## 4.1 Установка Git на компьютер

### 1. Установка Git на macOS

Перед тем как устанавливать Git надо проверить, а не установлен ли он уже, с помощью команды: git --version

Если результатом будет версия Git, то есть ответ, типа: git version 2.38.1

тогда можно еще попробовать обновить Git, а то вдруг это не самая последняя версия. А если Git вообще не установлен, то его, конечно, нужно поставить. И установка и обновление происходит с помощью уже знакомого нам homebrew.

Чтобы установить Git - выполните в терминале команду: brew install git

А чтобы обновить Git - команду: brew upgrade git

### 2. Установка Git на Ubuntu

Если команда git --version

покажет, что гит не установлен, то есть ответ будет:

Command 'git' not found, but can be installed with:

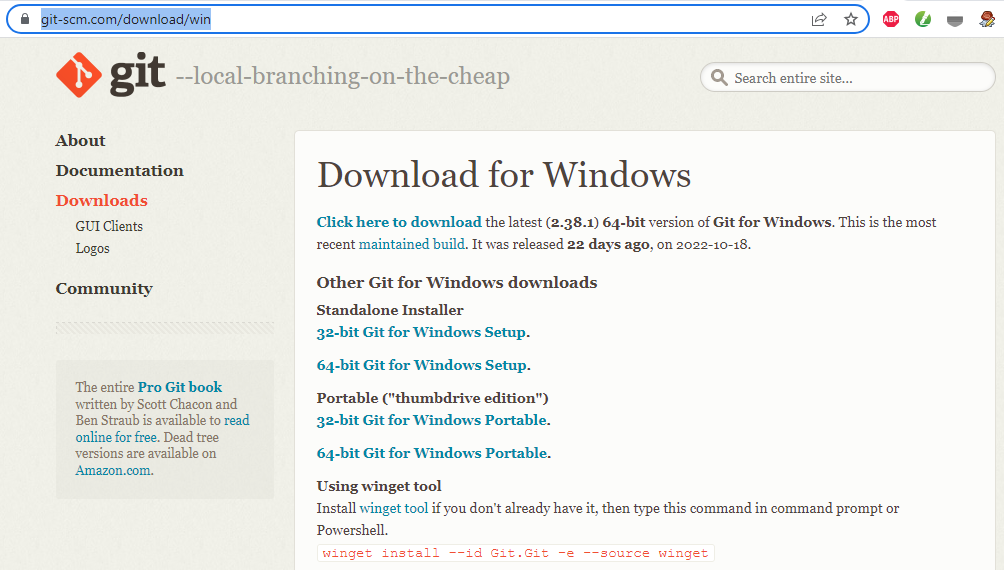
apt install git

Please ask your administrator.

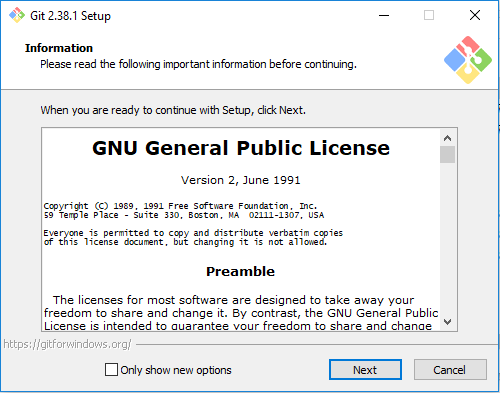
уже в нем самом и будет содержаться инструкция, что делать. Нужно с правами суперпользователя выполнить команду: apt install git

### 3. Установка Git на Windows

Скачать Git для Windows можно по ссылке: <https://git-scm.com/download/win>



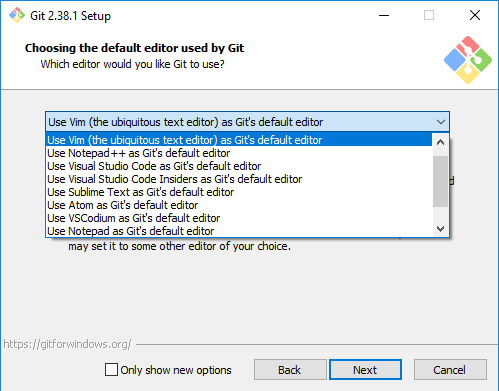
В процессе установки можно оставить все по умолчанию, для наших задач хватит.



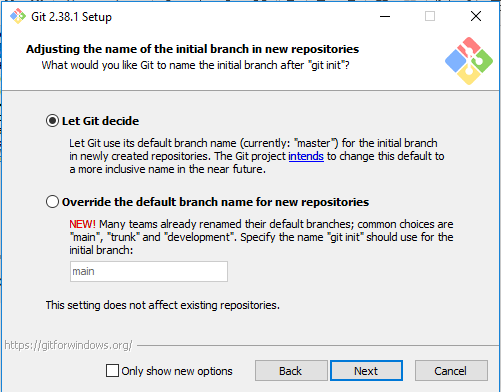
В какой-то момент вам будет предложено выбрать редактор, в котором, по умолчанию, будут открываться файлы Git. Если ничего не выбирать, то таким редактором будет Vim. Многим он кажется сложным, наверное, потому что они не знают как из него выйти :) Можете выбрать какой-то другой.

Но если оставите Vim - выйти из него без сохранения можно будет командой: :q! или с сохранением: :wq

Но вообще, конечно, Vim - это очень мощный инструмент, которым имеет смысл научиться пользоваться также, как и командной строкой, хотя бы потому, что когда вы будете подключаться к удаленному серверу, чаще всего никакого графического интерфейса взаимодействия с сервером не будет - все через терминал и Vim, который по умолчанию есть в любой системе. Впрочем, сейчас не об этом. Сейчас Git на Windows устанавливаем.



Еще в процессе установки Git предложит вам решить, как будет называться ветка, в которую будут добавляться коммиты по умолчанию. Раньше такая ветка всегда называлась master, но в свете новых модных веяний ее можно переименовать, например, в main. Тут решайте сами. В принципе, если очень захочется, переименовать любую ветку можно будет и в процессе работы. Лично я сторонник классики и использую название master.



После установки нужно будет перезагрузить компьютер, чтобы Git стал доступен глобально из командной строки. Как обычно, версию Git можно будет проверить командой:

git version

**Как работает Git**

Ну, вот, значит, услышали мы, что без гита современному программисту никак нельзя, установили, значит, этот самый Git на компьютер. А дальше что? Как это все работает? Какая логика должна быть в наших действиях, чтобы эффективно гитом пользоваться?

На пальцах. Git - это программа, которая следит за репозиториями и всеми изменениями в них. То есть, если вы инициируете Git в корне проекта, который хотите отслеживать, то в проекте появляется репозиторий - специальная директория для хранения изменений. Добавили новый файл или непустую директорию - Git запомнил; как-то поменяли существующий файл - Git запомнил; что-то удалили - Git снова запомнил. Причем, программа эта с ручным управлением, то есть без вашего ведома никакие изменения не фиксируются и вы сами решаете когда и какие из них нужно сохранить. "Закоммитить" - если пользоваться терминологией гита. Еще можно смотреть на Git как на базу данных, в которой хранятся все изменения по отслеживаемому проекту.

Согласно идеологии Git, файлы в проекте, где есть репозиторий Git могут быть:

1. Неотслеживаемыми (Untracked)
2. Отслеживаемыми

**Неотслеживаемые** файлы - это те, которые есть в директории проекта, но которые еще не добавлены в репозиторий, то есть Git их видит, сообщает о том, что они есть в проекте, но не хранит их в репозитории и не следит за их изменениями. Если мы скопируем репозиторий, чтобы перенести, например, на другой компьютер, этих файлов там не будет.

Среди неотслеживаемых, наверное, можно выделить еще один подкласс файлов - игнорируемые - это те файлы, о которых мы сообщили гиту, что их не надо отслеживать в принципе. Названия таких файлов или шаблоны названий прописываются в **специальном файле .gitignore**, которому будет посвящен отдельный шаг чуть дальше.

**Отслеживаемые файлы** - это те, которые находятся под версионным контролем, то есть те, о которых Git знает, и за изменениями в которых следит. Они могут находиться в трех состояниях:

1. Неизмененные (Unmodified)
2. Измененные (Modified)
3. Подготовленные к коммиту (Staged)

Когда мы только инициализируем Git в корне проекта - все файлы имеют статус неотслеживаемых, и чтобы начать их отслеживать:

- сначала нужно добавить их в так называемый индекс, где они получат статус staged,

- а уже из индекса с файлов делается "слепок" текущего состояния, который сохраняется в гит с временными метками, данными автора "слепка" и комментариями. Состояние, зафиксированное в Git остается там навсегда, с возможностью вернуться к нему в любой момент, даже если в будущем в проекте не останется ни одного файла, которые в нем были изначально.

Еще раз. Мы инициализируем Git в новом проекте. Постепенно добавляем в проект файлы, что-то с ними делаем. Получили часть работы, которую нужно зафиксировать - добавляем файлы сначала в индекс, а уже из индекса фиксируем (коммитим) их состояние на текущий момент, добавляя при этом комментарии к коммиту.

Если нам нужно поделиться проектом с кем-то - достаточно зафиксировать состояние всех необходимых файлов проекта и клонировать репозиторий на удаленный сервер, откуда, в свою очередь, кто-то сможет клонировать репозиторий к себе на локальный компьютер. Через удаленный сервер проект будет синхронизирован между всеми участниками. Важно только периодически свои коммиты отправлять на сервер и получать с сервера новые коммиты других участников.

## Конфигурирование Git

Надеюсь, из предыдущего шага стало более-менее понятно как все устроено. Прежде чем начать пользоваться Git - его требуется немного настроить.

Первым делом нужно **инициировать создание репозитория Git** в проекте, который вы собираетесь отслеживать. То есть, поступить можно, например, так:

1. Создаем папку, в которой будет находиться проект (либо через командную строку, либо через графический интерфейс операционной системы)

2. Запускаем VS Code

3. В VS Code выбираем **File - Open Folder...** - ищем созданную папку и открываем

4. В VS Code запускаем терминал **Terminal - New Terminal**

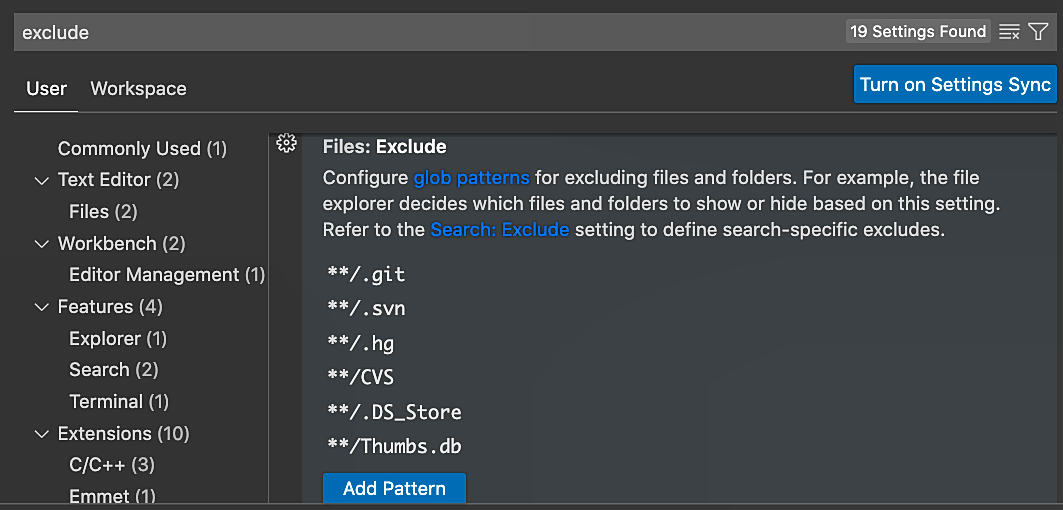
5. В терминале вы сразу окажетесь внутри папки проекта

Понятно, что в папку проекта можно попасть и просто через терминал, последовательно исполняя команду cd <имя папки>, но т.к. мы с вами настраивали инструменты для разработки - удобно все делать в одном месте.

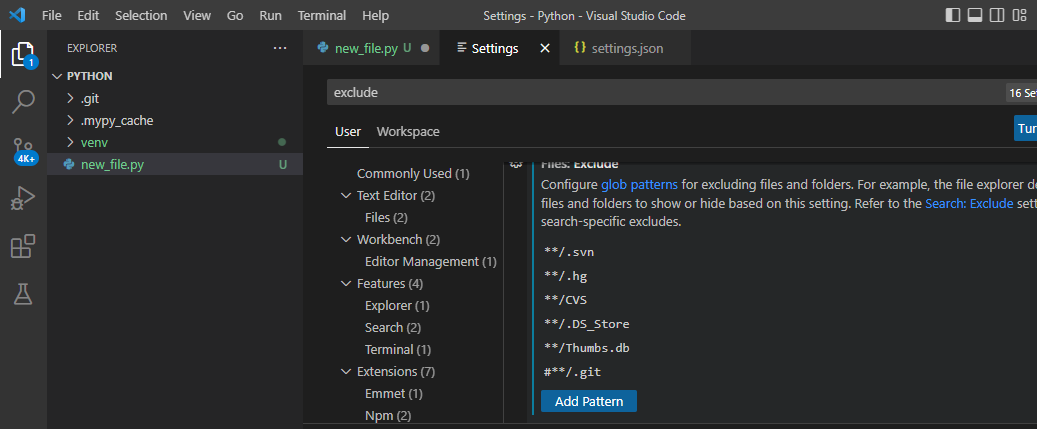
Теперь, находясь в корне проекта, нужно **инициализировать Git командой: git init**

После ее выполнения в директории проекта будет создан репозиторий .git.

Скорее всего вы не увидите его в VS Code, потому что, по умолчанию, VS Code репозитории не показывает. Но он там есть :) Если очень хочется, чтобы репозиторий был виден в VS Code - можно перейти в настройки, набрать в строке поиска "exclude" и найти раздел Files: Exclude.



Чтобы сделать Git-репозитории видимыми в VS Code, нужно либо удалить паттерн \*\*/.git, либо исправить, например, добавив вначале значок решетки #. После того как вы это сделаете и примените новые настройки, папка .git отобразится в проводнике VS Code.



Убедились, папка есть. Теперь давайте зададим глобальные настройки пользователя - **username** и **email**. Тогда для каждого нового проекта они будут подхватываться автоматически. По этим данным всегда можно будет понять, кто именно сделал те или иные изменения, то есть в каждый коммит будет добавляться не только дата и изменения, но и их автор.

Набираем команды:

git config --global user.name "Mikhail Kryzhanovskiy"

и

git config --global user.email "kms101@yandex.ru"

Понятно, что вместо Mikhail Kryzhanovskiy и kms101@yandex.ru в кавычках вы должны будете ввести свои имя и почту. Можно увидеть только что добавленные значения параметров конфигурации, а также те, что были там ранее, с помощью команды:

git config --list

Ответ будет примерно следующего содержания:

Air-Mihail:Python mikhail$ git config --list

credential.helper=osxkeychain

user.name=Mikhail Kryzhanovskiy

user.email=kms101@yandex.ru

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

core.ignorecase=true

core.precomposeunicode=true

**Примечание 1.** Чтобы получить справку по Git прямо в терминале - можно набрать команду git --help или, что одно и то же, git -h. Чтобы **получить справку** по конкретной команде Git - можно выполнить команду:

git -h <команда>

**Примечание 2.** Если опция к команде может быть записана без пробелов, кавычки ставить не обязательно. То есть, в нашем случае, когда мы в глобальные настройки пользователя добавляли email - можно было кавычки не писать и команда

git config --global user.email [kms101@yandex.ru](mailto:kms101@yandex.ru)

равнозначна команде

git config --global user.email [kms101@yandex.ru](mailto:kms101@yandex.ru)

**Примечание 3.** Глобальные настройки - это настройки в рамках одного пользователя в операционной системе. То есть если компьютером пользуются несколько пользователей с разными учетными записями - у каждого пользователя могут быть свои глобальные настройки, которые не будут пересекаться. Также в Git могут быть системные (--system) и локальные (--local) настройки. Системные настройки - это настройки в масштабах всей операционной системы, а локальные - в масштабах одного проекта.

**Примечание 4.** Команда git config --list выводит все настройки из конфигов всех уровней. Если же надо посмотреть настройки какого-то определенного уровня, то к команде добавляется один из флагов (--system, --global, --local), например, чтобы посмотреть глобальные настройки, команда будет:

git config --list –global

**Примечание 5.** Чтобы посмотреть сразу все настройки и где они хранятся - можно выполнить команду:

git config --list --show-origin

**Коммиты**

**Коммиты** в идеологии Git - **это зафиксированные изменения**.

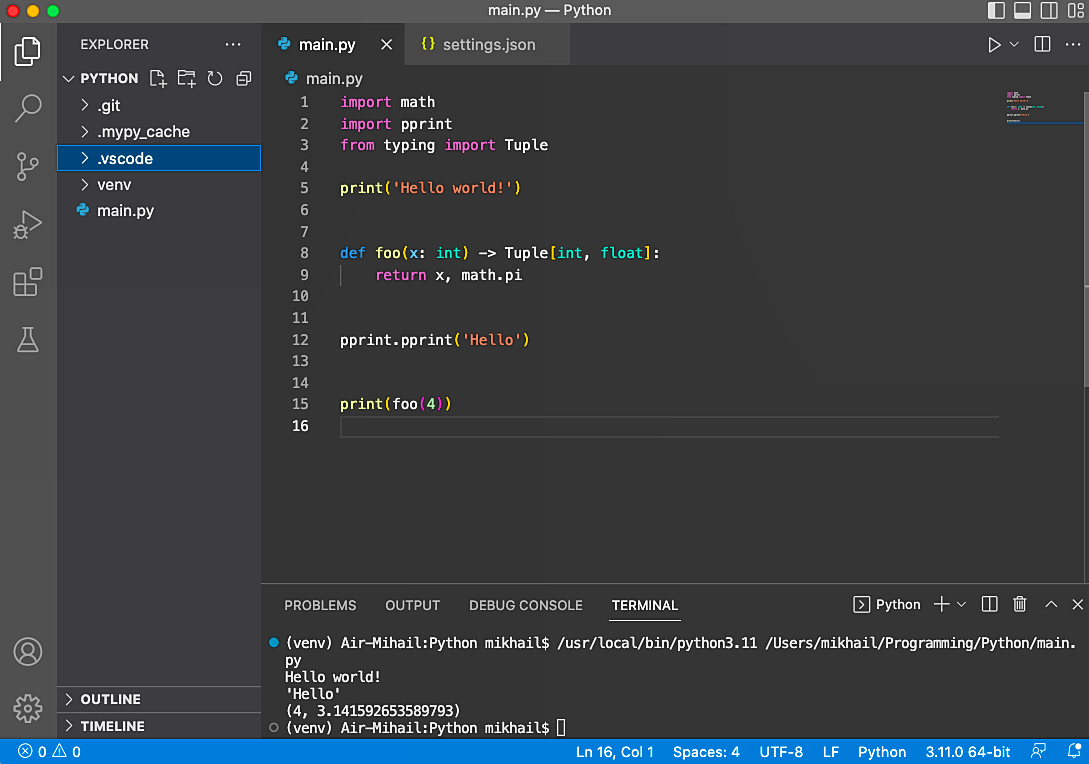
То есть, допустим, вы поработали с проектом, исправив какой-то баг - имеет смысл зафиксировать факт того, что баг был исправлен. Или вы отрефакторили громоздкую функцию, разбив ее на несколько небольших и более понятных - также имеет смысл сообщить гиту, команде и будущему себе, что рефакторинг функции выполнен. Добавили новую фичу - коммит, оптимизировали небольшой участок кода - коммит, дали старым переменным новые хорошие имена - коммит. И так далее. Если придерживаться правила, что любой логически законченный блок работы должен быть зафиксирован коммитом - и вам, и команде будет намного легче ориентироваться в коде проекта и проще находить его недостатки, если вдруг что-то идет не так.

Коммитить, в целом, не очень сложно, если не вдаваться глубоко в детали.

В общем случае, при условии, что в проекте уже инициализирован Git, **алгоритм** такой:

1. Работаем над проектом и понимаем, что пора зафиксировать какое-то изменение
2. С помощью команды git add <имя\_файла> добавляем файл в индекс
3. С помощью команды git commit -m "<комментарий к коммиту>" фиксируем текущее состояние

Давайте посмотрим как это работает на практике. От предыдущих уроков у нас уже осталась директория с питоновским файлом, виртуальным окружением, репозиторием Git и служебными папками.



Чтобы **посмотреть в каком сейчас состоянии у нас файлы в проекте** с точки зрения Git - нужно выполнить команду:

git status

|  |  |
| --- | --- |
| Получим  список  неотслеживаемых файлов. | https://ucarecdn.com/3b99e78b-1973-4a6f-ab24-8300c04b69d4/-/preview/-/enhance/83/ |

Git сообщает, что индекс пуст, то есть коммитить нечего, но при этом есть неотслеживаемые файлы. Тут сразу хочу обратить внимание вот на что.

|  |  |
| --- | --- |
| Смотрите, в проводнике VS Code у нас показаны 4 папки и один файл, а неотслеживаемых объектов Git показал всего три. Ну, допустим, с папкой .git понятно - нет смысла отслеживать папку, которая и так отслеживается, а почему папка .mypy\_cache не попала в список неотслеживаемых файлов? Дело в том самом файле .gitignore, который был автоматически создан при установке линтера mypy. В этом можно убедиться, если зайти в папку: | https://ucarecdn.com/966043a1-19fb-4546-a865-17060f2d2cda/-/preview/-/enhance/74/ |

Файлу .gitignore посвящен следующий шаг, просто заранее снимаю ваш вопрос, если он вдруг возник.

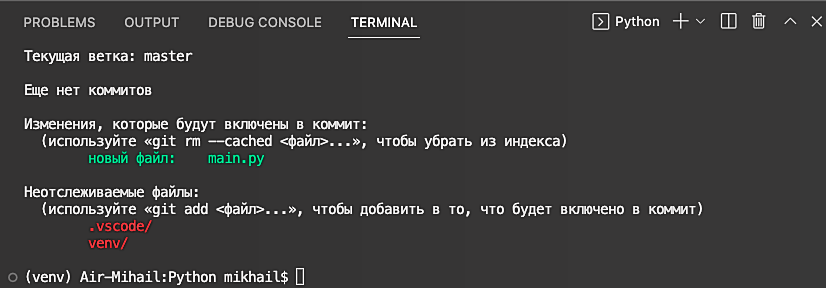
Итак, файлы виртуального окружения и служебные файлы VS Code отслеживать нет необходимости, нужные проекту зависимости хранят в файле requirements.txt, про который мы обязательно вспомним, когда будем разворачивать бота на удаленном сервере.

Поэтому, на текущий момент у нас есть только один файл для отслеживания - это main.py. Давайте **добавим его в индекс**. Для этого выполним команду:

git add main.py

И снова посмотрим статус репозитория. Теперь видно, что файл main.py позеленел и его можно добавить в очередной коммит. Также есть подсказка, что если какой-либо файл нужно удалить из индекса - можно выполнить команду:

git rm --cached <файл>



Но мы пока ничего удалять не будем, а наоборот, закоммитим текущее состояние файла командой:

git commit -m "Это наш первый коммит в проекте"

Получим сообщение, типа:

(venv) Air-Mihail:Python mikhail$ git commit -m "Это наш первый коммит в проекте"

[master (корневой коммит) 17368ea] Это наш первый коммит в проекте

1 file changed, 15 insertions(+)

create mode 100644 main.py

И давайте сразу проверим статус Git. Сообщение будет вида:

Текущая ветка: master

Неотслеживаемые файлы:

(используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)

.vscode/

venv/

индекс пуст, но есть неотслеживаемые файлы

(используйте «git add», чтобы проиндексировать их)

Обратите внимание на **число 17368ea** в сообщении, которое появилось после того, как мы создали первый коммит. Это именно число, правда, в 16-ричной системе счисления. Это часть идентификатора коммита, по которому можно посмотреть что это за коммит. Если выполнить команду (не забудьте только указать номер вашего, а не моего коммита):

git show 17368ea

|  |  |
| --- | --- |
| Можно увидеть информацию по коммиту. | https://ucarecdn.com/cbbacddd-6a4c-4a9d-8ec2-e1087cf4c703/-/preview/-/enhance/81/ |

**Примечание 1.** Если в индекс нужно добавить все неотслеживаемые и модифицированные файлы проекта, то можно пользоваться командой:

git add .

**Примечание 2.** Если **закоммитить** надо не весь индекс, а только **некоторые файлы**, например, чтобы разбить выполненную работу на несколько разных коммитов, можно выполнить команду:

git commit -m <файл\_1> <файл\_2> <файл\_3> "Комментарий к коммиту"

Это также является ответом на вопрос, почему сразу нельзя коммитить из рабочей директории, зачем нужен какой-то индекс. Вот, как раз для того, чтобы иметь возможность большую проделанную работу разбить на небольшие, логически завершенные части с помощью нескольких коммитов. Впрочем, Git позволяет коммитить и минуя индекс, но это уже более продвинутая история.

**Примечание 3.** После добавления файлов в индекс - закоммитятся они именно в состоянии, в котором были добавлены в индекс. То есть, если вы добавили файл в индекс, а потом продолжили работать с ним - в индексе будет версия файла, которая была на момент добавления в индекс. Когда вы закоммитите файл из индекса - git status покажет, что в этом файле снова есть изменения и его опять можно добавить в индекс.

**Примечание 4.** Флаг -m во время коммита говорит о том, что не нужно открывать текстовый редактор для написания комментария, а можно взять его сразу из опций команды (мы его записывали в кавычках). Если этот флаг не указать, то откроется текстовый редактор, в котором нужно будет написать комментарий к коммиту.

## Файл .gitignore

Не **все файлы следует отслеживать** в Git, а тем более, отправлять на GitHub. Во-первых, потому что это могут быть чувствительные данные, типа токенов, паролей доступа, приватных ключей и т.п. Во-вторых, потому что это могут быть служебные файлы, создаваемые IDE в процессе разработки или файлы кэша Python. В-третьих, у вас могут быть в проекте файлы, в которых вы тестируете работу каких-нибудь библиотек, функций и так далее, чтобы потом перенести рабочий вариант в предназначенные для этого модули.

Чтобы Git не отслеживал то, что ему не следует отслеживать - создают специальный файл .gitignore, в котором прописывают то, что отслеживать не нужно. Чаще всего файл .gitignore помещается в корень проекта и все пути в нем указываются от корня. Содержимое такого файла может выглядеть, например, так:

### Python template

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# C extensions

\*.so

# Distribution / packaging

.Python

env/

build/

develop-eggs/

dist/

downloads/

eggs/

.eggs/

lib/

lib64/

parts/

sdist/

var/

wheels/

\*.egg-info/

.installed.cfg

\*.egg

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.pytest\_cache/

.coverage

.coverage.\*

.cache

nosetests.xml

coverage.xml

\*,cover

# Sphinx documentation

docs/\_build/

# virtualenv

.venv

venv/

ENV/

# JetBrains

.idea/

# Current project

experiment.py

# Doc's

docs/html

# i18n/l10n

\*.mo

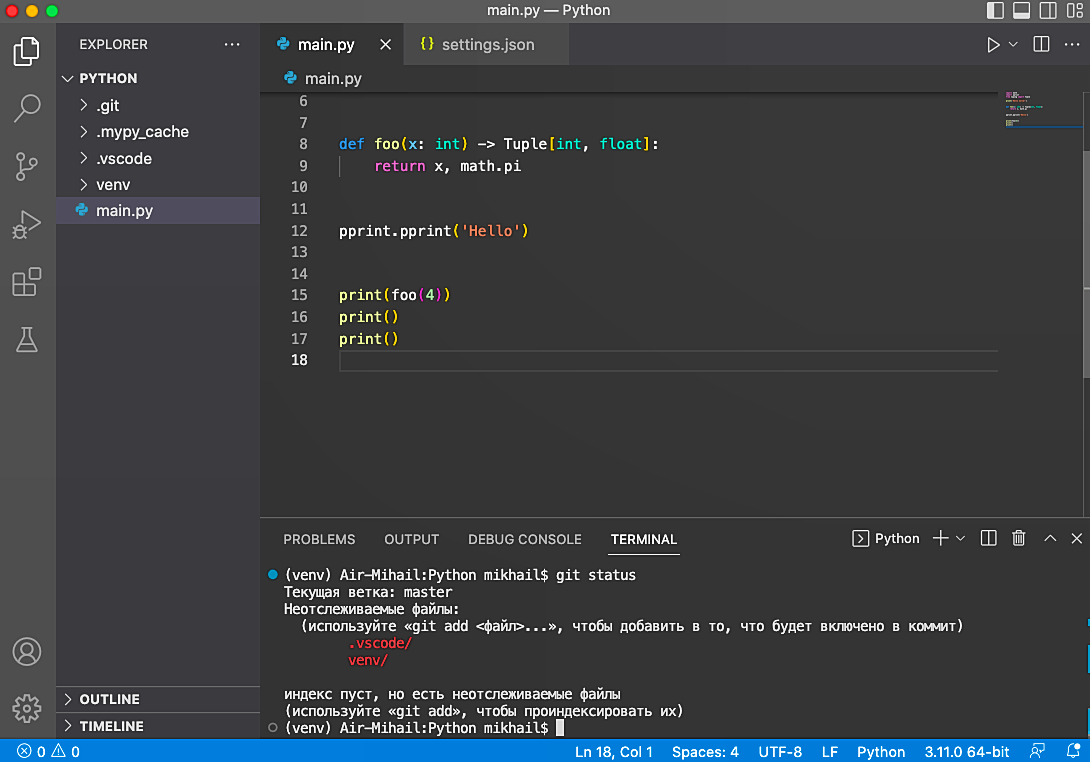
# pynev

.python-version

Нам, конечно, такой большой не понадобится, но здесь я просто хочу показать пример. Строки, которые начинаются со значка # - это комментарии, как и в Python. Остальные строки, начинающиеся каждая с новой строки, это названия файлов и директорий, которые Git не должен отслеживать. Также, помимо прямых названий файлов и директорий, используются шаблоны. Например, запись вида \*.co говорит гиту игнорировать файлы с любыми названиями, если у них расширение .co.

В общем, то, что вы не хотите добавлять в систему контроля версий - то и записывайте в .gitignore.

А в качестве небольшой практики, давайте с вами уберем из списка отслеживаемых директорию виртуального окружения и служебную папку VS Code. Если сейчас выполнить команду git status - можно увидеть, что папки не отслеживаются, но Git предлагает добавить их в индекс.



Нам это, понятное дело, не нужно, поэтому, как вы могли догадаться, достаточно создать в корне проекта этот самый файл .gitignore и добавить в него названия папок - каждое с новой строки.

.vscode/

venv/

Если теперь сохранить файл .gitignore и выполнить команду git status, то папок .vscode/ и venv/ мы не увидим, зато увидим теперь среди неотслеживаемых сам файл .gitignore. Его, кстати, тоже можно добавить в самого себя, тогда и он будет игнорироваться гитом.

