Основы Computer Science:
git, bash и другое.

Агафонов Артём
email agafonov.ad@phystech.edu

1 Знакомство с Shell. Изучение базовых команд.

1.1 Работа с Shell

1.1.1 Текстовый интерфейс

При запуске терминала обычно мы видим следующее:

```
(base) a.agafonov@a-agafonov ~ %.
```

Давайте разберемся, какую информацию нам сообщает терминал:

- `a.agafonov` – имя пользователя;
- `a-agafonov` – имя машины;
- `~` – сокращение для домашней (home) директории (см. подраздел 1.2);
- `%` для ZSH или `$` для Bash показывает, что вы не root пользователь(см. подраздел 1.5);
- `(base)` – появляется только при установленной conda. Вероятно, она у вас установлена, если вы запускали jupyter notebook. `(base)` говорит, что сейчас используется conda энвайромент `base`.

  Базовые команды:

  - `conda env list` – список всех доступных энвайроментов;
  - `conda create -name nlp_project` – создание энвайромента с именем `nlp_project`
  - `conda activate nlp_project` – переключение на энвайромент `nlp_project`;
  - `conda install scikit-learn` – установка sklearn в активный энвайромент.

1.1.2 Выполнение команд

С помощью команды `date` в командной строке можно узнать дату:

```
(base) a.agafonov@a-agafonov cs22 % date
Thu Sep  8 01:04:28 +04 2022
```

Пример выполнения команды с аргументами

```
(base) a.agafonov@a-agafonov cs22 % echo hello
```

Команда `echo` выводит текстовую строку, указанную в качестве аргумента. Оболочка анализирует команду, разделяя ее пробелами, и запускает программу слева направо, начиная с первого слова и передавая каждое последующее слово в качестве аргумента. Есть несколько способов указать аргумент, содержащий пробелы: `"Hello, world"`, `'Hello, world'`, `Hello\world`.

1.2 Навигация и работа с файловой системой

1.2.1 Навигация по файловой системе

`ls` (list storage) – вывод содержимого директории (папки). Например, в данный момент мы находимся в директории `cs22` и хотим увидеть ее содержимое.

```
(base) a.agafonov@a-agafonov cs22 % ls
agenda.txt        lecture1        lecture2        lecture3        lecture4
```

Команда `pwd` (print working directory) показывает полный (абсолютный) путь к директории, в которой мы находимся.

```
(base) a.agafonov@a-agafonov cs22 % pwd
/Users/a.agafonov/Documents/teaching/made/cs22
```

Полным путем (absolute path) называется путь, начинающийся с /, любой другой называется относительным (relative path). В Linux и Mac / – корень (root) файловой системы. Таким образом, полный путь показывает по каким директориям надо пройти, чтобы попасть в необходимое место. Например, полный путь до папки `cs22` будет `/Users/a.agafonov/Documents/teaching/made/cs22`, а до файла `agenda.txt` – `/Users/a.agafonov/Documents/teaching/made/cs22/agenda.txt`. Относительный путь связан с директорией в которой мы находимся сейчас. Относительный путь до файла `agenda.txt` из директории `cs22` есть просто `agenda.txt`.
При указании путей можно использовать следующие символы:

- `.` – текущая директория;
- `..` – родительская директория.

Для смены директории используется команда `cd`. Примеры:

```
(base) a.agafonov@a-agafonov cs22 % ls
agenda.txt        lecture1        lecture2        lecture3        lecture4

(base) a.agafonov@a-agafonov cs22 % cd ../..

(base) a.agafonov@a-agafonov teaching % ls
made        mipt

(base) a.agafonov@a-agafonov teaching % pwd
/Users/a.agafonov/Documents/teaching

(base) a.agafonov@a-agafonov teaching % cd ./made/cs22/lecture1

(base) a.agafonov@a-agafonov lecture1 % ls
commands.txt        text.pdf        text.tex

(base) a.agafonov@a-agafonov lecture1 % cd /Users/a.agafonov/opt/anaconda3/bin

(base) a.agafonov@a-agafonov bin % ls
2to3
2to3-3.8
Assistant.app
...
```

Что если нам надо перейти в какую-то папку и вернуться обратно, надо ли запоминать путь до нее? Оказывается, что нет. Команда `pushd` позволяет изменить директорию (как и `cd`), а `popd` возвращает обратно.
Перейти в директорию, название которой содержит пробел, можно следующими способами

```
(base) a.agafonov@a-agafonov tmp % cd My\ Folder
(base) a.agafonov@a-agafonov tmp % cd 'My Folder'
(base) a.agafonov@a-agafonov tmp % cd "My Folder"
```

1.2.2 Права доступа к файлам и каталогам

Чтобы получить больше информации о файлах и дректориях можно использовать команду `ls` с флагом `-l`.

```
(base) a.agafonov@a-agafonov cs22 % ls -l
total 8
-rw-r--r--  1 a.agafonov  staff  987 Sep  8 00:22 agenda.txt
drwxr-xr-x  5 a.agafonov  staff  160 Sep  7 05:23 lecture1
drwxr-xr-x  2 a.agafonov  staff   64 Sep  7 05:03 lecture2
drwxr-xr-x  2 a.agafonov  staff   64 Sep  7 05:03 lecture3
drwxr-xr-x  2 a.agafonov  staff   64 Sep  7 05:03 lecture4
```

Какую информацию мы получили?

- Первый символ – признак директории. `d` (directory) говорит, что перед нами директория, `-` – файл.
- Последующие три символа – права доступа *владельца файла и каталога*. Первый символ – чтение (`r`), второй – запись (`w`), третий – исполнение (`x`). `rw-` указывает, что владелец может читать и писать в файл, но не исполнять его.
- Следующие три символа – права доступа для *членов группы владельца*. Например, для директории `r-x` означает, что можно читать (выводить содержимое) и выполнять поиск в этой директории.
- Последние три символа – права доступа для *всех остальных пользователей*.
- Дальше идут имя владельца файла, имя группы владельца, размер файла, дата и время создания, имя файла.

Для изменения прав доступа используют команду `chmod`.

1.2.3 Команды для работы с файлами и директориями

Другие команды для работы с файлами и директориями:

- `touch` – создание пустого файла;
- `cat` – просмотр файла;
- `cp` – копирование файла/директории;
- `mv` – перемещение и переименование файлов и директорий;
- `rm` – удаление файла/директории;
- `mkdir` – создание пуской директории.

Чтобы получить больше информации о команде, ее флагах и аргументах можно воспользоваться `man`. Например, `man touch`. Чтобы выйти из описания команды достаточно нажать `q`.

1.3 Как Shell исполняет команды?

Когда пользователь вводит команду, bash (или другая оболочка) ищет программу, соответсвующую команде в директориях, указанных в переменной окружения `PATH`.

```
(base) a.agafonov@a-agafonov cs22 % echo $PATH
/Users/a.agafonov/opt/anaconda3/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:

(base) a.agafonov@a-agafonov cs22 % which python
/Users/a.agafonov/opt/anaconda3/bin/python
```

```
(base) a.agafonov@a-agafonov cs22 % python --version
Python 3.8.5


(base) a.agafonov@a-agafonov cs22 % ls /usr/bin/python
/usr/bin/python

(base) a.agafonov@a-agafonov cs22 % /usr/bin/python --version
Python 2.7.16
```

Shell по очереди смотрит все пути в `PATH` и исполняет первый попавшийся файл с заданным названием. Из примера выше мы видим, что файл `python` есть в нескольких директориях, указанных в переменной `PATH`. Эти программы отличаются между собой. Чтобы узнать какой именно файл исполнялся, можно воспользоваться командой `which`.

1.4 Перенаправление ввода/вывода

С помощью перенаправления ввода/вывода мы можем перенаправить вывод одной программы в файл или на стандартный ввод другой программы.

```
(base) a.agafonov@a-agafonov lecture1 % echo hello > example.txt
(base) a.agafonov@a-agafonov lecture1 % cat example.txt
hello
```

Символ `>` означает, что вывод программы `echo hello` будет записан в файл `example.txt`. Причем файл будет переписан! Как нам добавить запись в файл? Ответом послужит следующий пример:

```
(base) a.agafonov@a-agafonov lecture1 % cat example.txt
hello
(base) a.agafonov@a-agafonov lecture1 % echo world >> example.txt
(base) a.agafonov@a-agafonov lecture1 % cat example.txt
hello
world
```

Командой `cat note.txt` файл читается. Это равносильно `cat < note.txt`, где знак меньше `<` обозначает перенаправление ввода.

Однако равносильно по результату – не значит равносильно по механизму реализации. При использовании знака именно командная оболочка, в данном случае Bash, занимается перенаправлением данных из файла. В то время как программа `cat` (или любая другая) работает в обычном режиме, "думая"что данные она получает со стандартного ввода.

`|` оператор позволяет «связывать» программы так, чтобы выходные данные одной были вводными данными другой. Пример:

```
(base) a.agafonov@a-agafonov lecture1 % cat example.txt| wc -l
       2
```

1.5 Суперпользователь

В Unix-подобных системах пользователь root является особенным. Он может читать, менять и исполнять любой файл в системе. Будучи пользователем root очень легко что-нибудь сломать в системе. Поэтому для выполнения программ, требующих прав root, намного рациональнее использовать программы `sudo` (do as super-user) и `su` (super-user). Тем не менее, перед исполнением программы от имени root пользователя важно убедиться, что вы действительно хотите это сделать! Так выполнение под админом команды `rm -rf /` ведет к удалению всех файлов. Пример использования `sudo`:

```
(base) a.agafonov@a-agafonov lecture1 % shutdown
shutdown: NOT super-user
(base) a.agafonov@a-agafonov lecture1 % sudo shutdown
Password:
```

1.6 Некоторые команды

Запрос с помощью `curl` и обработка текста `grep`.

```
(base) a.agafonov@a-agafonov ~ % curl --head --silent google.com
HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Wed, 14 Sep 2022 05:53:05 GMT
Expires: Fri, 14 Oct 2022 05:53:05 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
(base) a.agafonov@a-agafonov ~ % curl --head --silent google.com | grep --ignore-case content-type
Content-Type: text/html; charset=UTF-8
(base) a.agafonov@a-agafonov ~ % curl --head --silent google.com | grep --ignore-case content-type |
UTF-8
```

Поменять на MacOS формат скриншота:

```
defaults write com.apple.screencapture type JPG
```

Вывести календарь

```
(base) a.agafonov@a-agafonov cs22 % ncal
    September 2022
Mo     5 12 19 26
Tu     6 13 20 27
We     7 14 21 28
Th  1  8 15 22 29
Fr  2  9 16 23 30
Sa  3 10 17 24
Su  4 11 18 25
```

Попросить MacOS произности Hello World!

```
say Hello World!
```

Команда `convert` представляет из себя простую и эффективную работу с картинками, гифками и видео. Рассмотрим последовательность команд:

```
(base) a.agafonov@a-agafonov parrot % ls
parrot.gif        source
(base) a.agafonov@a-agafonov parrot % convert parrot.gif ./source/parrot.png
(base) a.agafonov@a-agafonov parrot % ls source
parrot-0.png        parrot-2.png        parrot-4.png        parrot-6.png        parrot-8.png
parrot-1.png        parrot-3.png        parrot-5.png        parrot-7.png        parrot-9.png
```

`convert` преобразовал gif в набор исходных картинок. Давайте соберем их обратно

```
(base) a.agafonov@a-agafonov parrot % convert -dispose previous
-delay 100 ./source/*.png parrot100.gif
(base) a.agafonov@a-agafonov parrot % open parrot100.gif
```

1.7 Упражнения

1. Что делает команда `cat < hello.txt > hello2.txt`?
2. Создайте два текстовых файла с какими-либо данными. С помощью программы `cat` объедините данные этих файлов в третьем файле.
3. - Создайте пустой файл `filename`.
   - Запишите в созданный ранее файл построчно следующее:

     ```
     #!/bin/sh
     curl --head --silent https://missing.csail.mit.edu
     ```
   - Попробуйте запустить скрипт введя команду `./filename`. Почему он не выполнился?
   - Запустите скрипт как `sh filename`. Получилось? Почему?
   - Используя `chmod` сделайте возможным запуск скрипта при вводе `./filename`, без необходимости ввода `sh semester`. Как ваша оболочка узнала, что файл может быть интерпретирован sh?
4. Используя `|` и `>`, запишите дату "last modified" из вывода скрипта из предыдущего задания в файл с именем `last-modified.txt`.
5. Напишите команду `ls`, которая выводит список файлов в следующем виде:
   - Отображает все файлы, включая скрытые;
   - Размер файлов представлен в удобном для человека формате (пример: 454M вместо 454279954);
   - Порядок файлов - по дате изменения (от более новых - к старым);
   - Вывод раскрашен.

1.8 Источники

[1] Колисниченко Д. Н. Командная строка Linux и автоматизация рутинных задач. – БХВ-Петербург, 2012.

[2] The Missing Semester of Your CS Education - Lecture 1 - MIT, 2020.

[3] Mendel Cooper. Advanced Bash-Scripting Guide.

[4] Шапошникова С. Введение в Linux и Bash. Курс.

2 Shell инструменты и скрипты

В этой теме мы в основном сфокусирумся на умении комбинировать команды в скрипт. Shell скрипт – просто текстовый файл с набором команд.

2.1 Переменные

Переменные – основа любого языка программирования, они есть и в bash. Рассмотрим следующий пример:

```
(base) a.agafonov@a-agafonov lecture2 % day=01
(base) a.agafonov@a-agafonov lecture2 % echo day
day
(base) a.agafonov@a-agafonov lecture2 % echo $day
01
(base) a.agafonov@a-agafonov lecture2 % echo ${day}
01
(base) a.agafonov@a-agafonov lecture2 % month=09
(base) a.agafonov@a-agafonov lecture2 % echo ${day}.${month}.2022
01.09.2022
```

Важно отличать имя переменной (`variable`) и ссылку на её значение (`$variable`). Написание `${variable}` по сути эквивалентно `$variable`, но оно часто используется когда переменную надо вызвать как часть строки.

Строки в bash тажке могут быть определены через двойные кавычки " или одинарные '. Какие между ними есть различия поймем на следующем примере

```
(base) a.agafonov@a-agafonov lecture2 % var=abc
(base) a.agafonov@a-agafonov lecture2 % echo $var
abc
(base) a.agafonov@a-agafonov lecture2 % echo '$var'
$var
(base) a.agafonov@a-agafonov lecture2 % echo "$var"
abc
```

Получается, двойные кавычки не меняют механизм подстановки значения в переменную, а одинарное заставляют интерпретатор воспринимать ссылку как набор символов.

2.1.1 Типизация

Bash не производит разделения переменных по типам. По сути, переменные Bash являются строковыми переменными, но, в зависимости от контекста, Bash допускает целочисленную арифметику с переменными. Определяющим фактором здесь служит содержимое переменных.

```
(base) a.agafonov@a-agafonov lecture2 % a=10
(base) a.agafonov@a-agafonov lecture2 % let a+=1
(base) a.agafonov@a-agafonov lecture2 % echo $a
11
(base) a.agafonov@a-agafonov lecture2 % let 'a += 1'
(base) a.agafonov@a-agafonov lecture2 % echo $a
12
(base) a.agafonov@a-agafonov lecture2 % let "a += 1"
(base) a.agafonov@a-agafonov lecture2 % echo $a
13
(base) a.agafonov@a-agafonov lecture2 % a=${a}str
(base) a.agafonov@a-agafonov lecture2 % let a+=1
zsh: bad math expression: operator expected at `str`
```

Переменную можно объявить как целочсиленную с помощью команды `declare -i`.

```
(base) a.agafonov@a-agafonov lecture2 % declare -i b=0
(base) a.agafonov@a-agafonov lecture2 % echo b
b
(base) a.agafonov@a-agafonov lecture2 % echo $b
0
(base) a.agafonov@a-agafonov lecture2 % b+=42
(base) a.agafonov@a-agafonov lecture2 % echo $b
42
```

2.2 bash скрипты

2.2.1 Как устроены bash скрипты?

Давайте создадим следующий файл: `example`

```
# Some comment
pwd
whoami
ps -cp "$$" -o command="" # which shell is used
```

Как нам его запустить? Первый вариант – указать в командной строке, какой оболочкой пользоваться.

```
(base) a.agafonov@a-agafonov lecture2 % bash ./example
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2
a.agafonov
bash
(base) a.agafonov@a-agafonov lecture2 % sh ./example
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2
a.agafonov
sh
(base) a.agafonov@a-agafonov lecture2 % zsh ./example
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2
a.agafonov
zsh
```

Другой вариант – указать оболочку в первой строке файла. Например, это может быть `#!/bin/bash`, `#!/bin/zsh` или любая другая оболочка.

```
((base) a.agafonov@a-agafonov lecture2 % cat example
#!/bin/bash
# Some comment
pwd
whoami
ps -cp "$$" -o command="" # which shell is used
(base) a.agafonov@a-agafonov lecture2 % ./example
zsh: permission denied: ./example
(base) a.agafonov@a-agafonov lecture2 % ls -la ./example
-rw-r--r--  1 a.agafonov  staff  86 Sep 13 17:20 ./example
(base) a.agafonov@a-agafonov lecture2 % chmod -x ./example
(base) a.agafonov@a-agafonov lecture2 % ./example
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2
a.agafonov
bash
```

В примере выше мы столкнулись с проблемой, что файл не был исполняемым. Чтобы это исправить мы использовали команду `chmod`.

2.2.2 Локальные переменные и переменные окружения

Переменные окружения – это переменные, которые определены для текущей оболочки и наследуются любыми дочерними оболочками или процессами. Переменные окружения используются для передачи информации в процессы, которые порождаются из оболочки.

Локальные переменные (переменные оболочки) – переменные, область видимости которых ограничена блоком кода или телом функции.

В более общем контексте, каждый процесс имеет некоторое "окружение"(среду исполнения), т.е. набор переменных, к которым процесс может обращаться за получением определенной информации.

Для вывода переменных окружения можно использовать команды `env` и `printenv`. Разница между этими двумя командами проявляется только в их более конкретной функциональности. Например, с помощью `printenv` вы можете запросить значения отдельных переменных:

```
(base) a.agafonov@a-agafonov lecture2 % printenv SHELL
/bin/zsh
```

Рассмотрим следующий пример. Пусть для запуска некоторой программы (например, luigi таска) нам нужно передать в нее конфиг. Давайте передадим его через переменные окружения. Создадим файл с конфигом `config.cfg` который содержит путь до датасета:

```
(base) a.agafonov@a-agafonov lecture2 % cat config.cfg
data_fp=/path/to/file
```

Создадим переменную окружения с помощью команды `export`:

```
(base) a.agafonov@a-agafonov lecture2 %
export CONFIG=/Users/a.agafonov/Documents/teaching/made/cs22/lecture2/config.cfg
```

Проверяем

```
(base) a.agafonov@a-agafonov lecture2 % printenv CONFIG
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2/config.cfg
```

Используя `export`, ваша переменная окружения будет установлена для текущего сеанса оболочки. Как следствие, если вы откроете другую оболочку или перезапустите свою систему, переменная окружения больше не будет доступна.

Создадим следующий файл `task`

```
(base) a.agafonov@a-agafonov lecture2 % cat task
#!/bin/bash
echo "config_fp=${CONFIG}"
```

Дадим права на запуск и выполним его

```
(base) a.agafonov@a-agafonov lecture2 % chmod +x ./task
(base) a.agafonov@a-agafonov lecture2 % ./task
config_fp=/Users/a.agafonov/Documents/teaching/made/cs22/lecture2/config.cfg
```

Теперь давайте объявим переменную без `export` (это будет локальная переменная) и попробуем обратиться в ней в другом файлике.

```
(base) a.agafonov@a-agafonov lecture2 % touch file
(base) a.agafonov@a-agafonov lecture2 % cat > file
#!/bin/bash
echo $variable
(base) a.agafonov@a-agafonov lecture2 % ./file
(base) a.agafonov@a-agafonov lecture2 % export variable
(base) a.agafonov@a-agafonov lecture2 % ./file
123
```

Видим, что без `export` дочерний процесс данной оболочки не видит переменную.

Еще раз отметим, что переменные окружения определены для текущей оболочки и наследуются любыми `дочерними` оболочками или процессами. Давайте создадим файл `example_export` с программой `export variable=123` и запустим его.

```
(base) a.agafonov@a-agafonov lecture2 % cat example_export
#!/bin/bash
export variable=123
(base) a.agafonov@a-agafonov lecture2 % chmod +x ./example_export
(base) a.agafonov@a-agafonov lecture2 % ./example_export
(base) a.agafonov@a-agafonov lecture2 % echo $variable
```

Видим, что переменная `variable` не определена.

Команда `set` выдает список всех переменных окружения и оболочки с их значениями. Вывод может быть большим, поэтому стоит воспользоваться командой `set | less`, чтобы сделать вывод постраничным.

2.2.3 source

Давайте попробуем распарсить `config.cfg`. Первый вариант воспользоваться `cut`.

```
(base) a.agafonov@a-agafonov lecture2 % cat task
#!/bin/bash
echo "config_fp=${CONFIG}"
cat ${CONFIG} | cut -d "=" -f2
(base) a.agafonov@a-agafonov lecture2 % ./task
config_fp=/Users/a.agafonov/Documents/teaching/made/cs22/lecture2/config.cfg
/path/to/file
(base) a.agafonov@a-agafonov lecture2 %
```

Как еще можно решить эту задачу? Давайте внутри `task` запустим `config.cfg` и попробуем вытащить из него переменную `data`. Это можно сделать с помощью `source` (или его аналога `.` (точка)). `source` запускает сценарий оболочки, указанный в качестве аргумента. Данную команду можно использовать как `include` в обычных языках программирования.

```
(base) a.agafonov@a-agafonov lecture2 % cat task
#!/bin/bash
echo "config_fp=${CONFIG}"
source ./config.cfg
echo $data_fp
(base) a.agafonov@a-agafonov lecture2 % ./task
config_fp=/Users/a.agafonov/Documents/teaching/made/cs22/lecture2/config.cfg
/path/to/file
```

2.2.4 Установка постоянных переменных окружения

Как мы видели, переменные окружения не были постоянными при перезапуске оболочки. С помощью системных файлов, которые читаются и выполняются в определенных условиях, можно сделать изменения постоянными.

Самый популярный способ постоянной установки переменных среды – добавить их в файл `/.bashrc` (`/.zshrc` для оболочки zsh). `/.bashrc` (`/.zshrc`) представляет из себя скрипт, который каждый раз выполняется при запуске оболочки.

Давайте добавим какую-нибудь папку в `PATH`, например,
`export PATH=/Users/a.agafonov/Documents/teaching/made/cs22/lecture2:$PATH`. Теперь при перезапуске терминала и открытии новых окон этот путь появится в `PATH`. Чтобы добавить эту переменную без перезапуска терминала нужно восползоваться `source /.bashrc`.

2.2.5 PYTHONPATH

Еще одной важной переменной окружения является `PYTHONPATH`. В ней хранятся пути, по которым проходится интерпретатор python в поисках модулей. Когда интерпретатор начинает выполнение, он парсит значение этой переменной и кладёт в `sys.path`.

```
(base) a.agafonov@a-agafonov python_examples % cat myfunc.py
import sys

def print_version():
        print(sys.version)
(base) a.agafonov@a-agafonov python_examples % cd dir
(base) a.agafonov@a-agafonov dir % ls
runner.py
(base) a.agafonov@a-agafonov dir % cat runner.py
from myfunc import print_version

print_version()
(base) a.agafonov@a-agafonov dir % python runner.py
3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ]
```

2.3 Функции

Как и большая часть языков программирования, `bash` имеет операторы `if`, `case`, `while`, `for`. Также в `bash` есть функции. Рассмотрим следующую функцию

```
(base) a.agafonov@a-agafonov lecture2 % cat mcd
mcd (){
        mkdir -p $1;
        cd $1
}
```

Здесь $1 – первый аргумент скрипта/функции. В отличие от других языков сценариев, `bash` использует множество специальных переменных для ссылки на аргументы, коды ошибок и другие важные переменные. Ниже приведен список некоторых из них. Полный список можно найти здесь.

- `$0` – название скрипта
- `$1, ..., $9` – аргументы скрипта. К десятому аргументу и далее обращаться как `"$10"`
- `$@` – все аргументы
- `$#` – число аргументов
- `$?` – код возврата предыдущей команды. Значение 0 обычно означает, что все прошло хорошо; любое значение, кроме 0, означает ошибку.
- `$$` – pid текущего shell (самого процесса-сценария)
- `!!` - полное повторение предыдущей команды. Распространенное применение – когда команда не была выполнена из-за отсутствия прав, то можно повторить ее, просто вызвав `sudo !!`

Команды возвращают выходные данные в `stdout`, ошибки в `stderr` и код возврата.

Давайте воспользуемся нашей функцией `mcd`. Напомним, что для начала надо "импортировать"эту функцию с помощью `source`.

```
(base) a.agafonov@a-agafonov lecture2 % source mcd
(base) a.agafonov@a-agafonov lecture2 % mcd new_folder
(base) a.agafonov@a-agafonov new_folder % ls
(base) a.agafonov@a-agafonov new_folder % echo $?
0
```

Видим, что фунция `mcd` отработала без ошибок. Сделаем функцию, которая возвращает код 1.

```
(base) a.agafonov@a-agafonov new_folder % cat tmp
#!/bin/bash
exit 1
(base) a.agafonov@a-agafonov new_folder % ./tmp
(base) a.agafonov@a-agafonov new_folder % echo $?
1
```

Код возврата может использоваться для условного выполнения команды с помощью && (оператор И) и || (оператор ИЛИ). `True` программа всегда будет иметь код возврата 0, а команда `False` всегда будет иметь код возврата 1. Посмотрим на несколько примеров.

```
(base) a.agafonov@a-agafonov new_folder % ./tmp || echo "fail"
fail
(base) a.agafonov@a-agafonov new_folder % ./tmp && echo "not printed"
(base) a.agafonov@a-agafonov lecture2 % true && echo "ok"
ok
(base) a.agafonov@a-agafonov lecture2 % true || echo "not printed"
```

2.3.1 Условные операторы и циклы

Давайте напишем функцию, принимающую на вход два числа и суммирующую их. Если на вход не подано чисел или их больше 2, то будем выдавать -1. Если число одно то удвоим его.

```
(base) a.agafonov@a-agafonov lecture2 % cat subst
(base) a.agafonov@a-agafonov lecture2 % cat add2num
#!/bin/bash
function addnum {
        if [ $# -eq 0 ] || [ $# -gt 2 ]
        then
                echo -1
        elif [ $# -eq 1 ]
        then
                echo $(($1 + $1))
        else
                echo $(($1 + $2))
        fi
}
```

Рассмотрим пример цикла `for`, который парсит содержимое `PATH`

```
(base) a.agafonov@a-agafonov lecture2 % cat ./path_parser
#!/bin/bash
IFS=":"
for path in $PATH
do
        echo $path
done
(base) a.agafonov@a-agafonov lecture2 % ./path_parser
/usr/local/Caskroom/google-cloud-sdk/latest/google-cloud-sdk/bin
/Users/a.agafonov/opt/anaconda3/bin
...
```

2.3.2 Вызов bash-функций из командной строки

В разделе 2.2.4 мы добавили в `PATH` следующий путь `/Users/a.agafonov/Documents/teaching/made/cs22/lecture2`. Именно по этому пути у нас лежит файл `mcd` с одноименной командой. Чтобы сделать этот файл исполняемым из командной строки надо добавить в `/.bashrc` (`/.zshrc`) строчку `source mcd`. Если папка с исходником не указана в `PATH`, то после `source` надо указать полный путь до файла. Теперь мы можем вызывать эту команду из любого места.

2.3.3 Подстановки

В bash есть возможность извлекать информацию из вывода команд и назначать её переменным, что позволяет использовать эту информацию где угодно в файле сценария. Делается это с помощью конструкции `$()`. Пример

```
(base) a.agafonov@a-agafonov lecture2 % cat subst
#!/bin/bash
mydir=$(pwd)
echo $mydir
(base) a.agafonov@a-agafonov lecture2 % ./subst
/Users/a.agafonov/Documents/teaching/made/cs22/lecture2
```

2.4 Источники

[1] Mendel Cooper. Advanced Bash-Scripting Guide.

[2] Колисниченко Д. Н. Командная строка Linux и автоматизация рутинных задач. – БХВ-Петербург, 2012.

[3] The Missing Semester of Your CS Education - Lecture 1 - MIT, 2020.

[4] Роббинс А. Bash. Карманный справочник для системного администратора.

[5] Bash script easy guide.

Домашнее задание 1

В этой задаче вы напишите скрипт для "автоматического оценивания"домашнего задания. Выход скрипта должен выглядить следующим образом:

```
Maximum score 50
Processing barrett ...
barrett has incorrect output (1 lines do not match)
barrett has earned a score of 45 / 50

Processing waters ...
waters has correct output
waters has earned a score of 50 / 50

Processing mason ...
mason did not turn in the assignment
```

Скачайте файл по ссылке. Его содержимое:

- `students/`: директория с домашними заданиями студентов для проверки. Домашнее задание должно называться `task1.sh`. Некоторые стедnты могли не сдать задание или назвать файл неправильно.
- `expected.txt`: файл содержит ожидаемый ответ домашнего задания. Это текст, который вы будете сравнивать с выходом домашних заданий студентов.

Вам нужно написать скрипт `hw1.sh`, который выставляет студенту оценку. У него должен быть один аргумент MAXPOINTS. Например, чтобы запустить проверку с максильным баллом 50 надо запустить:

```
% ./hw1.sh  50
```

Если аргумент не передан или их больше 2, то скрипт должен вывести сообщение об ошибке и завершиться с кодом ошибки:

```
% ./hw1.sh
Usage ./hw1.sh MAXPOINTS
```

Аргумент можно не валидировать. Предполагайте, что аргумент всегда положительное целое число. Также предполагайте, что в директории откуда вы запускаете скрипт есть директория `students` и файл `expected.txt`.

Проверка проходит следующим образом

- Скрипт `hw1.sh` запускает файл `task1.sh` каждого студента.
- Скрипт `hw1.sh` сравнивает выход `task1.sh` с `expected.txt` с помощью `diff`. Запусакайте `diff` так, чтобы он игнорировал все пробелы. Будем считать, что одна строка выхода `diff` содержащая < или >, говорит о наличии одной ошибки в домашнем задании. Пример:

  ```
  wright has correct output

  gilmour has incorrect output (2 lines do not match)
  ```

  За каждую несовпавшую строку вычтите по 5 баллов из MAXPOINTS. Например, если выход `diff` содержит 5 строк с < или >, то надо вычесть 25 баллов. Если студент теряет баллов больше чем MAXPOINT, то он получает 0 баллов. Количество баллов за домашннее задание также должно быть выведено

  ```
  wright has earned a score of 50 / 50

  gilmour has earned a score of 40 / 50
  ```

Если студент на сдал задание или назвал файл неверным образом, то выход должен быть следующим

```
mason did not turn in the assignment
```

Вы можете предположить, что никакая студенческая программа не пытается причинить вред вашему компьютеру, например, стереть все ваши файлы. Вы также можете предположить, что код учащихся не застрянет в каком-либо бесконечном цикле. Всли ваш скрипт может создавать временные файлы во время работы, но не забывайте добавить команду по их удалению в скрипт.

Для сдачи задания загрузите файл `hw1.sh` на портал.

3 Подключение к удаленным машинам и управление процессами

3.1 Подключение по ssh

На большинстве серверов используется ОС Linux. Графический интерфейс на серверах Linux не используется для экономии ресурсов, поэтому единственный способ работы с ними – командная строка. Протокол ssh позволяет вам выполнять команды в удаленной системе так, как будто вы это делаете в своей системе.

Для подключения по ssh требуются

- ip адрес сервера
- порт на котором ожидает подключения ssh сервер
- имя пользователя и пароль на удаленном сервере

Чтобы подключиться к удаленному серверу нужно набрать команду

```
% ssh username@ipadress
```

или

```
% ssh username@ipadress -p port
```

Если все введено верно, то программа запросит пароль. Пример

```
(base) a.agafonov@a-agafonov cs22 % ssh artem@127.0.0.1 -p 2222
artem@127.0.0.1's password:
```

Если пытаетесь подключится через ssh к этому серверу первый раз, то утилита также попросит подтвердить добавление нового устройства в свой список известных устройств, здесь нужно набрать yes и нажать Enter.

3.1.1 Аутентификация без пароля

Использование ssh пароля для входа на сервер не только неудобно но и небезопасно, потому что этот пароль в любой момент может быть подобран. Самый надежный и часто используемый способ аутентификации - с помощью пары ключей RSA. Секретный ключ хранится на компьютере, а публичный используется на сервере для удостоверения пользователя.

Ключ создается командой

```
(base) a.agafonov@a-agafonov cs22 % ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/a.agafonov/.ssh/id_rsa):
...
```

Во время создания ключа нужно будет ответить на несколько вопросов, расположение оставляйте по умолчанию, если хотите подключаться без пароля - поле Passphare тоже оставьте пустым.

Отправляем публичный ключ на сервер

```
ssh-copy-id -i ~/.ssh/id_rsa.pub artem@127.0.0.1 -p 2222
```

3.1.2 Аутентификация без username@remoteserver

Вместо постоянного ввода `username@remoteserver` мы можем воспользоваться псевдонимами. Для этого есть два пути.

Bash aliases. Добавьте в `~/.bashrc` или `~/.zshrc` строку

```
alias VM='ssh artem@127.0.0.1 -p 2222'
```

Применим изменения с помощью

```
(base) a.agafonov@a-agafonov cs22 % source ~/.zshrc
```

Теперь для подключения достаточно ввести команду `webserver`.

Конфиг файл ssh. Используем дефолтный конфиг ssh `~/.ssh/config`. Добавляем в него

```
(base) a.agafonov@a-agafonov cs22 % source ~/.zshrc
```

```
Host webserver
    Hostname 127.0.0.1
    User artem
    Port 2222
```

Теперь можем подключиться к серверу следующим образом

```
(base) a.agafonov@a-agafonov cs22 % ssh webserver
```

3.1.3 Выполнение команд

Мы привыкли подключаться к удаленному серверу и потом выполнять нужные команды. Но ssh позволяет сразу выполнить нужную команду без открытия терминала удаленной машины. Например:

```
(base) a.agafonov@a-agafonov lecture3 % ssh -p 2222 artem@127.0.0.1 ls
```

Или можем воспользоваться нашим псевдонимом `webserver`

```
(base) a.agafonov@a-agafonov lecture3 % ssh webserver ls
```

Можно воспользоваться операторм `|`

```
(base) a.agafonov@a-agafonov lecture3 % ssh webserver ls | grep Documents
```

Выполним на удаленном сервере локальный скрипт `localscript` с помощью перенаправления ввода Bash:

```
(base) a.agafonov@a-agafonov lecture3 % cat localscript
#!/bin/bash
echo $1 $2
(base) a.agafonov@a-agafonov lecture3 % ssh webserver "bash -s" < localscript "hi" "bye"
hi bye
```

Запустить удаленный (лежащий на сервере) скрипт `serverscript`

```
(base) a.agafonov@a-agafonov lecture3 % ssh webserver cat serverscript
#!/bin/bash
echo $1 $2
(base) a.agafonov@a-agafonov lecture3 % ssh webserver bash serverscript "hi" "bye"
hi bye
```

3.1.4 Передача файлов по ssh

Для передачи файла можно воспользоваться утилитой `scp`.

Передадим локальный файл на сервер:

```
% scp -P 2222 send.txt webserver:~/Folder
send.txt                                                  100%   13    18.0KB/s   00:00
```

Скачаем файл с сервера:

```
(base) a.agafonov@a-agafonov lecture3 % scp webserver:"~/Folder/download.txt" ./
download.txt                                              100%    9     0.3KB/s   00:00
```

Передать файл можно и без `scp`, воспользовавшись `|` и интерактивным режимом `cat`:

```
(base) a.agafonov@a-agafonov lecture3 % cat send.txt | ssh webserver "cat > cat_send.txt"
(base) a.agafonov@a-agafonov lecture3 % ssh webserver cat cat_send.txt
Hello, World
```

Более того, мы можем сжать файл перед отправкой, а сразу после передачи его распаковать. Так можно передавать целые папки.

```
(base) a.agafonov@a-agafonov lecture3 % tar -czf - Folder | ssh webserver tar -xvz -C /home/artem
```

3.1.5 ssh туннели и проброс портов

Запустим на сервере jupyter notebook

```
(base) artem@artem-VirtualBox:~$ jupyter notebook --no-browser
[I 12:39:42.237 NotebookApp] Jupyter Notebook 6.4.12 is running at:
[I 12:39:42.237 NotebookApp] http://localhost:8888/?token=d3e5c505ac66fbc148f6568e979a76681af3e246045
```

Чтобы иметь доступ с локальной машины к jupyter серверу, запущенному на удаленной машине, нужно создать тоннель. Например, на сервере jupyter слушает порт 8888. Тогда, чтобы получить доступ к интерфейсу с локальной машины и порта 8000, мы выполним

```
sh -L 8000:localhost:8888 username@remoteserver -p port
```

и перейдем на `locahost:8000` на локальной машине.

Пример:

```
ssh -L 8000:localhost:8888 webserver -N
```

Очистить порт можно командой

```
npx kill-port 8000
```

3.2 Загрузка файлов через curl и wget

3.2.1 wget

`wget` это утилита для загрузки файлов из интернета.

Пример згрузки одного файла и сохранение в нынешней директории

```
(base) a.agafonov@a-agafonov lecture3 % wget https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
--2022-09-21 13:15:12--  https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a9a
Resolving www.csie.ntu.edu.tw (www.csie.ntu.edu.tw)... 140.112.30.26
Connecting to www.csie.ntu.edu.tw (www.csie.ntu.edu.tw)|140.112.30.26|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2329875 (2,2M)
Saving to: ‘a9a’

a9a                     100%[============================>]   2,22M   482KB/s    in 6,4s

2022-09-21 13:15:20 (354 KB/s) - ‘a9a’ saved [2329875/2329875]
```

При загрузке выводится следующая информация:

- процент загрузки
- суммарное число загруженных битов
- скорость загрузки
- оставшееся время на загрузку.

С помощью опции `-O` можно указать имя файла

```
(base) a.agafonov@a-agafonov lecture3 % wget -O dataset
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a9a
```

С помощью опции `-c` можно продолжить прерванную загрузку файла

```
(base) a.agafonov@a-agafonov lecture3 % wget -c -O gisette
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/gisette_scale
```

Это удобно для того чтобы не перекачивать большой файл полностью.

Еще флаги:

- `--limit-rate=200k` – ограничить загрущку скоростью в 200KB
- `-b` – загрузка в фоновом режиме, выход сохраняется в `wget-log`. С помощью флага `-o` можно указать название лог файла.
- `--spider` – проверка, что файл существует и загрузка будет успешной.
- `-i` – загрузка нескольких файлов одновременно.

3.2.2 curl

`curl` - это утилита, которая позволяет делать различные сетевые запросы по различным протоколам данных (HTTP, FTP, SCP, ...). Смысл: сделать какой-то запрос и получить ответ.

Следующая команда получит содержимое URL-адреса и отобразит его в stdout

```
(base) a.agafonov@a-agafonov lecture3 % curl https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets
/binary/a1a
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 ...
```

Можно сохранить сожержимое в файл с помощью `> filename` или получить только несколько строк `| head -5`.

Другие опции для сохранения в файл:

- `-o filename` – сохранение в `filename`,
- `-O` – сохранение в файл с исходным названием,
- `-I` – запрос заголовков.

Также через `curl` можно отправлять PUT и POST запросы.

3.3 Управление процессами в shell

Часто программы могут выполняться очень долго и, соотвественно, могут занимать терминал. Представим такой случай, что вы решили что-то запустить еще в этом терминале, а он занят. Тут можно воспользоваться `ctrl+z` для приостановки процесса. `fg %1` (foreground) – продолжить процесс. Аргумент указывает на то, какую именно задачу следует запустить. Вы можете приостанавливать более чем одну задачу одновременно, каждая приостановленная задача имеет свой собственный номер, так что ими легко оперировать.

`jobs` выводит список всех приостановленных задач. Можно воспользоваться `bg %1` чтобы продолжить выполнение в фоне.

Но гораздо проще отправить задачу в фон с помощью символа апмерсанд &. Например,

```
(base) a.agafonov@a-agafonov lecture3 % ssh -L 8000:localhost:8888 webserver -N &
[1] 20520
(base) a.agafonov@a-agafonov lecture3 % jobs
[1]  + running    ssh -L 8000:localhost:8888 webserver -N
```

Недостаток всей этой системы один: при закрытии шелла все задачи прерываются. Решить эту проблему можно с помощью команды `disown`. Например, порт можно прокинуть как

```
(base) a.agafonov@a-agafonov lecture3 % ssh -L 8000:localhost:8888 webserver -N & disown
[1] 21996
```

Теперь при закрытии терминала процесс все еще будет запущен. Как его остановить? В `jobs` он отражаться не будет. Увидеть его можно с помощью `ps -ax`. Для остановки процесса можно воспользоваться `kill PID`.

```
(base) a.agafonov@a-agafonov lecture3 % ssh -L 8000:localhost:8888 webserver -N & disown
[1] 24996
(base) a.agafonov@a-agafonov lecture3 % ps -ax
  UID   PID  PPID        F CPU PRI NI       SZ    RSS WCHAN
  S             ADDR TTY           TIME CMD
  503 24996 22156     4006   0  31  5  4310728   2852 -
  SN                  0 ttys000    0:00.03 ssh -L 8000:loca
(base) a.agafonov@a-agafonov lecture3 % kill 24996
```

`nohup` – утилита, чтобы запускать программу независимо от шелла. Если команда запущена из терминала, то стандартный вывод будет перенаправлен в `nohup.out`.

Сама по себе `nohup` не отправляет команду в фон — она нужна скорей для того, чтобы обезопасить себя от неожиданных обрывов связи с терминалом (например, из-за разрыва SSH-соединения). Поэтому в конец команды можно дописать амперсанд &.

`wait` – команда, которая позволяет ожидать завершения определенного процесса. Часто при написании какого-то пайплайна обработки данных надо дождаться выполнения определенных команд, чтобы последующие команды использовали их результат. Например, `wait` можно использовать так `wait PID`, где PID – идентификатор процесса команды, для которой утилита ожидает завершения.

4 Основы git

4.1 Что такое git?

Git это система контроля версий (VCS, Version Control System). Система контроля версий это утилита, которая позволяет отслеживать и управлять изменениями файлов со временем.

Представьте себе такие крупные компании как Facebook, Apple, или VK. Тысячи разработчиков работают каждый день над тысячами файлов с кодом. Как отслеживать изменения этих файлов, управлять ими, комбинировать результаты работы нескольких разработчиков? Ответ: с помощью системы контроля версий. Git всего лишь одна из большого количества систем контроля версий. Например, есть Subversion, Mercurial, CVS и многие другие. Но git – самая популярная. Согласно голосованию пользователей Stackoverflow 87% пользователей предпочитают git другим VCS.

Git позволяет нам:

- отслеживать изменения в файлах;
- сравнивать различные версии проекта;
- откатываться к старым версиям проекта;
- коллаборировать и делиться изменениями с другими разработчиками проекта;
- комбинировать изменения.

Git можно пользоваться напрямую через терминал, или воспользоваться GUI (Graphical User Interface). Список доступных GUI можно найти по ссылке. Самые популярные из них, наверное, GitKraken и GitHub Desktop. Плюсы импользования GUI заключаются в удобной визуализации и относительной простоте. Минусы – зависимость от софта, сложность в исправлении серьезных ошибки.

Как установить git для командной строки.

4.2 Базовые команды git

4.2.1 Конфигурация имени, почты и текстового редактора

Первое, что нужно сделать после установки git – установить имя пользователя и электронную почту. Это нужно, например, для того, чтобы при внесении изменений в open source код было видно, какой юзер что внес и с ним можно было связаться по почте.

```
(base) a.agafonov@a-agafonov lecture4 % git config --global user.name "agafonovartem"
(base) a.agafonov@a-agafonov ~ % git config --global user.email "agafonov.ad@phystech.edu"
```

Можно поменять дефолтный текстовый редактор:

```
(base) a.agafonov@a-agafonov lecture4 %  git config --global core.editor "code --wait" #vscode
(base) a.agafonov@a-agafonov lecture4 % git config --global core.editor "vim" #vim
```

4.2.2 Инициализация git репозитория

Давайте создадим репозиторий для работы с git. Создаем директорию и переходим в нее. Первая команда, которую нужно знать – `git status`. Эта команда отображает состояние рабочей директории и файлов, содержащихся в ней. При запуске видим, что что пока что git эту репозиторию не отслеживает:

```
(base) a.agafonov@a-agafonov lecture4 % git status
fatal: not a git repository (or any of the parent directories): .git
```

Чтобы git отслеживал эту директорию надо воспользоваться командой `git init`.

```
(base) a.agafonov@a-agafonov lecture4 % git init
Initialized empty Git repository in /Users/a.agafonov/Documents/teaching/made/cs22/lecture3/.git/
(base) a.agafonov@a-agafonov lecture4 % git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

После еще одного запуска команды `git status` видим, что git начал отслеживать изменения в этой директории.

Тут важно сделать два замечания. Первое – теперь в этой директории появилась папка `.git`. В ней хранится вся история разработки проекта. Удалять ее не стоит. Второе – остерегайтесь инициализации git в дочернем репозитории. Во-первых, это совсем незачем делать, git отслеживает состояние дочерних папок. Во-вторых, git может вести себя некорректно.

4.2.3 Adding & Committing

По сути коммит (commit) это копия вашего проекта в конкретный момент времени. Можно его рассматривать как чекпоинт (сохранение) в компьютерных играх. На самом деле git не просто слепо копирует весь проект каждый раз, а ужимает коммит в набор изменений между текущей версией и предыдущей. Принятно логичестки разделять коммиты. Рассмотрим пример на Рис. 1. Мы сделали несколько изменений в проекте. Часть из них связана с изменением файлов, связанных с моделими, другая часть – добавление препроцессинга. Логично разбить эти изменения на два коммита: "изменение моделей"и "добавление препроцессинга".

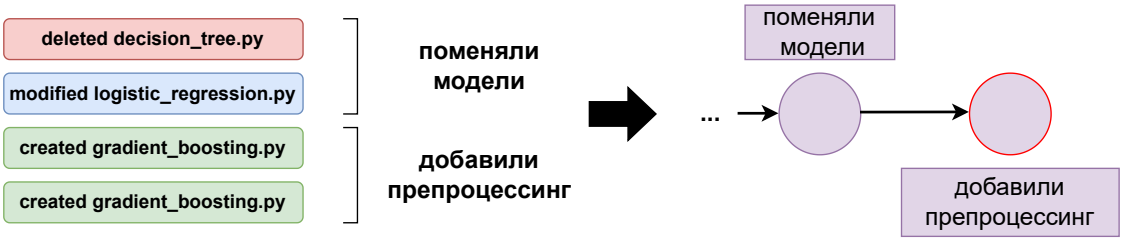


Рис. 1: Коммиты в git.

Рабочей процесс в данном случае состоит из нескольких команд: `git add` и `git commit`. `git add` позволяет выбрать, какие файлы "сохранить". То есть, когда вы готовы сделать "снэпшот"новой версии проекта, надо выбрать файлы с помощью `git add` а потом записать их в историю проекта с помощью `git commit` (см. Рис. 2). Заметьте, что `git add` не вносит никаких изменений в историю проекта, они будут записаны, только если выполнить `git commit`.

Рассмотрим пример:

```
(base) a.agafonov@a-agafonov repo % ls
dataloader.py        model.py
folder
(base) a.agafonov@a-agafonov repo % git add model.py
(base) a.agafonov@a-agafonov repo % git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   model.py
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        dataloader.py
(base) a.agafonov@a-agafonov repo % git add dataloader.py
(base) a.agafonov@a-agafonov repo % git status
On branch master
```

```
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   dataloader.py
        new file:   model.py
(base) a.agafonov@a-agafonov repo % git commit -m "model and dataloader added"
[master (root-commit) 2e06f04] model and dataloader added
 2 files changed, 15 insertions(+)
 create mode 100644 dataloader.py
 create mode 100644 model.py
```

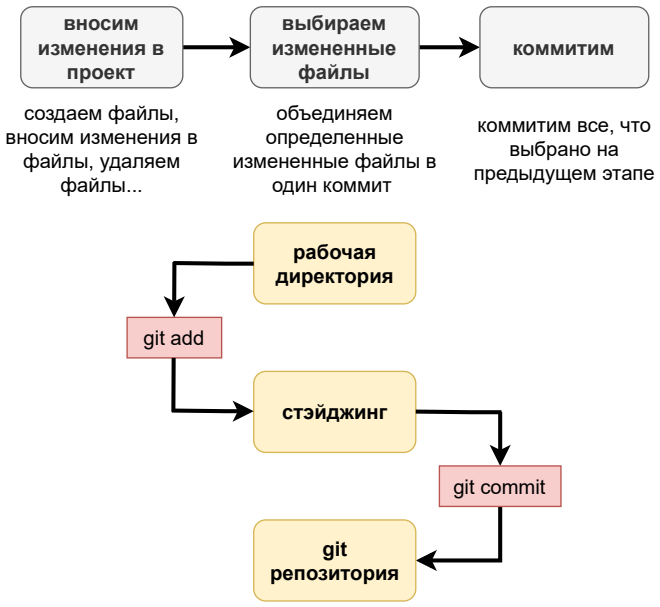


Рис. 2: Git add commit workflow.

В этом примере мы сделали коммит с добавлением двух новых файлов (git раньше их не видел и не отслеживал) `dataloader.py` и `dataloader.py`. Мы использовали флаг `-m` у команды `git commit`. Каждый коммит должен сопровождаться сообщением. `git commit -m "message"` позволяет написать сообщение прямо в командной строке. Если воспользоваться команой без этого флага откроется текстовый редактор для записи сообщения. Редактор мы установили в разделе 4.2.1.

Если вы забыли добавить какой-то файл в коммит, его можно отменить с помощью `git commit --amend`. Также можно воспользоваться `git reset`. Подробное описание можно найти здесь.

4.3 Лог

Команда `git log` позволяет посмотреть истрию коммитов.

```
(base) a.agafonov@a-agafonov repo % git log
commit 083da7ec9572e0ea4667eb62dd9303ec38515d1a (HEAD -> master)
Author: agafonovartem <agafonov.ad@phystech.edu>
Date:   Thu Sep 22 14:22:36 2022 +0400

    dataloader updated

commit 45ce0ddec4e3d6a809feb9cb51c8fa38a3cffc00
Author: agafonovartem <agafonov.ad@phystech.edu>
Date:   Thu Sep 22 14:18:34 2022 +0400

    cross validation added

commit 2e06f048957d213ce2ef920a10335d82252a00c6
Author: agafonovartem <agafonov.ad@phystech.edu>
Date:   Thu Sep 22 14:10:27 2022 +0400
```

```
    model and dataloader added
(base) a.agafonov@a-agafonov repo %
```

`git log` можно вызывать с флагом `--oneline`. Это опция используется для отображения выхода команды в виде один коммит на строку.

4.4 .gitignore

Мы можем прямо указать git какие файлы игнорировать. Например, вы не хотите отслеживать изменения в логах (`.log`), системных файлах (`.DS_Store`), конфигах IDE (`idea/workspace.xml`), датасетах и т.д. Пути к таким файлам можно прописать в специальный файл `.gitignore` в корневой папке вашего репозитория.

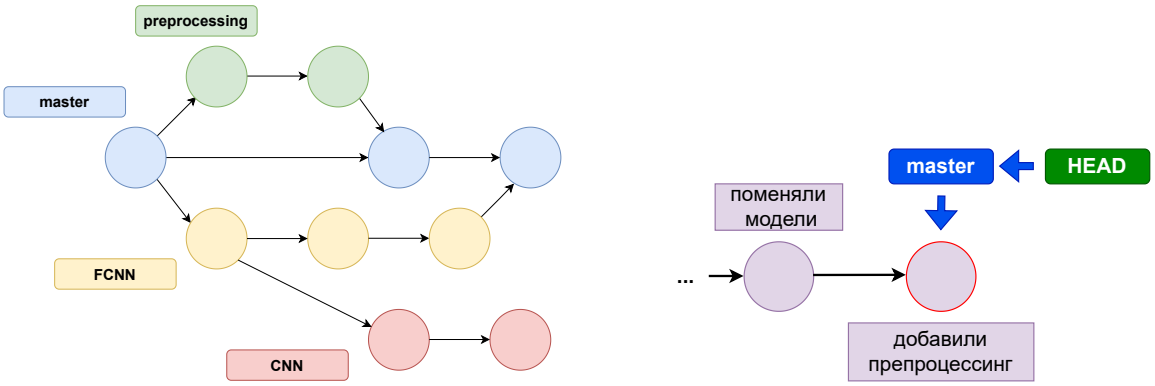
Пример:

```
(base) a.agafonov@a-agafonov repo % ls datasets
cifar10                 cifar100        mnist
(base) a.agafonov@a-agafonov repo % git status -uall
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        datasets/cifar10
        datasets/cifar100

nothing added to commit but untracked files present (use "git add" to track)
(base) a.agafonov@a-agafonov repo % vim ./.gitignore #добавляем пути к датасетам в gitignore
(base) a.agafonov@a-agafonov repo % git status -uall
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

4.5 Ветки

4.5.1 Зачем нужны ветки?

Предствим, что вы с командой решаете задачку на хакатоне. Кто-то должен заниматься препроцессингом, кто-то разичными моделями: полносвязной сеткой, сверточной и т.д. (см. Рис. 3(a)). Для таких случаев и нужны ветки (branch). Они позволяют вести разработку параллельно, где каждая ветка отвечает за какую-то свою фичу проекта.



(a) Каждый кружочек – коммит.

(b) Указатели `master` и `HEAD`.

Рис. 3: Ветки в git.

4.5.2 Ветка master

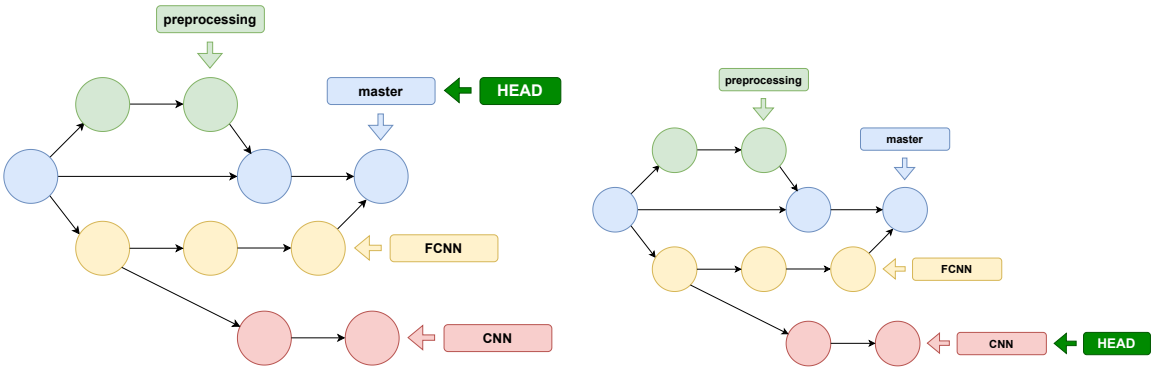
В первую очередь давайте разберемся в веткой `master`. Git автоматически создает эту ветку, в ней мы и вели разработку до сих пор. Это можно увидеть при выполнении команд `git status` и `git log`. Обычно в разработке ветка `master` содержит "чистый"код. Добавление фичей происходит в других ветках, а в `master` содержится лишь тот код, в котором уверены.

Давайте выполним `git log` в нашем проекте с моделями:

```
(base) a.agafonov@a-agafonov repo % git log
commit 3b34c7d04e58afff127d90a7c610314bfca2e3b5 (HEAD -> master)
...
```

Что означает `HEAD -> master`? `HEAD` – это просто указатель на нынешнюю "локацию"в репозитории. Он указывает на конкретную ветку. В случае нашего проекта это выглядит следующим образом (см. Рис. 3(b)).

Теперь мы понимаем, что ветка это просто перемещаемый указатель на один из коммитов. А `HEAD` – указатель на ветку в которой вы находитесь. Теперь Рис. 3(a) можно перерисовать следующим образом (см. Рис. 4).



(a) Находимся в ветке `master`.

(b) Находимся в ветке `CNN`.

Рис. 4: Ветки и `HEAD` в git.

4.5.3 Работа с ветками.

Команда `git branch` выводит список всех веток.

```
(base) a.agafonov@a-agafonov repo % git branch
* master
```

Рассмотрим следующий пример.

```
(base) a.agafonov@a-agafonov repo % git branch -c dev
(base) a.agafonov@a-agafonov repo % git branch
  dev
* master
(base) a.agafonov@a-agafonov repo % git switch dev
Switched to branch 'dev'
(base) a.agafonov@a-agafonov repo % git checkout master
Switched to branch 'master'
```

`git branch -c dev` создает ветку `dev`, с помощью `git switch dev` или `git checkout master` мы переключались между ветками. Перед переключением важно закоммитить изменения! Или воспользоваться `git stash` (см. описание).

Переименовать текущую ветку можно с помощью `git branch -m`, а удалить – `git branch -d`. Пример:

```
(base) a.agafonov@a-agafonov repo % git branch -m tmp
(base) a.agafonov@a-agafonov repo % git branch
  master
* tmp
(base) a.agafonov@a-agafonov repo % git switch master
Switched to branch 'master'
(base) a.agafonov@a-agafonov repo % git branch -d tmp
Deleted branch tmp (was 3b34c7d).
```

4.6 Merging

Команда `git merge` позволяет вам взять различные ветки разработки, созданные коммандой `git branch`, и интегрировать их в одну ветку. Например, на Рис. 4 мы смержили ветку `preprocessing` в `master` и ветку `FCNN` в `master`.

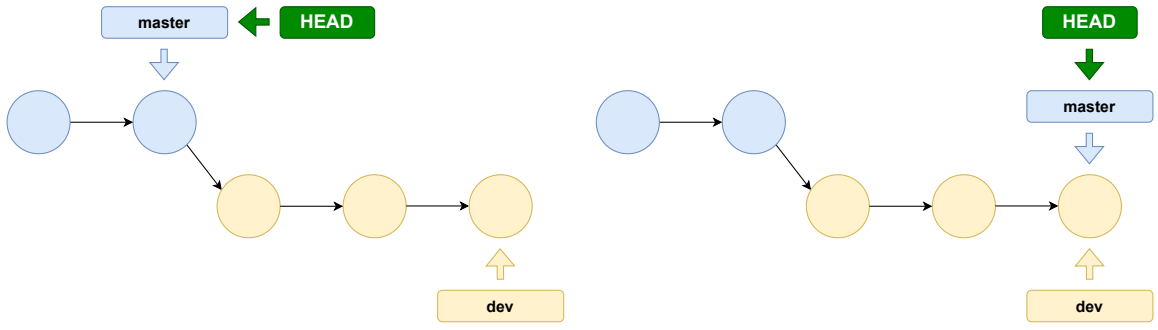
`git merge FCNN` мержит ветку `FCNN` в ткующую ветку (в которой мы находимся в данный момент). Поэтому перед использованием `git merge` надо воспользоваться `git checkout` или `git switch`, чтобы перейти в необходимую ветку. Стоит отметить, что

- мы мерджим ветки, не коммиты;
- мы всегда мерджим в текующую ветку (HEAD branch).

4.6.1 Fast-Forward Merge

Fast-forward мердж происходит, когда существует линейный путь коммитов от вершины текущей ветки до целевой ветки.

Например, пусть сейчас наш проект находится в состоянии как на Рис. 5(a). Мы находимся в ветке `master` и хотим смерджить `dev` в нее. Для этого мы выполняем команду `git merge dev`. Так как история коммитов линейна, выполняется fast-forward merge. Указатель `master` просто перемещается на последний коммит. Результат мерджа изображен на Рис. 5(b).

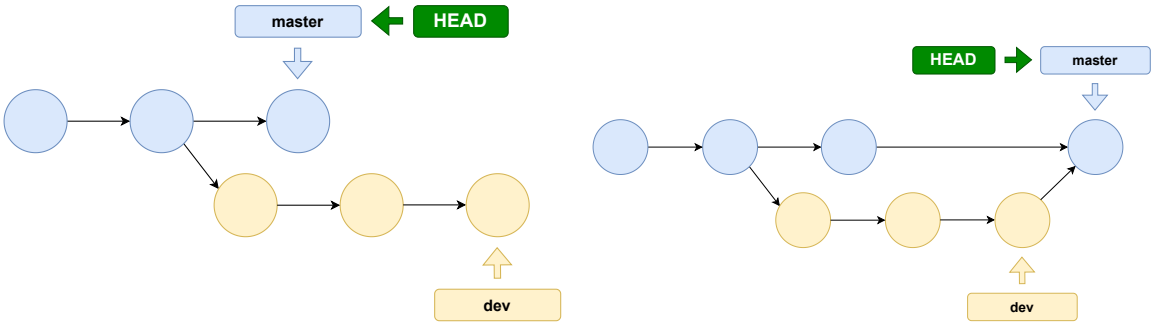


(a) До мерджа. (b) После мерджа

Рис. 5: Fast-forward merge.

4.6.2 Merge Conflicts

Если же две ветки разошлись, то fast-forward мердж применить невозможно (см. Рис 6(a)). Действительно, история проекта нелинейна, и просто переместить указатель как на Рис. 5 невозможно. В таком случае при мердже будет создан новый коммит (см. Рис 6(b)).



(a) До мерджа. (b) После мерджа

Рис. 6: Merge.

Если две ветки, которые вы пытаетесь объединить, изменили одну и ту же часть одного и того же файла, git не сможет определить, какую версию файла использовать. Когда возникает такая ситуация, вам нужно разрешить конфликты вручную. Рис. 5 можно проиллюстрировать следующим кодом

```
(base) a.agafonov@a-agafonov repo % git merge dev
Auto-merging model.py
CONFLICT (content): Merge conflict in model.py
Automatic merge failed; fix conflicts and then commit the result.
(base) a.agafonov@a-agafonov repo % git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   model.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Как мы видим, fast-forward merge не получился. Файл `model.py` теперь выглядит следующим образом:

```
1 <<<<<<< HEAD
2 def fit_predict(X_train, y_train, X_test, model):
3         model.fit(X_train, y_train)
4         return model.predict(X_test)
5 =======
6 class Model:
7         def __init__(self, model):
8                 self.model = model
9
10         def fit(self, X_train, y_train):
11                 self.model.fit(X_train, y_train)
12
13         def predict(self, X_test):
14                 preds = self.model.predict(X_test)
```

```
15                 return preds
16 >>>>>>> dev
```

Нам нужно самостоятельно решить, как этот файл должен выглядить: какую часть кода оставить от ветки `master`, какую от `dev`. В данный момент файл разделен на две части символами `=======`. `<<<<<<< HEAD` указывает начало кода из `HEAD` (ветки куда мы мерджим), `>>>>>>> dev` – конец кода из `dev` (ветки откуда мы мерджим). После того как мы разрешим конфликт (разделительные символы должны быть удалены), нам надо сделать `git add` и `git commit`.

```
(base) a.agafonov@a-agafonov repo % git add model.py
(base) a.agafonov@a-agafonov repo % git commit -m "merged dev into master"
[master d2b17c5] merged dev into master
```

4.7 Источники

[1] Докуметнация

[2] Туториал Atlassian

[3] The Git & Github Bootcamp Udemy course

Домашнее задание 2.

Дедлайн: 24 октября 6:00

Упражнение 1.

Цель: научаться выгружать данные параллельно и форматировать их.

Формат данных: В этом упражнении мы будем выгружать датасеты `a1a, a2a, a3a, a4a, a5a, a6a, a7a, a8a, a9a` с LibSVM. Эти датасеты храняться в текстовом формате. Пример:

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
```

`-1` здесь – таргет, остальные значения показывают в каком признаке стоит 1. Исходя из примера выше получается, что в признаках 3, 11, 14, 19, ..., 83 стоят единицы, в остальных – нули. Мы хотим перевести этот файл в формат `csv` с нулями и единицами на соотвествующих местах и колонкой `target`. Заметьте, что в LibSVM указано количество признаков. На него можно не обращать внимание, а считать что количество признаков это последняя колонка содержащая 1. Например для датасета `a1a` это 119. Пример `csv` файла.

Требования к скрипту: *Таким образом, вам надо написать скрипт, который бы скачивал датасеты и форматировал их параллельно.* То есть, как только датасет `a1a` скачан, его надо сразу же начать форматировать. При этом в параллели идет загрузка и форматирование других датасетов. Работа скрипта должна сопровождаться сообщениями (пример для двух датасетов `a1a`, `a2a`):

```
a1a is downloading # старт загрузки
a2a is downloading
a2a downloaded and now is reading # старт форматирования
a1a downloaded and now is reading
a1a is read # завершение форматирования
a2a is read
all files downloaded # все файлы загружены и отформатированы
```

Обратите внимание, что скрипт должен дождаться загрузки и форматирования всех датасетов и вывести `all files downloaded`. # – комментарий, его выводить не надо, цвета также не важны.

Как перевести в `csv`: Тут есть несколько опций. Например, есть вариант воспользоваться `awk`, который вам не рассказывался на занятиях. Также можно написать `python` скрипт для перевода, используя, например, библиотеку `argparse`. Вам может пригодится эта функция библиотеки `scikit-learn`. Вы можете воспользоваться любым удобным вам инструментом для форматирования. Пример перевода в `csv`.

Организация хранения файлов: Исходные датасеты должны быть загружены и сохранены в `./datasets/txt/`, форматированные – в `./datasets/csv/`. Названия должны быть `a1a` и `a1a.csv` соответсвенно. В самом начале работы скрипта директории должны быть созданы только в том случае, если они не существуют (посмотрите флаги команды `mkdir`).

Summary: План по написанию скрипта можно описать так:

1. создание директорий для хранения файлов;
2. выгрузка датасета (например, с помощью `curl` или `wget`). Обратите внимание, что стандартный выход (выход в терминал) этих команд надо спрятать, если файл уже был загружен его загрузку надо проигнорировать;
3. перевод датасета в формат `.csv`;
4. параллелизация пунктов 2, 3 по датасетам;
5. ожидание выполнения скрипта

Отправка решения: решение надо отправить в формате `.zip` через плафторму. Прикладывать папку `./datasets` не надо!

Упражнение 2.

Инициализация git репозитория, первые коммиты
Цель: создать новую git репозиторию и сделать несколько коммитов.

- Инициализируйте локально git репозиторию.
- Создайте файл и начните симулировать работу над каким-нибудь (придумайте сами) проектом. Сделайте несколько коммитов.
- Установите Git Kraken, сделайте скриншот текущего состояния проекта (визуализации истории коммитов, см. пример на семинаре) и добавьте его в репозиторию в ветку `master` в папку `./screenshots`

Fast-forward merge *Цель*: продемонстрировать ваше понимание fast-forward мерджа.

- Создайте новую ветку.
- Сделайте несколько коммитов.
- Сделайте fast-forward merge в мастер.
- Сделайте скриншот текущего состояния проекта в Git Kraken и добавьте его в репозиторию в ветку `master` в папку `./screenshots`

Merge commit
Цель: сделать мердж **без конфликтов** но с отдельным merge commit.

- Создайте новую ветку.
- Сделайте несколько коммитов.
- Сделайте merge в мастер без конфликтов, но с отдельным коммитом. Подсказку можно найти здесь.
- Сделайте скриншот текущего состояния проекта в Git Kraken и добавьте его в репозиторию в ветку `master` в папку `./screenshots`

Merge conflict
Цель: создать мердж конфликт и разрешить его.

- Создайте новую ветку.
- Измените ваш проект таким образом, чтобы при мердже ветки в `master` получался конфликт.
- Разрешите его и сделайте `merge commit`.
- Сделайте скриншот текущего состояния проекта в Git Kraken и добавьте его в репозиторию в ветку `master` в папку `./screenshots`

Github
Цель: залить ваш проект на github для проверки.

Пример можно посмотреть на 4 занятии или по ссылке. Обратите внимание, что на github должна быть видна вся история ваших коммитов и все созданные вами ветки. Добавьте следующих пользователей в вашу репозитроию (или просто сделайте ее открытой): `Arhimisha`, `oslanaslan`, `andyst75`, `Zmeyoff`, `agafonovartem98`.

Отправка решения: В качестве решения прислать ссылку на ваш репозиторий.