

Тема 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ *PYTHON*

Оглавление

1.1 История создания и краткая характеристика языка.....	2
1.2 Сферы применения.....	4
1.3 Python IDE	6
1.4 Работа с командой строкой на Python.....	7
1.4.1 Установка Python	7
1.4.2 Настройка окружения.....	12
1.5 Взаимодействие с операционной системой	17
1.5.1 Получение информации из ОС.....	18
1.5.2 Проверка существования пути	20
1.5.3 Создание директорий.....	22
1.5.4 Удаление файлов и директорий.....	24
1.5.5 Запуск на исполнение.....	25
1.5.6 Переименование.....	26
1.5.7 Содержимое директорий.....	27
1.5.8 Информация о файлах и директориях.....	28
1.5.9 Обработка путей.....	29
1.6 Хостинг ИТ-проектов и управление версиями.....	30
1.6.1 Создание репозитория GitHub.....	30
1.6.2 Загрузка проектов через веб-интерфейс GitHub.....	31
1.6.3 Загрузка проектов через интерфейс командной строки Git	34
1.6.4 Работа с локальным репозиторием	38
1.6.5 Управление версиями проектов.....	41

ТЕМА 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ *PYTHON*

1.1 История создания и краткая характеристика языка

Создание *Python* было начато Гвидо ван Россумом (*Guido van Rossum*) в 1991 году в процессе работы над распределенной операционной системой *Amoeba*, для которой требовался расширяемый язык с поддержкой системных вызовов. За основу были взяты *ABC* и *Modula-3*.

Название "*Python*" было выбрано в честь комедийных серий *BBC "Monty Python's Flying Circus"* («Летающий цирк Монти-Пайтона»).

Особенно активно язык совершенствуется в настоящее время, когда над ним работает не только команда создателей, но и целое сообщество программистов со всего мира. И все-таки последнее слово о направлении развития языка остается за Гвидо ван Россумом.

История версий *Python*:

- Гвидо Ван Россум опубликовал первую версию кода *Python* (версия 0.9.0) в 1991 году. Он уже включал в себя ряд полезных возможностей. Например, различные типы данных и функции для обработки ошибок.
- В версии *Python* 1.0, выпущенной в 1994 году, были реализованы новые функции для простой обработки списка данных: сопоставление, фильтрация и сокращение.
- *Python* 2.0 был выпущен 16 октября 2000 года с новыми полезными функциями для программистов, такими как поддержка символов *Unicode* и упрощенный способ циклического просмотра списка.
- 3 декабря 2008 года вышел *Python* 3.0. Эта версия включала функцию печати и дополнительную поддержку деления чисел и обработки ошибок.

Python является мультипарадигменным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование, функциональное программирование и асинхронное программирование. Задачи обобщенного программирования

решаются за счет динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений.

Основные архитектурные черты – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора (GIL), высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией *Python* является интерпретатор *CPython*, который поддерживает большинство активно используемых платформ, являющийся стандартом де-факто языка. Он распространяется под свободной лицензией *Python Software Foundation License*, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. *CPython* компилирует исходные тексты в высокоуровневый байт-код, который исполняется в стековой виртуальной машине. К другим трём основным реализациям языка относятся *Jython* (для *JVM*), *IronPython* (для *CLR/.NET*) и *PyPy*. *PyPy* написан на подмножестве языка *Python* (*RPython*) и разрабатывался как альтернатива *CPython* с целью повышения скорости исполнения программ, в том числе за счёт использования *JIT*-компиляции.

Поддержка *Python2* закончилась в 2020 году. На текущий момент активно развивается версия языка *Python3*. Разработка языка ведётся через предложения по расширению языка *PEP* (англ. *Python Enhancement Proposal*), в которых описываются нововведения, делаются корректировки согласно обратной связи от сообщества и документируются итоговые решения.

Язык портирован (адаптирован) и работает почти на всех известных платформах – от карманных персональных компьютеров до промышленных вычислительных систем. В отличие от многих портируемых систем, для всех основных платформ *Python* имеет поддержку характерных для данной платформы технологий. Более того, существует специальная версия *Python* для виртуальной

машины *Java* – *Jython*, что позволяет интерпретатору выполняться на любой системе, поддерживающей *Java*, при этом классы *Java* могут непосредственно использоваться из *Python* и даже быть написанными на *Python*. Также несколько проектов обеспечивают интеграцию с платформой *Microsoft.NET*, основные из которых – *IronPython* и *Python.NET*.

1.2 Сферы применения

Стандартными примерами использования *Python* являются:

- веб-разработка на стороне сервера;
- автоматизация с помощью скриптов *Python*;
- наука о данных и машинное обучение;
- разработка программного обеспечения;
- автоматизация тестирования программного обеспечения.

Веб-разработка на стороне сервера включает в себя сложные серверные функции, с помощью которых веб-сайты отображают информацию для пользователя. Например, веб-сайты должны взаимодействовать с базами данных и другими веб-сайтами, а также защищать данные при их отправке по сети.

Python полезен при написании серверного кода, поскольку он предлагает множество библиотек, состоящих из предварительно написанного кода для сложных серверных функций. Также разработчики используют широкий спектр платформ *Python*, которые предоставляют все необходимые инструменты для более быстрого и простого создания интернет-приложений. Например, разработчики могут создать «скелет» интернет-приложения за считанные секунды, потому что им не нужно писать код с нуля. Затем его можно протестировать с помощью инструментов тестирования платформы независимо от внешних инструментов тестирования.

Язык скриптов – это язык программирования, который автоматизирует задачи, обычно выполняемые людьми. Программисты широко используют скрипты *Python* для автоматизации многих повседневных задач, среди которых:

- одновременное переименование большого количества файлов;
- преобразование файла в другой тип файла;

- удаление повторяющихся слов в текстовом файле;
- выполнение базовых математических операций;
- отправка сообщений электронной почты;
- загрузка контента;
- выполнение базового анализа журналов;
- поиск ошибок в нескольких файлах.

Наука о данных извлекает ценную информацию из данных, а машинное обучение (*ML*) позволяет компьютерам автоматически учиться на данных и делать точные прогнозы. Специалисты по работе с данными используют *Python* для решения следующих задач:

- исправление и удаление неверных данных (очистка данных);
- извлечение и выбор различных характеристик данных;
- разметка данных добавляет данным значимые имена;
- поиск статистической информации в данных;
- визуализация данных с помощью диаграмм и графиков: линейных диаграмм, столбчатых диаграмм, гистограмм и круговых диаграмм.

Специалисты по работе с данными используют библиотеки *Python ML* для моделей машинного обучения и создания классификаторов, которые точно классифицируют данные. Классификаторы на основе *Python* используются в различных областях и применяются для выполнения таких задач, как классификация изображений, текста и сетевого трафика, распознавание речи и распознавание лиц. Специалисты по работе с данными также используют язык для глубокого обучения – передовой техники машинного обучения.

Разработчики программного обеспечения часто используют *Python* для различных задач разработки и программных приложений, среди которых:

- отслеживание ошибок в программном коде;
- автоматическая сборка программного обеспечения;
- управление программными проектами;
- разработка прототипов программного обеспечения;
- разработка настольных приложений с использованием библиотек графического пользовательского интерфейса (ГПИ);
- разработка игр: от простых текстовых игр до сложных видеоигр.

Тестирование программного обеспечения – процесс проверки соответствия фактических результатов программного обеспечения ожидаемым результатам, который позволяет убедиться, что программное обеспечение не содержит ошибок.

Разработчики используют среды модульного тестирования *Python* (*Unittest*, *Robot* и *PyUnit*) для тестирования написанных функций.

Тестировщики программного обеспечения используют *Python* для написания тестовых примеров для различных сценариев. Например, язык применяется для тестирования пользовательского интерфейса интернет-приложения, нескольких программных компонентов и новых функций.

Разработчики могут использовать несколько инструментов для автоматического запуска тестовых скриптов. Эти инструменты известны как инструменты непрерывной интеграции / непрерывного развертывания (*CI/CD*). Тестировщики и разработчики программного обеспечения используют инструменты *CI/CD* (*Travis CI* и *Jenkins*) для автоматизации процесса тестирования. Инструмент *CI/CD* автоматически запускает тестовые скрипты *Python* и сообщает о результатах тестирования всякий раз, когда разработчики вносят новые изменения в код.

1.3 *Python IDE*

Интегрированная среда разработки (*IDE*) – программное обеспечение, которое предоставляет разработчикам инструменты для написания, редактирования, тестирования и отладки кода.

PyCharm – продукт чешской компании *JetBrains* по разработке программных инструментов. У программы имеется как бесплатная версия для небольших приложений, так и платная профессиональная версия, подходящая для создания крупных приложений *Python* со следующим набором функций:

- автоматическое завершение и проверка кода;
- обработка и быстрое устранение ошибок;
- чистка кода без изменения функциональных возможностей;
- поддержка платформ интернет-приложений, таких как *Django* и *Flask*;

- поддержка других языков программирования, таких как *JavaScript*, *CoffeeScript*, *TypeScript*, *AngularJS* и *Node*;
- научные инструменты и библиотеки, такие как *Matplotlib* и *NumPy*;
- возможность запуска, отладки, тестирования и развертывания приложений на удаленных виртуальных машинах;
- отладчик для поиска ошибок в коде, профилировщик для выявления проблем с производительностью и средство запуска модульных тестов;
- поддержка баз данных.

Интегрированная среда разработки и обучения (*IDLE*) – интегрированная среда разработки *Python*, установленная по умолчанию. Среда разработана только на *Python* с использованием набора инструментов *Tkinter GUI* и совместима с *Windows*, *Unix* и *macOS*.

Spyder – *IDE* с открытым исходным кодом, которую используют многие специалисты и аналитики данных. Она применяется для всесторонней разработки с использованием функций расширенного анализа данных, визуализации и отладки. Среда имеет следующие особенности:

- редактор кода, поддерживающий несколько языков;
- интерактивная консоль *IPython*;
- базовый отладчик;
- научные библиотеки, такие как *Matplotlib*, *SciPy* и *NumPy*;
- возможность исследования переменных в коде;
- возможность просмотра документации в режиме реального времени.

При выборе *IDE* следует учитывать не только функциональные возможности, но и личные предпочтения.

1.4 Работа с командой строкой на Python

1.4.1 Установка Python

Для начала работы с *Python* в командной строке (терминале) *Windows* необходимо установить сам язык. Скачать установочный файл можно с помощью *Microsoft Store* (рис. 1.1).

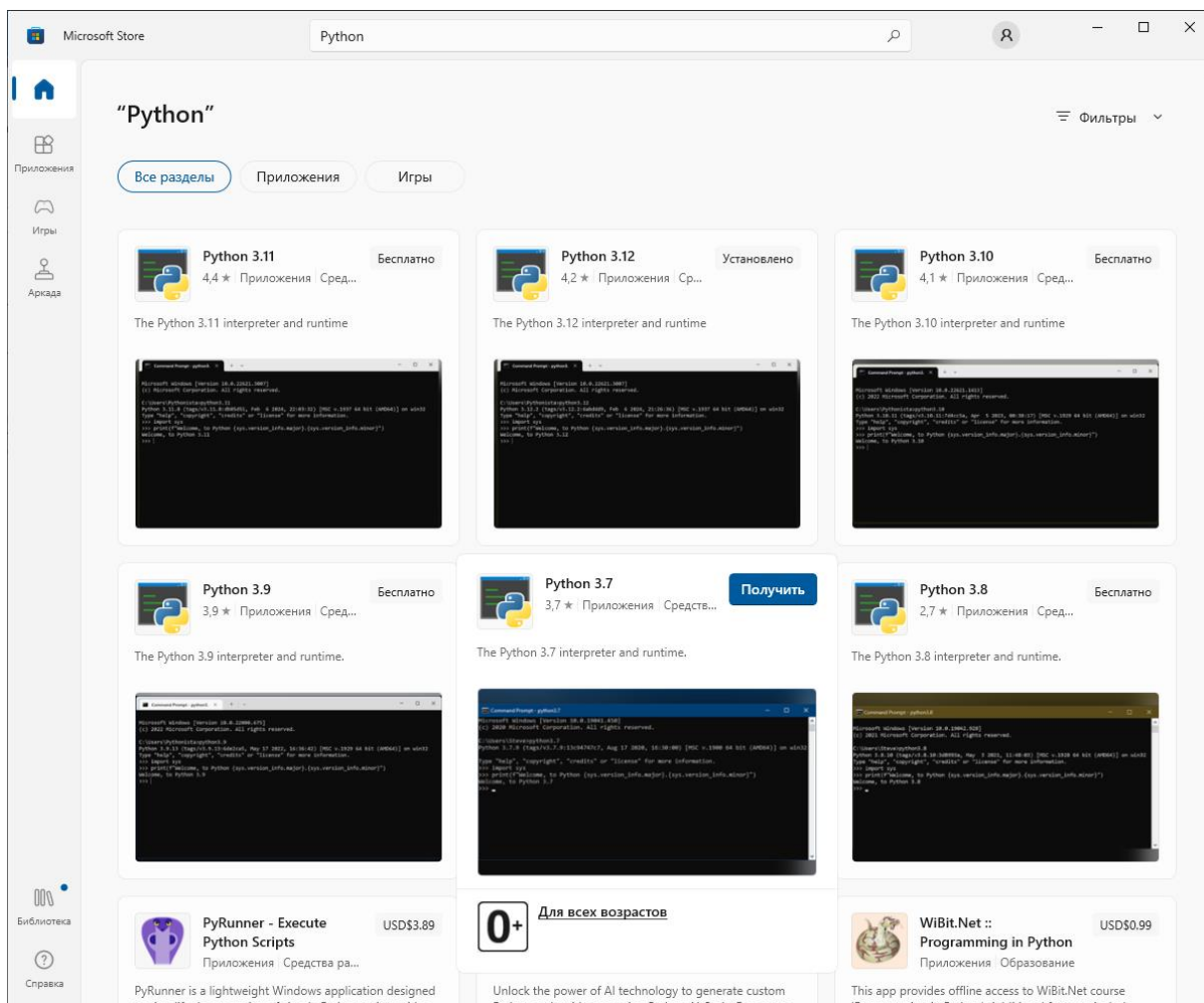


Рис. 1.1. Различные версии интерпретатора Python в Microsoft Store

Следующим шагом является запуск терминала *Windows* с помощью строки поиска (рис. 1.2) или сочетания горячих клавиш *Win+R* с последующим вводом команды *cmd.exe* в появившемся окне (рис. 1.3).

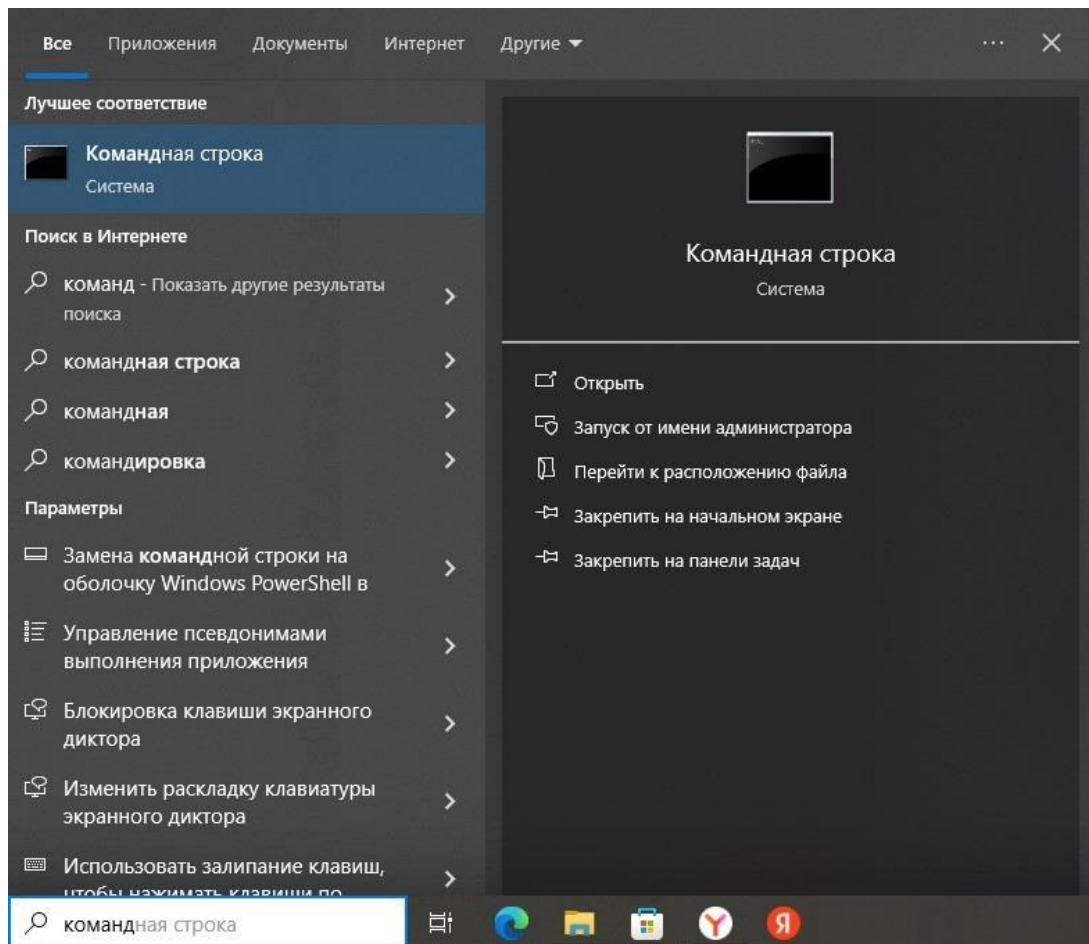


Рис. 1.2. Запуск командной строки с помощью интерфейса поиска

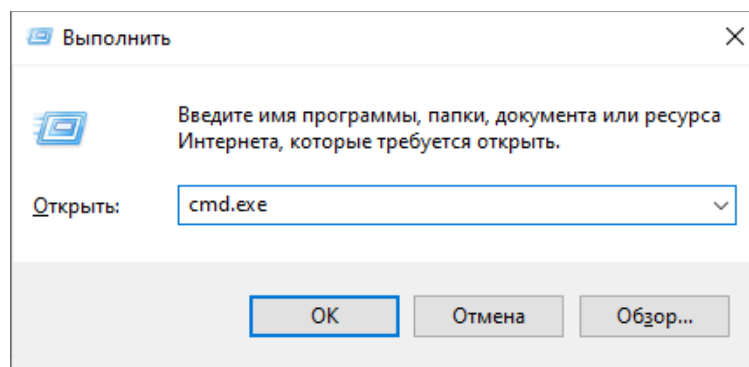


Рис. 1.3. Запуск командной строки через Win+R

Начать работу с интерпретатором языка *Python* следует с команды *python* или *python3* для *Windows* (рис. 1.4).

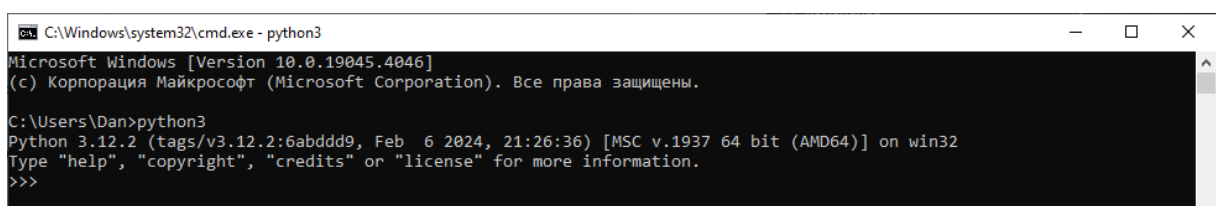
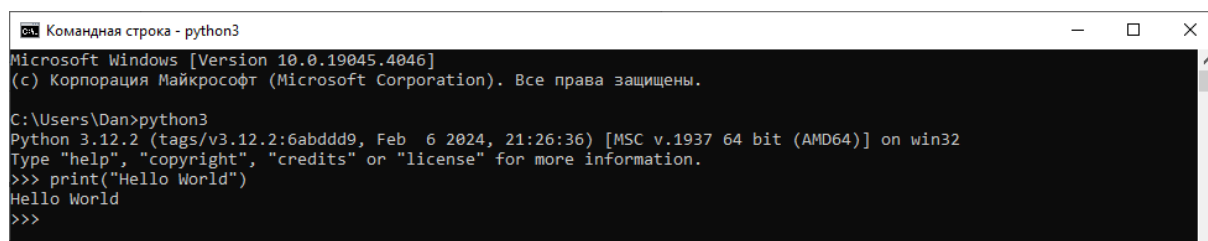


Рис. 1.4. Начало работы с интерпретатором Python

После запуска *Python* командная строка изменилась на символы ">>>". Данный факт означает возможность использования только команд рассматриваемого языка программирования.

Для возвращения к терминалу операционной системы требуется ввести *exit()* или использовать сочетание клавиш *Ctrl+Z* для *Windows*.

Рассмотрим классический пример начала работы с любым языком программирования: выводом на экран приветственного сообщения. Для этого в консоли необходимо ввести *print("Hello World")* и нажать клавишу *Enter*, после чего будут выведены слова "*Hello World*" (рис. 1.5).



```
Командная строка - python3
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>python3
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

Рис. 1.5. Демонстрация работы с *Python* на примере использования функции *print*

Приведенный пример позволяет заметить, что *Python* выдает результат работы инструкции немедленно. Набирать код каждый раз при запуске нецелесообразно, поэтому следует сохранять тексты программ в файлах с возможностью их запуска в дальнейшем.

Для написания и сохранения кода программ в файлах нужен редактор, одним из самых основных требований к которому является *подсветка синтаксиса*. Данный инструмент позволяет раскрасить разные элементы инструкций программы, что дает возможность программисту видеть структуру программу и ход ее выполнения.

В качестве удачного примера можно привести редактор *Visual Studio Code*, разработанный для *Windows*, *Linux* и *macOS* в виде программного обеспечения с открытым исходным кодом. К его достоинствам следует отнести многоязычный интерфейс пользователя и поддержку ряда языков программирования, подсветку синтаксиса, *IntelliSense*, рефакторинг, отладку, навигацию по коду, поддержку *Git* и другие возможности.

Расширение *Python* для *Visual Studio Code* с открытым исходным кодом появилось 2018 году (рис. 1.6).

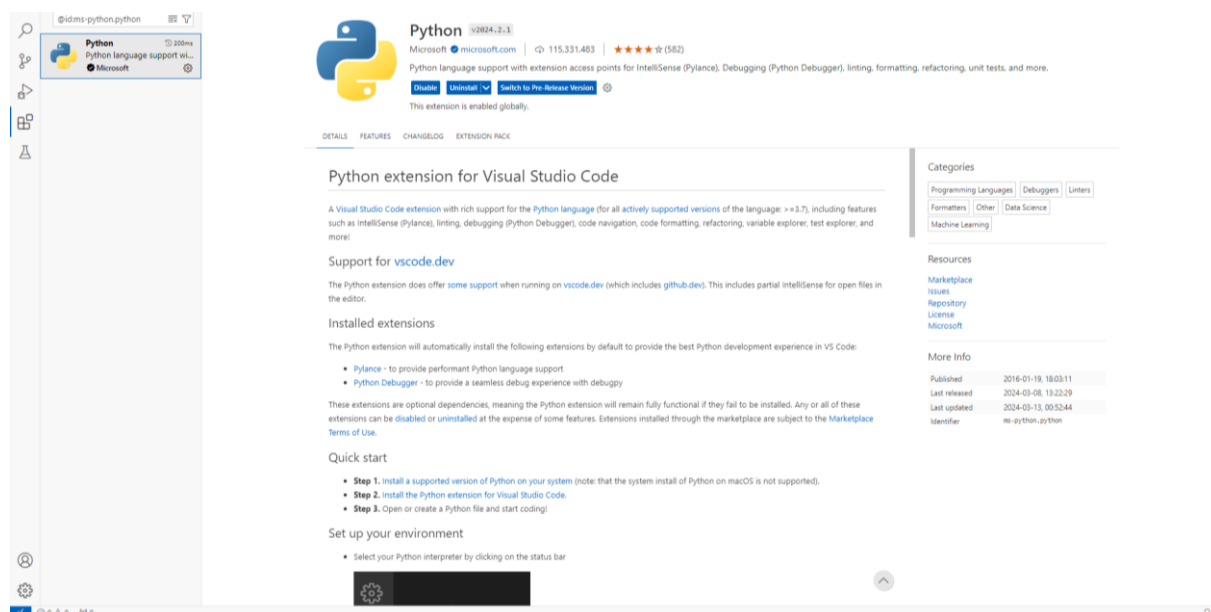


Рис. 1.6. Установка Python для Visual Studio Code

После запуска редактора необходимо создать файл и сохранить его в удобной директории (папке, каталоге) с осмысленным названием (например, *helloworld.py*). Важно, чтобы в названии присутствовало расширение *".py"*, определяющее *Python*-файл (рис. 1.7).

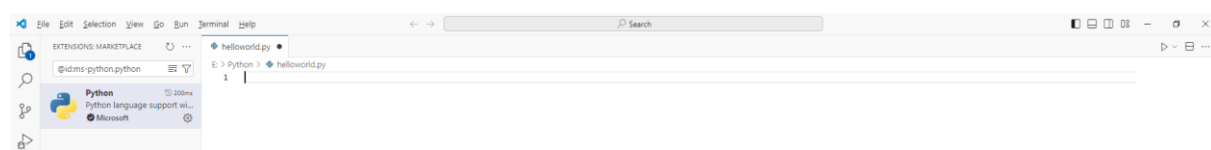


Рис. 1.7 Создание Python файла в Visual Studio Code

Сохранять файл можно в любую папку (директорию). Хорошей практикой является создание одной папки для сохранения всех дальнейших проектов.

На рис. 1.8 показан файл с программой на языке Python.

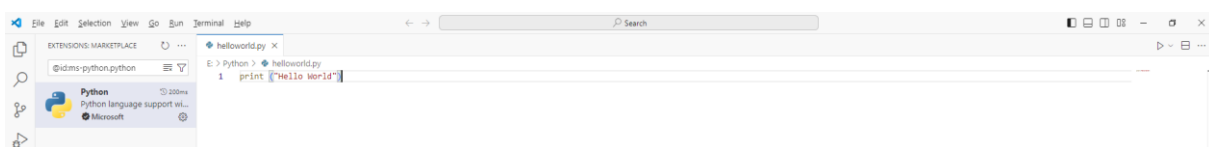


Рис. 1.8. Первая программа на языке Python в Visual Studio Code

Рассмотрим запуск написанной программы в терминале. После его запуска необходимо перейти в директорию, в которой сохранен файл, с помощью команды *cd*.

Для рассматриваемого случая команда будет выглядеть так: *cd /D E:\Python*, где */D* обязательная команда для перехода на другой диск, а затем следует записать команду: *python3 helloworld.py* (рис. 1.9).

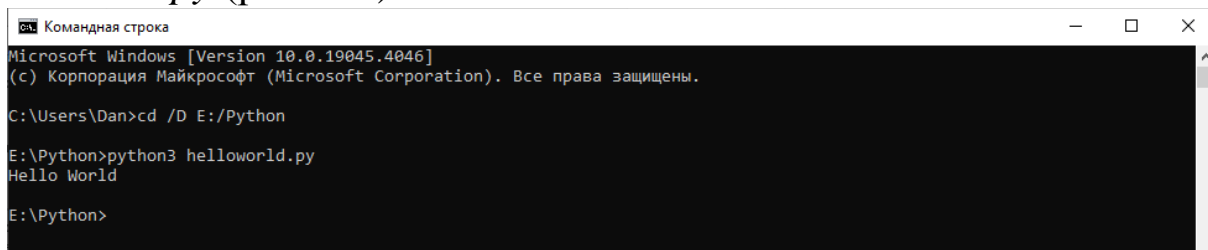
A screenshot of a Windows Command Prompt window titled "Командная строка". The window shows the following text: "Microsoft Windows [Version 10.0.19045.4046] (c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены. C:\Users\Dan>cd /D E:\Python E:\Python>python3 helloworld.py Hello World E:\Python>". The background is black, and the text is white.

Рис. 1.9. Запуск ранее созданной программы через терминал

Программа на языке *Python* состоит из *инструкций* или *выражений*. В рассмотренном примере имеется всего лишь одно выражение, отвечающее за вызов функции *print*.

Python никак не обрабатывает комментарии, кроме специального случая "*shebang*" в первой строке: первые два символа файла с программой – *#!*, за ними следует путь к некоторой программе. В данной ситуации для *Unix*-подобной системы создается указание и запуске программы именно в указанном интерпретаторе.

1.4.2 Настройка окружения

Создание виртуального окружения *venv Python* позволяет проблем с библиотеками разных версий путем изоляции проектов, каждый из которых будет запускаться в собственной виртуальной среде. Данный механизм повышает стабильность работы всех приложений.

Для создания виртуального окружения необходимо создать папку для нового проекта и перейти в нее после запуска командную строку. Следующим шагом является запись команды *python -m venv project1_env* (рис. 1.10), в которой *venv* – инструмент для создания виртуального окружения, *project1_env* – произвольное имя для данного окружения.

```
Командная строка - python -m venv project1_env
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\Dan>cd /D E:\Python\Project1
E:\Python\Project1>python -m venv project1_env
```

Рис. 1.10. Создание нового виртуального окружения

В результате выполнения в папке с проектом появится вложение с именем *project1_env* (рис. 1.11).

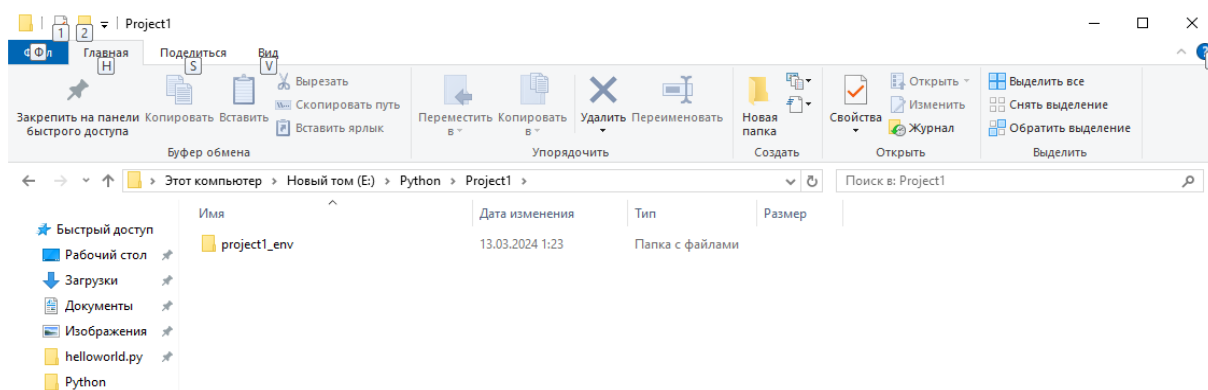


Рис. 1.11. Результат создания виртуального окружения

Для работы в рамках виртуального окружения его необходимо активировать с помощью команды:

project1_env\Scripts\activate

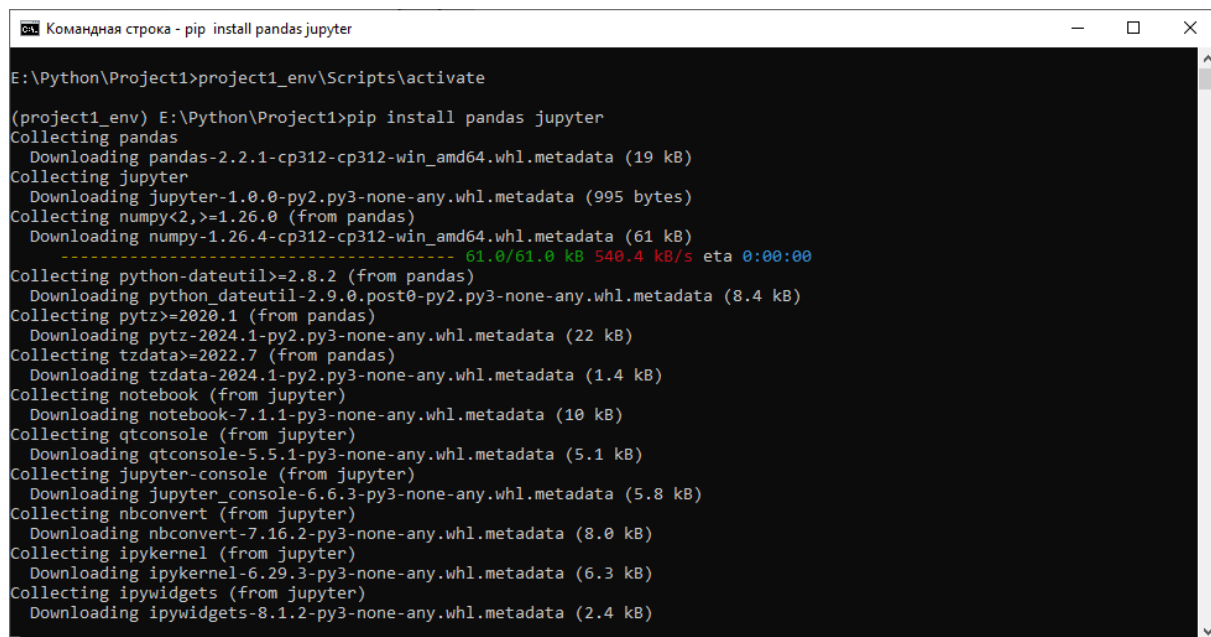
После активации название окружения должно появиться в скобках слева от пути к проекту (рис. 1.12).

```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\Dan>cd /D E:\Python\Project1
E:\Python\Project1>python -m venv project1_env
E:\Python\Project1>project1_env\Scripts\activate
(project1_env) E:\Python\Project1>
```

Рис. 1.12. Активация виртуального окружения

Далее следует установить пакеты, необходимые для конкретного проекта, например, *pandas* для работы с табличными данными и *jupyter* для интерактивной работы с данными в браузере с помощью блокнота. Команда установки нескольких пакетов имеет вид: *pip install pandas jupyter* (рис. 1.13). Имена пакетов разделять запятыми не нужно.

Важно отметить, если *jupyter* не будет установлен для нового окружения, то интерпретатор *Python* будет подтягивать пакеты из глобального окружения.



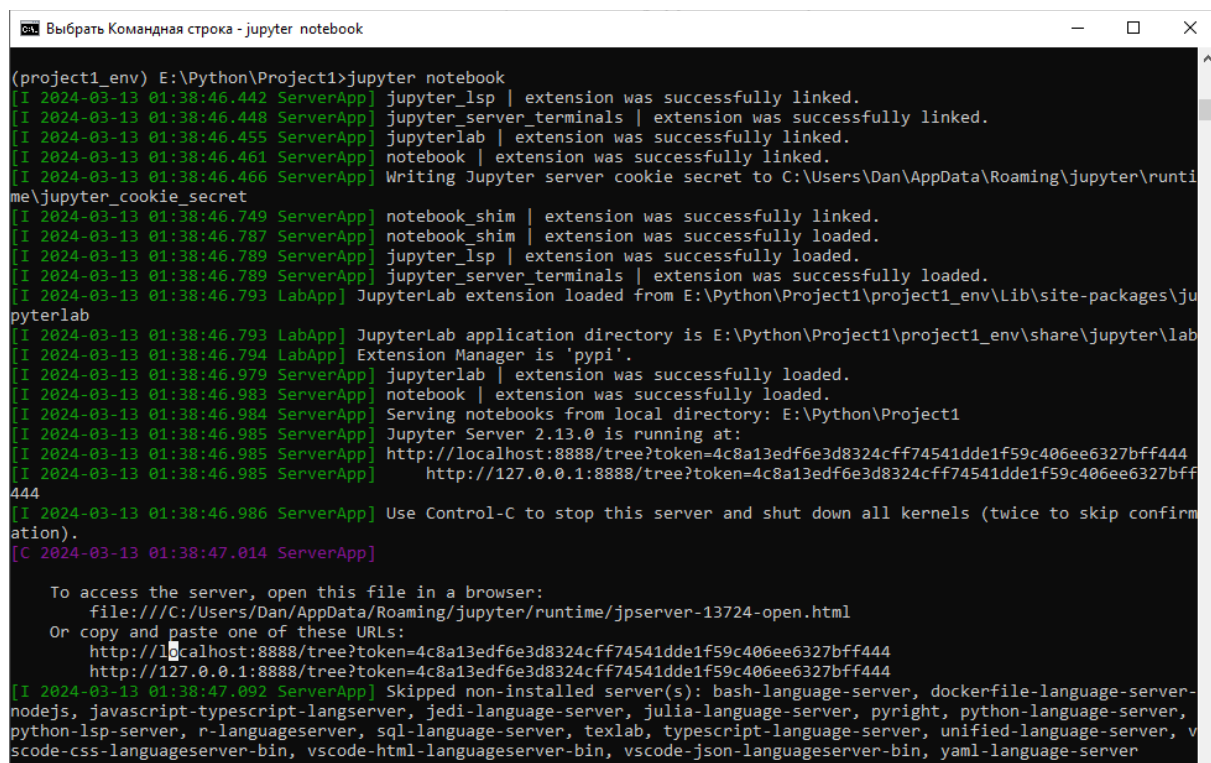
```
Командная строка - pip install pandas jupyter

E:\Python\Project1>project1_env\Scripts\activate

(project1_env) E:\Python\Project1>pip install pandas jupyter
Collecting pandas
  Downloading pandas-2.2.1-cp312-cp312-win_amd64.whl.metadata (19 kB)
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl.metadata (995 bytes)
Collecting numpy<2,>=1.26.0 (from pandas)
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
----- 61.0/61.0 kB 540.4 kB/s eta 0:00:00
Collecting python-dateutil>=2.8.2 (from pandas)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting pytz>=2020.1 (from pandas)
  Downloading pytz-2024.1-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting notebook (from jupyter)
  Downloading notebook-7.1.1-py3-none-any.whl.metadata (10 kB)
Collecting qtconsole (from jupyter)
  Downloading qtconsole-5.5.1-py3-none-any.whl.metadata (5.1 kB)
Collecting jupyter-console (from jupyter)
  Downloading jupyter_console-6.6.3-py3-none-any.whl.metadata (5.8 kB)
Collecting nbconvert (from jupyter)
  Downloading nbconvert-7.16.2-py3-none-any.whl.metadata (8.0 kB)
Collecting ipykernel (from jupyter)
  Downloading ipykernel-6.29.3-py3-none-any.whl.metadata (6.3 kB)
Collecting ipywidgets (from jupyter)
  Downloading ipywidgets-8.1.2-py3-none-any.whl.metadata (2.4 kB)
```

Рис. 1.13. Установка пакет *jupyter* и *pandas*

Для начала работы следует запустить блокнот с помощью команды *jupyter notebook* (рис. 1.14).



```
Выбор Командная строка - jupyter notebook

(project1_env) E:\Python\Project1>jupyter notebook
[I 2024-03-13 01:38:46.442 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-03-13 01:38:46.448 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-03-13 01:38:46.455 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-03-13 01:38:46.461 ServerApp] notebook | extension was successfully linked.
[I 2024-03-13 01:38:46.466 ServerApp] Writing Jupyter server cookie secret to C:\Users\Dan\AppData\Roaming\jupyter\runtim
me\jupyter_cookie_secret
[I 2024-03-13 01:38:46.749 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-03-13 01:38:46.787 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-03-13 01:38:46.789 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-03-13 01:38:46.789 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-03-13 01:38:46.793 LabApp] JupyterLab extension loaded from E:\Python\Project1\project1_env\Lib\site-packages\ju
pyterlab
[I 2024-03-13 01:38:46.793 LabApp] JupyterLab application directory is E:\Python\Project1\project1_env\share\jupyter\lab
[I 2024-03-13 01:38:46.794 LabApp] Extension Manager is 'pypi'.
[I 2024-03-13 01:38:46.979 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-03-13 01:38:46.983 ServerApp] notebook | extension was successfully loaded.
[I 2024-03-13 01:38:46.984 ServerApp] Serving notebooks from local directory: E:\Python\Project1
[I 2024-03-13 01:38:46.985 ServerApp] Jupyter Server 2.13.0 is running at:
[I 2024-03-13 01:38:46.985 ServerApp] http://localhost:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff444
[I 2024-03-13 01:38:46.985 ServerApp] http://127.0.0.1:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff
444
[I 2024-03-13 01:38:46.986 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirm
ation).
[C 2024-03-13 01:38:47.014 ServerApp]

To access the server, open this file in a browser:
file:///C:/Users/Dan/AppData/Roaming/jupyter/runtime/jpserver-13724-open.html
Or copy and paste one of these URLs:
http://localhost:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff444
http://127.0.0.1:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff444
[I 2024-03-13 01:38:47.092 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-
nodejs, javascript-typescript-languageserver, jedi-language-server, julia-language-server, pyright, python-language-server,
python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server, v
scode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

Рис 1.14. Запуск *Jupyter Notebook*

В результате должен запускаться браузер (по умолчанию), если этого не произошло, или ссылка оказалась нерабочей, то необходимо скопировать один из адресов, из блока, выделенного ниже, и вставить в строку поиска браузера (рис. 1.15).

```
Or copy and paste one of these URLs:  
http://localhost:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff444  
http://127.0.0.1:8888/tree?token=4c8a13edf6e3d8324cff74541dde1f59c406ee6327bff444
```

Рис. 1.15. Ссылки на вкладку с Jupyter Notebook

Пример работы с *Jupyter Notebook* показан на рис. 1.16-7.17.

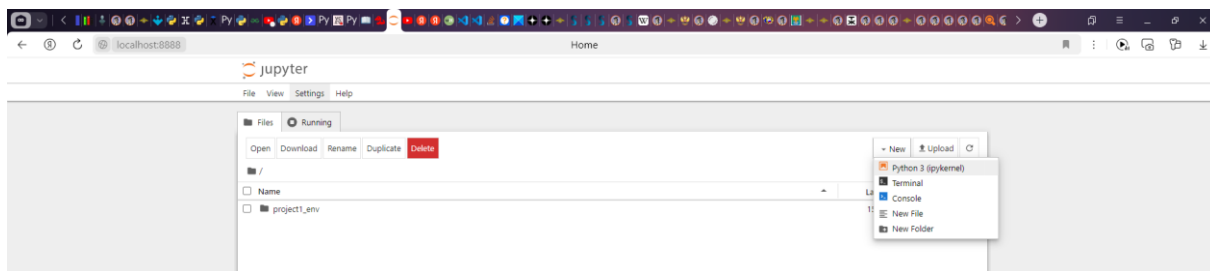


Рис. 1.16. Создание нового файла

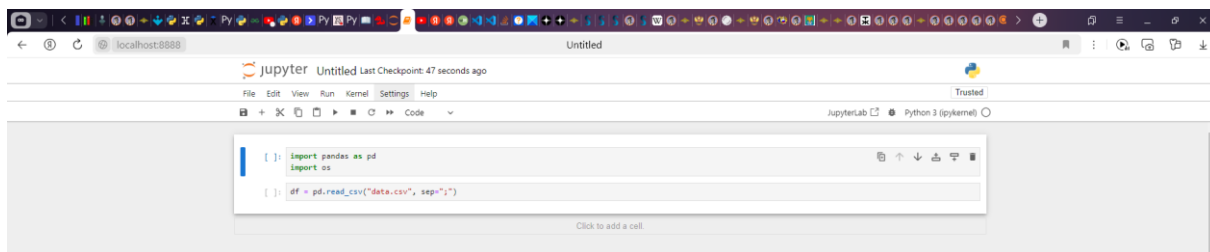


Рис. 1.17. Добавление инструкций кода

Для продолжения работы после перерыва необходимо:

1. Перейти в папку с проектом.
2. Активировать виртуальное окружение.
3. Запустить *jupyter notebook*.

Для выполнения указанных действий можно создать *переменную окружения* в разделе панели управления «Изменение переменных среды», найти и открыть который можно с помощью строки поиска (рис. 1.18).

В появившемся окне необходимо нажать на кнопку «Переменные среды», а затем «Создать» и внести необходимые данные (рис. 1.19): имя новой переменной *start_notebook* и ее значение.

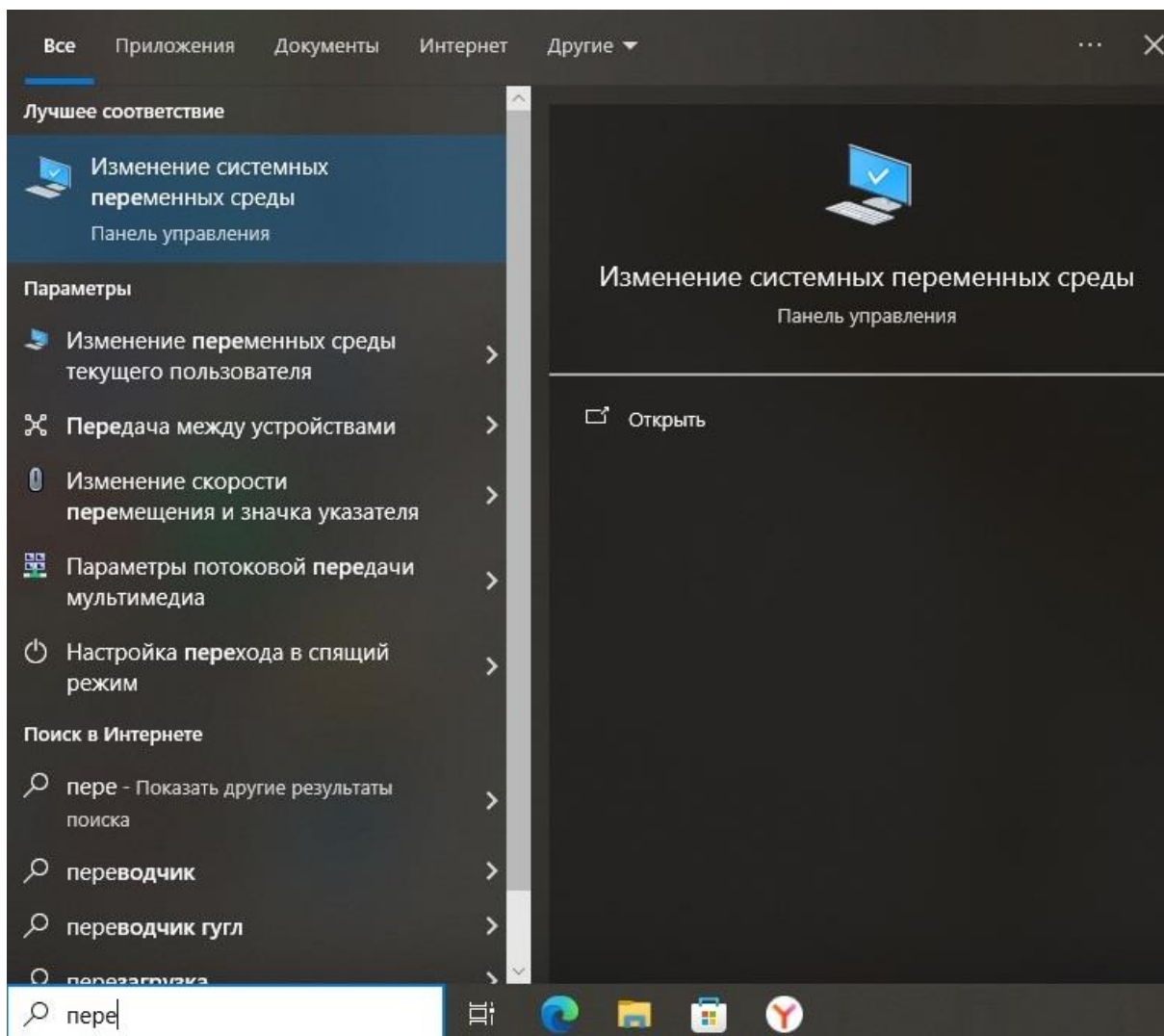


Рис. 1.18. Поиск и запуск раздела «Изменения системных переменных среды»

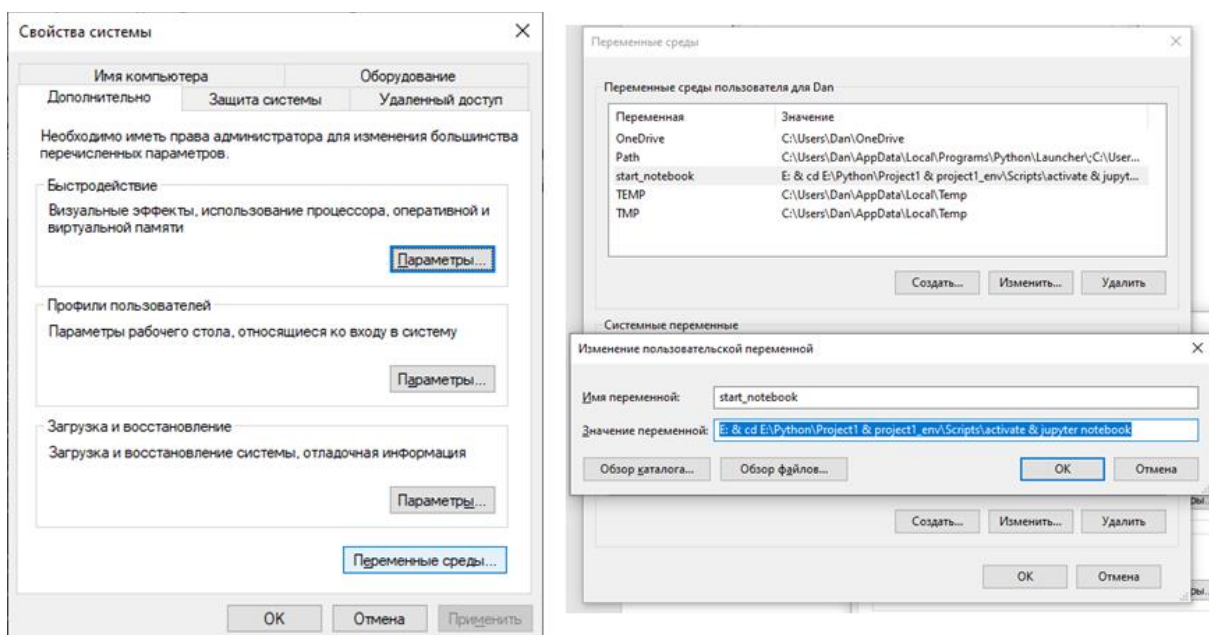


Рис. 1.19. Создание переменной окружения

Значение переменной можно связать с последовательностью команд с помощью оператора '&'.

Для рассматриваемого примера в качестве значения переменной указаны следующие команды

1. E: — переход на диск с папкой конкретного проекта.
2. cd E:\Python\Project1 — переход к папке с проектом.
3. project1_env\Scripts\activate — активация виртуального окружения.
4. jupyter notebook — запуск блокнота.

Таким образом, для продолжения работы в следующий раз потребуется ввести лишь имя переменной, заключенное в парные знаки '%' (рис. 1.20).

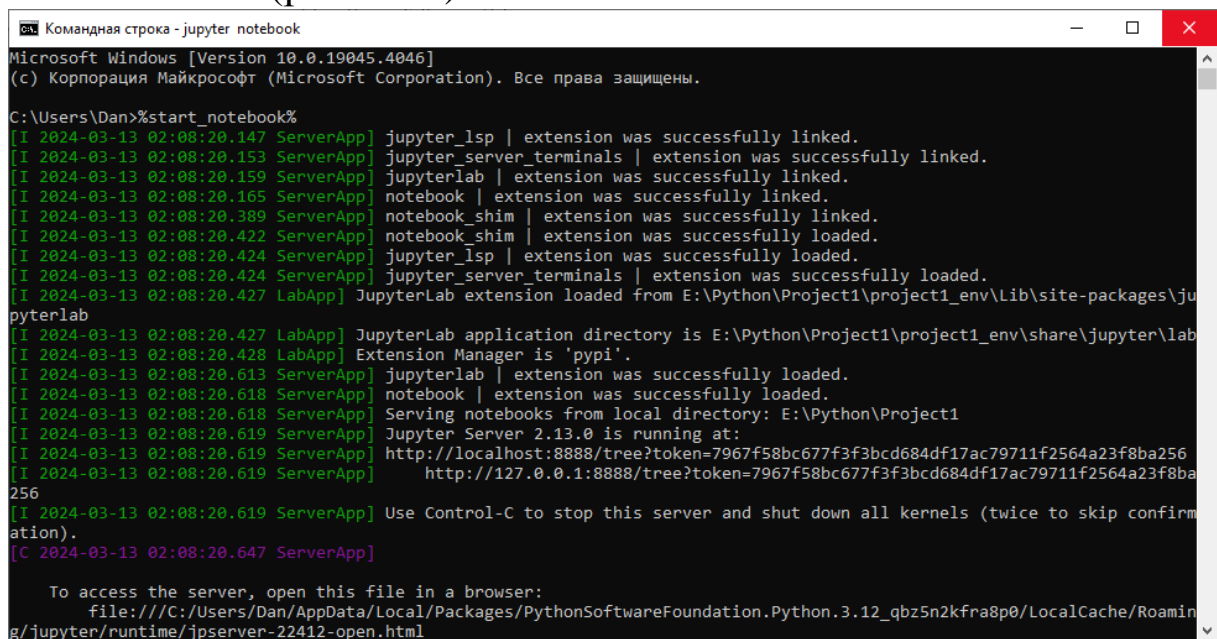


Рис. 1.20. Демонстрация работы переменной окружения

Результатом выполненных действий стали виртуальное окружение и переменная среды для быстрого доступа к рабочему проекту.

1.5 Взаимодействие с операционной системой

Модуль *os* из стандартной библиотеки языка программирования *Python* обычно используется для работы с установленной операционной системой (ОС), а также файловой системой компьютера. Он содержит много методов для взаимодействия с файлами и папками на жестком диске. Программы, работающие с

модулем *os*, не зависят от типа ОС и являются легко переносимыми на другую платформу.

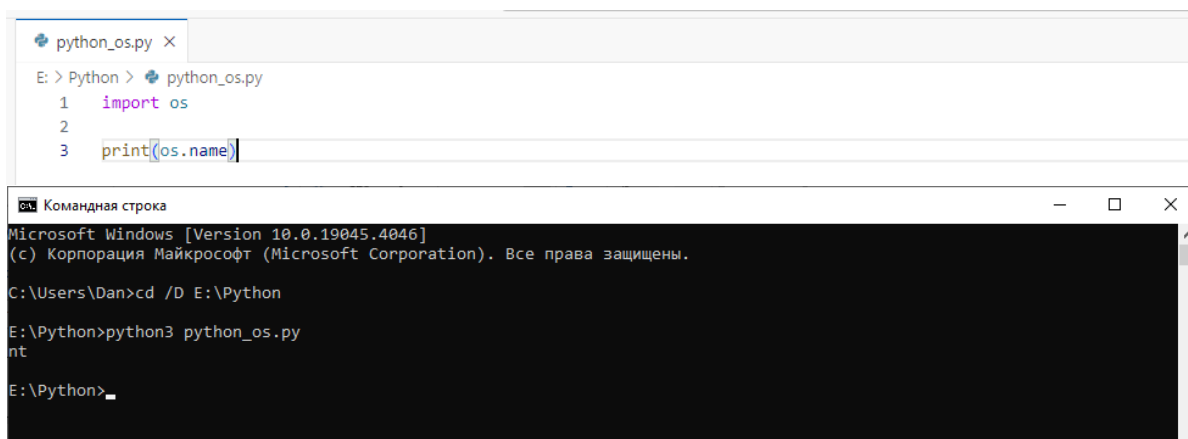
Модуль *os* в *Python* – библиотека функций для работы с операционной системой. Методы в ее составе позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами. При вызове функций *os* необходимо учитывать, что некоторые из них могут не поддерживаться текущей ОС.

Ниже показаны наиболее популярные приемы работы с работой с модулем *os*, позволяющие получать данные об операционной системе, файлах и папках, хранимых в памяти на жестком диске компьютера.

Для демонстрации работы применяется редактор *Visual Studio Code* с последующим запуском исполняемого файла на терминале *Windows*.

1.5.1 Получение информации из ОС

Метод *name* позволяет узнать имя текущей ОС. В зависимости от установленной платформы, он вернет ее короткое наименование в строковом представлении. Следующая программа (рис. 1.21) была запущена на компьютере с ОС *Windows 10*, поэтому результатом работы функции *name* является строка *nt*. Вывод данных осуществляется с помощью стандартной функции *print*.



The image shows two overlapping windows. The top window is a Visual Studio Code editor with a file named `python_os.py`. The code inside is:

```
1 import os
2
3 print(os.name)
```

The bottom window is a Windows Command Prompt. It shows the following commands and output:

```
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

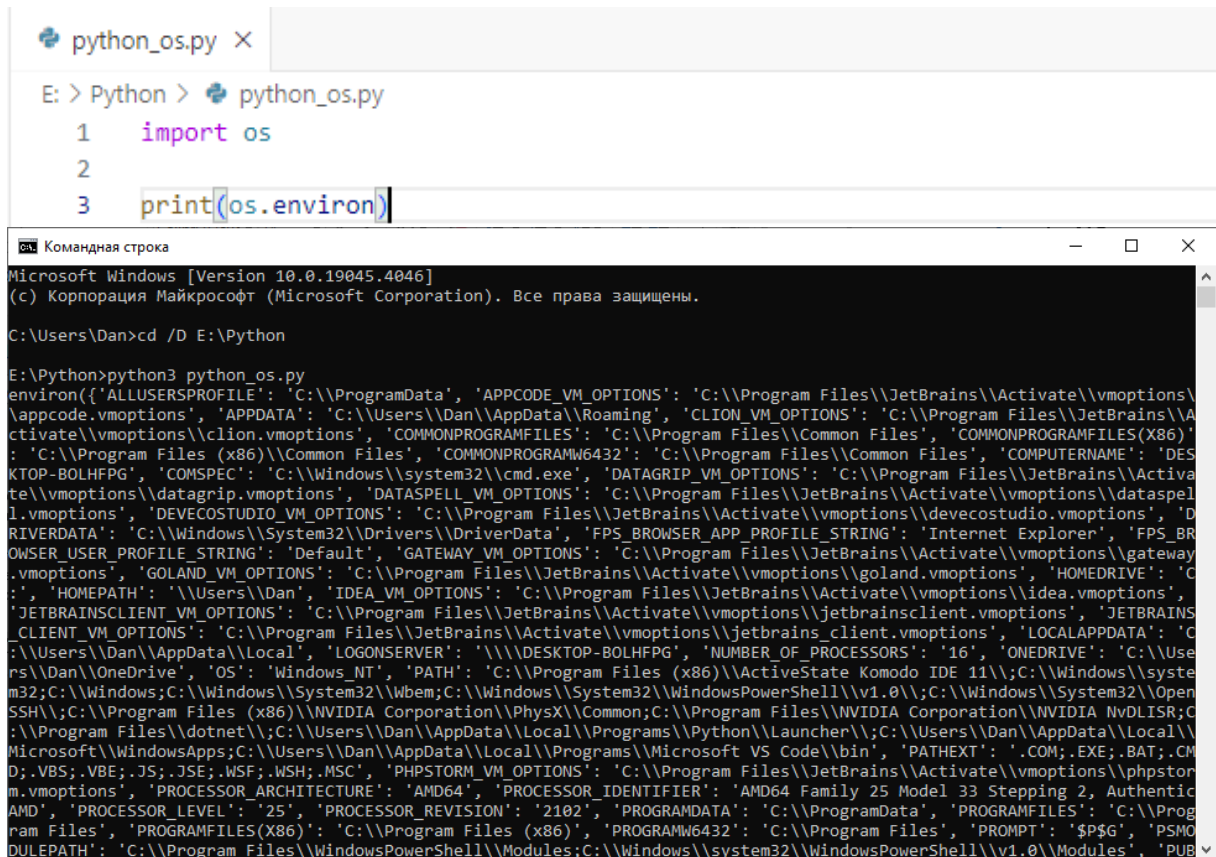
E:\Python>python3 python_os.py
nt

E:\Python>
```

Рис. 1.21. Вывод наименования текущей ОС

Получить сведения о конфигурации компьютера (название системного диска, адрес домашней директории, имя системы и

т.д.) позволяет метод *environ*. Результатом обращения к нему является большой словарь с переменными окружения, выводимый на консоль или сохраняемый в строковую переменную (рис. 1.22).



```
python_os.py X
E: > Python > python_os.py
1 import os
2
3 print(os.environ)
```

```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

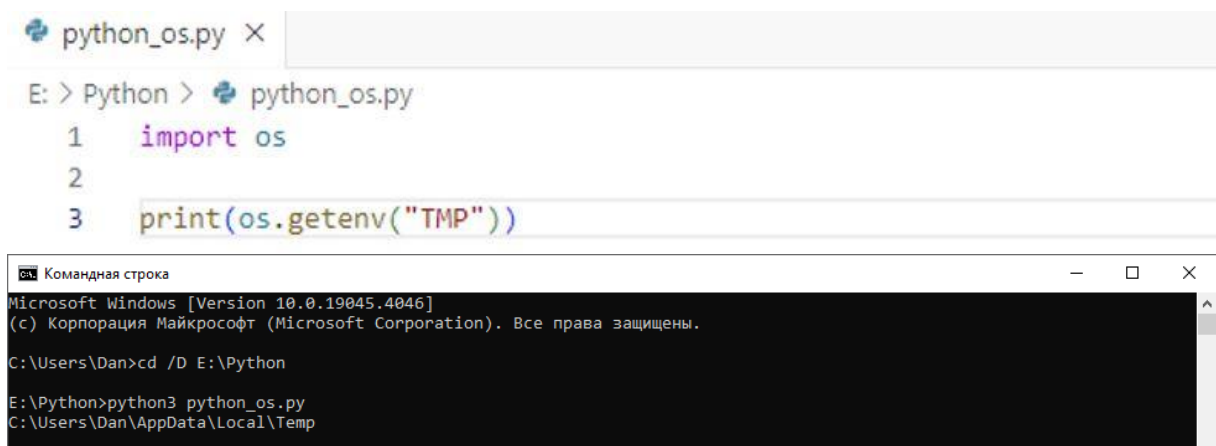
C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
environ({'ALLUSERSPROFILE': 'C:\\ProgramData', 'APPCODE_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\appcode.vmoptions', 'APPDATA': 'C:\\Users\\Dan\\AppData\\Roaming', 'CLION_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\clion.vmoptions', 'COMMONPROGRAMFILES': 'C:\\Program Files\\Common Files', 'COMMONPROGRAMFILES(X86)': 'C:\\Program Files (x86)\\Common Files', 'COMMONPROGRAMW6432': 'C:\\Program Files\\Common Files', 'COMPUTERNAME': 'DESKTOP-BOLHFPG', 'COMSPEC': 'C:\\Windows\\system32\\cmd.exe', 'DATAGRIP_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\datagrip.vmoptions', 'DATASPELL_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\dataspell.vmoptions', 'DEVECOSTUDIO_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\devcostudio.vmoptions', 'DRIVERDATA': 'C:\\Windows\\System32\\Drivers\\DriverData', 'FPS_BROWSER_APP_PROFILE_STRING': 'Internet Explorer', 'FPS_BROWSER_USER_PROFILE_STRING': 'Default', 'GATEWAY_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\gateway.vmoptions', 'GOLAND_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\goland.vmoptions', 'HOMEDRIVE': 'C:', 'HOMEPATH': '\\Users\\Dan', 'IDEA_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\idea.vmoptions', 'JETBRAINSCLIENT_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\jetbrainsclient.vmoptions', 'JETBRAINSCLIENT_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\jetbrains_client.vmoptions', 'LOCALAPPDATA': 'C:\\Users\\Dan\\AppData\\Local', 'LOGONSERVER': '\\\\DESKTOP-BOLHFPG', 'NUMBER_OF_PROCESSORS': '16', 'ONEDRIVE': 'C:\\Users\\Dan\\OneDrive', 'OS': 'Windows_NT', 'PATH': 'C:\\Program Files (x86)\\ActiveState Komodo IDE 11\\;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Windows\\System32\\OpenSSH\\;C:\\Program Files (x86)\\NVIDIA Corporation\\PhysX\\Common;C:\\Program Files\\NVIDIA Corporation\\NVIDIA NvDLISR;C:\\Program Files\\dotnet\\;C:\\Users\\Dan\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\Dan\\AppData\\Local\\Microsoft\\WindowsApps;C:\\Users\\Dan\\AppData\\Local\\Programs\\Microsoft VS Code\\bin', 'PATHEXT': '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC', 'PHPSTORM_VM_OPTIONS': 'C:\\Program Files\\JetBrains\\Activate\\vmoptions\\phpstorm.vmoptions', 'PROCESSOR_ARCHITECTURE': 'AMD64', 'PROCESSOR_IDENTIFIER': 'AMD64 Family 25 Model 33 Stepping 2, AuthenticAMD', 'PROCESSOR_LEVEL': '25', 'PROCESSOR_REVISION': '2102', 'PROGRAMDATA': 'C:\\ProgramData', 'PROGRAMFILES': 'C:\\Program Files', 'PROGRAMFILES(X86)': 'C:\\Program Files (x86)', 'PROGRAMW6432': 'C:\\Program Files', 'PROMPT': '$P$G', 'PSMODULEPATH': 'C:\\Program Files\\WindowsPowerShell\\Modules;C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\Modules', 'PUB
```

Рис. 1.22. Получение сведений о конфигурации компьютера

Доступ к различным переменным среды осуществляется с помощью метода *getenv*.

На рис. 1.23 показан вывод сведений о переменной *TMP*.



```
python_os.py X
E: > Python > python_os.py
1 import os
2
3 print(os.getenv("TMP"))
```

```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
C:\Users\Dan\AppData\Local\Temp
```

Рис. 1.23. Получение сведений о переменной среде *TMP*

По умолчанию рабочей директорией программы является каталог, в котором содержится документ с исходным кодом. Благодаря этому можно не указывать абсолютный путь к файлу, если он находится именно в этой папке. Получить сведения о текущей директории позволяет метод *getcwd*, возвращающий в качестве результата полный адрес рабочего каталога на жестком диске (рис. 1.24).



Рис. 1.24. Получение сведений о текущей директории

1.5.2 Проверка существования пути

Метод *exists* позволяет избежать ошибок, связанных с отсутствием определенного файла или директории, которые должны быть обработаны программой, путем предварительной проверки их существования. В качестве аргумента передается путь к нужному объекту. Результатом вызова является булево значение *true/false*, сообщающее о наличии/отсутствии указанного объекта в памяти компьютера.

Следует отметить, *exists* проверяет только существование файла, но не наличие доступа программы к нему. Для подтверждения доступа к файлу следует использовать обработку исключения *IOError* при открытии файла.

На рис. 1.25 продемонстрирована проверка наличия текстового файла *test.txt* в корневом каталоге *E*.



Рис. 1.25. Проверка наличия файла *test.txt*

Наряду с проверкой существования объекта важно проверять возможность его дальнейшей обработки. Проверить, является ли определенный объект файлом, можно с помощью метода *isfile*, принимающего адрес.

На рис 1.26 выполняется проверка объекта *test.txt* на соответствие категории файла.

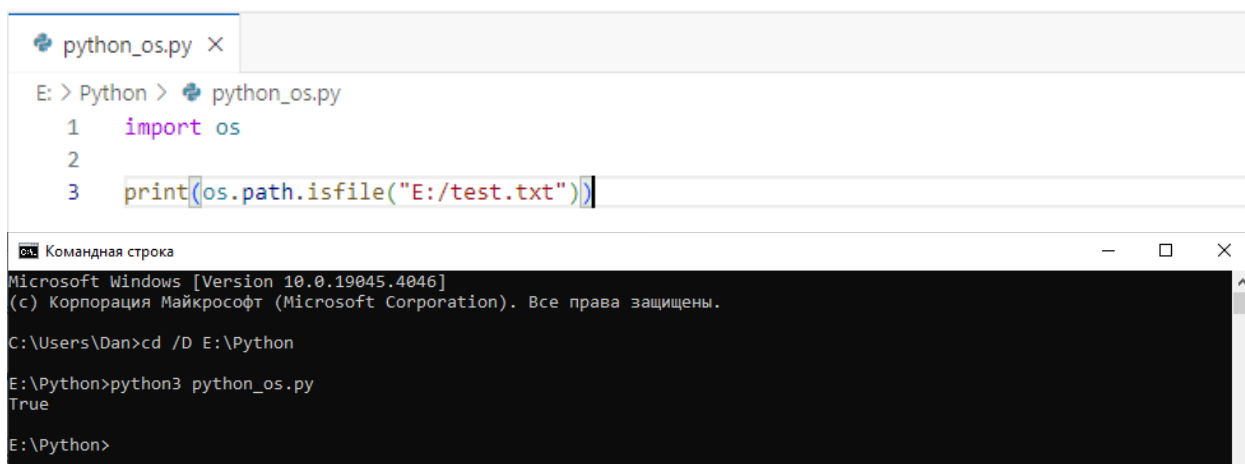


Рис. 1.26. Проверка объекта на принадлежность к файлам

Аргумент метода *isfile* может принимать байтовые или текстовые строки, а также объекты, представляющие путь к файловой системе, например, *pathlib.PurePath*.

Аналогичные действия можно выполнить и для проверки объекта на принадлежность к классу директорий, вызвав для его адреса метод *isdir* из библиотеки *os* (рис. 1.27).



Рис. 1.27. Проверка объекта на принадлежность к классу директорий

Как можно заметить, в данном случае *print* выводит на экран булево значение *False*, поскольку *test.txt* не является директорией.

1.5.3 Создание директорий

Возможности модуля *os* позволяют не только отображать информацию об уже существующих в памяти объектах, но и генерировать абсолютно новые. Например, с помощью метода *mkdir* можно создать папку, указав для нее нужный путь.

На рис. 1.28-1.29 продемонстрирован пример создания в корневом каталоге диска новой папки под названием *folder1* через *mkdir*.

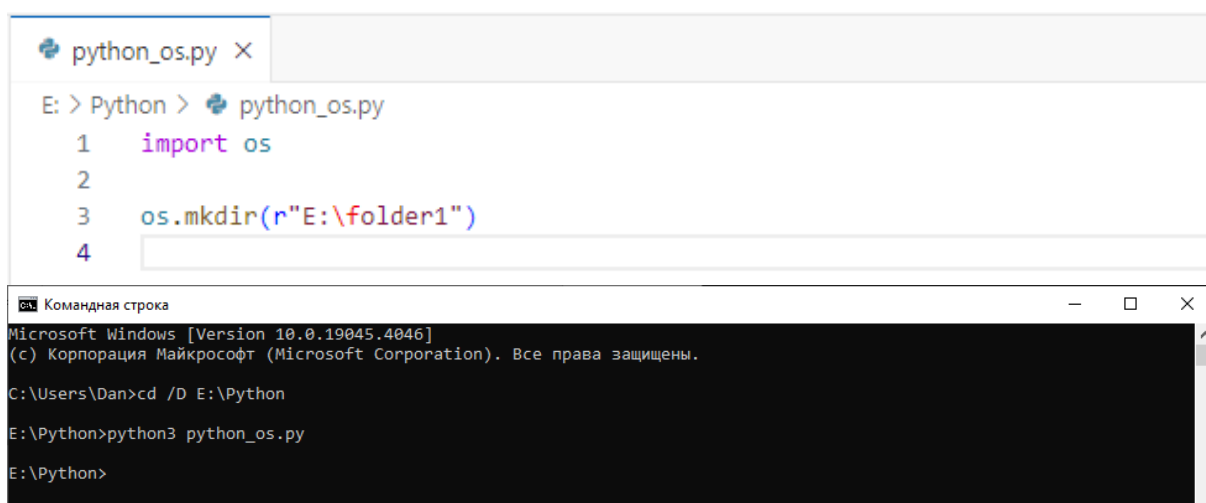


Рис. 1.28. Создание новой папки

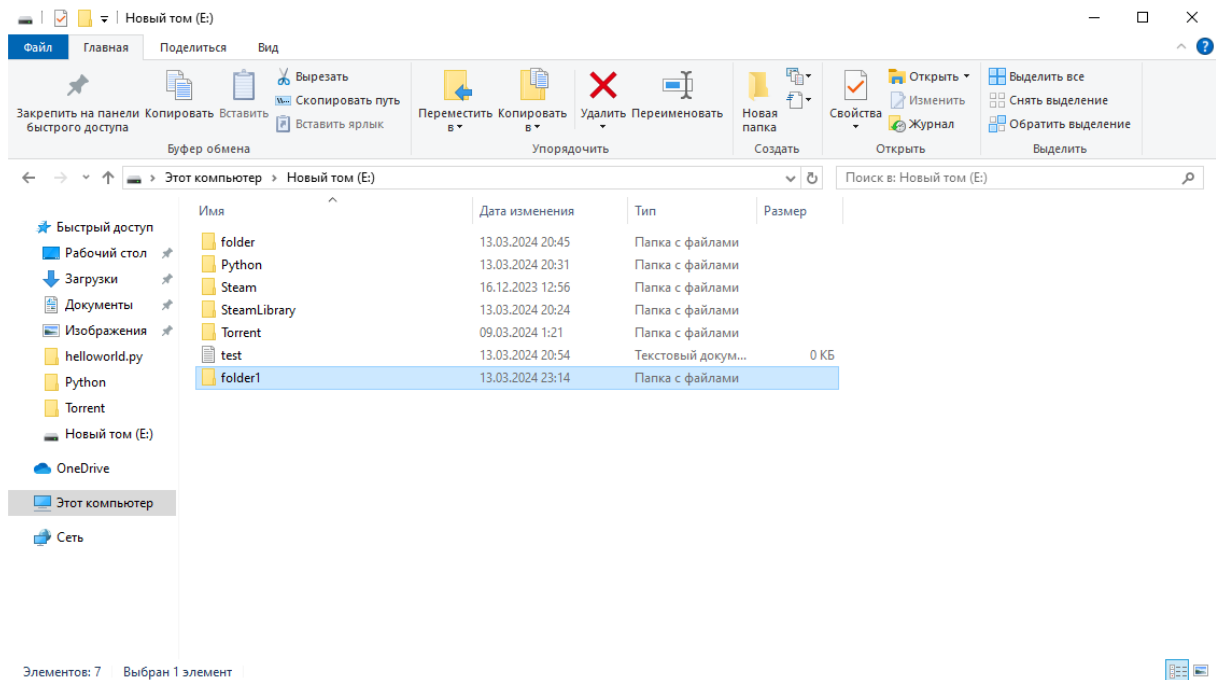


Рис. 1.29. Результат выполнения метода *mkdir*

Метод *makedirs* позволяет создавать сразу несколько новых папок в неограниченном количестве, если предыдущая директория является родительской для следующей.

На рис. 1.30 показана генерация вложенных папок *folder*, *first*, *second* и *third*.

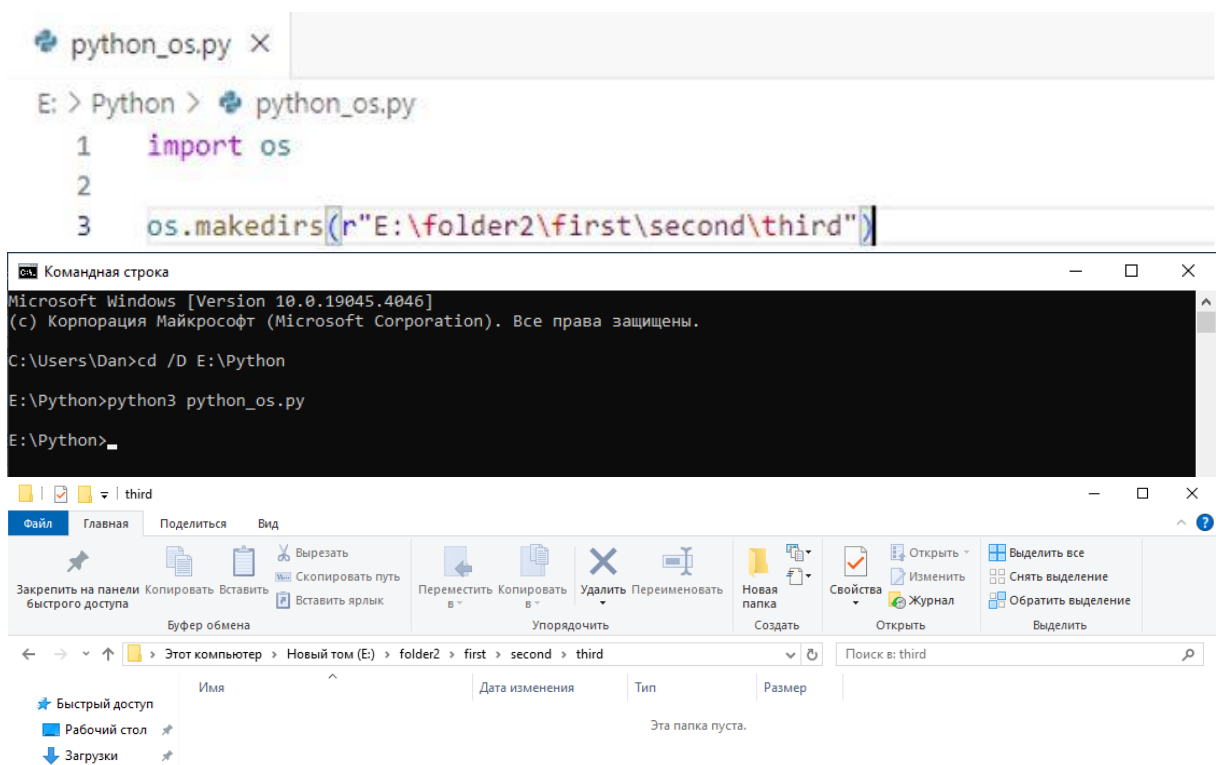


Рис. 1.30. Генерация вложенных папок

1.5.4 Удаление файлов и директорий

Функционал модуля *os* поддерживает удаление файлов и каталогов с помощью пользовательского интерфейса без поиска файла вручную. Однако при выполнении операции удаления следует помнить о возможности возникновения ситуации, при которой нельзя будет восстановить объекты.

Удаление файла можно выполнить методом *remove*, в качестве аргумента которого задают абсолютный либо относительный путь к объекту.

На рис. 1.31 демонстрируется удаление документа *test.txt* из корневой директории диска *E* компьютера.

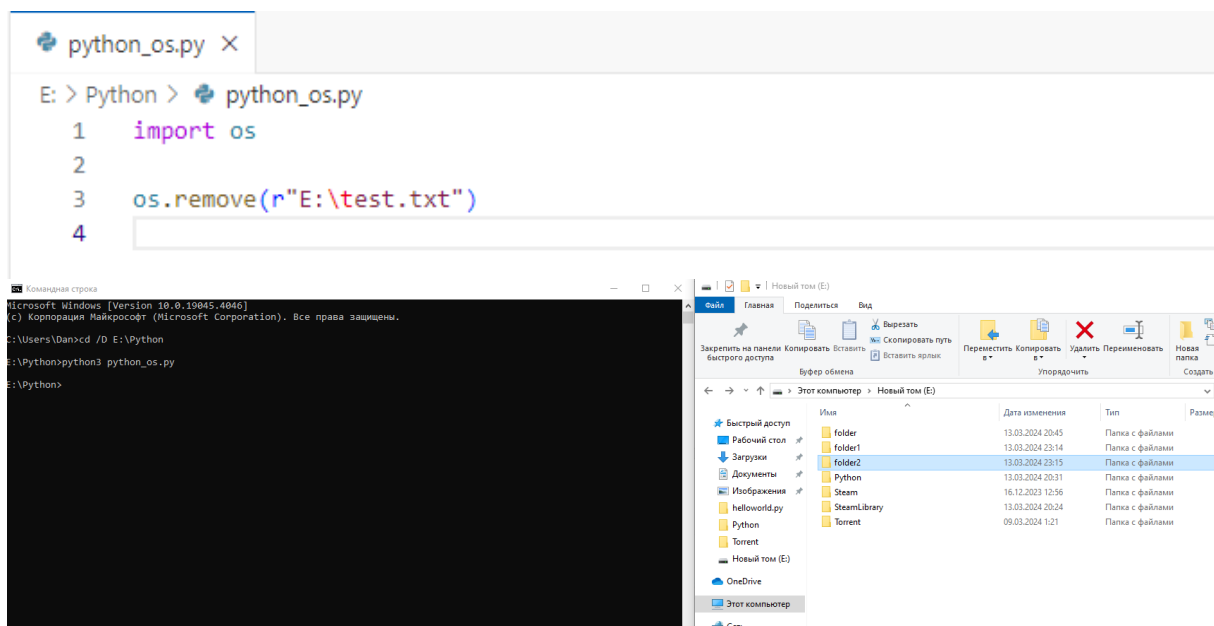


Рис. 1.31. Удаление файла

Если удаляемого файла не существует, будет возвращено исключение *FileNotFoundError*.

Для удаления каталога следует воспользоваться методом *rmdir* модуля *os*. Однако программа не позволит удалить папку, в которой хранятся другие объекты, и выдаст исключение с ошибкой *OSError*. Для удаления папок, содержащих вложения следует использовать метод *rmtree()* модуля *shutil*.

Пример корректного удаления пустого каталога представлен на рис. 1.32.

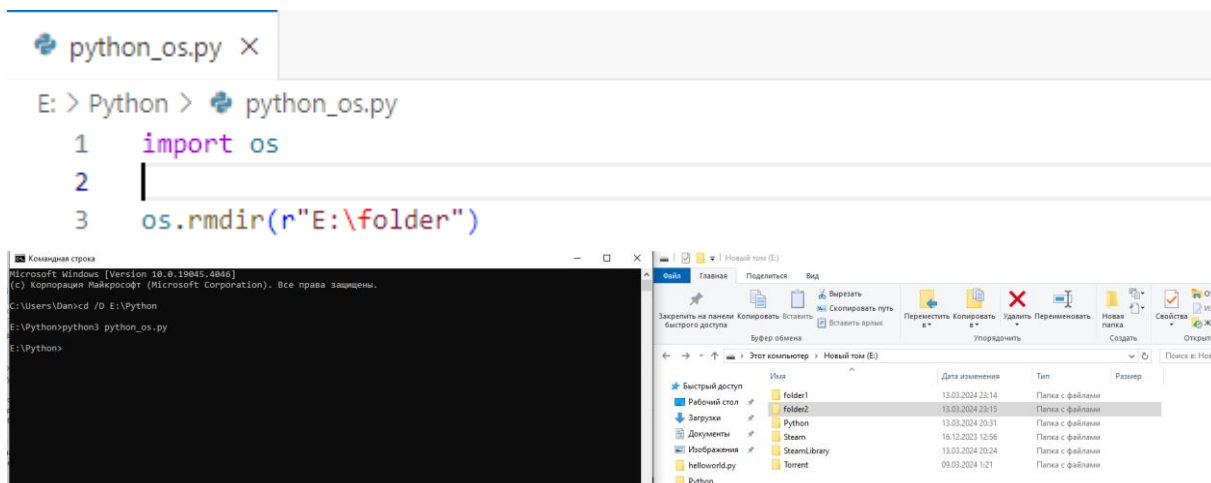


Рис. 1.32. Удаление пустого каталога

Для удаления вложенных друг в друга пустых каталогов следует использовать метод *removedirs*. В качестве аргумента указывается путь к конечной папке.

На рис. 1.33 показан пример удаления ранее созданных вложенных каталогов *folder*, *first*, *second* и *third*.

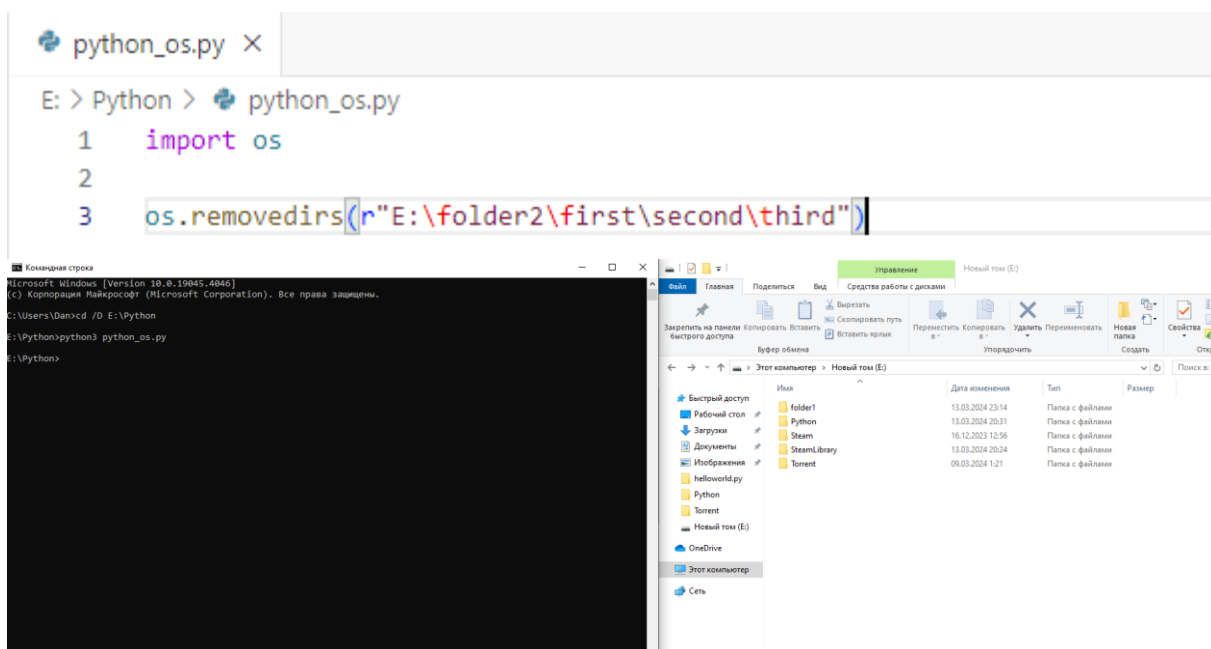


Рис. 1.33. Удаление вложенных пустых каталогов

1.5.5 Запуск на исполнение

Метод *startfile* модуля *os* позволяет выполнить запуск отдельных файлов и каталогов напрямую из программы. Аргументом является адрес необходимого объекта. Программное обеспе-

чение, необходимое для открытия документа, определяется средой автоматически. Если аргументом метода является ссылка на директорию, для открытия будет использован встроенный менеджер файлов. На рис. 1.34 приведен пример запуска файла *test.txt* стандартным блокнотом с помощью *startfile*.



Рис. 1.34. Запуск файла *test.txt* при помощи функции *startfile*

1.5.6 Переименование

Метод *rename* модуля *os* позволяет изменить название любого файла или каталога. При вызове передаются два разных аргумента. Первый из них соответствует пути к старому наименованию документа, а во втором хранится его новое название. Данный метод может генерировать исключение, если по указанному пути нет файла.

Пример переименования директории *folder1* в *catalog* представлен на рис. 1.35.

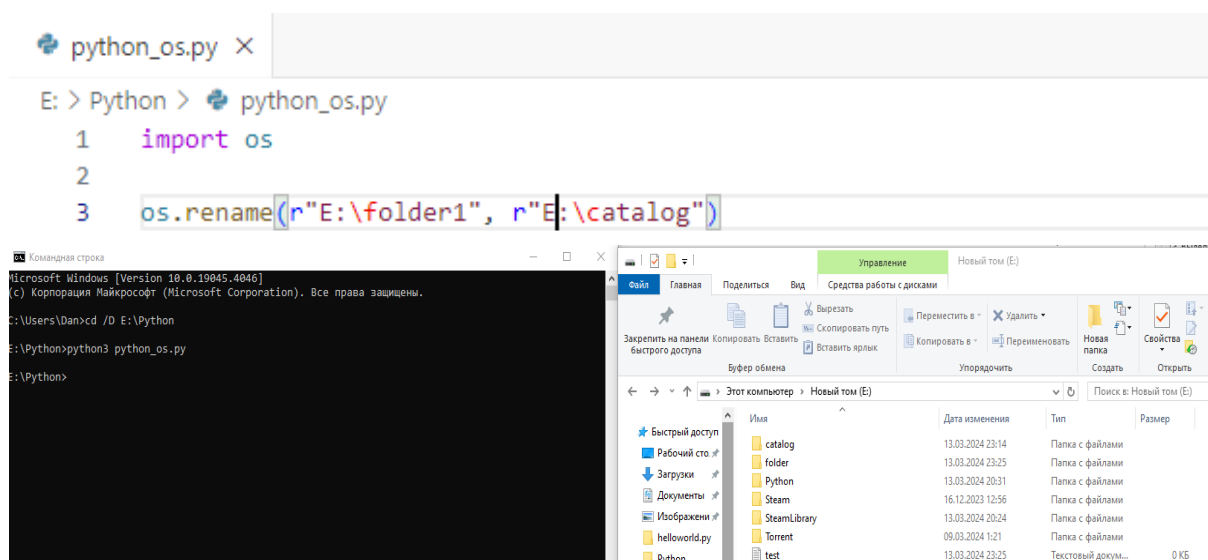


Рис. 1.35. Переименование каталога

Метод *renames* позволяет переименовать несколько вложенных каталогов сразу. В первом аргументе при вызове передается путь к конечной директории, во втором – новые имена всей цепочки (рис. 1.36).

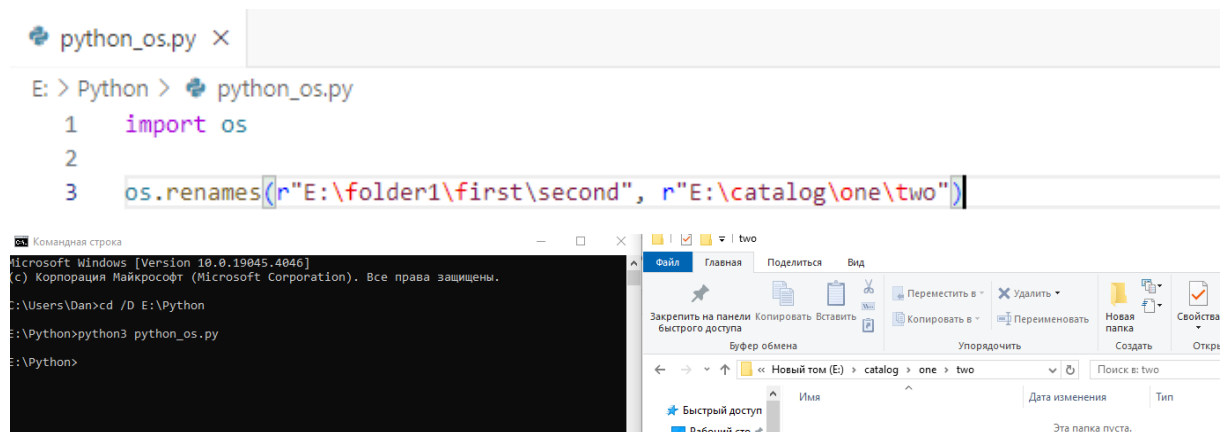


Рис. 1.36. Переименования вложенных каталогов

1.5.7 Содержимое директорий

Проверить наличие в каталоге определенных объектов позволяет метод *listdir*, результатом выполнения которого является список с информацией о файлах и папках. В качестве аргумента при вызове передается путь к нужному каталогу. Вывод результатов выполняется с помощью *print* (рис. 1.37).

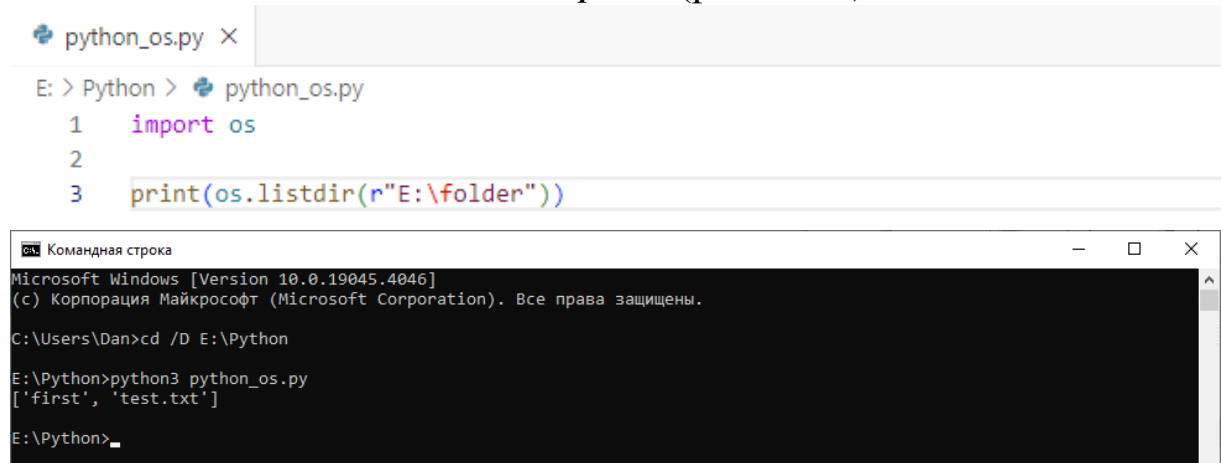


Рис. 1.37. Получение информации об объектах в заданном каталоге

Получение информации обо всех подкаталогах и файлах в них можно осуществить с помощью метода *walk* и вложенных циклов (рис. 1.38).

```
python_os.py X
E: > Python > python_os.py > ...
1  import os
2
3  for root, directories, files in os.walk(r"E:\folder"):
4      print(root)
5      for directory in directories:
6          print(directory)
7      for file in files:
8          print(file)
```

```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
E:\folder
first
E:\folder\first
second
E:\folder\first\second
third
E:\folder\first\second\third
test.txt
E:\Python>
```

Рис. 1.38. Получение информации о вложениях каталоге

1.5.8 Информация о файлах и директориях

Метод *stat* позволяет получить основные сведения об объекте. В качестве аргумента при вызове передается расположение файла или каталога. Результатом работы метода являются данные о размере объекта в байтах, типе доступа и режиме работы (рис. 1.39).

```
python_os.py X
E: > Python > python_os.py
1  import os
2
3  print(os.stat(r"E:\test.txt"))
```

```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
os.stat_result(st_mode=33206, st_ino=281474976985169, st_dev=10704654707144693496, st_nlink=1, st_uid=0, st_gid=0, st_size=0, st_atime=1710363622, st_mtime=1710361536, st_ctime=1710363622)
E:\Python>
```

Рис. 1.39. Получение информации об объекте на диске

1.5.9 Обработка путей

Функционал модуля *os* позволяет не только проверять существование пути, но и проводить манипуляции с его составляющими. Метод *split* возвращает кортеж строк, соответствующих иерархии каталогов и имени файла.

На рис. 1.40 показан пример преобразования пути к текстовому документу *test.txt* в папке *folder* на диске *E* в две строки.



The image shows a Python IDE window with a file named `python_os.py`. The code in the editor is as follows:

```
E: > Python > python_os.py
1  import os
2
3  print(os.path.split(r"E:\folder\test.txt"))
```

Below the IDE is a Windows Command Prompt window. It shows the following commands and output:

```
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
('E:\\folder', 'test.txt')

E:\Python>
```

Рис. 1.40. Преобразование пути к файлу в отдельные строки

Обратное действие по созданию пути выполняет функция `join` (рис. 1.41)



The image shows a Python IDE window with a file named `python_os.py`. The code in the editor is as follows:

```
E: > Python > python_os.py
1  import os
2
3  print(os.path.join(r"E:\folder", "test.txt"))
```

Below the IDE is a Windows Command Prompt window. It shows the following commands and output:

```
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd /D E:\Python

E:\Python>python3 python_os.py
('E:\\folder', 'test.txt')

E:\Python>python3 python_os.py
E:\folder\test.txt

E:\Python>
```

Рис. 1.41. Формирование пути к файлу

1.6 Хостинг ИТ-проектов и управление версиями

На начальных этапах изучения любого языка программирования или при создании небольшого проекта для личного пользования исходные коды можно успешно хранить на локальном компьютере.

Однако в случае проектов с частыми обновлениями, множеством версий, большим количеством файлов, необходимостью синхронизации разработки и удобного развертывания следует задуматься о размещении на специальных платформах – хостингах.

1.6.1 Создание репозитория *GitHub*

GitHub – построенная на базе популярной системы контроля версий *Git* облачная платформа для хостинга ИТ-проектов и совместной разработки, а также полноценная социальная сеть для разработчиков.

Первым шагом к использованию сервиса *GitHub* является регистрация нового пользователя. Доступ к личному кабинету активируется после подтверждения электронной почты.

Для размещения проекта нужно создать репозиторий с помощью кнопки *New* (рис. 1.42) и заполнить основные сведения о нем: название, описание, тип репозитория (рис. 1.43).

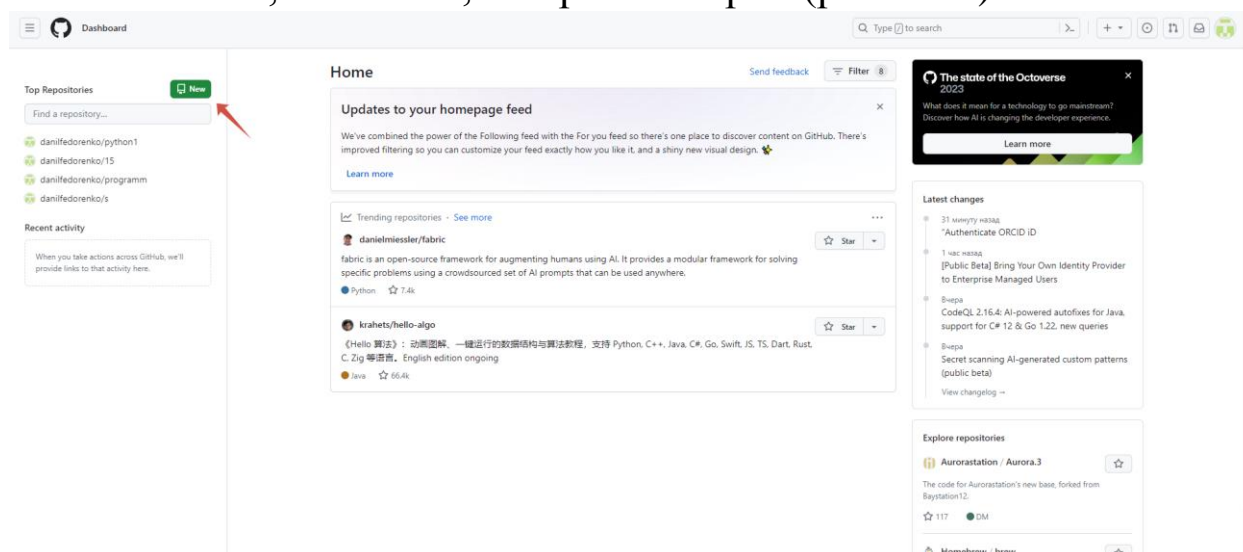


Рис. 1.42. Интерфейс создания нового репозитория

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * danilfedorenko / Repository name * PythonProject2
 PythonProject2 is available.

Great repository names are short and memorable. Need inspiration? How about [upgraded-pancake](#)?

Description (optional)

☒ **Public**
 Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
 You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
 This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
 .gitignore template: None

Choose which files not to track from a list of templates. [Learn more about choosing files.](#)

Choose a license
 License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

Рис. 1.43. Заполнение данных о репозитории

После создания репозитория следует загрузить исходный код и файл *requirements.txt*, используемый в *Python*-проектах для указания пакетов или библиотек, необходимых для работы над проектом, а также списка зависимостей и их версий.

1.6.2 Загрузка проектов через веб-интерфейс GitHub

В ряде случаев требуется просто сохранить проект или поделиться им. Работа с веб-интерфейсом *GitHub* – самый простой способ загрузки файлов в ранее созданный репозиторий. Сначала необходимо открыть страницу проекта и нажать ссылку "uploading an existing file" (рис. 1.44).

PythonProject2 Public

Pin Unwatch 1 Fork 0 Star 0

Set up GitHub Copilot
 Use GitHub's AI pair programmer to autocomplete suggestions as you code.
[Get started with GitHub Copilot](#)

Add collaborators to this repository
 Search for people using their GitHub username or email address.
[Invite collaborators](#)

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/danilfedorenko/PythonProject2.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

Рис. 1.44. Загрузка файлов через интерфейс GitHub

Далее в открывшееся окно можно либо перетащить файлы, либо указать их через диалоговое окно по ссылке "*choose your files*" (рис. 1.45).

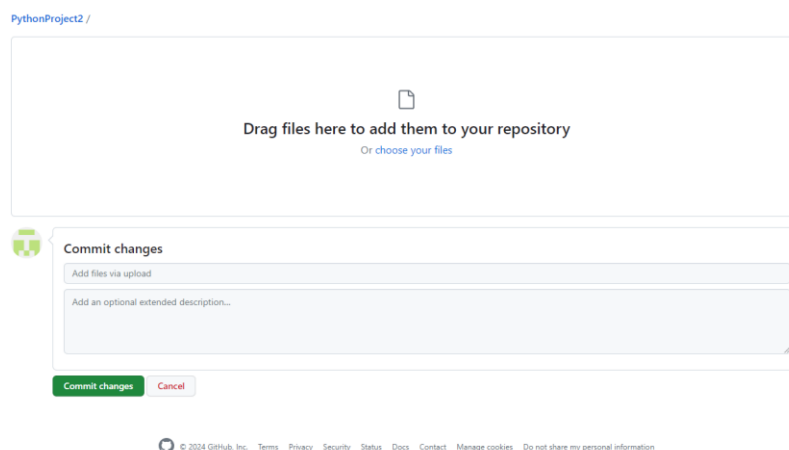


Рис. 1.45. Окно выбора файлов для добавления

После выбора файлов и каталогов нужно нажать "Открыть" (рис. 1.46), после чего они отобразятся в репозитории (рис. 1.47).

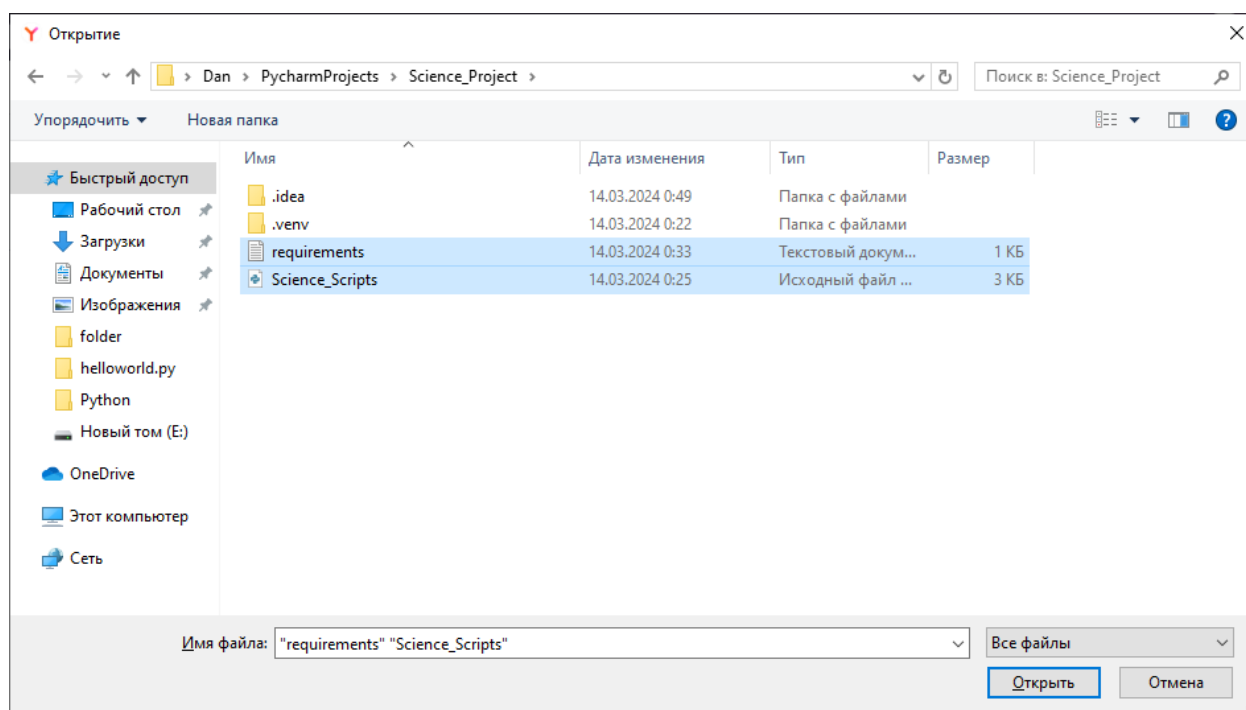


Рис. 1.46. Выбор файлов для добавления

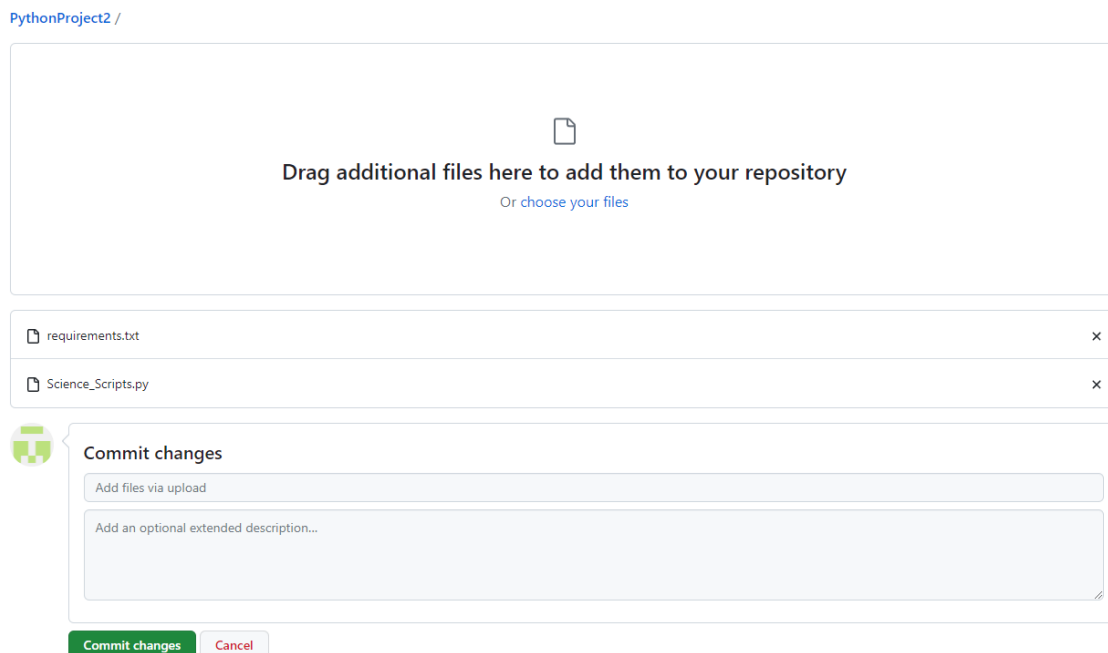


Рис. 1.47. Отображение выбранных файлов в репозитории

После загрузки всех файлов проекта требуется зафиксировать изменения в репозитории нажатием кнопки "Commit changes" внизу страницы (рис. 1.48).

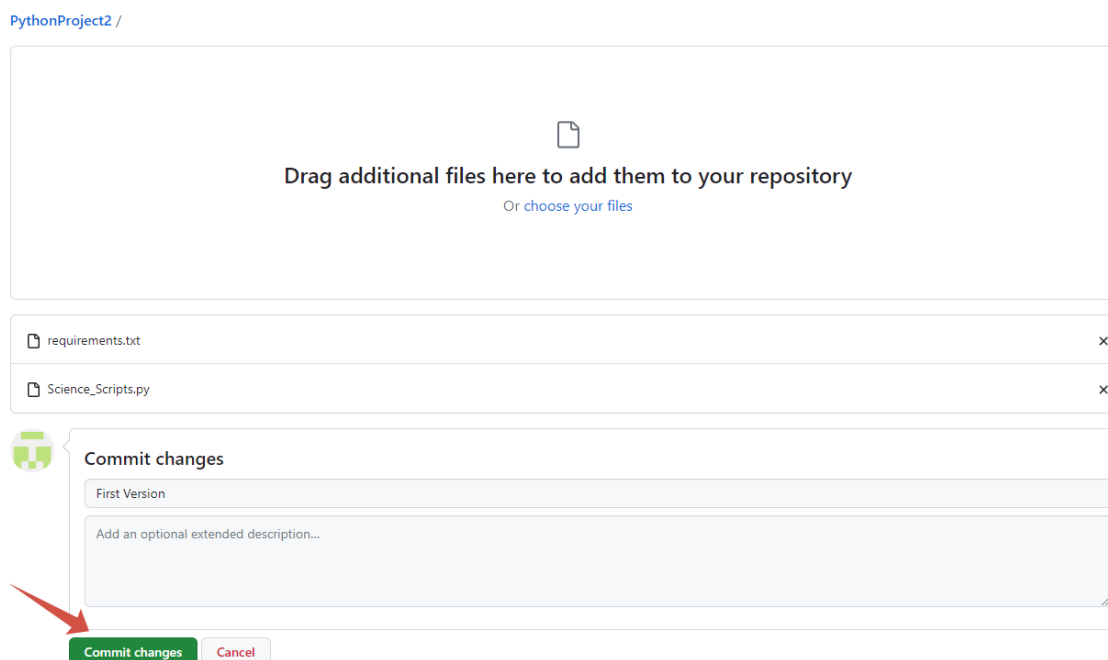


Рис. 1.48. Фиксация изменений в репозитории

При желании можно изменить автоматическое сообщение "*Add files via upload*", например, записать "Первая версия проекта".

После некоторого ожидания загрузки отобразится страница репозитория с файлами проекта (рис. 1.49).

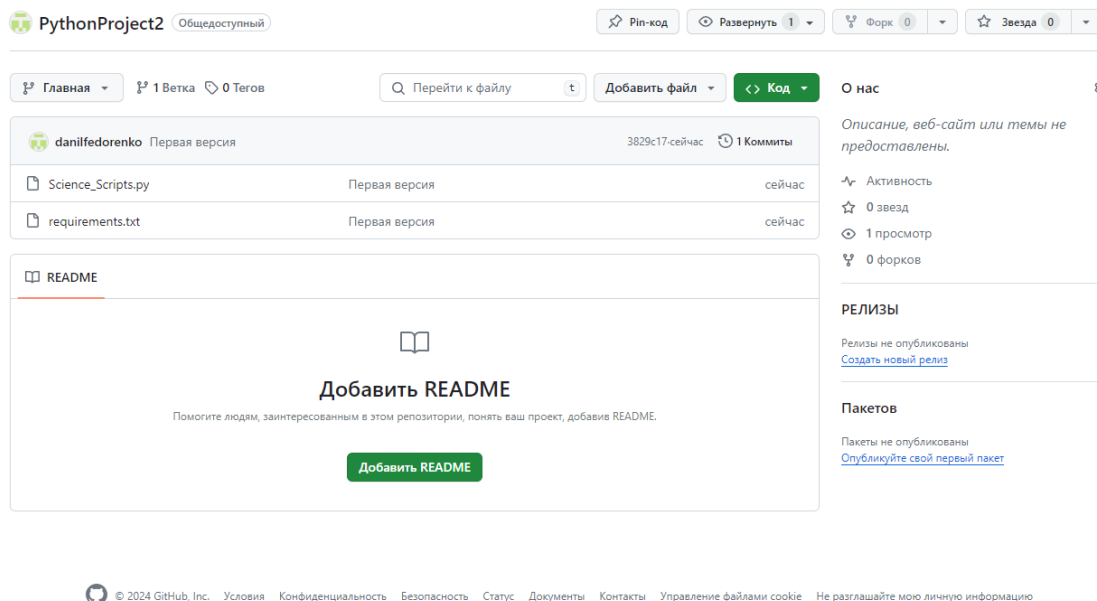


Рис. 1.49. Отображение страницы репозитория с файлами проекта

1.6.3 Загрузка проектов через интерфейс командной строки Git

Еще один способ загрузки файлов в репозиторий требует навыков работы с командной строкой и установленной на компьютере системы контроля версий *Git*, установочный файл которой можно скачать на сайте *git-scm.com*. В процессе установки можно следовать рекомендуемым параметрам.

На странице проекта необходимо переключиться на ссылку "HTTPS" (рис. 1.50) и далее следовать инструкциям из предыдущего раздела.

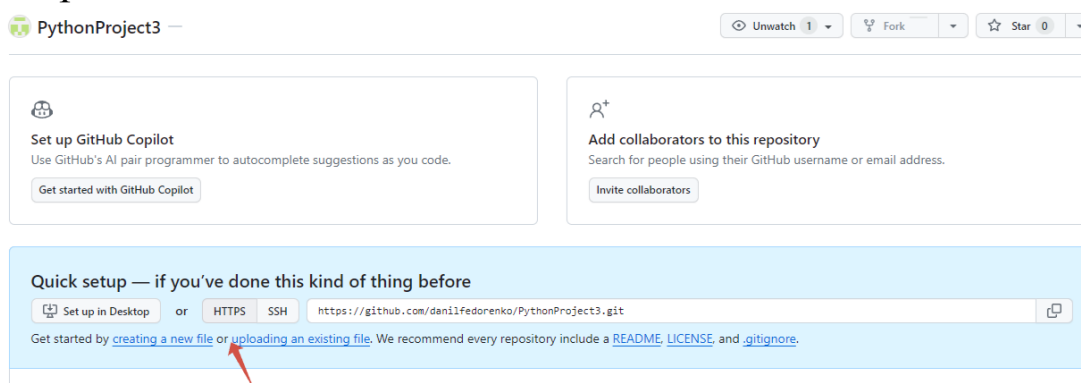
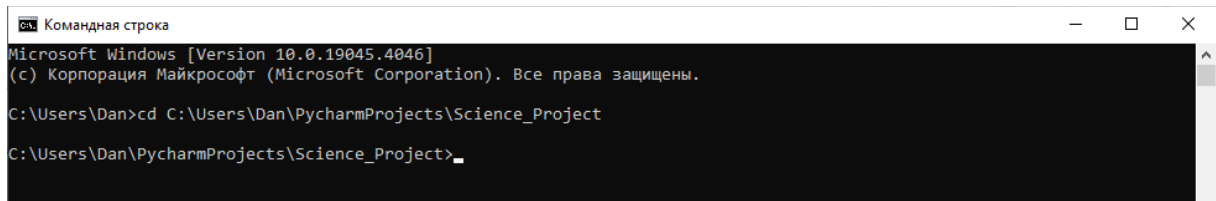


Рис. 1.50. Переключение на ссылку HTTPS

Следующим этапом выполняется запуск командной строки. Перед началом работы рекомендуется ввести свой адрес электронной почты с помощью команды `git config --global user.email "you@example.com"` и задать имя пользователя с помощью команды `git config --global user.name "user name"`, после чего осуществляется переход в папку с проектом (рис. 1.51).

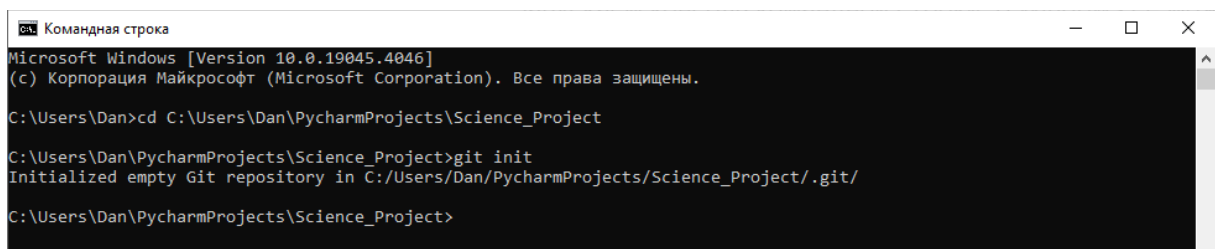


```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project
C:\Users\Dan\PycharmProjects\Science_Project>_
```

Рис. 1.51. Переход в папку с проектом

С помощью команды `git init` выполняется инициализация пустого локального репозитория (рис. 1.52).

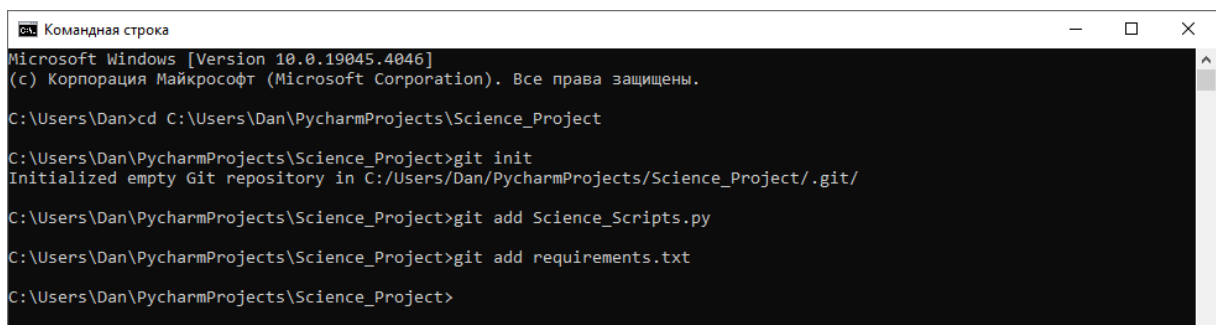


```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project
C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/
C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.52. Инициализация пустого локального репозитория

Добавление в локальный репозиторий файлов проекта выполняется командой `git add` (рис. 1.53).

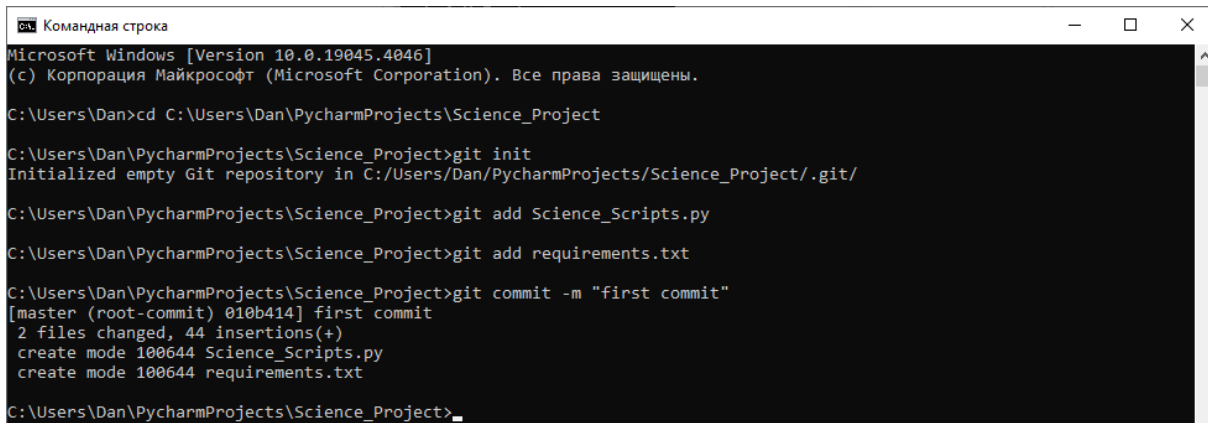


```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project
C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/
C:\Users\Dan\PycharmProjects\Science_Project>git add Science_Scripts.py
C:\Users\Dan\PycharmProjects\Science_Project>git add requirements.txt
C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.53. Добавление файлов проекта в локальный репозиторий

С помощью команды `git commit` осуществляется фиксирование добавленных в репозиторий файлов (рис. 1.54).



```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project

C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/

C:\Users\Dan\PycharmProjects\Science_Project>git add Science_Scripts.py

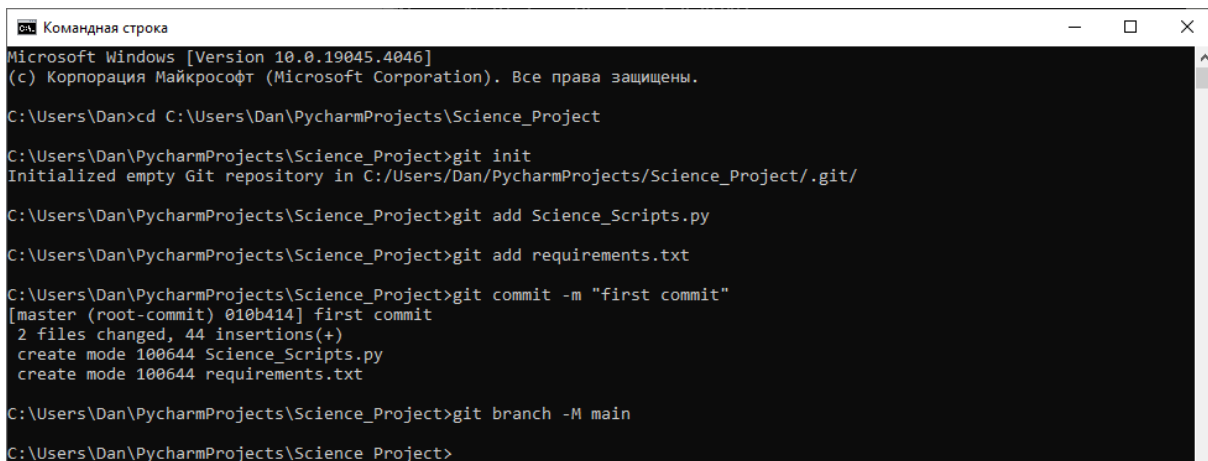
C:\Users\Dan\PycharmProjects\Science_Project>git add requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git commit -m "first commit"
[master (root-commit) 010b414] first commit
 2 files changed, 44 insertions(+)
 create mode 100644 Science_Scripts.py
 create mode 100644 requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.54. Фиксирование добавленных в репозиторий файлов

Далее необходимо переименовать главную ветку проекта в *main* с помощью команды *git branch -M main* (рис. 1.55).



```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project

C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/

C:\Users\Dan\PycharmProjects\Science_Project>git add Science_Scripts.py

C:\Users\Dan\PycharmProjects\Science_Project>git add requirements.txt

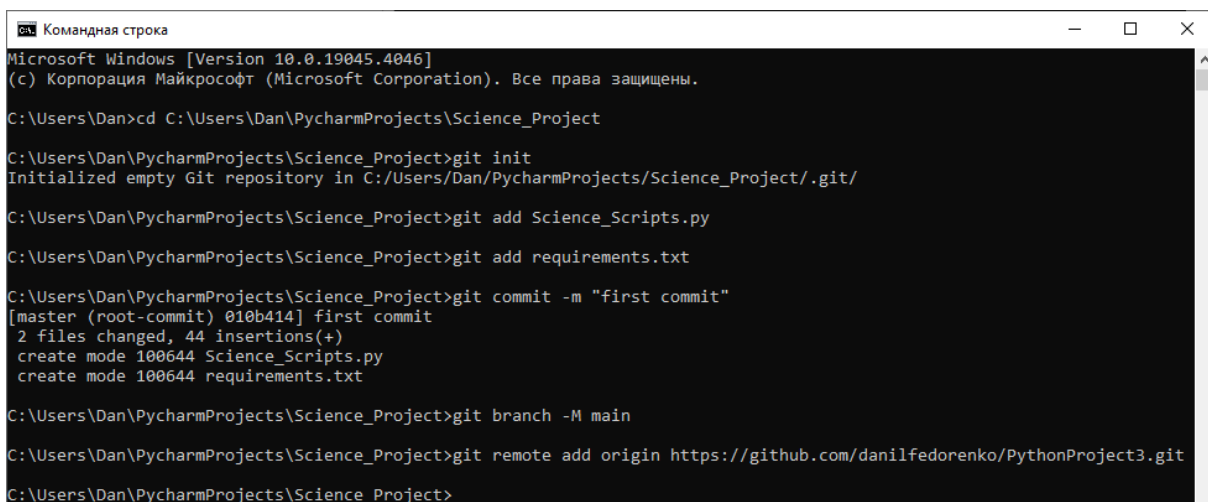
C:\Users\Dan\PycharmProjects\Science_Project>git commit -m "first commit"
[master (root-commit) 010b414] first commit
 2 files changed, 44 insertions(+)
 create mode 100644 Science_Scripts.py
 create mode 100644 requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git branch -M main

C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.55. Переименование главной ветки проекта

Команда *git remote add origin* назначает ранее созданный репозиторий *GitHub* удаленным репозиторием для текущего проекта (рис. 1.56).



```
Командная строка
Microsoft Windows [Version 10.0.19045.4046]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Dan>cd C:\Users\Dan\PycharmProjects\Science_Project

C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/

C:\Users\Dan\PycharmProjects\Science_Project>git add Science_Scripts.py

C:\Users\Dan\PycharmProjects\Science_Project>git add requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git commit -m "first commit"
[master (root-commit) 010b414] first commit
 2 files changed, 44 insertions(+)
 create mode 100644 Science_Scripts.py
 create mode 100644 requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git branch -M main

C:\Users\Dan\PycharmProjects\Science_Project>git remote add origin https://github.com/danilfedorenko/PythonProject3.git

C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.56. Добавление удаленного репозитория

Завершающим шагом является добавление локального репозитория к *GitHub* командой *git push -u origin main* (рис. 1.57). В ходе выполнения данного процесса могут быть запрошены учетные данные пользователя.

```
Командная строка
C:\Users\Dan\PycharmProjects\Science_Project>git init
Initialized empty Git repository in C:/Users/Dan/PycharmProjects/Science_Project/.git/

C:\Users\Dan\PycharmProjects\Science_Project>git add Science_Scripts.py

C:\Users\Dan\PycharmProjects\Science_Project>git add requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git commit -m "first commit"
[master (root-commit) 010b414] first commit
 2 files changed, 44 insertions(+)
 create mode 100644 Science_Scripts.py
 create mode 100644 requirements.txt

C:\Users\Dan\PycharmProjects\Science_Project>git branch -M main

C:\Users\Dan\PycharmProjects\Science_Project>git remote add origin https://github.com/danilfedorenko/PythonProject3.git

C:\Users\Dan\PycharmProjects\Science_Project>git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 1.29 KiB | 1.29 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/danilfedorenko/PythonProject3.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

C:\Users\Dan\PycharmProjects\Science_Project>
```

Рис. 1.57. Добавление локального репозитория в *GitHub*

В результате выполненных действий на странице проекта в *GitHub* будут отображаться добавленные файлы (рис. 1.58).

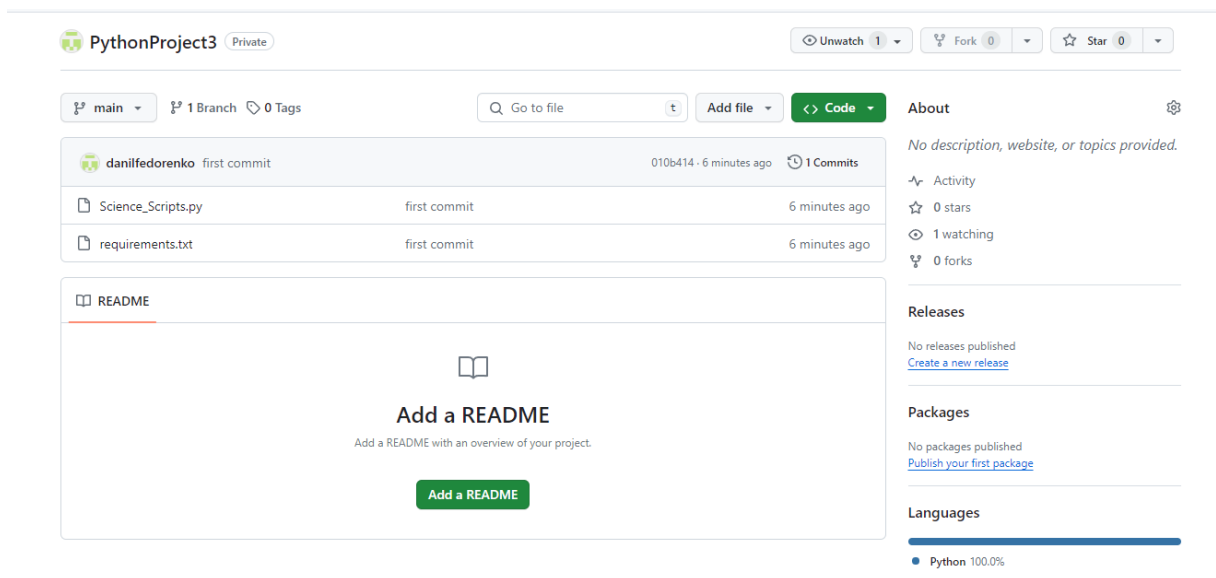


Рис. 1.58. Результат добавления файлов к проекту

1.6.4 Работа с локальным репозиторием

Ранее были упомянуты команды создания локального репозитория и добавления в него файлов.

В рассматриваемом далее примере проект находится по адресу *C:/Python*.

После выполнения команды *git init* в директории проекта создается новый каталог с именем *.git*. В нем находится несколько файлов и подкаталогов.

Команда *git add* осуществляет добавление в систему управления версий указанного файла. В ходе данного процесса генерируется хеш-значение, в примере для добавленного файла *Lab.py* полученное значение равно *4e4f5503f495f7e74765a3d9449cb97b3a9bcd2* (рис. 1.59).

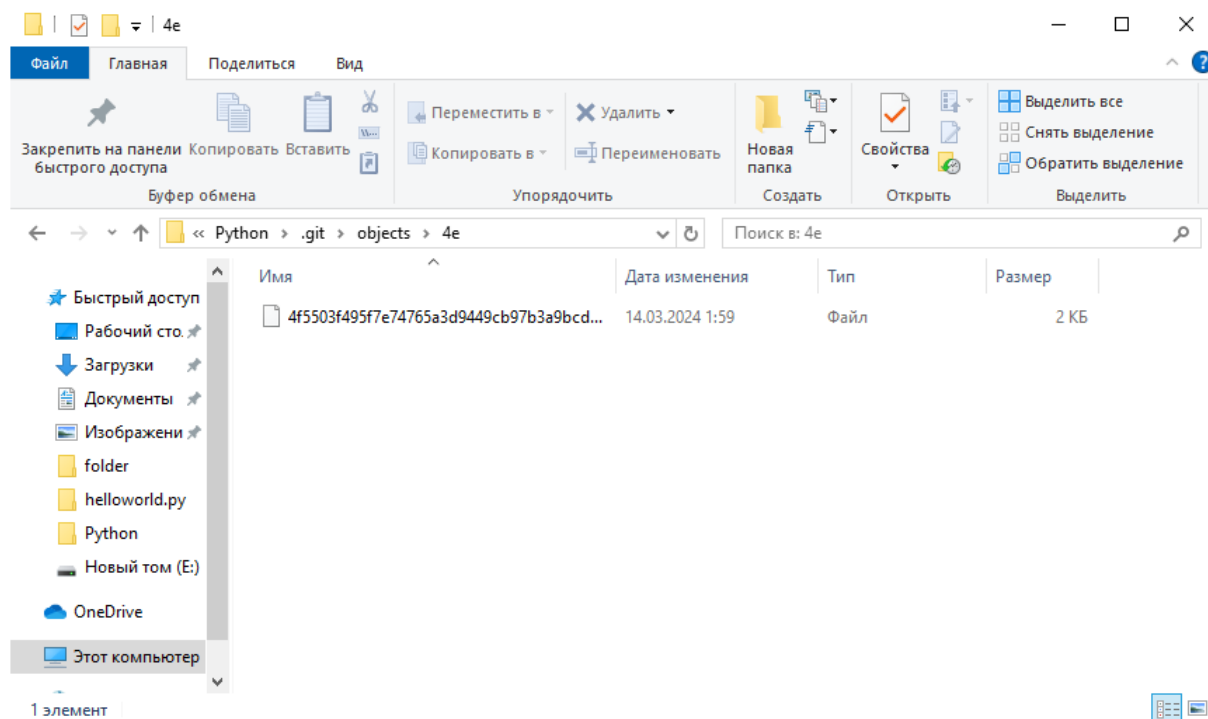
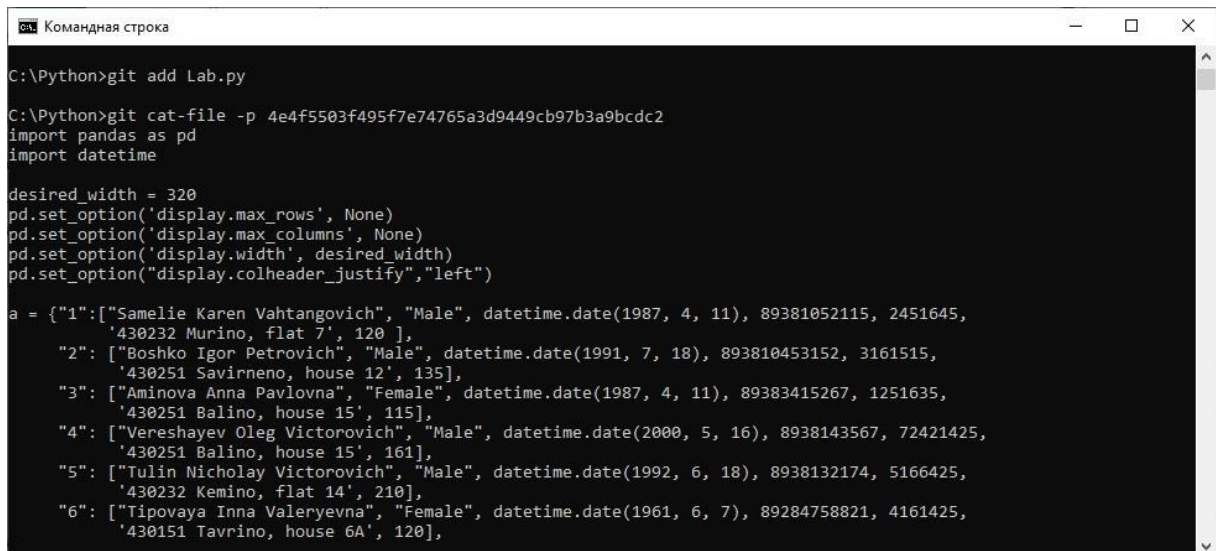


Рис. 1.59. Расположение файла, добавленного в локальный репозиторий

Добавленные файлы хранятся в каталоге *.git/objects/xx/yyyyyyyy*, при этом первые две цифры хеша используются для указания директории, а остальное хеш-значение является именем файла.

Полученный файл является архивом, который можно распаковать и вывести на экран, указав полное хеш-значение (рис. 1.60).



```
C:\Python>git add Lab.py

C:\Python>git cat-file -p 4e4f5503f495f7e74765a3d9449cb97b3a9bcd2
import pandas as pd
import datetime

desired_width = 320
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', desired_width)
pd.set_option("display.colheader_justify", "left")

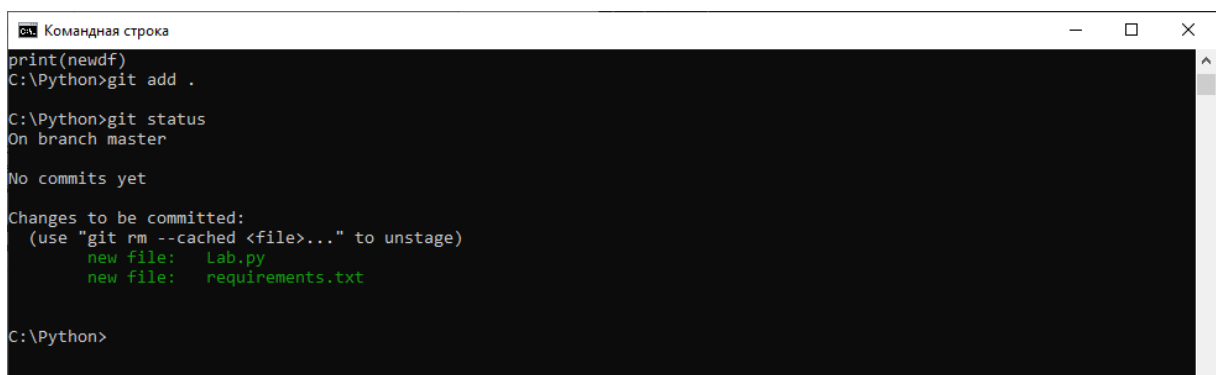
a = {"1":["Samelie Karen Vahtangovich", "Male", datetime.date(1987, 4, 11), 89381052115, 2451645,
'430232 Murino, flat 7', 120 ],
      "2":["Boshko Igor Petrovich", "Male", datetime.date(1991, 7, 18), 893810453152, 3161515,
'430251 Savirnenno, house 12', 135],
      "3":["Aminova Anna Pavlovna", "Female", datetime.date(1987, 4, 11), 89383415267, 1251635,
'430251 Balino, house 15', 115],
      "4":["Vereshayev Oleg Victorovich", "Male", datetime.date(2000, 5, 16), 8938143567, 72421425,
'430251 Balino, house 15', 161],
      "5":["Tulin Nicholay Victorovich", "Male", datetime.date(1992, 6, 18), 8938132174, 5166425,
'430232 Kemino, flat 14', 210],
      "6":["Tipovaya Inna Valeryevna", "Female", datetime.date(1961, 6, 7), 89284758821, 4161425,
'430151 Tavrino, house 6A', 120],
```

Рис. 1.60 Вывод на экран текста из файла локального репозитория

Команда *git add* используется для добавления в репозиторий всех файлов из текущей директории, команда *git add -all* – для добавления файлов из всех вложенных каталогов.

После добавления файлов все изменения находятся во временном хранилище *staging (cached) area*, которое используется для накопления изменений и создания версий проектов (*commit*).

Для просмотра текущего состояния применяется команда *git status* (рис. 1.61).



```
C:\Python>git add .

C:\Python>git status
On branch master

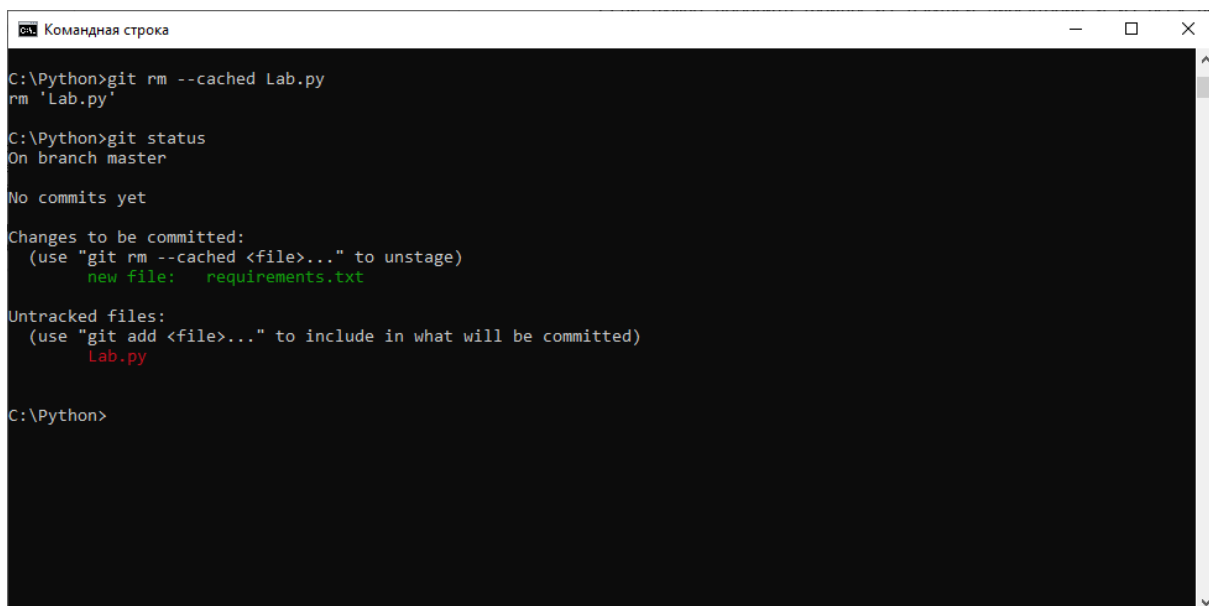
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Lab.py
        new file:   requirements.txt

C:\Python>
```

Рис. 1.61. Отображение информации о текущем состоянии репозитория

Отмена добавления файла *Lab.py* в *staging area* выполняется с помощью команды *git rm --cached Lab.py* (рис. 1.62).



```
C:\Python>git rm --cached Lab.py
rm 'Lab.py'

C:\Python>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   requirements.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Lab.py

C:\Python>
```

Рис. 1.62. Отмена добавления файла в репозиторий

Для предотвращения попадания в систему временных файлов следует создать на том же уровне, что и *.git* директория файл *.gitignore* (рис. 1.63).

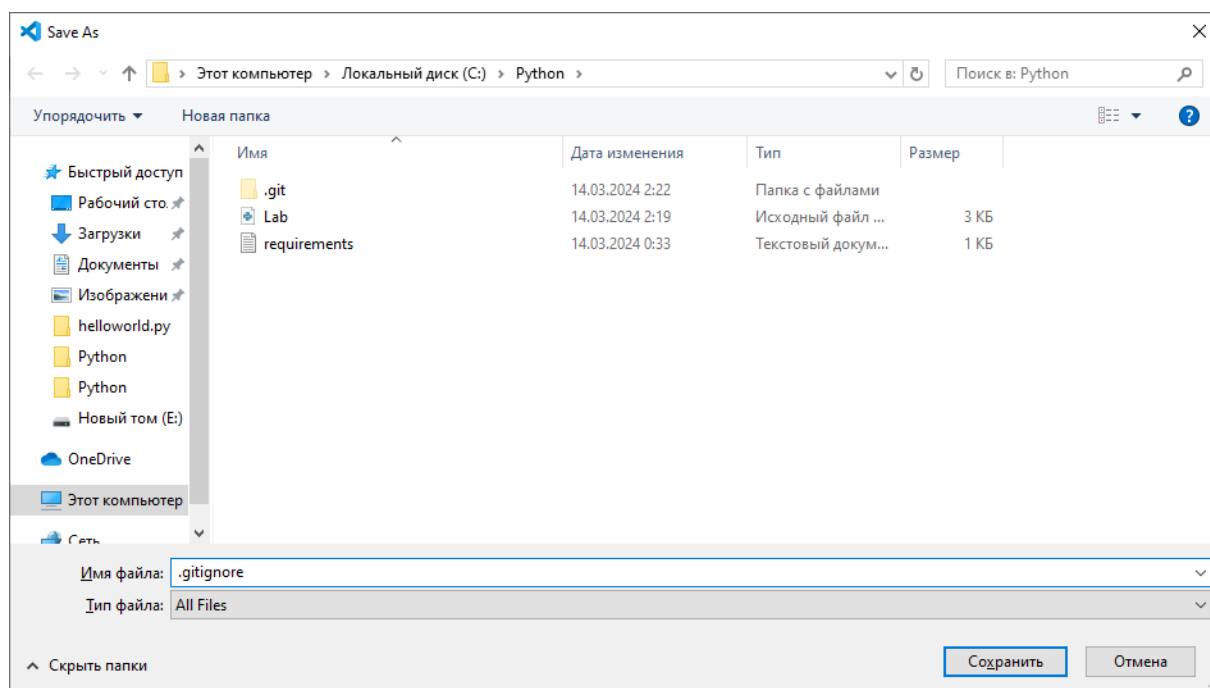


Рис. 1.63. Создание файла *.gitignore*

Например, если в *.gitignore* добавить строчку **.рус*, то все файлы с расширением *.рус* не будут добавляться в репозиторий (рис. 1.64).



Рис. 1.64. Заполнение файла *.gitignore*

1.6.5 Управление версиями проектов

После добавление требуемых файлов в *staging area* можно создать версию проекта, выполнив команду *git commit* (рис. 1.65).

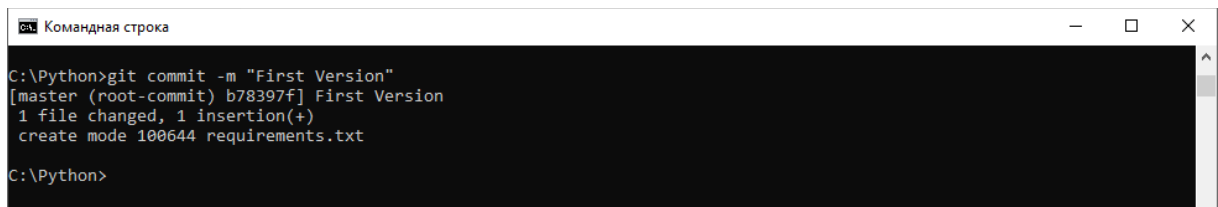


Рис. 1.65. Создание первой версии проекта

На рис. 1.65 создание версии содержит опцию *'-m'*, которая позволяет написать сообщение вместе с командой, не открывая стороннего редактора.

Каждая новая версия сопровождается комментарием. Первую созданную версию нельзя отменить, но можно исправить.

Для внесения изменений в последнюю созданную версию используется команда *git commit -m "comment" --amend* (рис. 1.66).

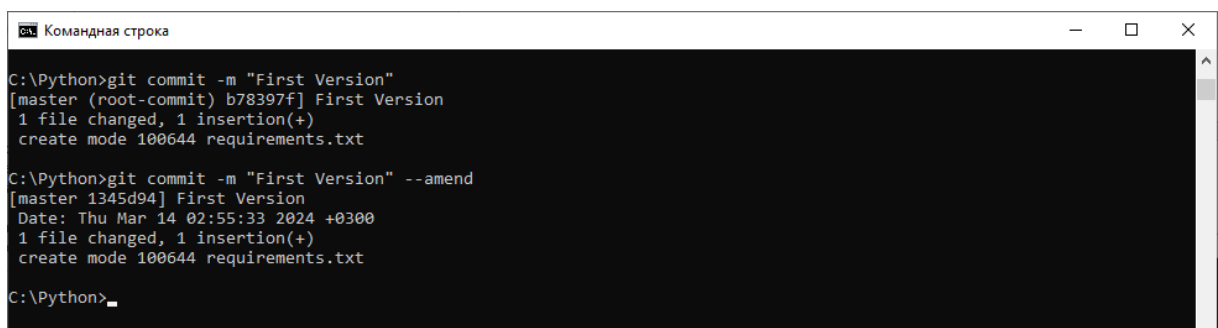
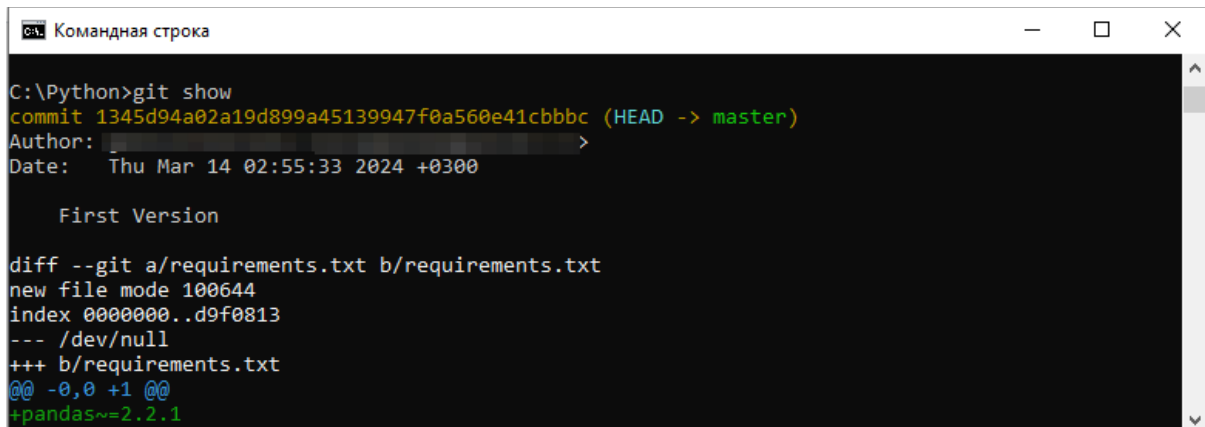


Рис. 1.66. Добавление изменений в последнюю версию проекта

Если дополнительно использовать ключ *--noedit*, комментарий вводить заново не нужно. В этом случае команда будет иметь следующий вид: *git commit --amend --no-edit*.

Для просмотра внесенных изменений можно выполнить команды `git show` (рис. 1.67) или `git show --name-only`, где ключ `--name-only` используется для отображения только имен измененных файлов (рис. 1.68). Без него по каждому измененному файлу будет выдан список всех изменений.



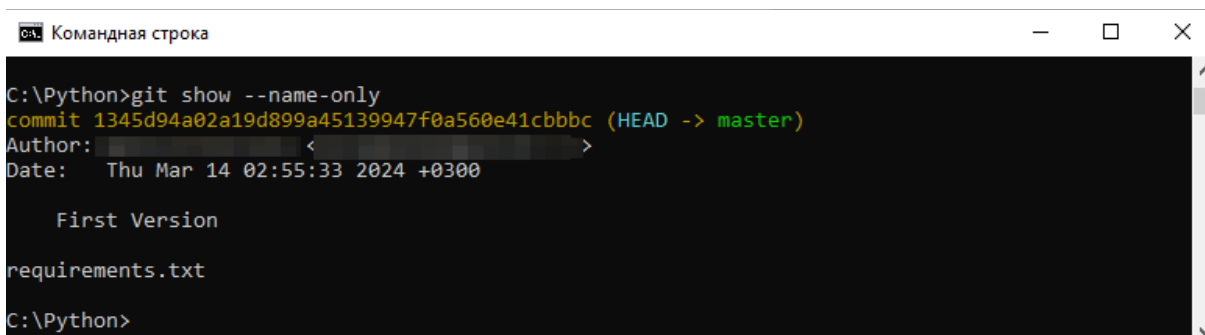
```
Командная строка

C:\Python>git show
commit 1345d94a02a19d899a45139947f0a560e41cbbbc (HEAD -> master)
Author: <[redacted]>
Date: Thu Mar 14 02:55:33 2024 +0300

    First Version

diff --git a/requirements.txt b/requirements.txt
new file mode 100644
index 0000000..d9f0813
--- /dev/null
+++ b/requirements.txt
@@ -0,0 +1 @@
+pandas~=2.2.1
```

Рис. 1.67. Внесенные изменения в последнюю версию проекта



```
Командная строка

C:\Python>git show --name-only
commit 1345d94a02a19d899a45139947f0a560e41cbbbc (HEAD -> master)
Author: <[redacted]>
Date: Thu Mar 14 02:55:33 2024 +0300

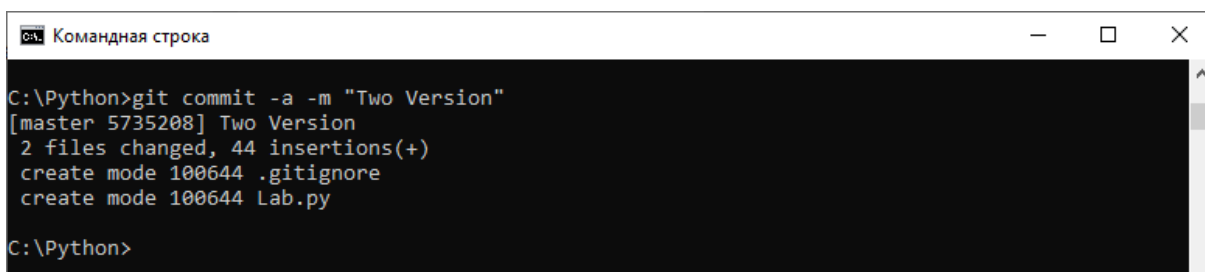
    First Version

requirements.txt

C:\Python>
```

Рис. 1.68. Отображение имени измененного файла

Если изменения затронули только уже добавленные файлы, их зафиксировать можно одной командой `git commit -a -m "Two Version"` (рис. 1.69).



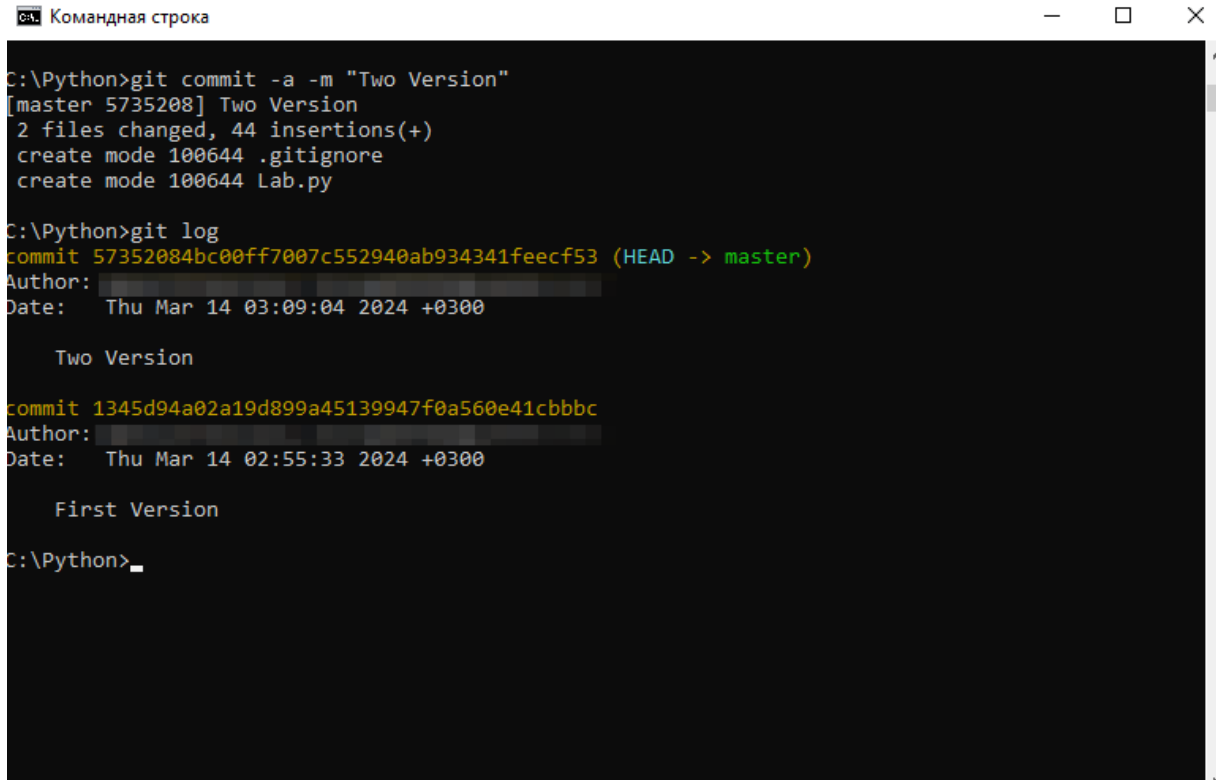
```
Командная строка

C:\Python>git commit -a -m "Two Version"
[master 5735208] Two Version
 2 files changed, 44 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Lab.py

C:\Python>
```

Рис. 1.69. Внесение изменений в добавленные файлы

Просмотр всех изменений осуществляется командой *git log* (рис. 1.70) или *git log --oneline*, где ключ *--oneline* нужен для уменьшения количества информации выводимой на экран (рис. 1.71).



```

C:\Python>git commit -a -m "Two Version"
[master 5735208] Two Version
 2 files changed, 44 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Lab.py

C:\Python>git log
commit 57352084bc00ff7007c552940ab934341feecf53 (HEAD -> master)
Author: 
Date:   Thu Mar 14 03:09:04 2024 +0300

    Two Version

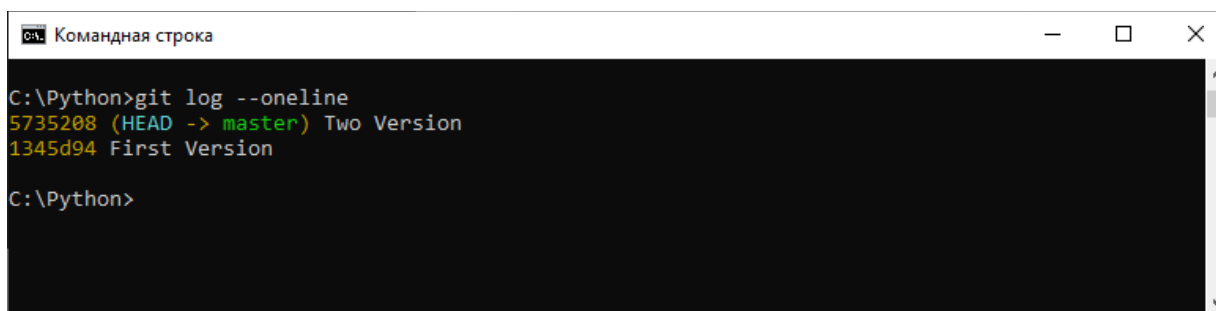
commit 1345d94a02a19d899a45139947f0a560e41cbbbc
Author: 
Date:   Thu Mar 14 02:55:33 2024 +0300

    First Version

C:\Python>_

```

Рис. 1.70. Просмотр списка всех версий



```

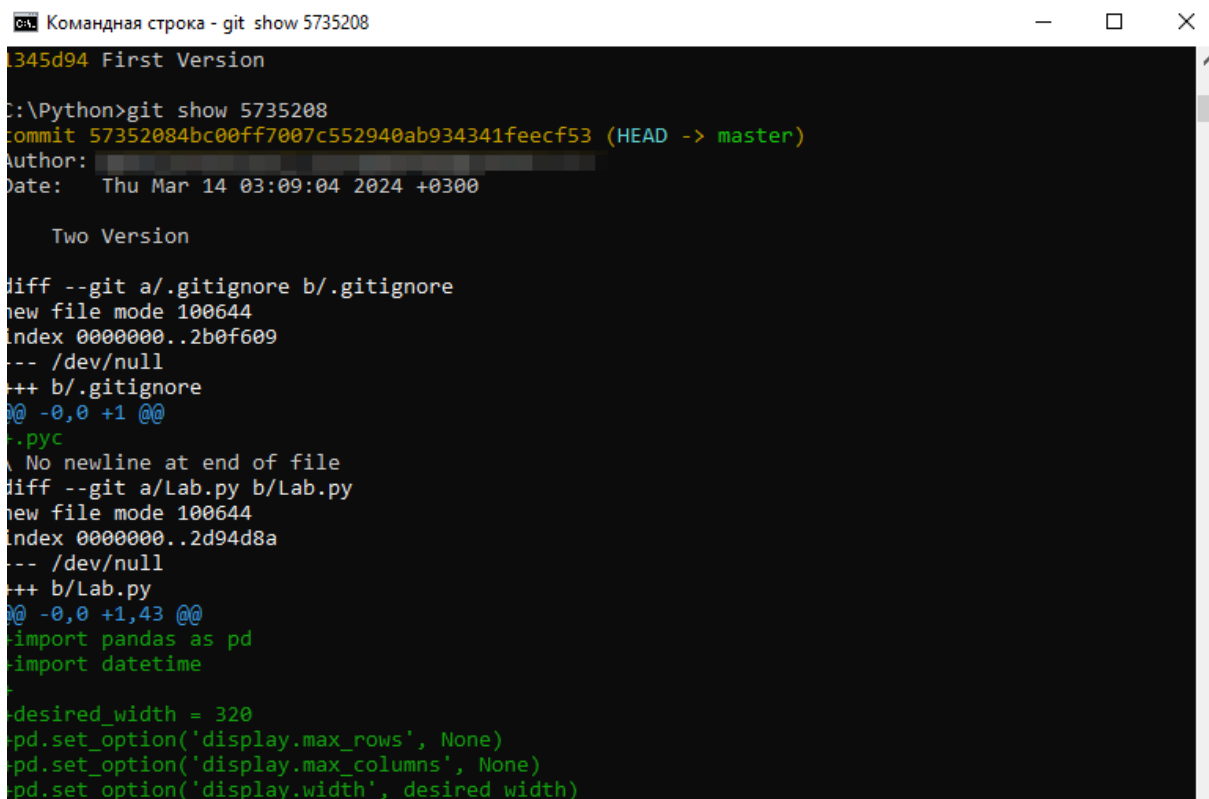
C:\Python>git log --oneline
5735208 (HEAD -> master) Two Version
1345d94 First Version

C:\Python>

```

Рис. 1.71. Упрощенная выдача информации о версиях

Для просмотра изменений по конкретной версии необходимо в команду *git show* добавить хеш-значение версии (рис. 1.72).



```
Командная строка - git show 5735208

1345d94 First Version

C:\Python>git show 5735208
commit 57352084bc00ff7007c552940ab934341feecf53 (HEAD -> master)
Author: 
Date: Thu Mar 14 03:09:04 2024 +0300

    Two Version

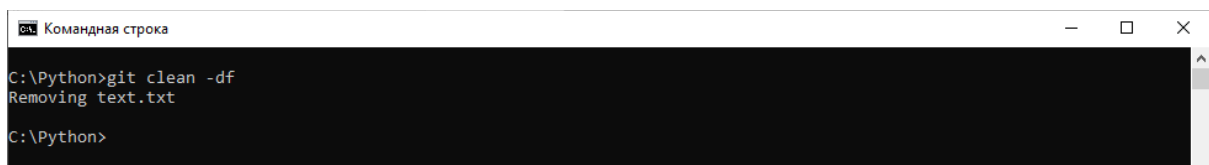
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..2b0f609
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+.pyc
\ No newline at end of file
diff --git a/Lab.py b/Lab.py
new file mode 100644
index 0000000..2d94d8a
--- /dev/null
+++ b/Lab.py
@@ -0,0 +1,43 @@
import pandas as pd
import datetime

desired_width = 320
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', desired_width)
```

Рис. 1.72. Просмотр изменений версии проекта

Для отмены последнего сохранения (кроме самого первого) можно воспользоваться *git reset Head~1*.

Удаление из каталога файлов, которые не относятся к проекту и не сохранены в репозитории, осуществляется командой *git clean -df* (рис. 1.73).



```
Командная строка

C:\Python>git clean -df
Removing text.txt

C:\Python>
```

Рис. 1.73. Удаление неиспользуемых файлов из папки с проектом

Рассмотренный материал демонстрирует главную функцию программного обеспечения контроля версий – возможность отслеживания всех вносимых в код изменений. При обнаружении ошибки разработчики могут вернуться назад и выполнить сравнение с более ранними версиями кода для исправления ошибок, сводя к минимуму проблемы для всех участников команды.