Для начала **открываем консоль:**

в папке, где создаем депозитарий для работы с **Git** жмем правую кнопку мыши и выбираем: “**Git Bash Here**”.

После открытия консоли прописываем в ней **уровень конфигурации** и **имя пользователя**:

* **git config --global user.name “Igor Zhemgulis”**

далее прописываем **email**:

* **git config --global user.email zhemgulis@gmail.com**

эта информация будет отражаться в **коммитах/commit.**

*Примечание:*

«**git config --system**» (для всех пользователей системы), «**git config --global**» (для конкретного пользователя системы), «**git config --local**» (только для текущего репозитария).

Очень важно понимать в ***каких состояниях могут находиться файлы*** нашего репозитария:

1. Они могут быть: **tracked** – отслеживаемые (изменения истории которых нам важно) и **untracked** – не отслеживаемые (то, что есть в папке нашего проекта, но нам не важно их отслеживать, временные файлы – например логи т.п.)
2. Далее мы работаем с отслеживаемыми файлами (изменяем, сохраняем, удаляем и т.п.) затем **коммитим т.е. делаем «снимок»** нашего проекта и **Git** запоминает его состояние. Далее круг повторяется у нас таких коммитов какое-то кол-во и при желании мы **можем откатиться** на любой коммит.
3. Тут, может быть, два состояния файла в которых он может находиться: когда файл был изменен, но еще не попал в коммит он находится в состоянии – **modified** (**измененный**). После коммита, когда **Git** уже сохранил состояние проекта он переходит в состояние – **committed** (**зафиксированное**). Потом файл может снова перейти в состояние modified и круг повторится.
4. Если взглянуть немного ближе мы увидим, что есть еще одно промежуточное состояние, в котором может находится файл – **staged** (**подготовленное**) к коммиту. А область, в которой находятся такие файлы – **staging area** (**index**). На самом деле, нельзя просто изменить файл и тут же сделать коммит. После того как мы создадим новый файл или изменим какой-то существующий нужно выполнить команду – **git add** (+название файла или название целой директории) только тогда этот файл или директория попадут в **staging area**.
5. Только после этого можно выполнить команду – **git commit -m** **“В этих кавычках пишем комментарии к коммиту”** (“**-m**” – команда для написания комментария к коммиту).
6. Желательно не делать огромных коммитов, один большой лучше разбить на несколько небольших так будет намного удобнее потом разбираться, где, что, когда и как было изменено и проще откатить для исправления небольшую часть с ошибкой, чем переделывать заново огромный кусок работы.
7. Например, у нас есть четыре файла готовые к коммиту – сначала мы можем сделать коммит отдельно для первых двух файлов: подготовим их к коммиту – **git add File-1 File-2**, теперь сам коммит – **git commit -m** **“примечание…”**, повторяем эти шаги для двух оставшихся файлов – **add File-3 File-4** и еще раз **git commit -m** **“примечание…”**. Таким образом при необходимости нам не нужно будет откатывать сразу все файлы, мы сможем это сделать только для двух из четырех, которые нас интересуют.
8. Мы рассмотрели то в каких состояниях могут быть файлы. А как узнать текущее состояние наших файлов? Для этого используем команду – **git status**. В результате мы можем, например получить то, что наш файл находится в состоянии – **untracked** (при этом он будет подсвечен красным шрифтом). Для того чтобы добавить его в **staging area** или **index** (подготовить к коммиту) используем команду – **git add** название файла, теперь после использования **git status** он уже будет подсвечен зеленым шрифтом и готов для коммита.

***Команды Git:***

* **mkdir git** - создание (**mkdir**) нового каталога с названием - “**git**” (можно просто как обычно создать папку в проводнике и работать уже в ней),
* **cd new\_dir** - переход в другую папку, указываем название папки куда переходим (**new\_dir**), этим способом можно переходить в паки которые непосредственно расположены в папке в которой мы находимся, для других папок надо задавать полный путь,
* **cd ..** - переход в папку на один уровень выше,
* **git --version** - посмотреть версию нашей программы Git,
* **ls -la** - для просмотра какие файлы и папки есть в данном каталоге (**ls**) далее минус (**-**) и два аргумента первый(**l**) выводит данные в виде списка, второй (**a**) показывает все скрытые данные,
* **git init** - команда создания самого репозитория (создаст папку с историей проекта “**.git**”). Причем мы можем применить эту команду для любой паки, где у нас уже есть какие-то проекты или просто файлы, и мы теперь хотим отслеживать все изменения в ней при помощи программы **Git**,
* **git log** - команда, которая показывает нам историю изменений в нашем репозитарии,
* **git log -p** - то же самое, но аргумент **-p** добавит выведенным коммитам некоторые подробности того, что было сделано в раках того или иного коммита,
* **git status** - многозначная команда показывает статус всех файлов, текущее состояние нашего проекта и т.д.,
* **git show** - многогранная команда в зависимости от аргумента меняет свое поведение. Вообще она показывает какой-то объект в его подробностях, например если в качестве аргумента в ней передать хеш коммита, то она предоставит более детальную информацию что было сделано в рамках этого коммита в репозитарии (например, что был добавлен новый файл – у нас в данном случае app.py).
* **vim** - это текстовый редактор, его основная задача — использовать клавиатуру для перемещения по тексту и выполнение модификаций с помощью команд,
* **vim** app.py - с помощью команды **vim** мы можем открыть файл в нашем репозитарии и посмотреть его содержимое. Что бы выйти из vim – сначала нажимаем ESC дальше вводим двоеточие «**:**» потом «**х**» и нажимаем ENTER. При вводе после двоеточия **х** файл будет сохранен, и мы из него выйдем. При воде **w** просто сохранен без выхода, а если ввести **q** просто выйдем из файла (при условии, что мы в нем ничего не меняли, в противном случае он запросит сохранение). Если нам нужно в файле, что-то отредактировать перед этим нужно нажать «**a**».
* Не забываем – сразу **после** произведенных **изменений** в файле их сразу **закоммитить не получится**, измененный файл сначала нужно отправить в индекс (**git add**) и только тогда произведенные в нем изменения можно будет отразить и зафиксировать в очередном коммите (**git commit -m** **“…”**).
* **git restore** app.py - данная команда откатывает изменения в файле app.py на состояние последнего коммита,
* **git restore --staged** app.py - эта команда откатывает индексирование файла (**staging area** или **index**), т.е. удаляет его из **индекса**,
* **git diff** - по сути данная команда показывает какие были изменения внесены в файл с момента последнего коммита. При ее запуске мы увидим, в числе прочего, какие строки в файл были добавлены – передними будет стоять знак плюс «**+**», а какие были удалены, перед ними соответственно будет стоять минус «**-**».
* Но если данный файл мы командой **git add** уже перевели в область **staging area** (**index**) т.е. добавили в область **index** = подготовили к коммиту, то просто запуск **git add** нам ничего не выведет т.к. файл со всеми его изменениями уже в индексе. Для того чтобы увидеть изменения в таком файле (который уже подготовлен к коммиту), в команду **git diff** надо добавить аргумент: **«--staged**». Команда будет выглядеть так: **git diff --staged**.
* **git commit -m** **“…”** - собственно сама команда для создания коммита (**-m** – аргумент для написания комментария к коммиту, сам комментарий пишется в кавычках)
* **git commit -am** **“…”** - такая команда за счет добавления аргумента «**а**» позволяет как бы сразу выполнить две команды: **git add** и **git commit -m** **“…”** т.е. перед коммитом все файлы, которые имеют статус **tracked** и которые были **изменены** (статус **modified**), будут добавлены в **индекс** и тут же будет выполнен сам коммит.
* **git mv** app.py **app\_01.py** - такой командой можно **переименовать** файл **из** app.py **в** app\_01.py или например его можно не только переименовать, но еще и **переместить** в другую папку: **git mv** app.py **C:/Users/Igor/Desktop/git/new\_dir/app\_01.py**.
* **git rm** app.py - команда удаляет указанный файл, но при этом если мы после такого удаления даже еще **не делали коммит** - файла в папке **уже не будет!** Как будто мы уже проделали с файлом изменения и проиндексировали это и нам якобы осталось только сделать сам коммит, но файла то уже нет... Соответственно и откатить коммит так, чтобы удаленный файл появился мы не сможем (я так предполагаю). Можно выполнить команду, которая удалит файл в коммите, но оставим его в каталоге (заменив его статус с **tracked** на **untracked**), сама команда: **git rm --cached** app.py.
* **git pull** - команда используется для извлечения и загрузки содержимого из удаленного репозитория и немедленного обновления локального репозитория этим содержимым.
* **git push -u origin main** - команда пуша локального депозитария на удаленный
* **git remote add origin https://github.com/zhemgulis/Shpori\_Python.git** - соединение с GH
* **git branch -M main** - создание новой ветки и присвоение ей статуса "основная"