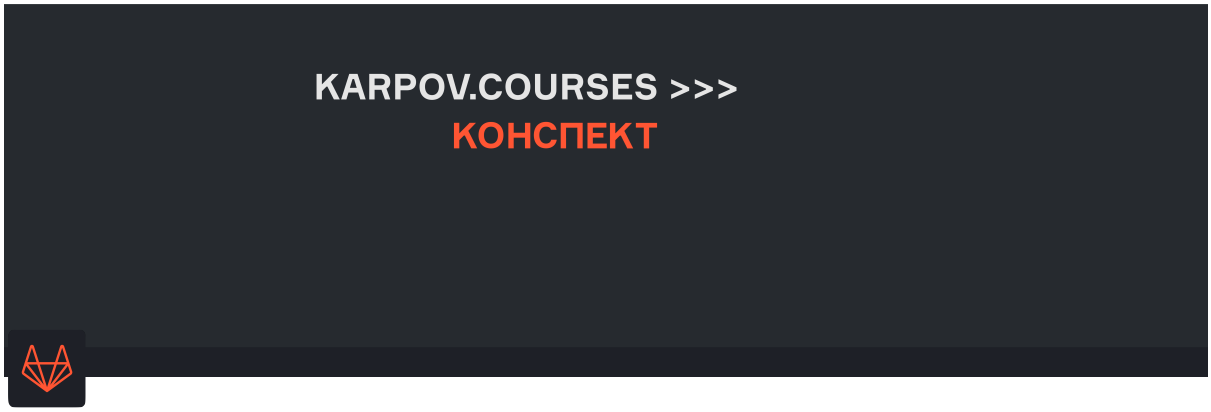

> Конспект > 1 урок > CLI

> Командная строка (терминал/консоль)

Командная строка это просто чудо! Она появилась несколько лет назад из своего предшественника-телетайпа и остаётся одним из самых эффективных способов работы с операционной системой (ОС). Причины её преимущества по сравнению с графическим интерфейсом (типа рабочего стола, папочек в проводнике) сходны с преимуществами языка программирования над какой-то программой

- широкие возможности автоматизации
- способность почти как угодно взаимодействовать с ОС
- лёгкая передача рецептов (набора действий для достижения какой-то цели)
- меньшее потребление ресурсов

Хотя есть и минусы

- высокий порог входа
- большая сила – большая ответственность (об этом в других частях конспекта)
- не настолько fancy как графон, но это субъективно и быстро привыкаешь

Что такое эта ваша консоль?

Консоль это по сути окно, где мы можем печатать команды, которые будут выполнены компьютером. Консоль широко используется для повседневной работы в системе (создание файликов/папочек, копирование/перенос информации, установка ПО, печатание в файлах, взаимодействие с сетью и так далее). То есть работа в ней аналогична работе на рабочем столе.

Как возможна работа в консоли? Просто для каждого обычного действия есть своя команда, например для копирования файла `data_maining.py` из папки `analytics` в домашнюю папку вместо щелчка правой кнопки по файлу, нажатия скопировать, перехода в домашнюю папку и нажатия вставить, мы просто пишем `cp analytics/data_maining.py ~`

И получаем такой же результат. Поначалу может показаться непривычным и неудобным – воспринимайте это как заклинания, а проблемы при использовании команд, как проблемы при изучении магии!

Вдобавок к набору команд (этих команд действительно много, они запоминаются постепенно), консоль предоставляет язык программирования bash. Он Тьюринг-полный, но своеобразный, и его используют в основном для продвинутой работы в консоли. В нашем курсе он не понадобится (для многих задач достаточно утилит, например git для контроля версий). Более того, далеко не на каждой работе нужно его продвинутое знание.

О том зачем вам вообще сдался этот терминал мы поговорим в следующей главе)

Больше информации

Обучательная статья по башу для тех, кому релевантно

> Зачем нужна консоль

Она нужна из-за ряда причин:

1. Перечисленные в предыдущем шаге преимущества
2. Устройства серверов

Так мы плавно подошли к серверам

Сервер

Сервер это просто особенно мощный компьютер, которым, как правило, пользуется больше одного человека одновременно. Типичные параметры сервера: несколько десятков-сотен ядер и сотни-тысячи гигабайт оперативной памяти.Понятное дело, сервера ставят не для того, чтобы поиграть в Cyberpunk2077 на максималках (для этого есть хорошие стационарные компьютеры): их назначение – computational-heavy задачи типа сборки генома, докинга (предсказание взаимодействия молекул), тренировки крупных моделей машинного обучения, прогоны симуляций, проверки математических гипотез, анализа астрофизических данных или любых других данных, которые больше оперативной памяти обычного компьютера. Помимо этого, сервера используется в любой команде, разрабатывающей один продукт, даже если каждому из них по отдельности не нужны большие вычислительные мощности.

Все суперкомпьютеры (или почти все) это сервера или кластеры (объединение компьютеров в одну вычислительную систему). Исходно они создавались для того, чтобы можно было шарить и эффективно распределять ресурсы. Поэтому там нет графического интерфейса (он потребляет ресурсы, которые можно потратить на вычисления).

Доминирующая часть ОС на серверах это UNIX, в частности Linux. Windows проиграла этот рынок из-за отсутствия ОС на сервера и качества своего софта, и сейчас пытается делать какие-то решения, но это ничтожная доля, которой можно пренебречь. К слову, на Windows есть терминал (cmd), но он "авторский" – на нём несовместимый с линуксом и гораздо менее богатый набор команд.

Таким образом, знание Linux (как преемника UNIX) весьма полезно, потому что именно он используется на серверах.

Больше информации

> Как открыть консоль

Локально

На линуксе/маке можно открыть терминал так: нажмите Ctrl + Alt + T, либо поищите среди своих программ Terminal/Konsole или что-то подобное.

На Windows есть урезанный аналог – введите cmd в поиске. Но рассмотренные нами команды не применяются в ней! Вам может помочь cygwin

На jupyterhub'e

Нажмите New справа сверху и выберите там Terminal. Всё готово)

Внешний вид

Представляет чёрное полотно, где отображаются ваши команды и аутпут в ответ на них.

Вы видите промпт, где слева идёт

`название пользователя@название сервера:нынешняя директория$`

После этого названия стоит курсор, где вы пишите команду, например

```
jupyter-a.ilin@lab:~$ my_command
```

Обзор терминалов

> Использованные на уроке базовые команды (и немного неиспользованных)

pwd

Сокращение от print working directory. Делает то, чем назван) Не требует дополнительных указаний

```
jupyter-a.ilin@lab:~$ pwd
/home/jupyter-a.ilin
```

ls

Сокращение от list – выводит файлы и директории в указанной папке. Если не указать папку, то выведет всё в директории, где вы находитесь

```
jupyter-a.ilin@lab:~$ ls
analytics  introduction_python  shared  test
```

```
jupyter-a.ilin@lab:~$ ls analytics
data_maining.py  data_preprocessing  README.md
```

Также к большинству команд можно добавлять опции, меняющие их поведение. Например -lah выведет прямо всё содержимое папки с подробной информацией в колонку с human-readable размером файлов

```
jupyter-a.ilin@lab:~$ ls -lah
total 72K
drwxrwx--- 12 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 19 00:14 .
drwxrwxr-x 67 root           root           4.0K Jun 15 20:42 ..
drwxr-xr-x  4 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 19 00:08 analytics
-rwxrwx---  1 jupyter-a.ilin jupyter-a.ilin   75 May 21 14:06 .bash_history
-rwxrwx---  1 jupyter-a.ilin jupyter-a.ilin  220 Apr  4  2018 .bash_logout
-rwxrwx---  1 jupyter-a.ilin jupyter-a.ilin 3.7K Apr  4  2018 .bashrc
drwxrwx---  6 jupyter-a.ilin jupyter-a.ilin 4.0K May 23 13:09 .cache
-rwxrwx---  1 jupyter-a.ilin jupyter-a.ilin    0 Jan 12 05:15 .cloud-locale-test.skip
drwxrwx---  3 jupyter-a.ilin jupyter-a.ilin 4.0K May 22 09:32 .config
-rw-r--r--  1 jupyter-a.ilin jupyter-a.ilin   53 Jun 18 19:08 .gitconfig
drwxrwx---  5 jupyter-a.ilin jupyter-a.ilin 4.0K May 15 16:12 introduction_python
drwxrwx---  2 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 10 15:29 .ipynb_checkpoints
drwxrwx---  5 jupyter-a.ilin jupyter-a.ilin 4.0K May 15 15:59 .ipython
drwxrwx---  4 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 13 02:23 .jupyter
drwxrwx---  3 jupyter-a.ilin jupyter-a.ilin 4.0K May 11 17:49 .local
-rwxrwx---  1 jupyter-a.ilin jupyter-a.ilin  807 Apr  4  2018 .profile
-rw-------  1 jupyter-a.ilin jupyter-a.ilin   16 Jun 18 18:53 .python_history
lrwxrwxrwx  1 jupyter-a.ilin jupyter-a.ilin   16 May  5 15:16 shared -> /srv/data/shared
drwx------  2 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 18 18:42 .ssh
drwxrwx---  3 jupyter-a.ilin jupyter-a.ilin 4.0K Jun 18 13:39 test
```

cat

Сокращение от concatenate – выдаёт содержимое файла (файлов)

```
jupyter-a.ilin@lab:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDq8g0AQEhSHjKDrgjRE+gKZxPJp2p47QWuBRWUwvDzZahg1gGgQdcV4UW4KZ15Q/O5OW5uvyNw2ylOvpJw8Cfz1D3
```

cd

Означает change directory – позволяет перемещаться по папкам. Аналогичен щёлканью по папкам) После выполнения меняется папка, где вы находитесь

```
jupyter-a.ilin@lab:~$ cd analytics/
jupyter-a.ilin@lab:~/analytics$
```

При использовании без аргумента перемещает в домашнюю директорию

mkdir

Сокращение от make directory – то есть создает директории) Прямо как щёлкнуть правой кнопкой, создать папку, ввести название

```
jupyter-a.ilin@lab:~$ mkdir my_new_dir
jupyter-a.ilin@lab:~$
```

Преимущество командной строки над графическим интерфейсом можно продемонстрировать в этом запуске - мы можем передать кучу папок сразу, и они все создадутся

```
jupyter-a.ilin@lab:~$ mkdir my_new_dir1 my_new_dir2 my_new_dir3
jupyter-a.ilin@lab:~$ ls
analytics  introduction_python  my_new_dir  my_new_dir1  my_new_dir2  my_new_dir3  shared  test
```

rm

Сокращение от remove – удаляет файлы или папки*Note: ПЕРВЫЕ ПОЛГОДА ПЕРЕД ПРИМЕНЕНИЕМ КАКОЙ-ТО КОМАНДЫ ПОГУГЛИТЕ ЧТО ОНА ДЕЛАЕТ*rm может быть очень опасна, будьте внимательны.

Без опций rm может удалять только файлы. Чтобы удалить папку добавьте `r` (означает recursive)

```
jupyter-a.ilin@lab:~$ rm -r my_new_dir my_new_dir1 my_new_dir2 my_new_dir3
jupyter-a.ilin@lab:~$ ls
analytics  introduction_python  shared  test
```

Остерегайтесь выполнения команды `rm -rf /` (удалит всё без запроса подтверждения). А также остерегайтесь тех, кто вам это посоветует

> Асимметричное шифрование

Шифрование вообще говоря интересная тема, хотя и ненужная в нормальном обществе. Не будем углубляться в его применение и особенности, отметим лишь, что одно из использований – связь компьютеров между собой (в том числе с сервером). Асимметричное шифрование одно из самых мощных средств для шифрования, которое спасёт вас от взлома большинством злоумышленников.

Note: если именно вас захотят взломать, то почти ничего не поможетNote2: если светить пароли или ключи, то тоже почти ничего не поможет, хотя тут справиться проще)

Для генерации ключей используется утилита `ssh-keygen`. Ниже приведена генерация ключа (в местах, где просят ввести пароль, введите какой-нибудь пароль для более надёжного соединения)

```
jupyter-a.ilin@lab:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jupyter-a.ilin/.ssh/id_rsa):
Created directory '/home/jupyter-a.ilin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jupyter-a.ilin/.ssh/id_rsa.
Your public key has been saved in /home/jupyter-a.ilin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:UB6TkDdNHtKbHvKhWfWulOmsDc0gmuuDQoSDjSBxRLM jupyter-a.ilin@lab.karpov.courses
The key's randomart image is:
+---[RSA 4096]----+
|.+=   .o==o      |
|o. o  .o+=o..    |
|++E   .....+ .   |
|= o    .. *   .  |
| o      SB.o +   |
|  .    oo.o++ .  |
| .   .o   .+o.   |
|  . . ..   o+    |
|   . .o.  ...    |
+----[SHA256]-----+
```

Теперь в папке .ssh хранится пара ключей – публичный (id_rsa.pub) и приватный (id_rsa)

```
jupyter-a.ilin@lab:~$ ls .ssh/
id_rsa  id_rsa.pub
```

Ни в коем случае не показывайте никому (в сети) свой приватный ключ. Публичный можно светить

Далее скопируйте содержимое публичного ключа, и введите его в гитлабе (подробнее в видео)

```
jupyter-a.ilin@lab:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDq8g0AQEhSHjKDrgjRE+gKZxPJp2p47QWuBRWUwvDzZahg1gGgQdcV4UW4KZ15Q/O5OW5uvyNw2ylOvpJw8Cfz1D3
```

Инструкция по генерации ключей

Ещё одна инструкция по генерации ключей

Асимметричное шифрование на шоколадках

Объяснение асимметричного шифрования

> git

git – это система контроля версий и одноимённая команда. Она используется, чтобы удобно иметь историю разработки и удобно разработку в команде.

История разработки

Помните как вы писали диплом? С большой вероятностью это происходило так:

1. вы приносите научнику новую версию диплома
2. он делает правки
3. вы переделываете диплом
4. повторяете пункт 1, пока всех всё не устроит

Отдельный привет длинным названиям файлов в ворде типа диплом_модифицированный_исправленный_переправленный =)

Может возникнуть ситуация, когда нужно вернуть то, что было пару версий назад и это трудно искать, или их может даже уже не быть(

С гитом такого не возникает – вся история модификаций ваших файлов отслеживается (если вы коммитете конечно, об этом позже). И вы можете достать содержимое отслеживаемых файлов разных версий, что положительно сказывается на сохранности информации)

Разработка в команде

Помимо истории версий, гит позволяет разобраться с работой в одних файлах несколькими персонами и получить работающую готовую версию.

Больше информации

> Как работает гит

В простом варианте гит работает следующим образом:

- в репозитории вы делаете изменения
- сохраняете состояние репозитория с изменениями
- коммитите его (заливаете изменения как готовые)
- работаете дальше

Это позволяет наладить вам историю изменений, и возвращаться к предыдущим коммитам.

Создание веток позволяет независимо от других работать в репозитории, а потом их объединять:

- создаём ветку
- работаем в ней
- объединяем

В результате мы можем работать в общем репозитории и объединять нашу работу с вкладом других участников команды)

Больше информации

Небольшой курс по гиту

Анимированные объяснения гита

gitbook

> Команды git'a

git clone

Клонирует репозиторий с сервера (сайта) к вам в папку

```
git clone link
```

После применения этой команды в 1-ый раз с нашим сервером вас спросят хотите ли вы с ним контактировать – напишите yes)

> Текстовый редактор vim

В консоли тоже можно редактировать файлы, и есть разные варианты, как это сделать. Один из самых олдскульных - с помощью редактора vim (вим). Его аналогами являются emacs и nano, которые считаются более дружелюбными к пользователю. Но у вима есть одно неоспоримое преимущество - вероятность, что он будет на машине (то есть будет установлен на системе, и будет доступен из терминала), стремится к 1. То есть, зная его хотя бы на самом базовом уровне, вы почти всегда сможете отредактировать что-то на компьютере или локальном сервере

Как открыть файл?

Напишите

```
vi filename
```

это откроет файл filename в редакторе вим. Написано vi, а не vim, так как vi это просто более новая версия vim'a

Как создать новый файл?

```
vi new_filename
```

откроет в виме пустой файл с именем new_filename, в котором будет сохранено то, что вы напишите

Редактирование файла

Находясь в виме нажмите `i` - это переведёт вас в режим записи, это будет отображено надписью снизу INSERT. После этого можно писать текст, который вам нужен

Выход из режима записи

Написав нужный текст, нажмите `Esc`, чтобы выйти из режима записи. Надпись INSERT снизу пропадёт

Сохранение файла

Для сохранения изменений нажмите

```
:w
```

и `Enter`

При этом нужно находится в обычном режиме (без надписи INSERT снизу)

Выход из вима

Находясь в обычном режиме, нажмите

```
:q
```

и `Enter`

Сохранение файла и выход из вима одновременно

Находясь в обычном режиме, нажмите

```
:wq
```

и `Enter`

> Скрипты

Почти всё это время мы работали с вами только в юпитерских ноутбуках. Это стандарт в мире анализа данных, но исходно их не было (и ими пользуются не все айтишники, работающие с питоном). Исходно были питоновские скрипты - это просто текстовые файлы с расширением `py`. Пример скрипта simulacrum.py

```
# This is a comment, next line is just print invocation
print('Hello there!')
```

В скриптах написан питоновский код (всё то, что мы писали в ячейках ноутбука) и их можно открывать для редактирования прямо в блокноте

Отличия питоновского скрипта от юпитерского ноутбука

- можно писать только код (нет возможности вставлять картинки или markdown)
- удобно редактировать прямо в блокноте
- нет разделения на ячейки (будет выполняться весь скрипт целиком)

Для запуска питоновского скрипта необходимо вызвать питон и передать ему скрипт (нечто похожее происходит и в юпитере под капотом)

Питон как консольная команда

В терминале мы можем запустить питоновский скрипт следующим образом

```
python simulacrum.py
```

Это запустит выполнение скрипта simulacrum.py в питоне. По сути аналогично тому как мы выполняли ячейку в юпитер ноутбуке, только в ячейке всё содержимое файла simulacrum.py

Также мы можем запустить питон в интерактивном режиме

```
python
```

Далее можно писать блоки кода, и после нажатия `Enter` питон сразу будет писать результат. Чтобы выйти из питона, напишите

`quit()`

и нажмите `Enter`

Либо нажмите `Ctrl + d`