Работа с локальным репозиторием с помощью консольного git

Содержание:

1. Установка Git

В разных операционных системах Git устанавливают по-разному:

- в Linux — через командную строку. Перейдите по ссылке https://git-scm.com/download/linux, найдите команду для вашей версии Linux.

- в Windows и macOS скачайте установочный файл с официального сайта https://git-scm.com/.

Если программа-установщик предложит варианты применения Git, выбирайте «Использовать Git из командной строки» (*Use Git from... the Command Prompt*): задействовать все команды Git можно только из командной строки.

В Windows вместе с Git вам предложат установить Git Bash. Это специальная утилита, которая позволит вам использовать Bash под Windows, и все команды будут как на macOS. Мы предлагаем работать именно в нём.

2. Стартовая настройка Git

Откройте терминал для macOS или Git Bash для Windows. Чтобы убедиться, что Git установлен, запросите информацию о программе:

```
git --version
```

Вы увидите сообщение:

```
git version 2.17.1 # это пример; может быть иная версия Git
```

Следующим шагом укажите информацию о себе. Чтобы другие разработчики понимали, кто изменяет код, нужно сообщить программе Git ваши данные: имя пользователя и электронный адрес. Их мы пропишем в файле `.gitconfig` — в нём хранятся глобальные настройки программы. Для этого в командной строке запустите утилиту *git config* с опцией `--global` (англ. *global*, «глобальные»). Свои данные укажите как значения свойств `user.name` и `user.email`:

```
git config --global user.name "Analitik Datascientistov"
# вводите своё имя или ник латиницей и в кавычках
```

```
git config --global user.email analitikds@yandex.com
# здесь нужно ввести свой реальный e-mail
```

Проверьте, что получилось: выполните команду `git config` с опцией `--list` (англ. *list*, «список»):

```
git config --list
# вывели в окно командной строки список всех свойств конфига
```

В открывшемся перечне (Windows) или новом файле (macOS) среди прочих свойств вы увидите свои данные:

```
user.name=Analitik Datascientistov
user.email=analitikds@yandex.com
```

Выход из режима просмотра файла осуществляется нажатием клавиши [**Q**] (сработает при включённой латинской раскладке).

3. Создание, отображение, добавление

Организуем для нашего проекта рабочую директорию — папку с именем *yandex-praktikum-projects*.

Лишь папки недостаточно, нужно ещё сообщить программе Git, что файлы в этой папке нужно отслеживать. Этот процесс называется инициализацией и запускается командой *git init*.

Инициализируем Git вызовом **git init** (от англ. *initialization*, «инициализация»).

```
# Сначала создаём папку для проекта, назовём её yandex-praktikum-projects.

mkdir yandex-praktikum-projects
# создали папку yandex-praktikum-projects в текущей директории

cd yandex-praktikum-projects
# перешли в созданную папку yandex-praktikum-projects

git init
# инициализировали git в папке yandex-praktikum-projects
```

Команда *git init* создаёт в папке с проектом скрытую поддиректорию **.git,** где хранится вся информация об изменении файлов.

4. Создание локального репозитория Git

Версионный контроль в Git предполагает, что любой файл репозитория находится в одном из четырёх состояний:

1. **неотслеживаемый** (англ. *untracked*),

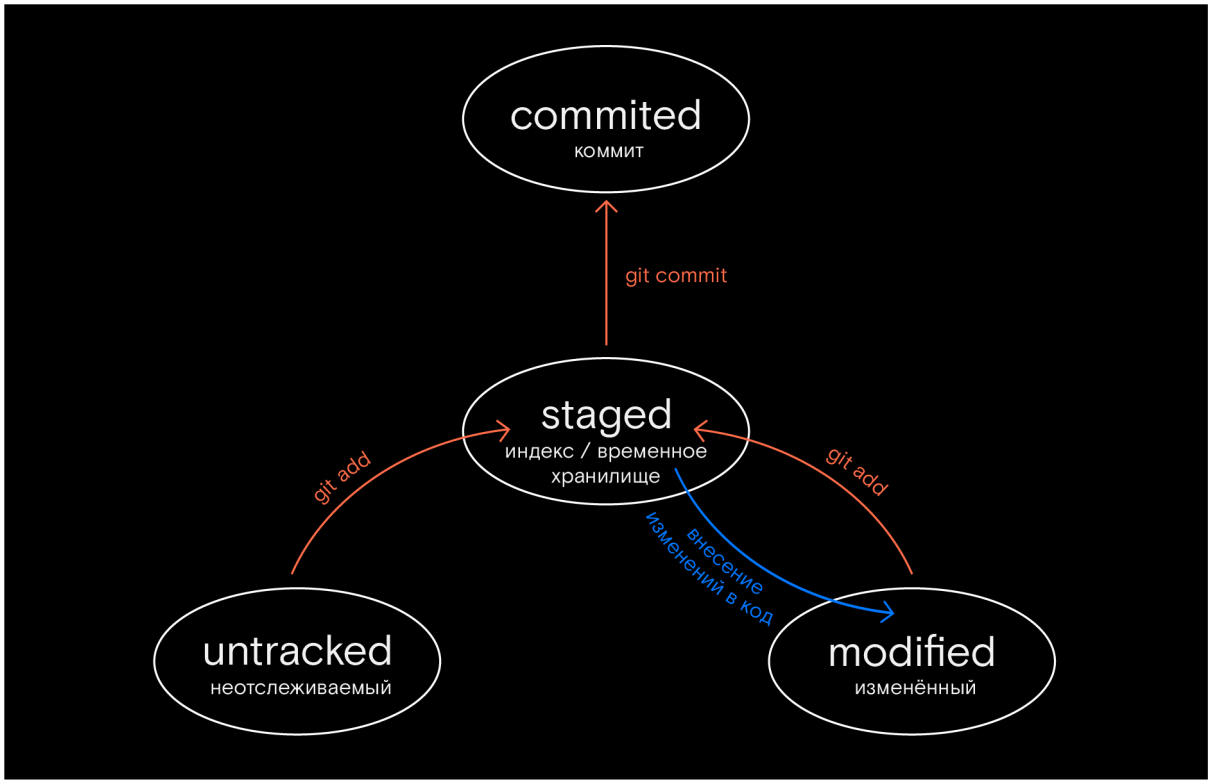
2. **добавленный** в индекс, индексируемый (англ. *staged*, «выдвинутый на плацдарм»),
3. **изменённый** (англ. *modified*),
4. **боевой**, на жаргоне разработчиков «коммит» или «закоммиченный» (англ. *committed*, «брошенный в бой»).

Логика работы такая: как только в инициализированной папке появляется файл, он попадает в состояние неотслеживаемый. Что бы вы ни делали с этим файлом, Git будет игнорировать изменения.

Чтобы Git обратил внимание на этот файл и стал учитывать вносимые в него изменения, файл нужно добавить в индекс. После этого файл перейдёт в состояние «добавленный», а если после этого внести в него изменения — в состояние «изменённый».

Когда мы закончим изменение файла, нам нужно его «сохранить» — сообщить программе Git, что актуальное состояние файла нужно запомнить. Позже это позволит нам вернуться к этой версии, если что-то пойдёт не так.

Такое сохранение называют **коммитом**:



Команды Git, которые нам предстоит изучить, выполняют одну из трёх задач:

- изменяют состояние файла;

- отображают информацию о файле;
- показывают разницу между его версиями.

5. Команды Git

5.1. git add

git add (англ. *add*, «добавить») — команда для добавления файлов в индекс **Staging Area** (англ. «плацдарм», «место временного сосредоточения», «временное хранилище»). После этой команды нужно указать имя файла, судьбу которого поручено отслеживать: `git add название_файла`.

Можно добавить все файлы сразу прибегнув к опции **--all** (англ. all, «все»). Файлы из вложенных каталогов так же добавятся в индекс:

`git add --all`

Опцию **--all** можно заменить точкой: `git add .`

```
git add project1.ipynb
# добавили файл project1.ipynb

git add --all
# добавили все файлы

git add .
# добавили все файлы
```

5.2. git commit

Чтобы сохранить состояние файлов нужно сделать коммит — зафиксировать все сделанные изменения в «боевой версии» и оставить комментарий. Делают это специальной командой **git commit** (англ. *commit*, «бросить в бой») с ключом **-m** (от англ. *message*, «послание»). После этого ключа требуется комментарий в кавычках:

```
git commit -m "My first commit"
# сделали первый коммит
# текст комментария - "My first commit", в переводе "мой первый коммит"
# комментарии лучше писать латинским алфавитом, чтобы они корректно отображались в ком
андной строке
```

Пишите комментарии так, чтобы потом вам было легко разобраться, какие изменения были сделаны в каждом коммите.

5.3. git clone

Самое время опубликовать проект на гитхабе! Скопируйте репозиторий из облака на локальное хранилище командой **git clone** (англ. *clone*, «клонировать»):

```
git clone [адрес, откуда копируем] [путь до папки, куда копируем]
```

Откройте командную строку и клонируйте репозиторий в память своего компьютера. Не забывайте: если вы уже находитесь в той папке, куда нужно клонировать, путь указывать не надо. Достаточно указать ссылку на клонируемый проект:

```
git clone https://github.com/<user>/<repository>.git
# скопировали репозиторий с сайта github в текущую директорию
```

После клонирования на компьютере появится папка с названием репозитория.
Зайдём в неё из командной строки:

```
cd <repository>.git
# перешли в папку с проектом
```

Скопируйте все файлы с проектами в этот репозиторий. Добавьте все файлы:

```
git add --all
# добавили всё
```

Сделайте коммит:

```
git commit -m "First commit"
# сделали коммит с комментарием First commit
```

5.4. git push

Вот и настал момент загрузки вашего сайта на удалённый репозиторий. Для этого есть команда **git push** (англ. *push*, в значении «от себя»). Так как мы скопировали готовый репозиторий с сервера, для первой публикации изменений будем вызовем команду *git push* с ключом *-u* и двумя аргументами:

- первый аргумент — имя сервера, с которого вы скопировали репозиторий: **origin** (англ. «источник»);
- второй аргумент — имя ветки: **master**;
- ключ **-u** связывает локальную ветку с веткой удалённого репозитория. Этот ключ нужен, только если вы публикуете новые ветки.

Опубликуем новую ветку:

```
git push -u origin master
```

Если всё прошло хорошо, вы увидите добавленные файлы в своём репозитории на сайте GitHub.

Если вы храните свои репозитории на разных компьютерах, то когда вы публикуете сделанные на одном компьютере изменения, на другом — их нужно получить командой *git pull*.

5.5. git pull

Команда **git pull** (англ. *pull*, в значении «на себя») заносит все созданные коммиты в вашу рабочую ветку. Выполнение этой команды может привести к конфликтам, так как все изменения при получении сразу сливаются с вашей веткой.

```
git pull
```

6. .gitignore

При работе с локальным репозиторием иногда хочется положить в папку с репозиторием дополнительные документы или даже папки, но так, чтобы они не попали в удалённый репозиторий на Github. Для этого нужно создать файл `.gitignore`. Это - простой текстовый документ, в каждой строчке которого написана маска файлов, которые не должны попасть на удалённый репозиторий после команды *git pull*.

Ниже есть примеры шаблонов в строках файла `.gitignore`

```
# комментарий – эта строка игнорируется

# игнорировать файлы с расширением .pyc,
# * - любое количество символов (ноль и больше),
*.pyc
# НО отслеживать файл main.pyc
```

```
# несмотря на то, что мы игнорируем все .pyc файлы с помощью предыдущего правила
!main.pyc
# ? - ноль или один символ
# Исключить файлы text.txt, test.txt, tet.txt и т.д.
te?t.txt
# игнорировать только файл main.py находящийся в корневом каталоге
# не относится к файлам вида <папка>/main.py
/main.py
# игнорировать все файлы в каталоге .idea/
.idea/
# игнорировать файлы с расширением .txt, только в папке doc, но не в подпапках папки doc
doc/*.txt
# игнорировать все .txt файлы в каталоге doc/ и всех его подкаталогах
doc/**/*.txt
```

Если вы добавили файл в репозиторий и опубликовали изменения командой *git pull*, но не добавили шаблон для файла в *.gitignore*, то он будет опубликован на github. После добавления шаблона в *.gitignore*, изменения в файле не будут отражаться на github, но сам файл не удалится с репозитория.

Чтобы удалить файл, удалите его, зафиксируйте и опубликуйте изменения. Затем добавьте файл и добавьте его шаблон в *.gitignore*. После этого снова опубликуйте изменения.

Если у вас возникли затруднения в процессе изучения консольного git, вам может помочь интерактивный тур по гит https://githowto.com/ru