Chapter 1: Initial Set Up

djangoforbeginners.com/initial-setup

Install Django 4.0, Python 3.10, Git, and virtual environments

This chapter covers how to properly configure your Windows or macOS computer to work on Django projects. We start with an overview of the *Command Line*, a powerful text-only interface that developers use extensively to install and configure Django projects. Then we install the latest version of Python, learn how to create dedicated virtual environments, and install Django. As a final step, we will explore using Git for version control and working with a text editor. By the end of this chapter you will have created your first Django project from scratch and more importantly be able to create and modify new Django projects in just a few keystrokes.

The Command Line

If you have ever seen a television show or movie where a hacker is furiously typing into a black window: that's the command line. It is an alternative to the mouse or finger-based graphical user interface familiar to most computer users. An everyday computer user will never need to use the command line but software developers do because certain tasks can only be done with it. These include running programs, installing software, using Git for version control, or connecting to servers in the cloud. With a little practice, most developers find that the command line is actually a faster and more powerful way to navigate and control a computer.

Given its minimal user interface–just a blank screen and a blinking cursor–the command line is intimidating to newcomers. There is often no feedback after a command has run and it is possible to wipe the contents of an entire computer with a single command if you're not careful: no warning will pop up! As a result, the command line must be used with caution. Make sure not to blindly copy and paste commands you find online; only rely on trusted resources for any command you do not fully understand.

In practice, multiple terms are used to refer to the command line: Command Line Interface (CLI), console, terminal, shell, or prompt. Technically speaking, the *terminal* is the program that opens up a new window to access the command line, a *console* is a text-based application, the *shell* is the program that runs commands on the underlying operating system, and the *prompt* is where commands are typed and run. It is easy to be confused by these terms initially but they all essentially mean the same thing: the command line is where we run and execute text-only commands on our computer.

On Windows, the built-in terminal and shell are both called *PowerShell*. To access it, locate the taskbar on the bottom of the screen next to the Windows button and type in "powershell" to launch the app. It will open a new window with a dark blue background and a blinking cursor after the `>` prompt. Here is how it looks on my computer.

```
PS C:\Users\wsv>
```

Before the prompt is `PS` which refers to PowerShell, the initial `C` directory of the Windows operating system, followed by the `Users` directory and the current user which, on my personal computers, is `wsv`. Your username will obviously be different. At this point, don't worry about what comes *to the left* of the `>` prompt: it varies depending on each computer and can be customized at a later date. The shorter prompt of `>` will be used going forward for Windows.

On macOS, the built-in terminal is called appropriately enough *Terminal*. It can be opened via Spotlight: press the `Command` and `space bar` keys at the same time and then type in "terminal." Alternatively, open a new Finder window, navigate to the *Applications* directory, scroll down to open the *Utilities* directory, and double-click the application called Terminal. This opens a new screen with a white background by default and a blinking cursor after the `%` prompt. Don't worry about what comes *to the left* of the `%` prompt. It varies by computer and can be customized later on.

```
Wills-Macbook-Pro:~ wsv%
```

If your macOS prompt is `$` instead of `%` that means you are using `Bash` as the shell. Starting in 2019, macOS switched from *Bash* to *zsh* as the default shell. While most of the commands in this book will work interchangeably, it is recommended to look up online how to change to zsh via `System Preferences` if your computer still uses Bash.

Shell Commands

There are many available shell commands but most developers rely on the same handful over and over again and look up more complicated ones as needed.

In most cases, the commands for Windows (PowerShell) and macOS are similar. For example, the command `whoami` returns the computer name/username on Windows and just the username on macOS. As with all shell commands, type the command itself followed by the `return` key. Note that the `#` symbol represents a comment and will not be executed on the command line.

```
# Windows
> whoami
wsv2021/wsv

# macOS
% whoami
wsv
```

Sometimes, however, the shell commands on Windows and macOS are completely different. A good example is the command for outputting a basic "Hello, World!" message to the console. On Windows the command is `Write-Host` while on macOS the command is `echo`.

```
# Windows
> Write-Host "Hello, World!"
Hello, World!

# macOS
% echo "Hello, World!"
Hello, World!
```

A frequent task on the command line is navigating within the computer filesystem. On Windows, the default shell should show the current location but it can also be outputted with `Get-Location`. On macOS use `pwd` (print working directory).

```
# Windows
> Get-Location

Path
----
C:\Users\wsv

# macOS
% pwd
/Users/wsv
```

You can save your Django code anywhere you like but for convenience we will place our code the `desktop` directory. The command `cd` (change directory) followed by the intended location works on both systems.

```
# Windows
> cd onedrive\desktop
> Get-Location

Path
----
C:\Users\wsv\onedrive\desktop

# macOS
% cd desktop
% pwd
/Users/wsv/desktop
```

Tip: The `tab` key will autocomplete a command so if you type `cd d` and then hit `tab` it will automatically fill in the rest of the name. If there are more than two directories that start with `d`, hit the `tab` key again to cycle through them.

To make a new directory use the command `mkdir` followed by the name. We will create one called `code` on the Desktop and then within it a new directory called `ch1-setup`.

```
# Windows
> mkdir code
> cd code
> mkdir ch1-setup
> cd ch1-setup

# macOS
% mkdir code
% cd code
% mkdir ch1-setup
% cd ch1-setup
```

You can check that it has been created by looking on your Desktop or running the command `ls`. The full Windows output is slightly longer but is shortened here for conciseness.

```
# Windows
> ls
testdir

# macOS
% ls
testdir
```

Tip: The `clear` command will clear the Terminal of past commands and outputs so you have a clean slate. The `tab` command autocompletes the line as we've discussed. And the `↑` and `↓` keys cycle through previous commands to save yourself from typing the same thing over and over again.

To exit you could close the Terminal with your mouse but the hacker way is to use use the shell command `exit` instead. This works by default on Windows but on macOS the Terminal preferences need to be changed. At the top of the screen click on `Terminal`, then `Preferences` from the drop down menu. Click on `Profiles` in the top menu and then `Shell` from the list below. There is a radio button for "When the shell exits:". Select "Close the window."

```
# Windows
> exit

# macOS
% exit
```

Kinda cool, right? With practice, the command line is a far more efficient way to navigate and operate your computer than using a mouse. For this book you don't need to be a command line expert: I will provide the exact instructions to run each time. But if you are curious, a complete list of shell commands for each operating system can be found over at `ss64.com`.

Install Python 3 on Windows

On Windows, Microsoft hosts a community release of Python 3 in the Microsoft Store. In the search bar on the bottom of your screen type in "python" and click on the best match result. This will automatically launch Python 3.10 on the Microsoft Store. Click on the blue "Get" button to download it.

To confirm Python was installed correctly, open a new Terminal window with PowerShell and then type `python --version`.

```
> python --version
Python 3.10.2
```

The result should be at least Python 3.10. Then type `python` to open the Python interpreter from the command line shell.

```
> python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 19:00:18)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits", or "license" for more information.
>>>
```

Install Python 3 on Mac

On Mac, the official installer on the Python website is the best approach. In a new browser window go the Python downloads page and click on the button underneath the text "Download the latest version for Mac OS X." As of this writing, that is Python 3.10. The package will be in your `Downloads` directory. Double click on it which launches the Python Installer and follow through the prompts.

To confirm the download was successful, open up a new Terminal window and type `python3 --version`.

```
% python3 --version
Python 3.10.2
```

The result should be at least 3.10. Then type `python3` to open the Python interpreter.

```
% python3
Python 3.10.2 (v3.10.2:a58ebcc701, Jan 13 2022, 14:50:16)
 [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python Interactive Mode

From the command line typing either `python` on Windows or `python3` on macOS will bring up the Python Interpreter, also known as Python Interactive mode. The new prompt of `>>>` indicates that you are now inside Python itself and **not** the command line. If you try any of the previous shell commands we ran– `cd`, `ls`, `mkdir` –they will each raise errors. What *will* work is actual Python code. For example, try out both `1 + 1` and `print("Hello Python!")` making sure to hit the `Enter` or `Return` key after each to run them.

```
>>> 1 + 1
2
>>> print("Hello Python!")
Hello Python!
```

Python's interactive mode is a great way to save time if you want to try out a short bit of Python code. But it has a number of limitations: you can't save your work in a file and writing longer code snippets is cumbersome. As a result, we will spend most of our time writing Python and Django in files using a text editor.

To exit Python from the command line you can type either `exit()` and the `Enter` key or use `Ctrl + z` on Windows or `Ctrl + d` on macOS.

Virtual Environments

Installing the latest version of Python and Django is the correct approach for any new project. But in the real world, it is common that existing projects rely on older versions of each. Consider the following situation: *Project A* uses Django 2.2 but *Project B* uses Django 4.0? By default, Python and Django are installed *globally* on a computer meaning it is quite a pain to install and reinstall different versions every time you want to switch between projects.

Fortunately, there is a straightforward solution. *Virtual environments* allow you to create and manage separate environments for each Python project on the same computer. There are many areas of software development that are hotly debated, but using virtual environments for Python development is not one. **You should use a dedicated virtual environment for each new Python project**.

There are several ways to implement virtual environments but the simplest is with the venv module already installed as part of the Python 3 standard library. To try it out, navigate to the existing `ch1-setup` directory on your Desktop.

```
# Windows
> cd onedrive\desktop\code\ch1-setup

# macOS
% cd ~/desktop/code/ch1-setup
```

To create a virtual environment within this new directory use the format `python -m venv <name_of_env>` on Windows or `python3 -m venv <name_of_env>` on macOS. It is up to the developer to choose a proper environment name but a common choice is to call it `.venv`.

```
# Windows
> python -m venv .venv
> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

# macOS
% python3 -m venv .venv
```

If you use the command `ls` to look at our current directory it will appear empty. However the `.venv` directory *is* there, it's just that it is "hidden" due to the period `.` that precedes the name. Hidden files and directories are a way for developers to indicate that the contents are important and should be treated differently than regular files. To view it, try `ls -la` which shows all directories and files, even hidden ones.

```
> ls -la
total 0
drwxr-xr-x  3 wsv  staff   96 Dec  12 11:10 .
drwxr-xr-x  3 wsv  staff   96 Dec  12 11:10 ..
drwxr-xr-x  6 wsv  staff  192 Dec  12 11:10 .venv
```

You will see that `.venv` is there and can be accessed via `cd` if desired. In the directory itself is a copy of the Python interpreter and a few management scripts, but you will not need to use it directly in this book.

Once created, a virtual environment must be *activated*. On Windows an *Execution Policy* must be set to enable running scripts. This is a safety precaution. The Python docs recommend allowing scripts for the `CurrentUser` only, which is what we will do. On macOS there are no similar restrictions on scripts so it is possible to directly run `source .venv/bin/activate`.

Here is what the full commands look like to create and activate a new virtual environment called `.venv`:

```
# Windows
> python -m venv .venv
> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
> .venv\Scripts\Activate.ps1
(.venv) >

# macOS
% python3 -m venv .venv
% source .venv/bin/activate
(.venv) %
```

The shell prompt now has the environment name `(.venv)` prefixed which indicates that the virtual environment is active. Any Python packages installed or updated within this location will be confined to the active virtual environment.

To deactivate and leave a virtual environment type `deactivate`.

```
# Windows
(.venv) > deactivate
>

# macOS
(.venv) % deactivate
%
```

The shell prompt no longer has the virtual environment name prefixed which means the session is now back to normal.

Install Django

Now that Python is installed and we know how to use virtual environments it is time to install Django for the first time. In the `ch1-setup` directory reactivate the existing virtual environment.

```
# Windows
> .venv\Scripts\Activate.ps1
(.venv) >

# macOS
% source .venv/bin/activate
(.venv) %
```

Django is hosted on the Python Package Index (PyPI), a central repository for most Python packages. We will use `pip`, the most popular package installer, which comes included with Python 3. To install the latest version of Django use the command `python -m pip install django~=4.0.0`.

The comparison operator `~=` ensures that subsequent security updates for Django, such as 4.0.1, 4.0.2, and so on are automatically installed. Note that while it *is* possible to use the shorter version of `pip install <package>`, it is a best practice to use the longer but more explicit form of `python -m pip install <package>` to ensure that the correct version of Python is used. This can be an issue if you have multiple versions of Python installed on your computer.

```
(.venv) > python -m pip install django~=4.0.0
```

You might see a `WARNING` message about updating `pip` after running these commands. It's always good to be on the latest version of software and to remove the annoying `WARNING` message each time you use `pip`. You can either copy and paste the recommended command or run `python -m pip install --upgrade pip` to be on the latest version.

```
(.venv) > python -m pip install --upgrade pip
```

First Django Project

A Django project can have almost any name but we will use `django_project` in this book. To create a new Django project use the command `django-admin startproject django_project .`

```
(.venv) > django-admin startproject django_project .
```

It's worth pausing here to explain why you should add a period (`.`) to the end of the previous command. If you just run `django-admin startproject django_project` then by default Django will create this directory structure:

```
django_project/
    ├── django_project
    │   ├── __init__.py
    │   ├── asgi.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    ├── manage.py
    └── .venv/
```

Do you see the multiple `django_project` directories? First a top-level `django_project` directory is created and then *another* one within it that contains the files we need for our Django project. This feels redundant to me which is why I prefer adding a period to the end which installs Django in the current directory.

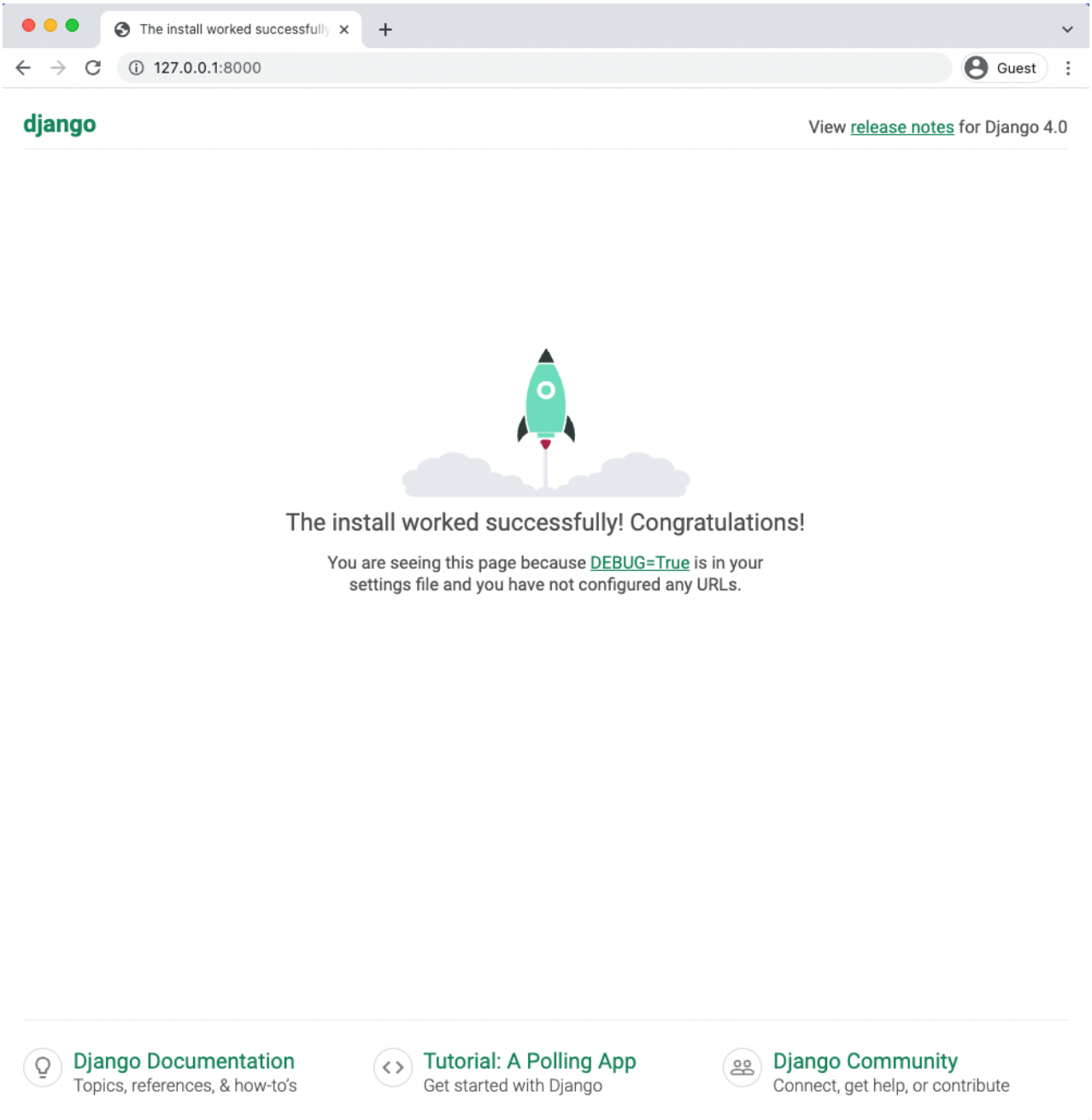
```
├── django_project
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py
└── .venv/
```

As you progress in your journey learning Django, you'll start to bump up more and more into similar situations where there are different opinions within the Django community on the correct best practice. Django is eminently customizable, which is a great strength, however the tradeoff is that this flexibility comes at the cost of seeming complexity. Generally speaking, it's a good idea to research any such issues that arise, make a decision, and then stick with it!

Now let's confirm everything is working by running Django's internal web server via the `runserver` command. This is suitable for local development purposes, but when it comes time to deploy our project's online we will switch to a more robust WSGI server like *Gunicorn*.

```
# Windows
(.venv) > python manage.py runserver

# macOS
(.venv) % python3 manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until
you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

December 12, 2021 - 15:26:23
Django version 4.0.0, using settings 'django_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-BREAK.
```

Don't worry about the text in red about `18 unapplied migrations.` We'll get to that shortly. The important part, for now, is to visit `http://127.0.0.1:8000/` in your web browser and make sure the following image is visible:



If you are on Windows you'll see the final line says to use `CONTROL-BREAK` to quit whereas on macOS it is `CONTROL-C`. Newer Windows keyboards often do not have a Pause/Break key in which case using the `C` key usually works.

Go ahead and stop the local server with the appropriate command and then exit the virtual environment by typing `deactivate` and hitting `Return`.

```
# Windows or macOS
(.venv) > deactivate
```

We'll get lots of practice with virtual environments in this book so don't worry if it's a little confusing right now. The basic pattern for any new Django project is to first make and activate a virtual environment, install Django, and then run `startproject`.

It's worth noting that only one virtual environment can be active in a command line tab at a time. In future chapters we will be creating a brand new virtual environment for each new project so either make sure to `deactivate` your current environment or open up a new tab for new projects.

Text Editors

The command line is where we execute commands for our programs but a text editor is where actual code is written. The computer doesn't care what text editor you use–the end result is just code–but a good text editor can provide helpful hints and catch typos for you.

There are many modern text editors available but a very popular one is Visual Studio Code, which is free, easy to install, and enjoys widespread popularity. If you're not already using a text editor, download and install VSCode from the official website.

An optional–but highly recommended–additional step is to take advance of the large ecosystem of extensions available on VSCode. On Windows, navigate to `File -> Preferences -> Extensions` or on macOS `Code -> Preferences -> Extensions`. This launches a search bar for the extensions marketplace. Enter "python" which will bring up the Microsoft extension as the first result. Install it.

A second extension to add is Black, which is a Python code formatter that has quickly become the default within the Python community. To install Black, open a Terminal window within VSCode by going to `Terminal -> New Terminal` at the top of the page. In the new terminal window opened at the bottom of the page, type `python -m pip install black`. Next, open up the VSCode settings by navigating to `File -> Preferences -> Settings` on Windows or `Code -> Preferences -> Settings` on macOS. Search for "python formatting provider" and select `black` from the dropdown options. Then search for "format on save" and enable "Editor: Format on Save". Black will now automatically format your code whenever a `*.py` file is saved.

To confirm this is working, use your text editor to create a new file called `hello.py` within the `ch1-setup` directory. located on your Desktop and type in the following using single quotes:

```
print('Hello, World!')
```

On save, it should be automatically updated to using double quotes which is Black's default preference: `print("Hello, World!")`. That means everything is working properly.

Install Git

The final step is to install *Git*, a version control system that is indispensable to modern software development. With Git you can collaborate with other developers, track all your work via commits, and revert to any previous version of your code even if you accidentally delete something important!

On Windows, navigate to the official website at `https://git-scm.com/` and click on the "Download" link which should install the proper version for your computer. Save the file and then open your Downloads folder and double click on the file. This will launch the Git for Windows installer. Click the "Next" button through most of the early defaults as they are fine and can always be updated later as needed. There are two exceptions however: under "Choosing the default editor used by Git" select VS Code not Vim. And in the section on "Adjusting the name of the initial branch in new repositories" select the option to use "main" as opposed to "master" as the default branch name. Otherwise the recommended defaults are fine and can always be tweaked later if needed.

To confirm Git is installed on Windows, close all current shell windows and then open a new one which will load the changes to our PATH variable. Type in `git --version` which should show it is installed.

```
# Windows
> git --version
git version 2.33.1.windows.1
```

On macOS, installing Git via Xcode is currently the easiest option. To check if Git is already installed on your computer, type `git --version` in a new terminal window.

```
# macOS
% git --version
```

If you do not have Git installed, a popup message will ask if you want to install it as part of "command line developer tools." Select "Install" which will load Xcode and its command line tools package. Or if you do not see the message for some reason, type `xcode-select --install` instead to install Xcode directly.

Be aware that Xcode is a very large package so the initial download may take some time. Xcode is primarily designed for building iOS apps but also includes many developer features need on macOS. Once the download is complete close all existing terminal shells, open a new window, and type in `git --version` to confirm the install worked.

```
# macOS
% git --version
git version 2.30.1 (Apple Git-130)
```

Once Git is installed on your machine we need to do a one-time *system* configuration by declaring the name and email address associated with all your Git commits. We will also set the default branch name to `main`. Within the command line shell type the following two lines. Make sure to update them your name and email address.

```
> git config --global user.name "Your Name"
> git config --global user.email "yourname@email.com"
> git config --global init.defaultBranch main
```

You can always change these configs later if you desire by retyping the same commands with a new name or email address.

Conclusion

Phew! It is no easy task to configure a software development environment. But fortunately it is a one-time pain and will pay many dividends done the road. We have now learned about the command line, Python interactive mode, installed the latest version of Python, Django and Git, and configured our text editor. Everything is ready for our first Django app in the next chapter.

Continue on to Chapter 2: Hello World app.