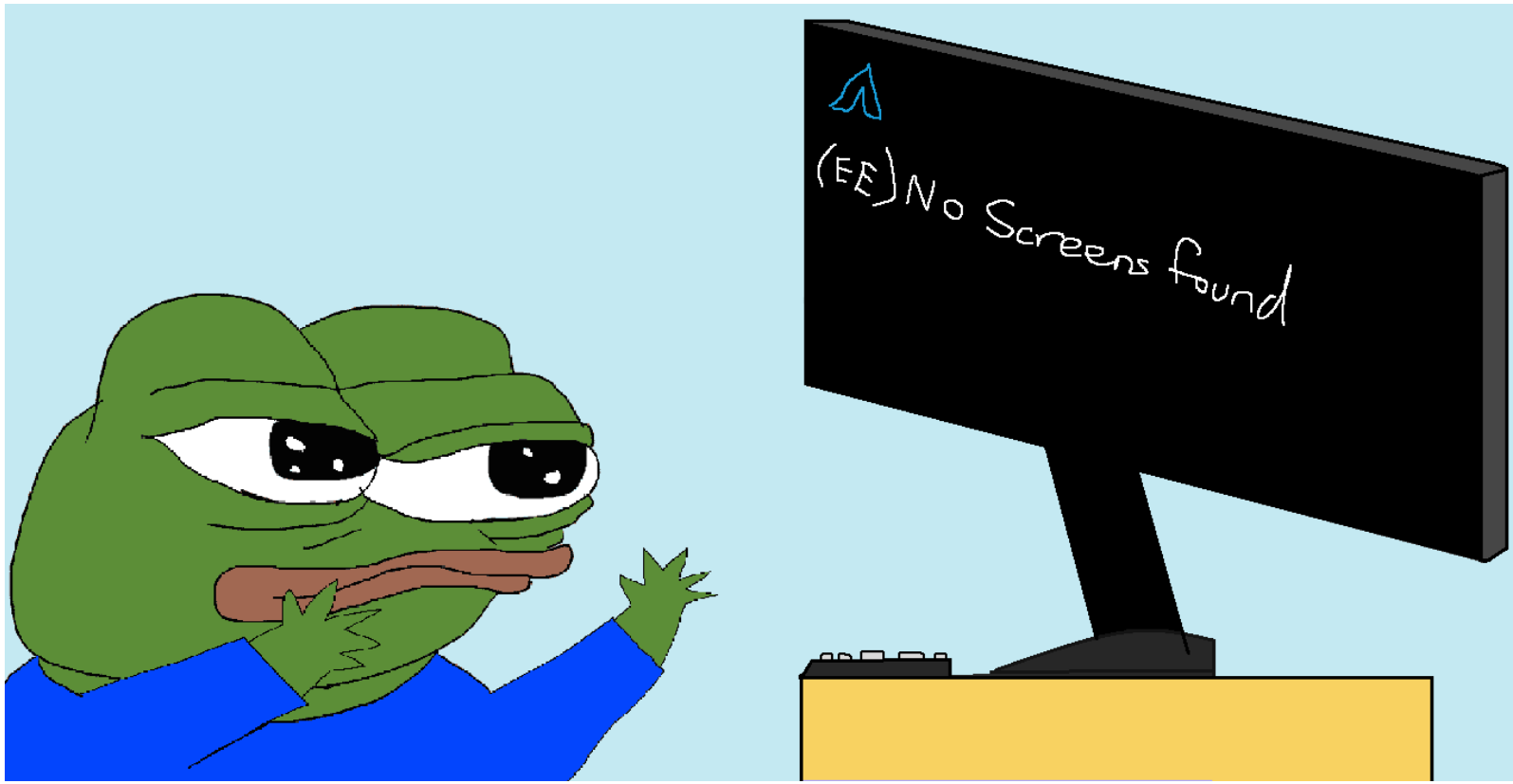
Оглавление

Введение
Архитектура
Постигаем основы bash
Процессы и их окружения
Управление вво-дом/выводом
Про файловую систему
Подводим итоги и пробуем!

На кого рассчитан спецкурс

- На людей, у которых уже был опыт/необходимость работы с командной строкой, но для которых это стало страшным сном.
- На тех, кто ещё не знаком, но очень интересно.

Немного о том, как это выглядит



Для чего мне нужна командная строка?

- Многие утилиты не имеют графического интерфейса или имеют ограничения по функционалу.

Для чего мне нужна командная строка?

- Многие утилиты не имеют графического интерфейса или имеют ограничения по функционалу.
- Утилиты командной строки удобнее сочетать друг с другом.

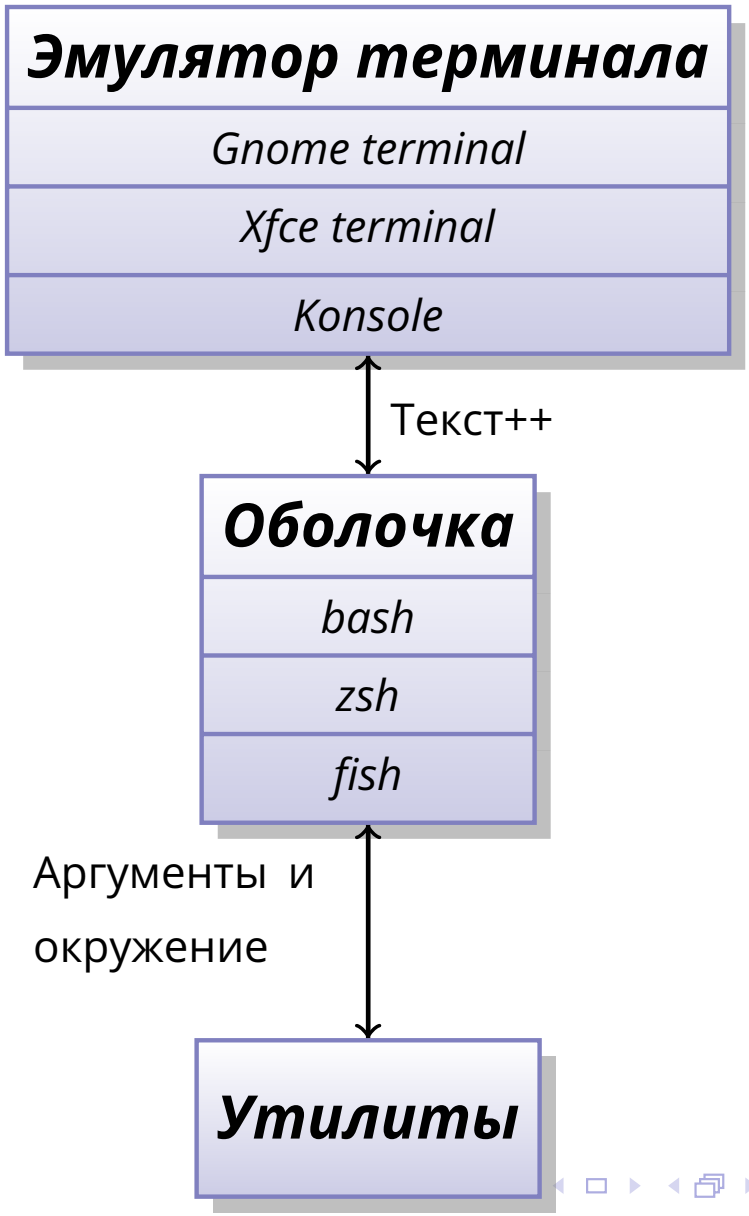
Для чего мне нужна командная строка?

- Многие утилиты не имеют графического интерфейса или имеют ограничения по функционалу.
- Утилиты командной строки удобнее сочетать друг с другом.
- Для работы командной строки нужно меньше ресурсов.

Для чего мне нужна командная строка?

- Многие утилиты не имеют графического интерфейса или имеют ограничения по функционалу.
- Утилиты командной строки удобнее сочетать друг с другом.
- Для работы командной строки нужно меньше ресурсов.
- Развитые возможности для автоматизации рутинных задач.

Раскладываем луч в спектр



Эмулятор терминала

Вопрос

А что, собственно, он эмулирует?

Эмулятор терминала

Вопрос

А что, собственно, он эмулирует?

Motorola MDT-9100

Эмулятор терминала

Ответ

Классический терминал внутри графического стека.

Эмулятор терминала

Ответ

Классический терминал внутри графического стека.

Грубо говоря, эмулятор общается с менеджером окон и графическим сервером выполняя просто функции типичные для любого окна. Достаточно важным является способность терминала отрисовывать шрифты со спецсимволами.
На самом деле эмулятор является программой переводящей текстовый поток данных и различные управляющие последовательности в графическое окно.

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.
- Возможности по определению контекста исполняемой команды.

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.
- Возможности по определению контекста исполняемой команды.
- Дополнительные возможности по редактированию вводимой команды.

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.
- Возможности по определению контекста исполняемой команды.
- Дополнительные возможности по редактированию вводимой команды.
- Различные подстановки перед исполнением команд

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.
- Возможности по определению контекста исполняемой команды.
- Дополнительные возможности по редактированию вводимой команды.
- Различные подстановки перед исполнением команд
- Всё???

Оболочка

Она является вашим средством обеспечения комфорта при работе с терминалом.

- Авто-дополнение путей файлов.
- Возможности по определению контекста исполняемой команды.
- Дополнительные возможности по редактированию вводимой команды.
- Различные подстановки перед исполнением команд
- Всё???
- Конечно нет, но об этом позже

Эти твои утилиты сейчас с нами в одной комнате?

Под утилитами в рамках этого спецкурса подразумеваются любые исполняемые файлы, работающие с текстовым вводом/выводом. Чаще всего это будут скомпилированные бинарные файлы.

Эти твои утилиты сейчас с нами в одной комнате?

Под утилитами в рамках этого спецкурса подразумеваются любые исполняемые файлы, работающие с текстовым вводом/выводом. Чаще всего это будут скомпилированные бинарные файлы.

Один из принципов UNIX Philosophy

```
Do One Thing And Do It Well, (DOTADIW).
```

Эти твои утилиты сейчас с нами в одной комнате?

Под утилитами в рамках этого спецкурса подразумеваются любые исполняемые файлы, работающие с текстовым вводом/выводом. Чаще всего это будут скомпилированные бинарные файлы.

Один из принципов UNIX Philosophy

```
Do One Thing And Do It Well, (DOTADIW).
```

Например,

- *sed* — утилита для потоковой обработки текста.
- *grep* — для поиска ключевых слов в файлах или директориях.

Примеры считывания аргументов и переменных окружения


```python
import os, sys
print("PWD:",  os.environ['PWD'])
print("ARGS:", sys.argv)
```

```cpp
#include <iostream>
int main(int argc, char* argv[]) {
  std::cout << "PWD: " << \
    std::getenv("PWD") << '\n';
  for (int i = 0; i < argc; ++i) {
    std::cout << argv[i] << ' ';
  }
  std::cout << '\n';
  return 0;
}
```

Откуда у меня команды? И что они делают

На самом деле каждое слово или символ введённый вами на клавиатуре является чем-то из списка. Для каждого определённого слова можно узнать введя команду:
`type <command>`.

- Обычные утилиты (cmd)

Откуда у меня команды? И что они делают

На самом деле каждое слово или символ введённый вами на клавиатуре является чем-то из списка. Для каждого определённого слова можно узнать введя команду:
`type <command>`.

- Обычные утилиты (cmd)
- Служебные команды оболочки (built-in): if, for...

Откуда у меня команды? И что они делают

На самом деле каждое слово или символ введённый вами на клавиатуре является чем-то из списка. Для каждого определённого слова можно узнать введя команду:
`type <command>`.

- Обычные утилиты (cmd)
- Служебные команды оболочки (built-in): if, for...
- Псевдонимы (alias)

Откуда у меня команды? И что они делают

На самом деле каждое слово или символ введённый вами на клавиатуре является чем-то из списка. Для каждого определённого слова можно узнать введя команду:
`type <command>`.

- Обычные утилиты (cmd)
- Служебные команды оболочки (built-in): if, for...
- Псевдонимы (alias)
- ...

Первые шаги. Навигация

Команда	Предназначение
`cd <path>`	Изменение рабочей директории
`ls`	Просмотр содержимого текущей директории
`pwd`	Просмотр пути до текущей директории

Первые шаги. Навигация

Команда	Предназначение
`cd <path>`	Изменение рабочей директории
`ls`	Просмотр содержимого текущей директории
`pwd`	Просмотр пути до текущей директории

Тип пути	Примерный вид
Абсолютный	`/path/to/file`
Относительный	`path/to/file` или `./path/to/file`

Первые шаги. Навигация

Команда	Предназначение
`cd <path>`	Изменение рабочей директории
`ls`	Просмотр содержимого текущей директории
`pwd`	Просмотр пути до текущей директории

Тип пути	Примерный вид
Абсолютный	`/path/to/file`
Относительный	`path/to/file` или `./path/to/file`

Отличие!

Абсолютные пути всегда начинаются с / и не зависят от того, каким пользователем и в какой директории они вызываются.

P.S.: директория является синонимом слова папка.

Укрощаем файлы

Команда	Функция
`mv <from path 1> <from path 2> ... <to path>`	Перемещает файлы.
`cp <from path 1> <from path 2> ... <to path>`	Копирует файлы.
`rm <del path 1> ...`	Удаляет файлы.

Укрощаем файлы

Команда	Функция
`mv <from path 1> <from path 2> ... <to path>`	Перемещает файлы.
`cp <from path 1> <from path 2> ... <to path>`	Копирует файлы.
`rm <del path 1> ...`	Удаляет файлы.

`cat <file>`	Выводит содержимое файла в stdout.
`touch <file>`	Создаёт файл или изменяет время доступа к нему.
`echo ...`	Выводит пост обработанные аргументы.

Укрощаем файлы

Команда	Функция
`mv <from path 1> <from path 2> ... <to path>`	Перемещает файлы.
`cp <from path 1> <from path 2> ... <to path>`	Копирует файлы.
`rm <del path 1> ...`	Удаляет файлы.

`cat <file>`	Выводит содержимое файла в stdout.
`touch <file>`	Создаёт файл или изменяет время доступа к нему.
`echo ...`	Выводит пост обработанные аргументы.

Обратите внимание, что `mv`, `rm` без аргументов работают только с файлами, чтобы они работали с папками необходимо передавать флаг запуска: `-r(recursive)`.

Джокерные символы(globbing)?

Спецсимволы в путях:

Конструкция	Функция
`..`	Обозначает родительскую директорию от префикса пути.
`~`	Используется только в начале пути. Обозначает домашнюю директорию пользователя.
`?`	Заменяет один любой символ в имени директории.
`*`	Заменяет любое число любых символов в имени директории.
`[<chars>]`	Заменяет один символов `<chars>`.

Забавный факт: ~`<user>` возвращает домашнюю директорию пользователя `user`.

Экранирование

Вопрос

А что нам делать, если мы хотим создать имя файла, содержащее пробелы?
Или имя файла, которое содержит звёздочку?

После исполнения команды `touch name with spaces.txt` у нас создадутся 3 файла: `name`, `with`, `spaces.txt`.
Для того, чтобы этого избежать, используется механизм экранирования: лишение некоторых символов служебной функции для оболочки и их интерпретация как обычный текст.

Экранирование

Есть три базовых вида экранирования:

1. `touch name\ with\ spaces`
2. `touch 'name with spaces'`
3. `touch "name with spaces"`

Почему экранирования двойными кавычками вообще удобно? Вспоминаем форматные строки в python или где-либо ещё: `echo ''Это переменная PATH: ${PATH}''`.

Пишем простейшее условие


```
if test -d test_path ; then
    echo "Directory"
else
    echo "Not directory"
fi
```

Пишем простейшее условие


```
if test -d test_path ; then
    echo "Directory"
else
    echo "Not directory"
fi
```

Тут:

- `test`, `echo` — built-in (встроенная в shell команды).
- `if`, `then`, `else`, `fi` — shell reserved word

Причём же тут коды возврата?

Простейшее условие, но сильно короче


```
command_1 && command_2
test -d test_path && echo "Directory"
```

Вторая команда испоняется, если код возврата первой ноль.

Простейшее условие, но сильно короче


```
command_1 && command_2
test -d test_path && echo "Directory"
```

Вторая команда испоняется, если код возврата первой ноль.

```
command_1 || command_2
test -d test_path || echo "Not directory"
```

Вторая команда испоняется, если код возврата первой НЕ ноль.

Простейшее условие, но сильно короче


```
command_1 && command_2
test -d test_path && echo "Directory"
```

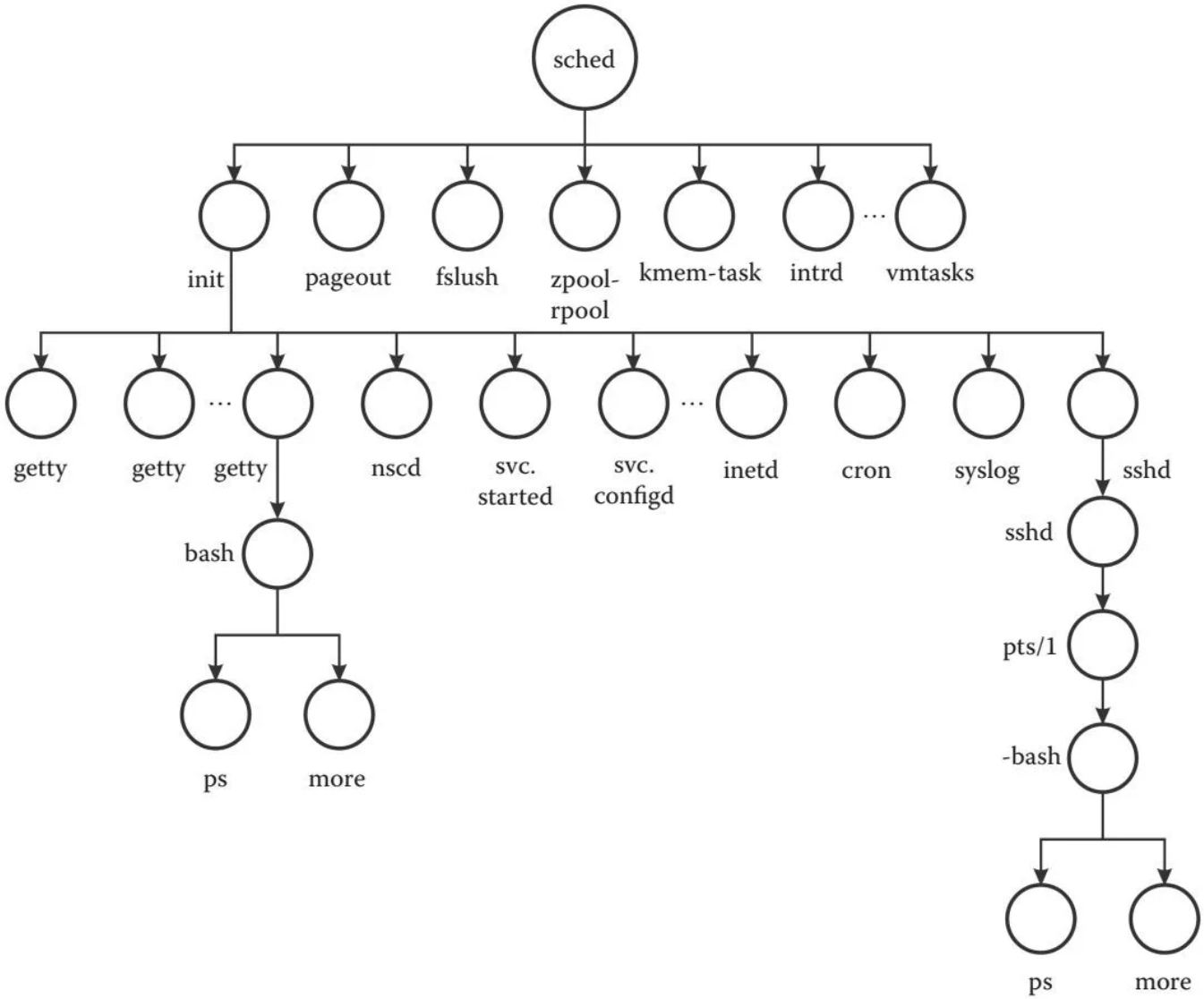
Вторая команда испоняется, если код возврата первой ноль.

```
command_1 || command_2
test -d test_path || echo "Not directory"
```

Вторая команда испоняется, если код возврата первой НЕ ноль.

```
test -d test_path && echo "Directory" || \
    echo "Not directory"
```

Иерархия процессов



Переменные окружения


```
PATH=/home/khaser/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/home/khaser/.dotnet/tools:/var/lib/flatpak/exp
erl:/usr/bin/core_perl:/opt/texlive/2021/bin/x86_64-linux
COLORTERM=truecolor
TERM=xterm-256color
LANG=en_US.utf8
QT_AUTO_SCREEN_SCALE_FACTOR=0
XDG_CURRENT_DESKTOP=i3
GRADLE_HOME=/usr/share/java/gradle
LC_IDENTIFICATION=ru_RU.UTF-8
WINDOWID=40019695
GTK3_MODULES=xapp-gtk3-module
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/khaser
USER=khaser
DESKTOP_SESSION=i3
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
HOME=/home/khaser
DOTNET_BUNDLE_EXTRACT_BASE_DIR=/home/khaser/.cache/dotnet_bundle_extract
LC_MEASUREMENT=ru_RU.UTF-8
XDG_VTNR=7
XDG_SEAT=seat0
I3SOCK=/run/user/1000/i3/ipc-socket.1023
LC_NUMERIC=ru_RU.UTF-8
GTK_MODULES=canberra-gtk-module
XDG_DATA_DIRS=/home/khaser/.local/share/flatpak/exports/share:/var/lib/flatpak/exports/share:/usr/local/share:/usr/share
VTE_VERSION=6800
XDG_SESSION_DESKTOP=i3
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LC_TIME=ru_RU.UTF-8
MAIL=/var/spool/mail/khaser
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
LOGNAME=khaser
LC_PAPER=ru_RU.UTF-8
DEBUGINFOD_URLS=https://debuginfod.archlinux.org
XDG_RUNTIME_DIR=/run/user/1000
DOTNET_ROOT=/usr/share/dotnet
SHELL=/bin/zsh
XDG_SESSION_TYPE=x11
LC_MONETARY=ru_RU.UTF-8
GTK2_RC_FILES=/home/khaser/.gtkrc-2.0
LC_TELEPHONE=ru_RU.UTF-8
EDITOR=vim
```

Export переменных окружения


```
 ~ $ MY_VAR=some
 ~ $ echo $MY_VAR
some
 ~ $ python print_var.py
Traceback (most recent call last):
  File "/home/khaser/print_var.py", line 3, in <module>
    print(os.environ["MY_VAR"])
          ~~~~~~~~~~~~~~~~~~~~
  File "<frozen os>", line 679, in __getitem__
KeyError: 'MY_VAR'
 ~ $ export MY_VAR
 ~ $ ./print_var.py
some
```

Ввод-вывод

Всего в UNIX-like системах всегда 3 стандартных потока ввода/вывода, а именно: **stdin stdout stderr**. По умолчанию, весь вывод, который вы видите в терминале от работающего процесса это **stdout** ∪ **stderr**. Очень часто приходится перенаправлять ввод-вывод, и для этого существует очень простой синтаксис.

Угадайка!

Подсказка:

id	stream
0	stdin
1	stdout
2	stderr

1	`./my_program < in_file`
2	`./my_program > out_file`
3	`./my_program >> out_file`
4	`./my_program &> out_file`
5	`./my_program 2>&1 > out_file`
6	`./my_program 2>/dev/null 1>&2`
7	`./my_program 1>&2 2>/dev/null`

Трубопровод (PIPEs)

Очень часто хочется просто отправить вывод первой команды на вход второй.

```
cat "part 1.txt" "part 2.txt" "part 3.txt" \
    > "full.txt"
grep "Пьер" "full.txt"
rm "full.txt"
```

Трубопровод (PIPEs)

Очень часто хочется просто отправить вывод первой команды на вход второй.

```
cat "part 1.txt" "part 2.txt" "part 3.txt" \
    > "full.txt"
grep "Пьер" "full.txt"
rm "full.txt"
```

```
cat "part 1.txt" "part 2.txt" "part 3.txt" | \
    grep "Пьер"
```

Все процессы соединённые PIPE, запускаются параллельно.

Минутка теории

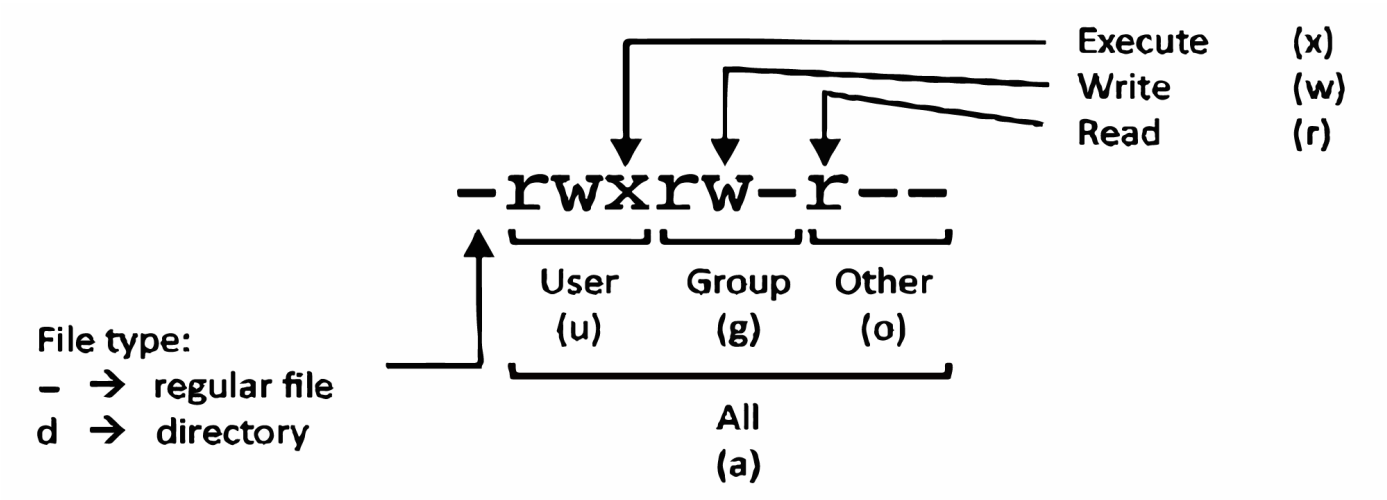

```
 …/lksh/BASH_course  stat main.tex
  File: main.tex
  Size: 16746           Blocks: 40         IO Block: 4096   regular file
Device: 8,7     Inode: 7239706     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  khaser)   Gid: ( 1000/  khaser)
Access: 2022-08-17 11:53:01.423603171 +0300
Modify: 2022-08-17 11:53:01.423603171 +0300
Change: 2022-08-17 11:53:01.423603171 +0300
 Birth: 2022-08-17 11:53:01.423603171 +0300
```

Давайте посмотрим на вывод команды `stat`, которая выводит подробную информацию о файле. Там много пока непонятных нам полей, давайте будем в них разбираться, начнём с прав доступа.

Знайте свои права!

Мы будем рассматривать только стандартную систему прав Unix.

У каждого файла или папки есть свой набор разрешений и все они выглядят примерно так:



Заметим, что эту строку можно воспринимать как три трибитных секции. Таким образом, набор прав, изображенных выше можно описать как 764.

ЧмОдим, чОвним

Для изменения прав доступа используется `chmod`.
Синтаксис очень простой: `chmod g+x file_1 file_2`.
Также частоиспользуемым используется вариант с числами: `chmod 775 file_1 file_2`.
Для изменения владельца и владеющей группы применяется следующие команды: `chown user:group file .`

~~Ярлыки~~ Ссылки

В отличии от всем привычных ярлыков в Windows, в Unix подобных системах есть целых два вида ярлыков. А именно есть Hardlink и Symlink. Symlink работает в точности как ярлык, но

~~Ярлыки~~ Ссылки

В отличии от всем привычных ярлыков в Windows, в Unix подобных системах есть целых два вида ярлыков. А именно есть Hardlink и Symlink. Symlink работает в точности как ярлык, но

Нюанс

Не надо писать в symlink относительные ссылки, без необходимости!!!

Ссылки пожёстче

Hardlink работает иначе.

1. Его можно создать только на файл.
2. Жёсткая ссылка ссылается не на исходный файл, а на тот же объект в файловой системе.
3. После создания жёсткой ссылки исходный файл и жёсткая ссылка на него являются неотличимыми, после удаления исходного файла, он всё её доступен по жёсткой ссылке.

Спасибо за внимание

Внешняя и внутренняя ссылки на хороший файл, где содержится вся необходимая начинающему информация.
Ссылка на сайт с практическими заданиями. Для них требуется уметь подключаться по SSH. На GNU/Linix, MacOS должно получиться подключиться через стандартную утилиту `ssh`.

```
ssh bandit0@bandit.labs.overthewire.org -p 2220
```

На windows необходимо устнановить putty, который можно скачать с оф. сайта.