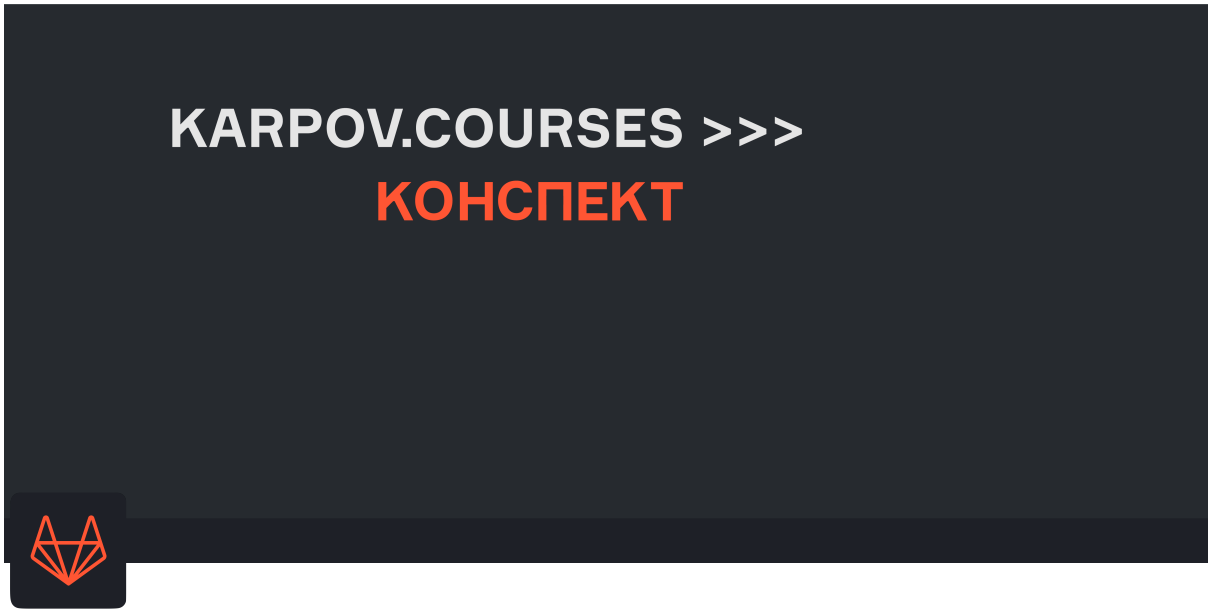

> Конспект > 1 урок > GIT

> 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)

Документация

git status

Выводит статус репозитория

Документация

git branch

Выводит существующие ветки. Если написать после branch название, то вы создадите ветвь с таким названием

Документация

git checkout

Перемещает вас в указанную после checkout ветвь

Документация

git add

Добавляет файл, указанный после add, в рассмотрение на коммит

Документация

git commit

Коммитит добавленные git add файлы

```
git commit -m "commit comment"
```

В `"commit comment"` пишется комментарий к коммиту

Документация

git push

Закидывает закомиченный код в репозиторий. Сначала идёт место, куда вы пушите, а потом что вы пишите

```
git push origin ilin/fix_akarpov_function
```

Так мы запушили закомиченные у нас изменения из ветки `ilin/fix_akarpov_function` в `origin`

Документация

git pull

Стягивает к вам последние изменения в общепринятом состоянии репозитория

```
git pull
```

Документация

Больше информации

> Настройка информации о себе

Для создания коммитов нужно указать своё имя и почту, чтобы другие разработчики знали, кто выполняет изменения. Команды ниже конфигурируют гит под вас

```
git config --global user.email "your@mail"
git config --global user.name "Your Name"
```

Так мы указали, что наша почта your@mail, а имя Your Name

Документация

Краткий обзор работы с git

Ещё один обзор

> Workflow при работе с git'ом

- в первый раз - склонировать репозиторий к себе, при желании посмотреть историю изменений
- обновить свою копию свежими изменениями с гитлаба
- создать ветку и внести изменения
- запушить изменения в гитлаб

Первый пункт делается один раз, когда вы получаете репозиторий, а пункты со второго по четвёртый представляют собой одну итерацию при работе

> Начало работы и получение информации о статусе

В начале нужно склонировать репозиторий, как мы делали в прошлом уроке. После этого репозиторий доступен на машине как папка

Допустим, нам нужно изменить какой-то скрипт или добавить новый. Для этого перейдём в папку, соответствующую репозиторию

```
cd analytics
```

analytics - название папки интересующего репозитория

Информация о состоянии репозитория

Команда `git status` позволяет узнать текущее состояние

```
git status
```

и выдает следующую информацию

On branch master - говорит, что мы находимся в ветке master (то есть основной ветке)Your branch is up to date with ‘origin/master’ - ветка обновлена с веткой masternothing to commit, working tree clean - никаких обновлений нет

Документация

`git log` - покажет всю историю коммитов

```
git log
```

Выйти из него можно, нажав `q`

Документация

Коммиты - это чекпойнты (как сохранения в игре), на которые мы можем перейти, получив состояние репозитория, которое было на момент коммита (сохранения)

> Обновление репозитория

Первое, что нужно сделать - выполнить команду `git pull`

```
git pull
```

Она стаскивает все последние состояния из репозитория и мы увидим все последние изменения в немAlready up to date - означает, что все последние обновления стянуты

Документация

> Концепция веток

Ветки (ветви, брэнчи) - это инструмент для командной работы в гите. Создавая ветвь, мы получаем параллельную вселенную, где сначала всё так же, как в основной ветке master. Меняя код в одной ветке, состояние

другой остаётся прежним, то есть ветки могут развиваться независимо друг от друга

При вызове команды

```
git branch
```

отображается список доступных нам веток и подсвечивается та, в которой мы сейчас находимся

В данном случаемaster - мы находимся в ветке master

Только что созданный репозиторий содержит только одну ветку - master, и она используется в качестве основной.

Документация

Создание веток

Если мы хотим внести какие-то изменения в репозитории - отредактировать код, добавить какие-то новые файлы - нужно создать новую ветку.По сути, создание ветки, – это способ внести изменения в проект и объединить их в итоге с остальным кодом. Ветка копирует все состояние из мастера.

Чтобы создать ветку надо написать всю ту же команду `git branch`, но дополнительно указать название ветки, которую нужно создать

```
git branch akarpov/my_new_branch
```

Для перехода между ветками используется команда `git checkout`

```
git checkout akarpov/my_new_branch
```

перенесёт нас в новую ветку

Документация

Для удаления ветки надо добавить опцию `d`. Если ветка не была запушена, но вы всё равно хотите её удалить, поставьте `D` вместо `d`

```
git branch -d akarpov/my_new_branch
```

Больше информации

> Внесение изменений (коммиты)

Предположим, что нам нужно отредактировать скрипт в репозитории. Прежде чем мы начнём редактировать, убедитесь, что создали новую ветку и перешли в неё

Редактировать можно или в юпитере, или через терминал (в виме). После внесения изменений и сохранения файла, необходимо запечатлеть изменения в виде коммита. Чтобы его создать нужно несколько шагов

Добавление файлов в коммит

Прежде всего посмотрите в оутпут команды `git status`, и убедитесь, что она говорит о том, что произошло

```
git status
```

Добавьте изменённый файл в будущий коммит

```
git add file
```

Это добавит file в число изменений, которые будут "сохранены" в коммите. Чтобы добавить все изменения, можно написать

```
git add .
```

Снова взгляните на

```
git status
```

и убедитесь, что всё идёт по плану

Если что-то пошло не так, то воспользуйтесь командой

```
git checkout .
```

которая откатит вас к состоянию последнего коммита.

Коммит

Для самого коммита используется команда

```
git commit  -m 'important concise comment'
```

Это закоммитит все файлы, которые были добавлены с помощью add, с комментарием important concise comment

Снова взгляните на

```
git status
```

А также посмотрите на

```
git log
```

> Обновление единого репозитория

Теперь настало время поменять код в репозитории у всех разработчиков

Команда

```
git push origin your_branch
```

запушит в репозиторий в гитлабе (origin) изменения, находящиеся в ветке your_branch. Также появится сообщение, что вы создали merge request, то есть запрос на слияние вашего кода с основным.Перейдите по ссылке из сообщения в браузере. В появившемся окне подтвердите merge request (submit merge request)

Документация

Зачем это надо

Merge request (MR) позволяет предотвратить внесение некорректных изменений, которые сломают проект.

Рядом с именем ветки есть кнопка Merge request. При нажатии на нее открывается диалог позволяющий задать описание, добавить комментарий и выбрать кому из разработчиков будет отправлен MR. Также можно установить, что нужно чьё-либо одобрение для принятия merge request и слияния с веткой master.

После этого более скилловые разработчики могут посмотреть реквэст и вмёрджить его в master

Когда merge выполнен, можно переключиться на master и спуллить новые изменения

> Merge conflict

Что будет, если два разработчика одновременно будут вносить изменения в один файл? И при этом один из них сам вмёрджит изменения в masterПри попытке второго разработчика сделать git merge master возникнет конфликт

Будет сообщениеBoth modified - тот файл которые оба разработчика поменялиТут два вариант решения конфликта, принять версию другого разработчика или настоять на своей

Допустим, мы захотели настоять на своей версииВ файле удаляем все комментарии и оставляем свою версию, затем выполняем последовательно

```
git add .
git commit -m ‘fix conflict with encoding’
git push origin akarpov/my_new_branch
```

И сабмиттим merge request

Если второй разработчик не согласен, то чтобы изменить файл надо повторить вышеперечисленные команды

Затем обновляем наш клонированный репозиторий

```
git checkout master
```

Краткий обзор работы с gitЕщё один обзор

> Ресурсы по гиту

Описание возможностей git'a

Мануал от Atlassian

Интерактивный учебник

Пройти git за три часа

> Мутирование пути, где питон ищет модули

```
import sys


sys.path.append('/path/to_your/folder')
```

Данный код добавит `/path/to_your/folder` в путь, где ищутся модули. После этого импорт вашего модуля, находящегося в этой папке, пройдёт успешно

То есть можно будет сделать импорт вашего файла, не находясь в той же папке, где он лежит

Больше информации