

An understanding of the technologies previously mentioned is required to build a full-blown FastAPI application. It also serves as an addition to your current skillset.

At the completion of this chapter, you will be able to set up and use Git, install and manage packages using pip, create an isolated development environment with Virtualenv, use Docker, and most importantly, scaffold a FastAPI application.

This chapter covers the following topics:

- Git basics
- Creating isolated development environments with Virtualenv
- Package management with pip
- Setting up and learning the basics of Docker
- Building a simple FastAPI application

Technical Requirement

You can find the code files for this chapter on GitHub at <https://github.com/PacktPublishing/Building-Python-Web-APIs-with-FastAPI/tree/main/ch01>

Git basics

Git is a version control system that enables developers to record, keep track, and revert to earlier versions of files. It is a decentralized and lightweight tool that can be installed on any operating system.

You will be learning how to use Git for record-keeping purposes. As each layer of the application is being built, changes will be made, and it's important that these changes are kept note of.

Installing Git

To install Git, visit the downloads page at <https://git-scm.com/downloads> and select a download option for your current operating system. You'll be redirected to an instructional page on how to install Git on your machine.

It is also worth noting that Git comes as a **CLI** and a **GUI** application. Therefore, you can download the one that works best for you.

Git operations

As mentioned earlier, Git can be used to record, track, and revert to earlier versions of a file. However, only the basic operations of Git will be used in this book and will be introduced in this section.

In order for Git to run properly, folders housing files must be initialized. Initializing folders enables Git to keep track of the content except otherwise exempted.

To initialize a new Git repository in your project, you need to run the following command in your terminal:

```
$ git init
```

To enable tracking of files, a file must first be added and committed. A Git commit enables you to track file changes between timeframes; for example, a commit made an hour ago and the current file version.

What Is a Commit?

A commit is a unique capture of a file or folder status at a particular time, and it is identified by a unique code.

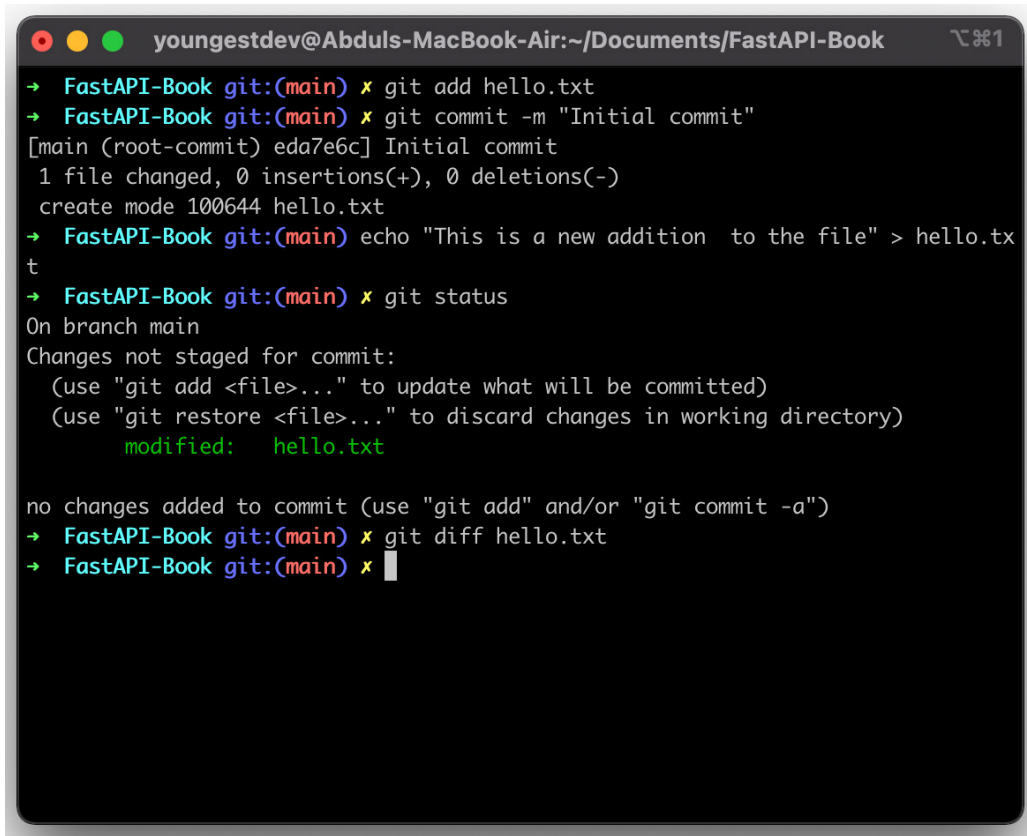
Now that we know what a commit is, we can go ahead and commit a file as follows:

```
$ git add hello.txt  
$ git commit -m "Initial commit"
```

You can track the status of your files after making changes by running the following command:

```
$ git status
```

Your terminal should look similar to the following:

A screenshot of a macOS terminal window titled 'youngestdev@Abduls-MacBook-Air:~/Documents/FastAPI-Book'. The terminal shows a series of Git commands and their outputs. The user starts by adding 'hello.txt', committing it with the message 'Initial commit', and then editing the file with 'echo'. Finally, they run 'git status' which shows the file as modified but not staged for commit. The terminal text is as follows:

```
→ FastAPI-Book git:(main) x git add hello.txt
→ FastAPI-Book git:(main) x git commit -m "Initial commit"
[main (root-commit) eda7e6c] Initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello.txt
→ FastAPI-Book git:(main) echo "This is a new addition to the file" > hello.txt
→ FastAPI-Book git:(main) x git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt

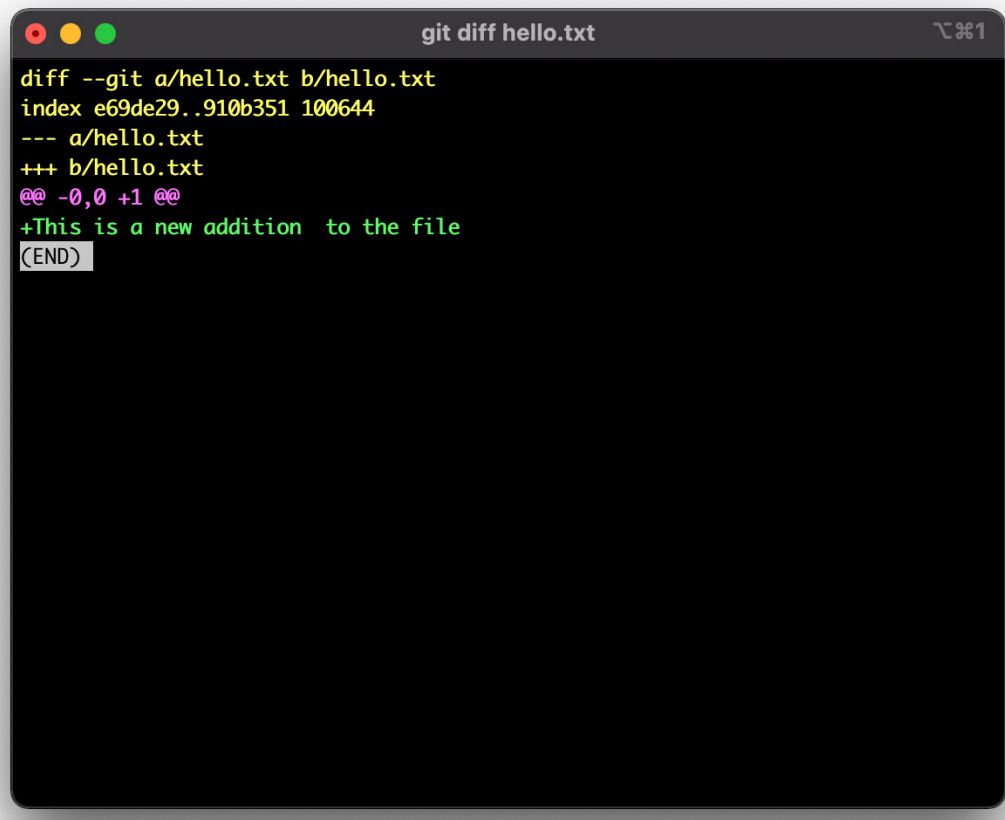
no changes added to commit (use "git add" and/or "git commit -a")
→ FastAPI-Book git:(main) x git diff hello.txt
→ FastAPI-Book git:(main) x
```

Figure 1.1 – Git commands

To view the changes made to the file, which can be additions or subtractions from the file contents, run the following command:

```
$ git diff
```

Your terminal should look similar to the following:

A terminal window titled "git diff hello.txt" with standard macOS window controls (red, yellow, green buttons) in the top-left corner. The terminal output is as follows:

```
diff --git a/hello.txt b/hello.txt
index e69de29..910b351 100644
--- a/hello.txt
+++ b/hello.txt
@@ -0,0 +1 @@
+This is a new addition to the file
(END)
```

The text is color-coded: "diff" is yellow, "index" is yellow, "a/hello.txt" is yellow, "b/hello.txt" is yellow, "0,0" is purple, "1" is purple, and the rest is green. The "(END)" prompt is highlighted with a grey background.

Figure 1.2 – Output from the git diff command

It is good practice to include a `.gitignore` file in every folder. The `.gitignore` file contains the names of files and folders to be ignored by Git. This way, you can add and commit all the files in your folder without the fear of committing files like `.env`.

To include a `.gitignore` file, run the following command in your terminal:

```
$ touch .gitignore
```

To exempt a file from being tracked by Git, add it to the `.gitignore` file as follows:

```
$ echo ".env" >> .gitignore
```

Common files contained in a `.gitignore` file include the following:

- Environment files (`*.env`)
- Virtualenv folder (`env`, `venv`)
- IDE metadata folders (such as `.vscode` and `.idea`)

Git branches

Branches are an important feature that enables developers to easily work on different application features, bugs, and so on, separately before merging into the main branch. The system of branching is employed in both small-scale and large-scale applications and promotes the culture of previewing and collaborations via pull requests. The primary branch is called the main branch and it is the branch from which other branches are created.

To create a new branch from an existing branch, we run the `git checkout -b newbranch` command. Let's create a new branch by running the following command:

```
$ git checkout -b hello-python-branch
```

The preceding command creates a new branch from the existing one, and then sets the active branch to the newly created branch. To switch back to the original main branch, we run `git checkout main` as follows:

```
$ git checkout main
```

Important Note

Running `git checkout main` makes `main` the active working branch, whereas `git checkout -b newbranch` creates a new branch from the current working branch and sets the newly created branch as the active one.

To learn more, refer to the Git documentation: <http://www.git-scm.com/doc>.

Now that we have learned the basics of Git, we can now proceed to learn about how to create isolated environments with **virtualenv**.