
Friendly 101 Documentation

Release 0.1

Friendly Django

July 22, 2015

1	Prerequisites	3
2	Compatibility	5
3	Contents	7
3.1	Introduction	7
3.2	Text editor	8
3.3	Command line	11
3.4	Command line tools	16
3.5	Bash profile	18
3.6	Homebrew	22
3.7	pip & virtualenv	26
3.8	Django	29

Friendly 101 is a tutorial and hands-on workshop by the [Friendly Django](#) Meetup group. It acts as an introduction to the [Django](#) web framework for beginners.

This documentation is saved in a [GitHub repository](#). If you see any errors in this text, please [file an issue](#). The project that results from this tutorial is included in the repository and currently [deployed to Heroku](#).

- <https://github.com/friendlydjango/friendly-101>
- <https://friendly-101.readthedocs.org/>
- <https://friendly-101.herokuapp.com/>

Prerequisites

- A Mac
- OS X 10.9 Mavericks or greater

Compatibility

The guide is written for the following versions of software:

- Python 2.7.9
- SQLite 3.8.7.4
- Django 1.7

3.1 Introduction

If you like photos on [Instagram](#), you're using Django.

If you pin ideas on [Pinterest](#), you're using Django.

If you read [National Geographic](#), you're using Django.

With Django, we can [read the news](#), [write comments](#), [listen to music](#), [download web browsers](#), [laugh at satire](#), [buy swag](#), [play guitar](#), [talk to our neighbors](#), [park the car](#), [follow hipsters](#), [learn about the universe](#) and a whole lot more.

Django has allowed for the creation of fun, interesting, and useful websites and web applications since its developers released it [10 years ago](#). So, what is it?

3.1.1 What is Django?

According to [its website](#):

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

At its core, Django is a [framework](#), which is a collection of best practices, conventions, shortcuts, and helpers for creating websites faster and of higher quality. Frameworks strike a middle ground between content management systems, which although convenient can make fundamental assumptions about the structure of your data, and going alone with custom code that is (hopefully) well suited but time consuming to create.

Django was [created](#) by [Adrian Holovaty](#) and [Simon Willison](#) while working at the online division of the [Lawrence Journal-World](#) newspaper in [Lawrence, Kansas](#), between 2003 and 2005. After becoming frustrated by developing their websites in the [PHP](#) programming language, Holovaty and Willison decided instead to use the [Python](#) programming language, eventually creating a reusable, generic framework in the face of the stringent deadlines of a newsroom, earning it the tagline “the web framework for perfectionists with deadlines.”

In 2005, Django was [released](#) as open-source software and quickly gained popularity. [DjangoCon](#), the first Django conference, was in 2008, and today the [Django Software Foundation](#) oversees all development effort and promotion of the framework.

Django adopts an [MVC](#) software pattern, which means it separates the data layer from the logic layer from the presentation layer. One of Django's ambitious principles is “[batteries included](#),” which means that all of its separate parts work together without relying on external software dependencies other than Python itself.

There's a lot more in Django's [FAQ](#) and in the [design philosophies](#). Other great historical references are Holovaty's [Snakes and Rubies](#) presentation, and Willison's [Django history answer](#) on Quora.

3.1.2 Who are you again?

Hello, my name is Rich Cornish. I first used Django in the fall of 2006 and quickly fell in love working with it. I then worked as an interaction designer at the same online division at the Lawrence Journal-World from 2007 to 2009.

Today I work as a user experience designer and feel privileged that I can share the lessons I learned about web development from the outside perspective of a designer. I graduated from the [University of Illinois at Urbana-Champaign](#) with a [master of science degree in journalism](#).

3.1.3 What is Friendly 101?

This mission of Friendly 101 is to be a guide for absolute beginners wanting to learn Django. Other texts have similar missions for other technologies, but this text is different because I endeavor to meet the following standards:

- Easy to read and understand

All text and graphics are written and presented in a way that is neither too technical nor too childish. The material assumes some use of a Mac computer, but nothing more is required. Organization, grammar, consistency, and style are critical.

- Review related technologies

Too many instructional texts silo themselves, burdening the reader to figure out the technology landscape. There is great value in understanding what's required, what's optional, what's related, and what are the alternatives.

- Link honorably

While reviewing related technologies is useful, the text is not an all-encompassing guide to all of them. After an overview and explanation of basic concepts, hyperlinks should be used often to point the reader to a trustworthy source to learn more.

It is my belief that people learn best when they create interesting, useful things as “quick wins,” and can then “work backward” to progressively learn more about how Python works in and outside the context of Django. When people feel more comfortable in Django, there are plenty of other [resources](#), [frameworks](#), and [libraries](#) for people to continue their journey.

3.2 Text editor

The first step in a web developer's journey is to use a good text editor.

3.2.1 What is a text editor?

A **text editor** is an application that allows you to write code, which is simply text without formatting options like bold or alignment. [Microsoft Word](#) and [TextEdit](#) are word processors, which are similar to text editors, but often come with features that make them inappropriate for writing code.

Like all good things in life, people have a wide range of opinions on which text editor is “the best.” And like all good things, finding “the best” text editor isn't important—what's important is finding the best text editor *for you*, dear beginner.

3.2.2 Sublime Text is your new friend

As a beginning developer, you should use a text editor that is fairly easy to understand and use, is in active development, offers add-ons when you become advanced, and hopefully doesn't cost much. For those reasons, I recommend **Sublime Text**.

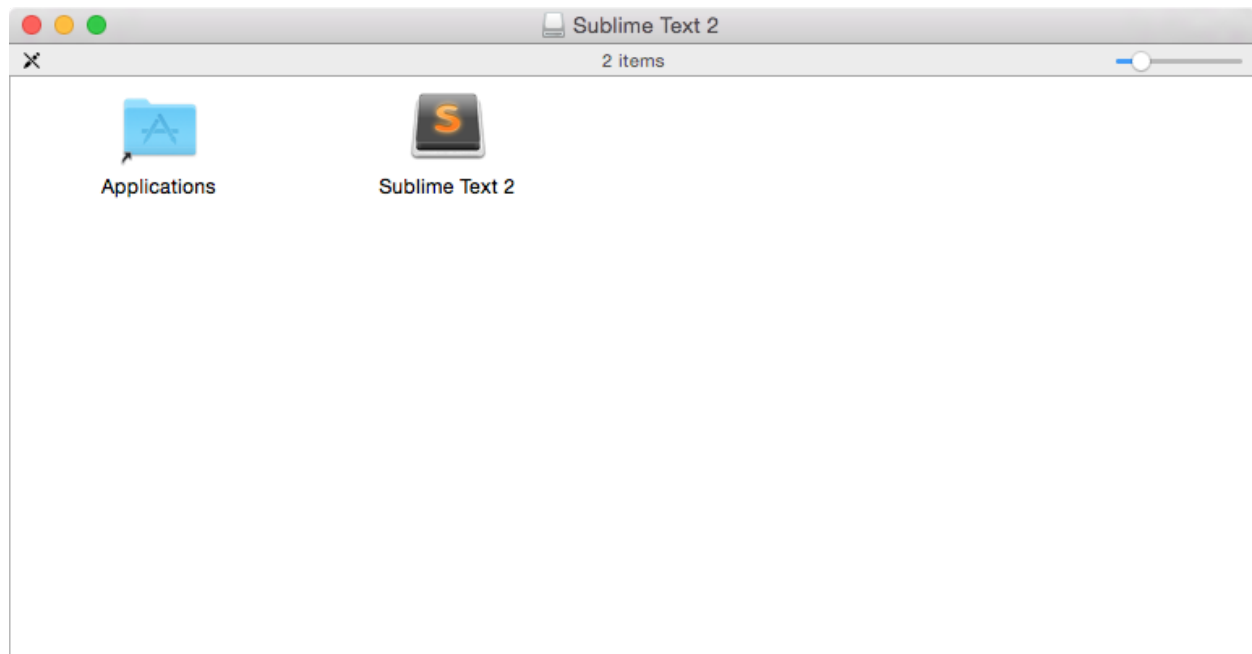
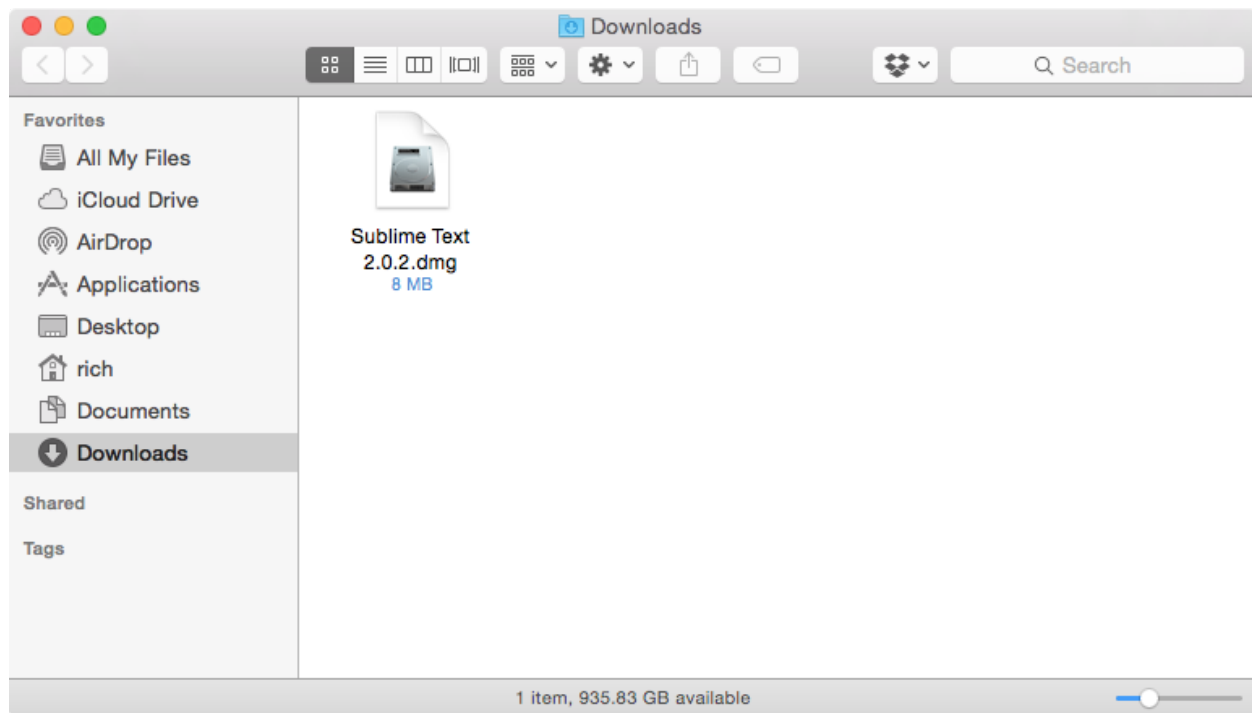
Sublime Text is a popular, open-source text editor that beginners and professionals alike use. Hundreds of [add-on packages](#), [themes](#), and [tutorials](#) exist for it. It's also free with minimal intrusion. Download Sublime Text from its website by clicking on the "Download for OS X" button.

The screenshot shows the Sublime Text website. At the top is a navigation bar with links: Home, Download, Buy, Blog, Forum, and Support. Below this is the title "Sublime Text" in a large, bold font. Underneath the title is a short description: "Sublime Text is a sophisticated text editor for code, markup and prose. You'll love the slick user interface, extraordinary features and amazing performance." Below this is a large image of the Sublime Text editor interface. The editor window shows a C++ file named "base64.cc" with code for a base64 encoding function. The status bar at the bottom of the editor window indicates "Line 31, Column 55", "Spaces: 4", and "C++". Below the editor window, there is a blue button labeled "Download for OS X" and the text "Version 2.0.2".

3.2.3 Installing Sublime Text

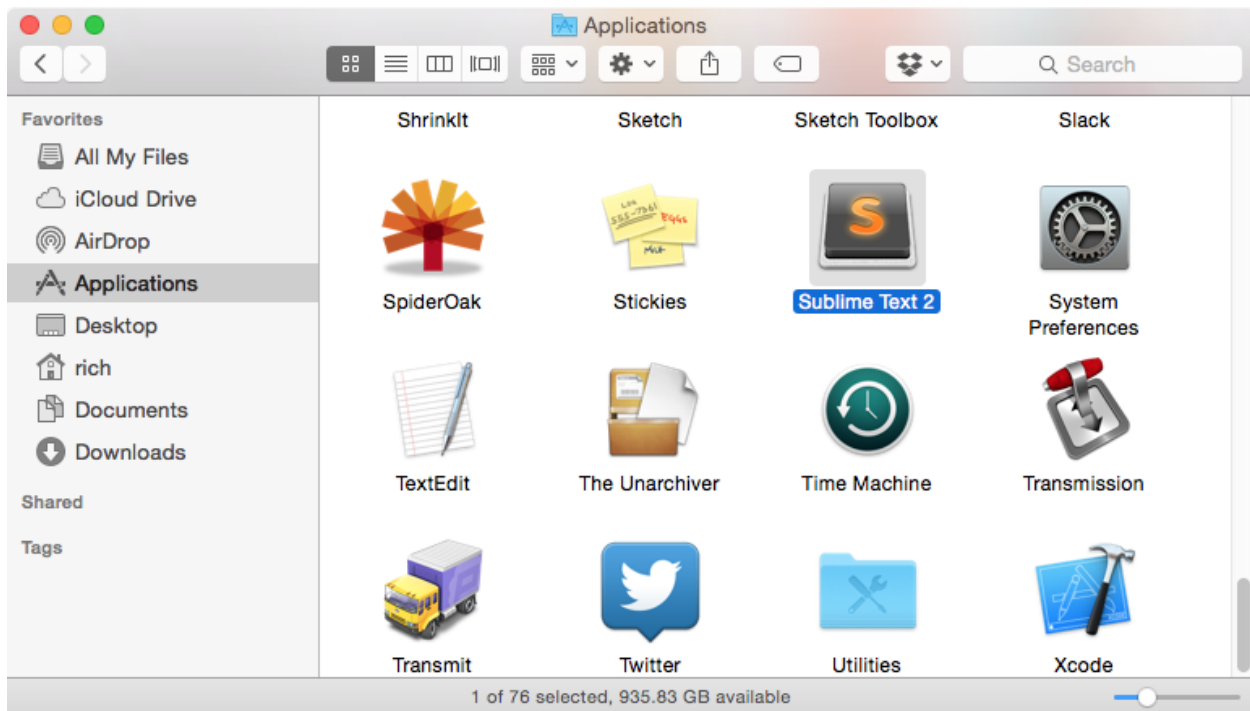
Find the DMG you downloaded, which is probably in your "Downloads" folder and was called "Sublime Text 2.0.2.dmg" at the time of this writing.

"DMG" stands for [Disk Image](#), and is the file format for OS X applications. Open the DMG by double clicking on it. The DMG will mount the Finder and a new window will appear. Drag the icon labeled "Sublime Text 2" into the shortcut folder labeled "Applications."



You can now close the windows, eject the mount by dragging it from the Desktop to the Trash, and trash the DMG.

Open a new Finder window and navigate to the “Applications” folder. Find the “Sublime Text 2” application icon and open it by double clicking on it.



Congratulations! You are now using a world-class text editor.

Sublime Text is free to use, but it pops up an occasional window reminding you to purchase it. You can click `Cancel` or press `Esc` when it appears, but feel free to eventually support the developers who work hard to make software free to use.

I recommend preserving the presence of the Sublime Text 2 icon in the Dock for easy access in the future. `CONTROL`-click on the Sublime Text 2 icon in your dock, click “Options,” and then “Keep in Dock.” You can also preserve the presence of Sublime Text 2 by simply dragging the icon into another place on the Dock.

It’s always good to remember that what’s fashionable today might not be fashionable tomorrow. A few years ago, `TextMate` was the text editor of choice until it languished in development. `Coda` has a small but dedicated audience. Veterans use `Vim` and `Emacs`, and newcomers include `Brackets` and `Atom`. You might even get a sheepish confession from someone using `nano`.

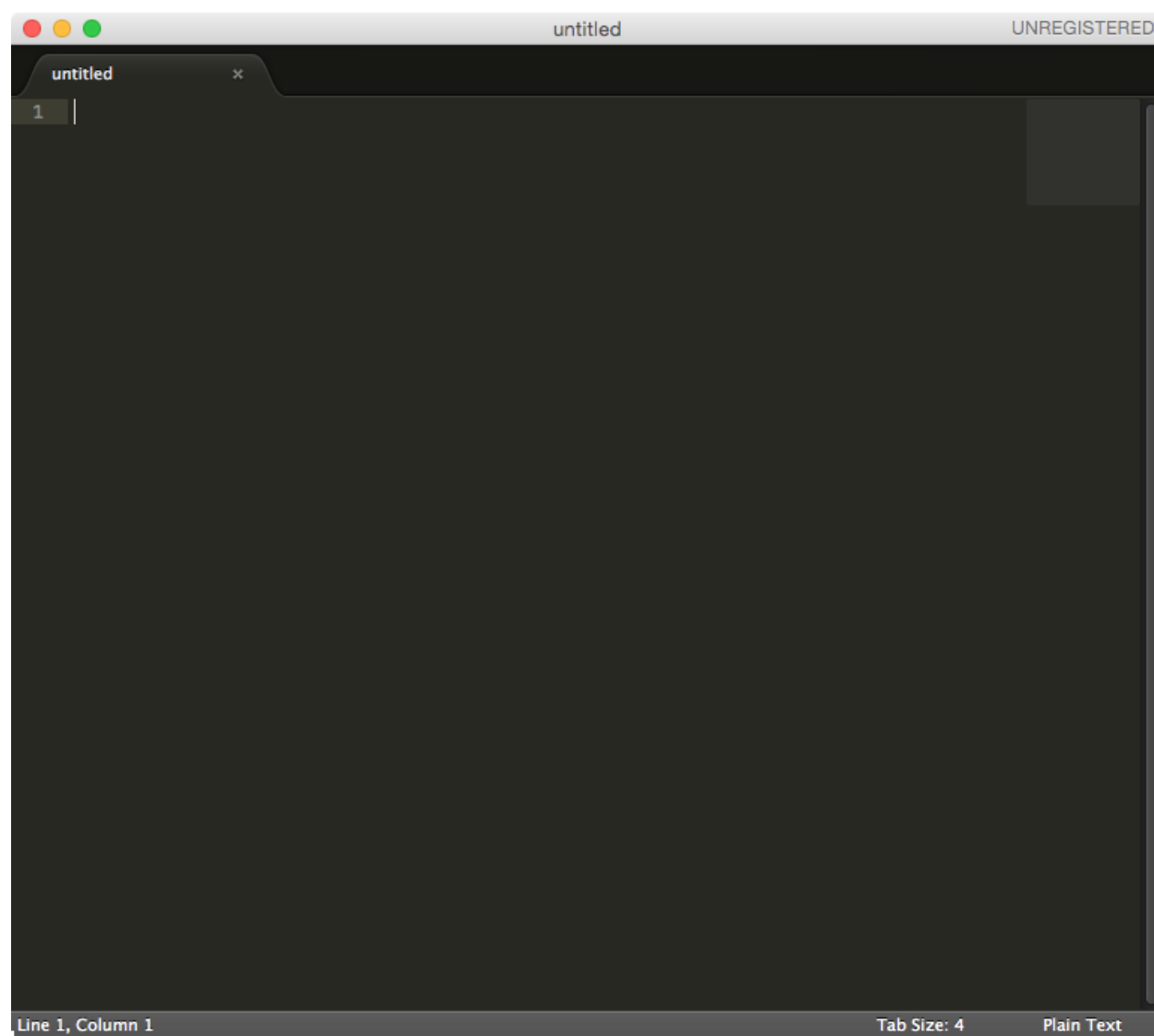
Developers like to brag about their text editor of choice, but always remember: What you write in your editor is far more important than your editor of choice.

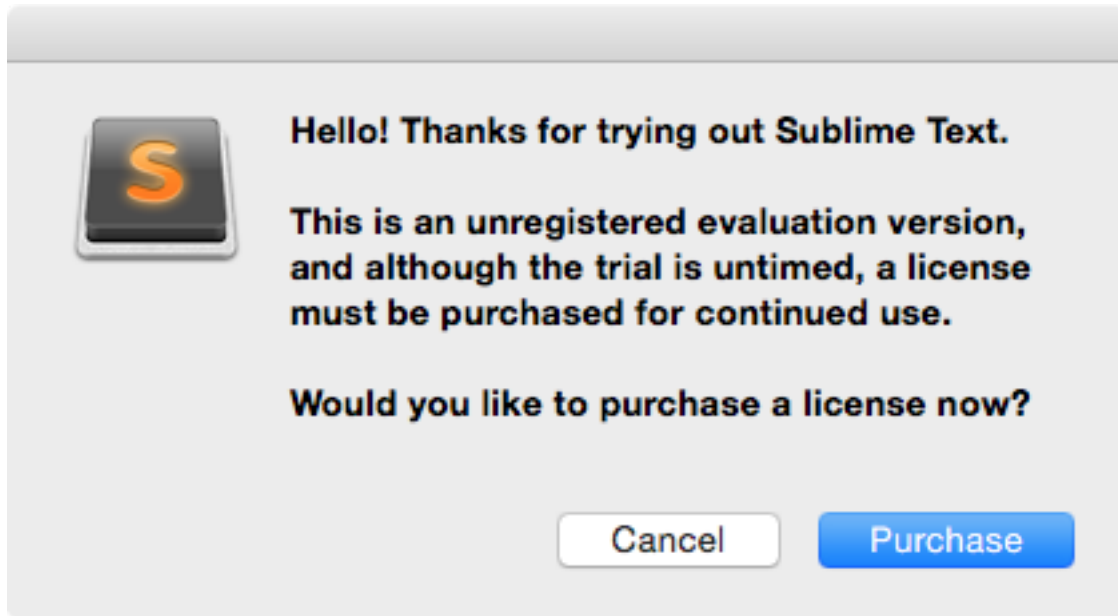
3.3 Command line

After a good text editor, the next tool a developer needs to understand is the `command-line interface`.

3.3.1 You won’t break your computer

A **command-line interface** is a program that allows developers to interact with a computer by running commands that are typed. The functionality shared by a command-line interface and the more familiar graphical user interface,





known in Apple’s OS X as the [Finder](#), are similar in many ways and identical in others.

Beginner developers often have a fear of the command line, thinking they will “break their computer” if they even open the application. Others think it’s an old, archaic tool that can’t possibly be useful to write modern software. Both ideas can be right and they can be wrong. As you’ll discover, command-line interfaces are merely ways of interacting with computers just as graphical user interfaces do. Understanding the strengths and weaknesses of one’s tools is one of the hallmarks of a good developer.

You won’t break your computer anymore than you would clicking around with a mouse.

3.3.2 You can be bashful now

The command-line interface we’ll work with is **Bash**. [Bash](#) is a shell, which is a category of programs that load command-line interfaces, much like [Chrome](#) is a web browser. The shell’s name comes from its function as a “wrapper” around the kernel, the core program that communicates with the hardware. [Bash](#) is the [most popular shell](#) today and the default shell inside the [Terminal](#) application on OS X. It’s worthy to note that shells other than Bash can be loaded inside of Terminal—a feat that web browsers don’t and will likely never do.

Open a Finder window, navigate to the “Applications” folder, then the “Utilities” folder, and open the “Terminal” application. It might look something like this, which shows a whole lot of white.

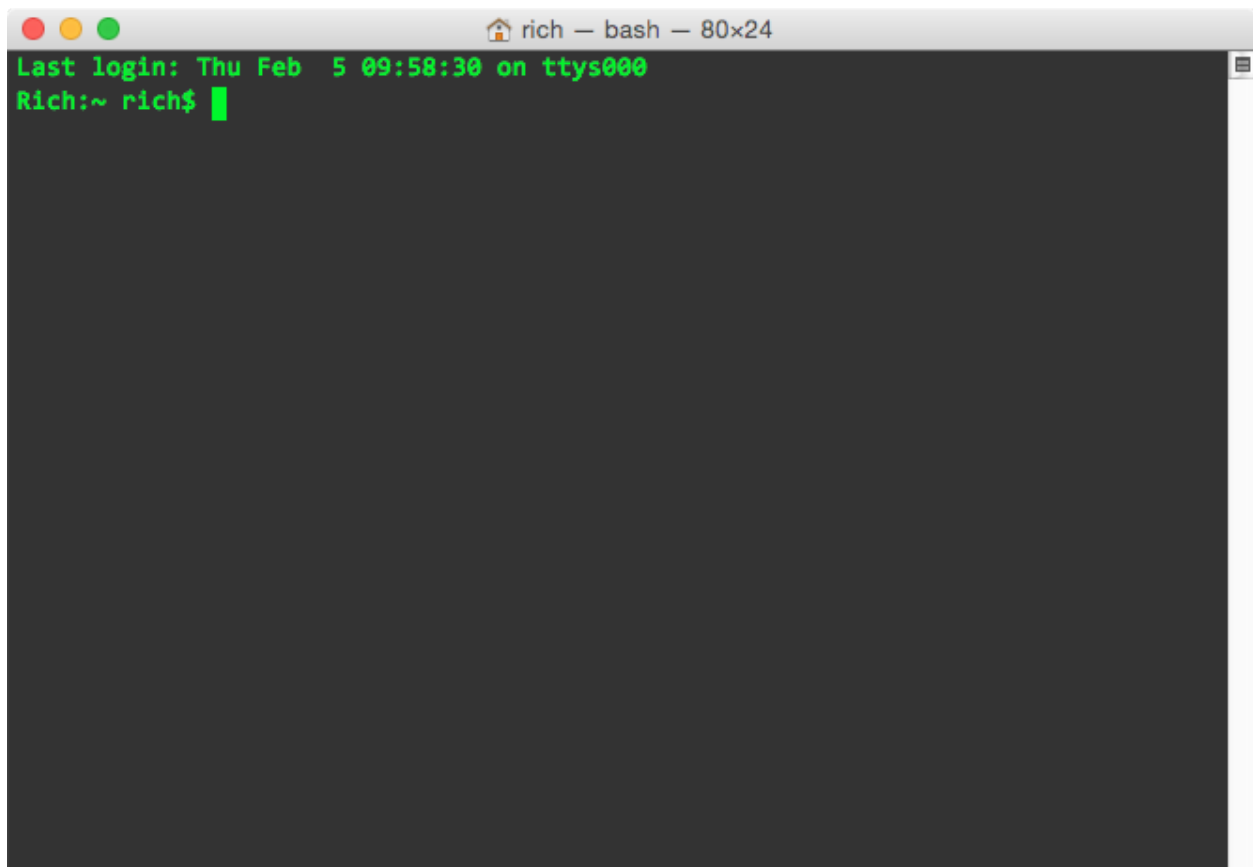
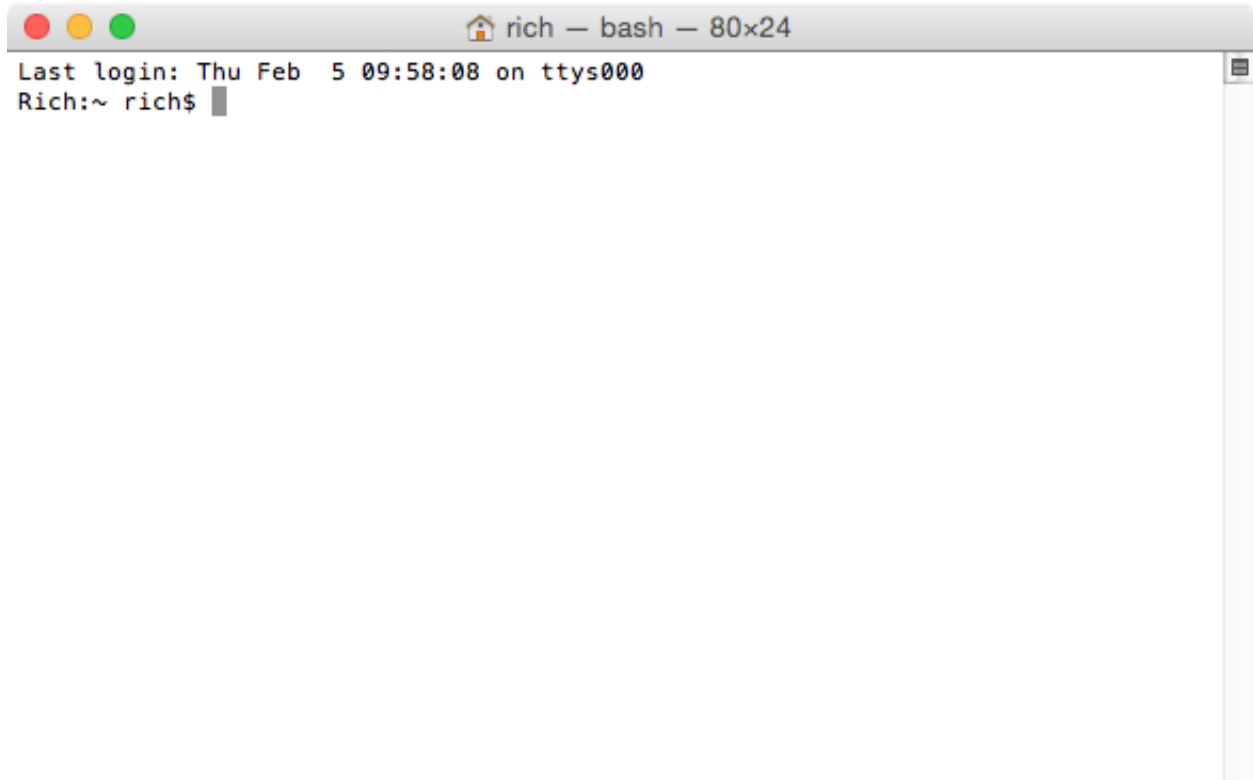
The first line tells me when I last started a Bash session (`Last login: Mon Feb 5 09:58:08`) and on which [Teletypes](#) (`ttys000`), although today the equivalent is simply a tab in Terminal, much like a tab in a web browser. The second line tells me the name of my computer on the network (`Rich`), my current directory (`~`, which is my home directory), the name of my user account (`rich`), and the symbol indicating a prompt for user input (`$`).

Feel free to trick it out in the preferences. I like to make mine look like [The Matrix](#).

I also recommend preserving the presence of the Terminal icon [in the Dock](#) for easy access in the future.

3.3.3 Common Bash commands

I provide all commands you should run when you need them, but it helps to remember the ones most commonly used.



Each command below is preceded by a comment, which is a line beginning with a hash (#) and explains the surrounding code. Note that the prompt, represented by the dollar sign (\$), is meant only to tell you that the following characters composed a Bash command and is not meant to be literally included as part of the command. The inclusion of the \$ indicating a Bash command is a common convention in the developer community.

Commands for directory changes (<directory> represents the name of a directory):

```
# Display (or print) the path of the current (or working) directory
$ pwd

# List the contents of a directory
$ ls <directory>

# Change the current directory
$ cd <directory>

# Make a directory
$ mkdir <directory>

# Open a directory in Finder
$ open <directory>

# Create an empty file or, if it exists, update last edit to now
$ touch <file>
```

Symbols for directory traversal:

```
# The parent directory
../

# The parent's parent directory
../../

# The root directory
/

# The current directory
.
```

These commands and others can be combined in interesting ways that would be difficult to replicate in a graphical user interface.

```
# List the items in the parent directory
$ ls ../

# Change to the parent's parent directory
$ cd ../../

# Make a directory in the root directory
$ mkdir /<directory>

# Open the current directory in Finder
$ open .
```

Note: You can type the first few letters of a file or directory and then press `tab` to cycle through or autocomplete possible items in the current directory.

Additional commands include:

```
# Move or rename
$ mv <directory or file> <destination>

# Copy
$ cp <directory or file> <destination>

# Delete (remove) a file (or directory with -r flag)
$ rm <file>

# Closes the Bash session
$ exit
```

If you feel like you need additional guidance, [The Command Line Crash Course](#) by Zed Shaw is excellent, and [SS64](#) lists all Bash commands.

3.3.4 But who are any of us, really?

Let's run our first command. Copy and paste the following and press `return`. Take care not to copy the `$` and adjacent space.

```
$ whoami
```

You should've gotten a response with the name of your user account.

```
$ whoami
Rich
```

Note: After entering your first command, you can hit the up arrow `↑` or down arrow `↓` key to cycle through previous commands.

3.3.5 Terminal, meet Sublime

Let's run a powerful command, a helpful trick that will connect Terminal to Sublime Text 2. Note that you might need to enter your OS X password, and again take care not to copy and paste the `$` and adjacent space.

```
$ sudo ln -s "/Applications/Sublime Text 2.app/Contents/SharedSupport/bin/subl" /usr/local/bin/subl
```

This command and this command alone will run as a superuser ("superuser do" or `sudo`) and create a link (`ln`) that is (`symbolic`). A [symbolic link](#) is the equivalent of an alias or shortcut in the Finder. From now on, when we type `subl` and the name of a directory or file in Terminal, that same directory or file will pop up in Sublime Text 2, ready for us to edit!

Just like text editors, people have varying opinions on shells. Bash is the most popular, but some developers claim [Z shell](#) can be more productive. Terminal comes with OS X, but some swear by [iTerm2](#). Get comfortable with what's most accessible and when you feel confident, explore what else is out there.

3.4 Command line tools

Now that you're a little familiar with the command-line interface, we're going to install some additional tools to compile any software we might need.

3.4.1 What is compiling?

If you’ve downloaded software before, you’re probably familiar with the song and dance by now.

1. Go to the website
2. Find the download link
3. Click the download link
4. Find the DMG
5. Open it
6. Open the mount
7. Copy the application into the Applications folder
8. Close the window
9. Find the application
10. Open the application
11. Unmount the DMG
12. Trash the DMG.

Whew.

The DMG you downloaded is known as a **binary**, which was *compiled* from source code that other developers wrote, likely from many files into a single neat, tidy file ready for hungry downloaders. Not all software is compiled—especially development software that you might need—and because of that, I recommend learning the basics of compiling your own software. Fortunately, the ability to compile your own software has come a long way and is a lot easier than it used to be.

3.4.2 Installing the tools

Compiling on OS X requires the [GNU Compiler Collection](#), or GCC, and is included in the **Command Line Tools** software by Apple. You can download Command Line Tools from the [Apple Developer website](#), but it is often more convenient to simply download and install it from the command line.

Open Terminal, copy and paste the following, and press return. Review the [Command line](#) lesson if necessary.

```
$ xcode-select --install
```

A Finder window should pop up. Click “Install” and agree to the license agreement of Command Line Tools. Once installed, confirm the installation of Command Line Tools:

```
$ xcode-select --print-path
/Library/Developer/CommandLineTools
```

If instead your path looked more like `/Applications/Xcode.app/Contents/Developer`, then you previously installed Xcode. Keep reading to understand what you need to do.

And now confirm GCC was installed along with Command Line Tools:

```
$ gcc --version
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/usr/include
Apple LLVM version 6.0 (clang-600.0.54) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin14.0.0
Thread model: posix
```

3.4.3 Do I need Xcode?

Previously Apple bundled Command Line Tools with [Xcode](#), a full suite of software development tools for developing native applications on OS X and iOS. The bundling forced developers to download the entire suite, which was about 2.5 GB in size.

Since the OS X release of Mavericks, developers can download Command Line Tools separately, which means developers *do not need* Xcode. However, *if you downloaded Xcode before*, then you will need to update Xcode either in the App Store application or with Software Update.

If you verified the installation of Command Line Tools with Xcode installed, you will instead receive confirmation that Xcode was installed, which is perfectly fine.

```
$ xcode-select --print-path
/Applications/Xcode.app/Contents/Developer
```

```
$ gcc --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-included-dir=/usr/
Apple LLVM version 6.0 (clang-600.0.54) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin14.0.0
Thread model: posix
```

3.5 Bash profile

You’ve probably noticed that when you turn on your computer that some applications start up with it. Some of them are required by the operating system and some you might have installed yourself. Some can make your computer’s fan sound like an airplane taking off.

You can see some of these in the “Login Items” tab of the “Users & Groups” section of your System Preferences.

Other applications have their own login items, which run only when that specific application is opened. We’re going to use Bash’s login items feature to make web development easier.

3.5.1 Why do I need a Bash profile?

The **Bash profile** is a file on your computer that Bash runs every time a new Bash session is created. This is useful because we need to run certain code every time before starting to work.

OS X doesn’t include a Bash profile by default, but if you already have one, it lives in your home directory with the name `.bash_profile`. And if you did have one, you probably never saw it because its name starts with a period. The Finder hides folders and files starting with a period to protect casual users from harming the operating system—but because you’re a developer now, we’re going to create a Bash profile!

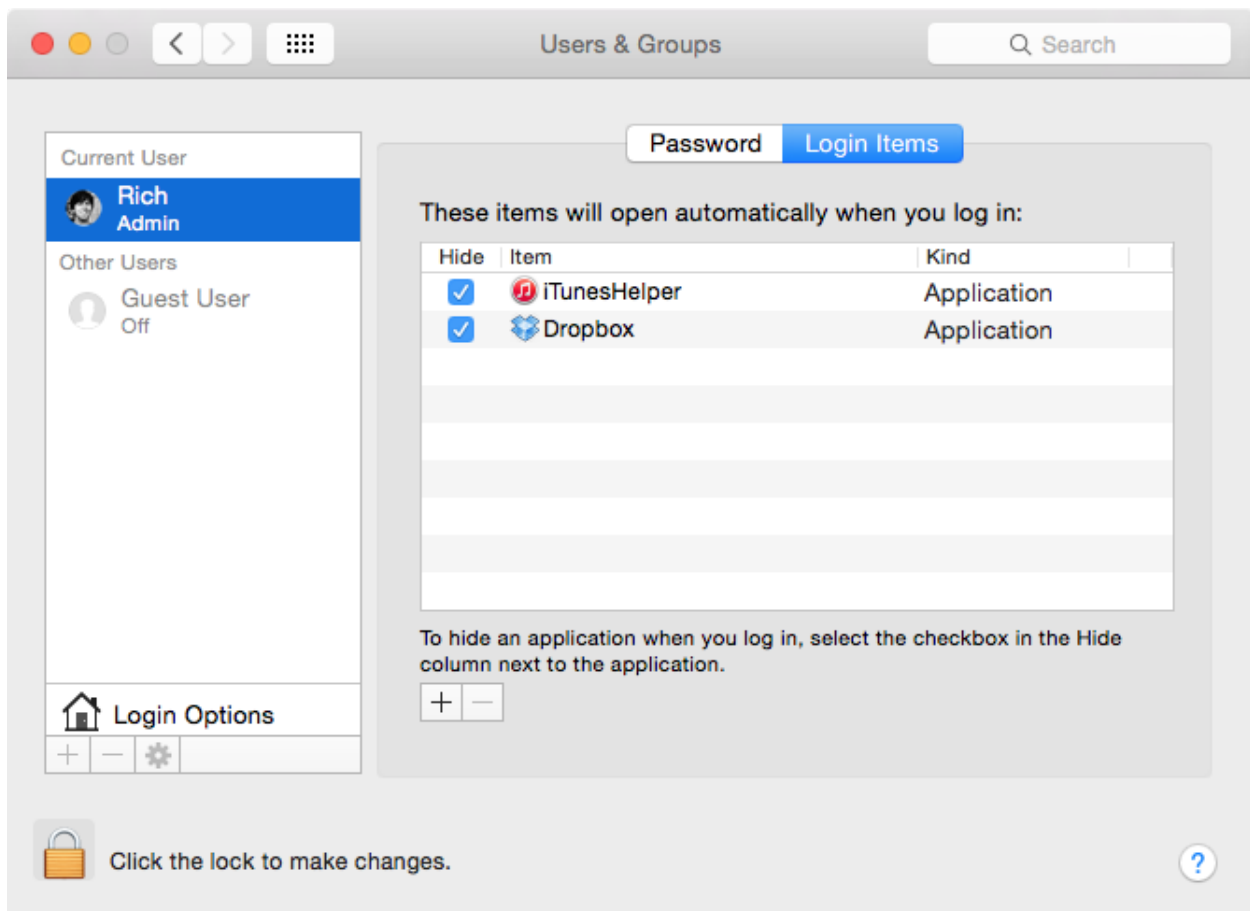
3.5.2 Creating your Bash profile

Open Terminal, copy and paste the following command, and press `return`.

```
$ subl ~/.bash_profile
```

If you received an error, review [Terminal, meet Sublime](#) of the [Command line](#) lesson.

The **tilda** (`~`) tells Bash to start traversing the file system from your home directory. In my personal case, an equivalent command would have been `subl /Users/Rich/.bash_profile`, which means `~` is the equivalent of `/Users/Rich`. Using `~` is a shortcut that makes it generic and usable for everybody to understand and use.



Your Bash profile will pop up in a Sublime Text window.

3.5.3 Understanding your PATH

The first edit to your Bash profile is to correct your `PATH`. `PATH` is an **environment variable**, which simply means that it represents some small bit of data while you use Terminal. Specifically, `PATH` contains a list of file system paths where the operating system can find programs to run.

When a developer runs a program in Bash, the operating system will sequentially look for the program in each of the paths that `PATH` contains, starting with the first path listed. If the operating system can't find the program in the first path, it looks for the same program in the second path, and so on, until either eventually finding and running the program or returning an error if the program couldn't be found.

`PATH` contains paths that are delimited by a colon (:). Therefore, the value of `PATH` might look something like:

```
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

You can see that `/usr/local/bin` is the first path, and `/usr/bin` is the second path. `/usr/local/bin` is where all programs *local* to your use of the operating system are located. Storing programs for your personal use in `/usr/local/bin` is a best practice and highly encouraged. Therefore, the `PATH` above is correct.

Likewise, storing programs in `/usr/bin` allows programs to be globally accessible by other users. Storing programs globally can sometimes be desirable, but in general it's discouraged and likely to cause confusion.

3.5.4 Correcting your PATH

In versions of OS X prior to Yosemite, Apple mistakenly switched the order of the paths, placing `/usr/bin` ahead of `/usr/local/bin`, causing much disruption and angst. Apple has since corrected the issue, but it's still worth changing because doing so won't harm the computer and will prevent problems from occurring again.

Copy and paste the following into your Bash profile.

```
# Paths
export PATH=/usr/local/bin:$PATH
```

The first line is a comment, which begins with a hash (#) and explains surrounding code. The right portion of the second line begins `/usr/local/bin`, which is the path we want to prioritize, followed by `:`, which joins paths, and finally `$PATH`, which evaluates the value of the existing `PATH`. By appending `$PATH` we can overwrite the original `PATH` without destroying its value, making everything nice and tidy!

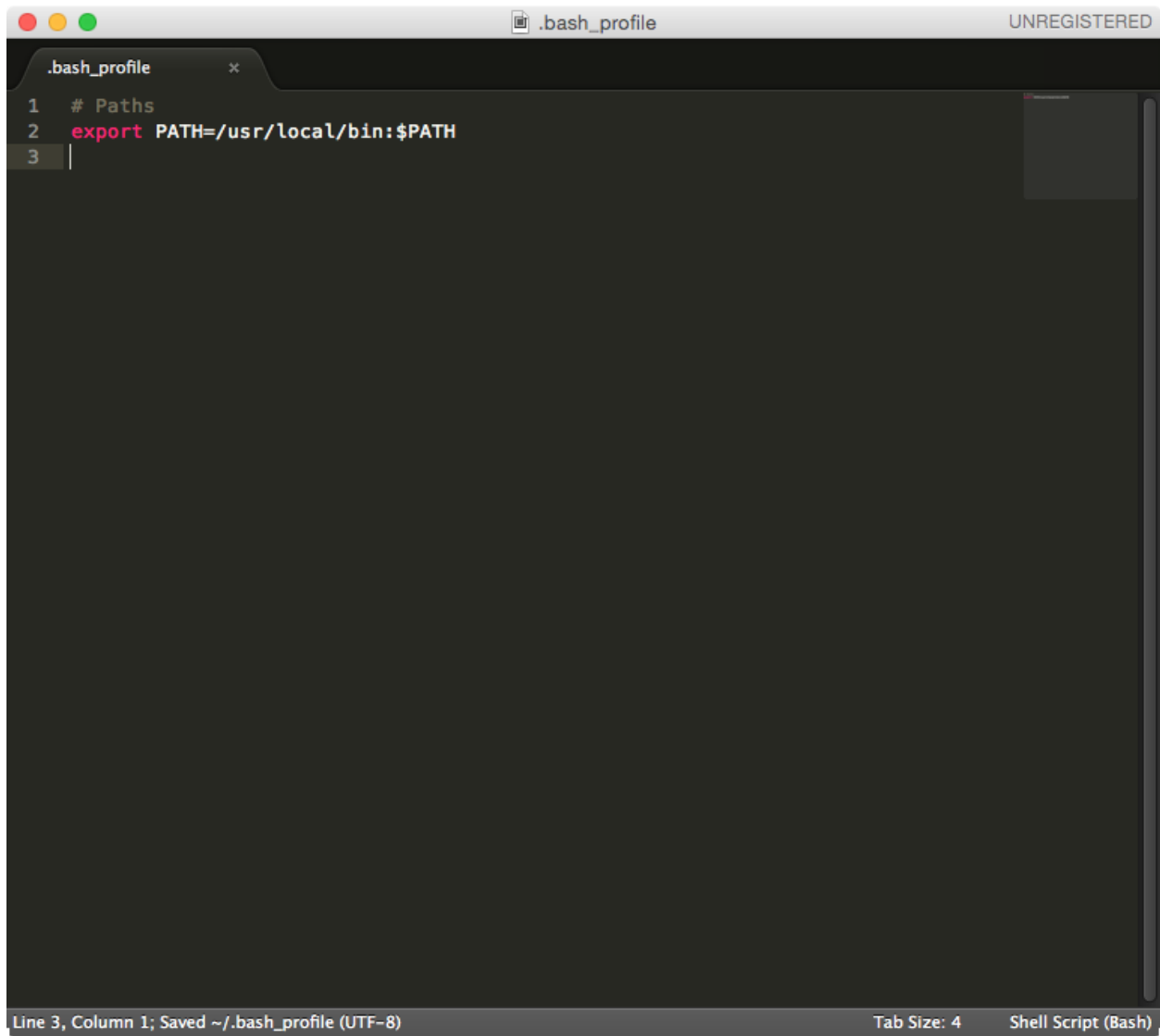
Next we assign the value `/usr/local/bin:$PATH` to `PATH` and export it at the same time. Exporting `PATH` ensures that the variable is loaded into memory and accessible.

Note: The difference between `$PATH` and `PATH` is subtle but worth pointing out. When you assign a value to a variable, then the variable should be called without `$`. If you want to *evaluate* the variable to get its value for use in Bash, prepend `$` to the variable name.

Save and close the file.

3.5.5 Sourcing your Bash profile

We edited our Bash profile, but it is critical to remember the code in Bash profile runs only when a *new Bash session is created*, which is called **sourcing**. Therefore, our changes will take effect when you quit Terminal and open it again to make sure that `PATH` is in fact exported. When Terminal is open again, you can check the value of `PATH` by running the `echo` command:



The image shows a text editor window with a dark theme. The title bar at the top indicates the file is `.bash_profile` and is `UNREGISTERED`. The editor has a single tab labeled `.bash_profile`. The content of the file is as follows:

```
1 # Paths
2 export PATH=/usr/local/bin:$PATH
3 |
```

The status bar at the bottom provides additional context: `Line 3, Column 1; Saved ~/.bash_profile (UTF-8)`, `Tab Size: 4`, and `Shell Script (Bash)`.

```
$ echo $PATH
```

The output might look like one of the two:

```
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

```
/usr/local/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

Again ensure that `/usr/local/bin` is listed prior to `/usr/bin`.

Note: Although Terminal allows a Bash profile to be sourced on command without restarting (`source ~/.bash_profile`) the method can be unreliable.

`PATH` is just one environment variable we changed in our Bash profile. There are a lot more, like `USER` for the current user (you!) and `HOME` for the path to the home directory of the current user. We will edit our Bash profile a few more times to run other important code in the future.

3.6 Homebrew

Several years ago, compiling one’s software was a [messy and complicated process](#). [Some tried](#) to create solutions, but it was said that the process was enough to [drive one to drink](#) until **Homebrew** had arrived.

3.6.1 What’s Hombrew?

Compiling software involves several implied steps, much like downloading and installing a binary: Download the source code, often a [ZIP](#) file of some kind, unzip the file, compile the source code, move the binaries to correct locations, arrange the symbolic links, delete the downloaded file, perform any other cleanup or special tasks, determine any dependencies, and do the whole process over again if dependencies do exist. Don’t forget about updating and uninstalling.

[Homebrew](#) handles all of these tasks as “the missing package manager for OS X” and “installs the stuff you need that Apple didn’t.” A **package manager** is a program that automates the installation and maintenance of software libraries, which are self-contained bundles of code. Packages contain not only source code, but also the license, [README](#), and other release notes. Homebrew manages [thousands of packages](#), each of which has a unique set of installation instructions called “formula.”

Homebrew installs all software to the `/usr/local/Cellar` directory and creates symbolic links in `/usr/local/bin` and `/usr/local/lib` that point back to your “Cellar.” It’s a very clean way to manage packages and automates an [existing best practice](#). And because we edited the `PATH` environment variable in [Understanding your PATH](#) to prioritize `/usr/local/bin`, all of the software Homebrew installs will take precedence. Perfect!

Homebrew is written in Ruby, but it can be used to compile almost any other software, including Python. In doing so, Homebrew depends on the pre-installed version of Ruby that came with OS X.

Warning: Homebrew is a package manager for larger “general purpose” packages, such as Python, SQLite, MySQL, PostgreSQL, or Git. Do not install packages whose languages have package managers of their own. For example, packages written in Python be installed not with Homebrew but with pip, a package manager for Python, and will be explained later.

3.6.2 Let's start brewing

As a precaution, set the ownership of the `/usr/local` directory to yourself. Correct ownership ensures Homebrew will be able to install packages on your behalf without errors. You should already have ownership, but sometimes an erroneous permission can slip through.

```
$ sudo chown -R `whoami` /usr/local
```

Please note that the `grave accent` or “backtick” (```), are *not* single quotation marks. Backticks are to the left of the number 1 key on many keyboards. You could also use the environment variable `$USER` (assuming it wasn't overwritten) or type in your username manually, but backticks are the easiest and least error prone method.

Now install Homebrew.

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

You should run `brew doctor` and any other on-screen instructions to make sure Homebrew is up to date and error free.

```
$ brew doctor
```

3.6.3 Installing Python

Django is written in the `Python` programming language, and as such *requires Python* to run. OS X comes with a pre-installed version of Python. Let's find out where that installation currently lives and which version it is.

```
$ which python
/usr/bin/python
$ python --version
Python 2.7.5
```

This version of Python is the globally accessible version and likely a little bit old by now. To remedy these issues, let's use Homebrew to install a newer version of Python.

```
$ brew install python
```

The success message should look something like:

```
python: stable 2.7.9 (bottled), HEAD
https://www.python.org
/usr/local/Cellar/python/2.7.9 (4855 files, 79M) *
  Poured from bottle
From: https://github.com/Homebrew/homebrew/blob/master/Library/Formula/python.rb
==> Dependencies
Build: pkg-config
Required: openssl
Recommended: readline , sqlite , gdbm
==> Options
--quicktest
  Run `make quicktest` after the build (for devs; may fail)
--universal
  Build a universal binary
--with-brewed-tk
  Use Homebrew's Tk (has optional Cocoa and threads support)
--with-poll
  Enable select.poll, which is not fully implemented on OS X (http://bugs.python.org/issue5154)
--without-gdbm
  Build without gdbm support
```

```
--without-readline
Build without readline support
--without-sqlite
Build without sqlite support
--HEAD
Install HEAD version
==> Caveats
Setuptools and Pip have been installed. To update them
  pip install --upgrade setuptools
  pip install --upgrade pip

You can install Python packages with
  pip install <package>

They will install into the site-package directory
  /usr/local/lib/python2.7/site-packages

See: https://github.com/Homebrew/homebrew/blob/master/share/doc/homebrew/Homebrew-and-Python.md

.app bundles were installed.
Run `brew linkapps python` to symlink these to /Applications.
```

You don't need to run `brew linkapps python` in the success message.

Let's find out where our new installation of Python lives and what version it is.

```
$ which python
/usr/local/bin/python
$ python --version
Python 2.7.9
```

Excellent! Because we set precedence in our Bash profile to look for programs in `/usr/local/bin`, and because Homebrew creates symbolic links to that location by default, we get our Homebrew installation whenever we reference Python from now on.

Note: Homebrew prevents multiple versions of Python to be installed at the same time. `pyenv` is a program that manages different versions of Python, much like the popular `rbenv` and `RVM` managers for Ruby. But because Homebrew installs Python 2.7.x by default, and because Python 3 is installed with the unique `brew install python3`, I don't recommend needing to install `pyenv`.

Note: Python 3 is the next major version of the Python programming language. It is a backward-incompatible upgrade; however migration guides for `Python` and `Django` exist. Updating code to Python 3 compatibility is a good idea in the long run, but Python 2 is expected to be supported [until 2020](#) at the time of this writing.

3.6.4 Installing SQLite

Django also requires a `SQL database`. `SQL`, which stands for Structured Query Language, is a category of programming languages that interact with `relational databases`.

`SQLite` is a good candidate for beginning developers and development on your computer because it's easier to use than its more complex but robust peers, like `PostgreSQL` and `MySQL`. By default, Django expects `SQLite` because it helps start development quickly. Install `SQLite` with Homebrew.

```
$ brew install sqlite
```

The success message should look something like:

```

sqlite: stable 3.8.7.4 (bottled)
http://sqlite.org/

This formula is keg-only.
Mac OS X already provides this software and installing another version in
parallel can cause all kinds of trouble.

OS X provides an older sqlite3.

/usr/local/Cellar/sqlite/3.8.7.4 (9 files, 2.1M)
  Poured from bottle
From: https://github.com/Homebrew/homebrew/blob/master/Library/Formula/sqlite.rb
==> Dependencies
Recommended: readline
Optional: icu4c
==> Options
--universal
  Build a universal binary
--with-docs
  Install HTML documentation
--with-fts
  Enable the FTS module
--with-functions
  Enable more math and string functions for SQL queries
--with-icu4c
  Enable the ICU module
--without-readline
  Build without readline support
--without-rtree
  Disable the R*Tree index module

```

Warning: Do not use SQLite in a production environment. SQLite supports a low number of concurrent database connections, which makes it a good candidate for development on your personal computer, but is not recommended for use on the web.

3.6.5 Troubleshooting Homebrew

Homebrew has a [troubleshooting checklist](#), but in general the following commands are the most helpful in keeping your brews up to date and trouble free.

```

# Search to see if a package is available
$ brew search <package>

# Display information about an installed package
$ brew info <package>

# Install a new package
$ brew install <package>

# Update installed packages
$ brew update

# Update to new major versions of installed packages
$ brew upgrade (<package>)

# Remove the old (existing but unused) versions of packages

```

```
$ brew cleanup (<package>)

# Delete stray symbolic links
$ brew prune

# Check all packages for installation integrity
$ brew doctor
```

It's possible to avoid installing Homebrew packages by visiting the respective websites of [Python](#), [SQLite](#), and others, and installing each DMG (or worse, compiling manually), but I highly recommend Homebrew for its convenience and ease of use.

3.7 pip & virtualenv

If you were reading carefully in *Installing Python* in the Homebrew lesson, you might have noticed the following lines in the success message.

```
Setuptools and Pip have been installed. To update them
  pip install --upgrade setuptools
  pip install --upgrade pip

You can install Python packages with
  pip install <package>

They will install into the site-package directory
  /usr/local/lib/python2.7/site-packages

See: https://github.com/Homebrew/homebrew/blob/master/share/doc/homebrew/Homebrew-and-Python.md
```

pip is the community-favored package manager for Python software, and is the [successor](#) to [Easy Install](#), which suffered from [several issues](#). Homebrew [installs pip automatically](#) for you. Sweet!

It might seem strange to use a package manager to have downloaded, well, *another* package manager, but each tool has specific capabilities that take advantage of the language's unique strengths. To make one all-encompassing package manager for all platforms and all languages would be unwieldy at best. Think of it like using one web browser to download another preferred web browser.

3.7.1 pip installs...what?

pip is a [recursive acronym](#) for “pip installs packages” and is, in and of itself, a Python package. If Homebrew didn't install pip automatically, we would have installed pip with Easy Install. Some common commands you'll run with pip:

```
# Search to see if a package is available
$ pip search <package>

# Display information about an installed package
$ pip show <package>

# Install a new package
$ pip install <package>

# Update an existing package
$ pip install <package> --update

# Install all packages from a requirements file
```

```
$ pip install -r requirements.txt

# Export a list of all currently installed packages to a requirements file
$ pip freeze > requirements.txt

# List all installed packages
$ pip list

# Uninstall a package
$ pip uninstall <package>
```

pip downloads Django and other Python software packages from the [Python Package Index](#), or PyPi (pronounced like “pie pie”), which is generally recognized by the community as the canonical source of Python software.

3.7.2 We are VR

If you installed packages solely with pip, pip would install them to `/usr/local/lib/python2.7/site-packages`, which is “local” to your computer use but *does not* separate packages from the possibly several Django (or in general Pythonic) projects you could write. The result is an inability to cleanly share your code with others because your project’s package dependencies are not cleanly separated. To solve this problem, the author of pip also created **virtualenv**.

virtualenv is a Python package that creates isolated development environments, preventing packages from colliding and conflicting with one another.

Some developers don’t use it, but I also recommend **virtualenvwrapper**, which is a collection of additional helpers that makes working with virtualenv easier. It’s another layer of abstraction, but I think the returns come back fairly quickly. virtualenv and virtualenvwrapper can be installed with pip in just one command.

```
$ pip install virtualenv virtualenvwrapper
```

pip installs virtualenv and virtualenvwrapper to `/usr/local/lib/python2.7/site-packages`. virtualenvwrapper requires additional settings in your Bash profile to ensure that it is available on the command line. Open your Bash profile.

```
$ subl ~/.bash_profile
```

Copy and paste the following lines, probably right after your PATH settings. Remember to restart Terminal.

```
# virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

The first line appends the hidden directory `.virtualenvs` to the path of the home directory `$HOME`, assigns it to the variable `WORKON_HOME`, and finally **exports it**. `.virtualenvs` is the name of the hidden directory in your OS X home folder where all of our virtual environments will be stored. `workon` will usually be the command to start working on a project.

It might make your head spin to think that a package’s only job is to isolate other packages, but it’s not very complicated. You should use pip to install virtualenv and virtualenvwrapper *globally* and only virtualenv and virtualenvwrapper globally. All other packages should be installed with pip but inside a virtual environment.

3.7.3 Making an environment

Let’s make a new virtual environment.

```
$ mkvirtualenv hello
New python executable in hello/bin/python2.7
Also creating executable in hello/bin/python
Installing setuptools, pip...done.
(hello)$
```

The name of my virtual environment was the ever-so creative `hello`. You can see that I entered my environment because `(hello)` prepends the `$` Bash prompt. Now whenever I install a package, it installs to the site packages directory of my virtual environment, which is `/Users/Rich/.virtualenvs/hello/lib/python2.7/site-packages`. Had I installed a package without being inside of my virtual environment, the package would have installed globally to `/usr/local/lib/python2.7/site-packages`.

It is worth noting that `virtualenvwrapper` automatically puts you inside a virtual environment whenever creating a new one. You won't totally understand these `virtualenvwrapper` commands now, but they're worth pointing out because we'll use some of them in the future (`<env>` or `hello` standing for the name of an environment).

```
# List all virtual environments
$ lsvirtualenv

# Make and enter a virtual environment
$ mkvirtualenv <env>

# Enter a virtual environment
$ workon <env>

# Change to a virtual environment's directory
# This would change you to ~/.virtualenvs/hello/
(hello)$ cdvirtualenv

# Change to a virtual environment's site packages directory
# This would change you to ~/.virtualenvs/hello/lib/python2.7/site-packages/
(hello)$ cdsitepackages

# Set a project directory
(hello)$ setvirtualenvproject <virtualenv directory> <project directory>

# Change to the project directory
(hello)$ cdproject

# Add a directory to the virtual environment's Python path
# Edits ~/.virtualenvs/hello/lib/python2.7/site-packages/_virtualenv_path_extensions.pth
(hello)$ add2virtualenv <directory>

# Exit a virtual environment
(hello)$ deactivate

# Remove a virtual environment
$ rmvirtualenv <env>
```

There is a lot more in `virtualenvwrapper`'s [command reference](#), but you can see that the naming conventions are similar to *Common Bash commands*. For example, while your virtual environment is active, you can change into the virtual environment directory by running `cdvirtualenv`, which, if you squint, kind of looks like the change directory command `cd`. You could have just as easily run `cd ~/.virtualenvs/hello/` but the shortcut `cdvirtualenv` can be easier to remember. Beginners are often confused by the phrase “cd into your virtual environment.” Someone is telling you to run `cdvirtualenv`—and not `cd virtualenv`, which wouldn't make sense!

Note: Virtual machine software like [VirtualBox](#) and [Vagrant](#) can be used with `pip` and `virtualenv` to further minimize

differences between development and production environments. Their use is often how development is done at larger organizations. [Getting Started with Django](#) by Kenneth Love has a great tutorial about using virtual machines with Django.

3.8 Django

You have your text editor, compiler, Bash profile, package manager, programming language, and database. *Whew!* Depending on your computer's prior setup, you might have been able to skip to this step, but we're being thorough to lay the foundation for success. It's time to install Django!

Let's find a suitable place for your Django project code. I prefer to save my websites to the `Sites` folder. The `Sites` folder used to be in your OS X home folder, but Apple removed it several versions ago. Let's put it back and change into it now. If it already exists, this command won't overwrite it.

```
$ mkdir -p ~/Sites/ && cd ~/Sites/
```

If you haven't already made and entered your virtual environment, do so now.

```
$ mkvirtualenv hello
New python executable in hello/bin/python2.7
Also creating executable in hello/bin/python
Installing setuptools, pip...done.
(hello)$
```

If you have an existing environment and exited it, enter it with `workon hello`.

3.8.1 Installing Django

We're all ready to go! Let's install Django with `pip`!

```
(hello)$ pip install django
Downloading/unpacking django
  Downloading Django-1.7.4-py2.py3-none-any.whl (7.4MB): 7.4MB downloaded
Installing collected packages: django
Successfully installed django
Cleaning up...
```

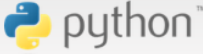
Without specifying a version of Django, that is `pip install django==1.7.4`, the latest version was installed, at the time of this writing 1.7.4. You can read about Django 1.7.4 on [PyPi's package page](#).

If you're interested in the nitty gritty: `pip` installed the file `Django-1.7.4-py2.py3-none-any.whl`, which indicates Django version 1.7.4 was installed (`Django-1.7.4`) and that it's compatible with Python versions 2 (`py2`) and 3 (`py3`). The `.whl` file extension indicates that the file is in the [wheel](#) format, which is a ZIP-format archive for Python packages and [improvement](#) over Python's previous `egg` format, which were similar ZIP-format archives created with `setuptools` and often used with Easy Install. `none` indicates purposeful omission of a hash string used in security to verify the package's validity. Django instead uses its own [MD5](#) hash string in the names of its files, which `pip` installs from a [simplified version](#) of PyPi's download page. If you scan diligently, you can see `Django-1.7.4-py2.py3-none-any.whl` in the list!

Now that Django is installed, we use the `django-admin` utility to start a new project.

```
(hello)$ django-admin startproject hello
```

But wait—how can Bash know where `django-admin` is and how to run this command?



» Package Index > Django > 1.7.4

PACKAGE INDEX >>

[Browse packages](#)
[Package submission](#)
[List trove classifiers](#)
[List packages](#)
[RSS \(latest 40 updates\)](#)
[RSS \(newest 40 packages\)](#)
[Python 3 Packages](#)
[PyPI Tutorial](#)
[PyPI Security](#)
[PyPI Support](#)
[PyPI Bug Reports](#)
[PyPI Discussion](#)
[PyPI Developer Info](#)

ABOUT >>

NEWS >>

DOCUMENTATION >>

DOWNLOAD >>

COMMUNITY >>

FOUNDATION >>

CORE DEVELOPMENT >>

Django 1.7.4

A high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Downloads ↓

Not Logged In

[Login](#)
[Register](#)
[Lost Login?](#)
 Use [OpenID](#) & [lp](#)

Status

Nothing to report

Latest Version: [1.8.2](#)

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Thanks for checking it out.

All documentation is in the "docs" directory and online at <https://docs.djangoproject.com/en/stable/>. If you're just getting started, here's how we recommend you read the docs:

- First, read docs/intro/install.txt for instructions on installing Django.
- Next, work through the tutorials in order (docs/intro/tutorial01.txt, docs/intro/tutorial02.txt, etc.).
- If you want to set up an actual deployment server, read docs/howto/deployment/index.txt for instructions.
- You'll probably want to read through the topical guides (in docs/topics) next; from there you can jump to the HOWTOs (in docs/howto) for specific problems, and check out the reference (docs/ref) for gory details.
- See docs/README for instructions on building an HTML version of the docs.

Docs are updated rigorously. If you find any problems in the docs, or think they should be clarified in any way, please take 30 seconds to fill out a ticket here: <https://code.djangoproject.com/newticket>

To get more help:

- Join the #django channel on irc.freenode.net. Lots of helpful people hang out there. Read the archives at <http://django-irc-logs.com/>.
- Join the django-users mailing list, or read the archives, at <https://groups.google.com/group/django-users>.

To contribute to Django:

- Check out <https://www.djangoproject.com/community/> for information about getting involved.

To run Django's test suite:

- Follow the instructions in the "Unit tests" section of docs/internals/contributing/writing-code/unit-tests.txt, published online at <https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/unit-tests/#running-the-unit-tests>

File	Type	Py Version	Uploaded on	Size
Django-1.7.4-py2.py3-none-any.whl (md5, pgp)	Python Wheel	py2.py3	2015-01-27	7MB
Django-1.7.4.tar.gz (md5, pgp)	Source		2015-01-27	7MB

Downloads (All Versions):
 12062 downloads in the last day
 182091 downloads in the last week
 788665 downloads in the last month

Author: Django Software Foundation
Home Page: <http://www.djangoproject.com/>
License: BSD
Categories
[Development Status :: 5 - Production/Stable](#)
[Environment :: Web Environment](#)

When you activated your virtual environment, the file `~/ .virtualenvs/hello/bin/activate` was automatically run, which in turn ran the commands `PATH="$VIRTUAL_ENV/___BIN_NAME___:$PATH"` and `export PATH`. `virtualenv` was smart enough to prepend the `bin` directory of the virtual environment (`$VIRTUAL_ENV/bin`) to the `PATH` environment variable (`$PATH`). While your virtual environment is active, the *new* `PATH` might look more like:

```
$ echo $PATH
/Users/Rich/.virtualenvs/hello/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

What about `django-admin`? When you ran `pip install django`, `pip` was smart enough during installation to recognize Django's own `django-admin` script file as an executable file by virtue of its placement in Django's `setup.py`, and then to copy it to your virtual environment's binary directory, `~/ .virtualenvs/hello/bin`. In other words, there were a lot of smart, hard-working developers to make sure everything lines up into one neat, little command!

Note: For the sake of simplicity, I recommend creating one Django project for every one virtual environment, and that you use the same name for your Django project as your virtual environment.

After running `django-admin startproject hello`, Django created a `hello` directory that looks like the tree structure below.

```
hello
-- manage.py
-- hello
    -- __init__.py
    -- settings.py
    -- urls.py
    -- wsgi.py
```

You can see that within `hello` a `manage.py` file was created alongside another `hello` directory. Within the second `hello` directory, other files like `__init__.py` and `settings.py` exist.

Every Django project comes with a `manage.py` file, which is the *utility you will use* to run commands. You could continue to use `django-admin`, but it's a global utility that could be run against *several* Django projects within a single virtual environment, and would need further configuration to run commands specific to our *single* project. For that reason, I recommend using `manage.py` from now on. Thanks, `django-admin`. It has been real.

Note: Beginners are often confused by the way in which Django project code interacts with the activated virtual environment. *Your* Django project code, represented by `~/Sites/hello/`, does *not* go into the directory of your virtual environment, `~/ .virtualenvs/hello/`. Your virtual environment merely isolates software packages you install, like Django, from other packages in their respective virtual environments. Your project directory contains code that *hooks into* Django, which `pip` installed to `~/ .virtualenvs/hello/lib/python2.7/site-packages/`.

To get all those awesome “*batteries included*” that come with Django, you have to *create the database tables* for them. Change into your project directory and run the `migrate` command.

```
(hello)$ cd ~/Sites/hello/
(hello)$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Django provides several of its own Django-specific Python packages—known as “Django apps”—for use right out

of the box. These apps include the `content types` app, the `authentication and authorization` app, the `admin` app, and the `sessions` app. Django’s own apps are decoupled from the framework, which means they can be removed, but it’s a good idea to leave them installed because various third-party apps can depend on them. Django’s default apps use the `publicly documented methods` by which all apps are loaded, which means that the framework “eats its own dog food”—a very good sign when software uses itself!

After migrating, run the local web server that comes with Django with `runserver`. The local web server simulates a production web server, which makes it great for clicking around on your simulated website without deploying it live to the web.

```
(hello)$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
February 09, 2015 - 21:50:56
Django version 1.7.4, using settings 'hello.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Warning: The `runserver` command is meant only for `testing your website on your personal computer`. It is **not** suitable for production use. From Django’s documentation: “We’re in the business of making Web frameworks, not Web servers, so improving this server to be able to handle a production environment is outside the scope of Django.”

3.8.2 The moment of truth

Time for the moment of truth! Open a web browser and visit `http://127.0.0.1:8000`. At long last you should see the “It worked!” page. Great job! Feel free to celebrate in the way befitting to you: back pats, high fives, fist bumps, etc.

You can go back to Terminal and quit the local server by pressing `CONTROL-C`. You can exit your virtual environment.

```
^C(hello)$ deactivate
```

3.8.3 Clean in house

I recommend setting a default project directory for your virtual environment. The default project directory is the directory you automatically change to when you start working on your project, and affords you to not think about which directories to traverse to start working.

You can set your project directory with `virtualenvwrapper`’s `setvirtualenvproject` command.

```
$ setvirtualenvproject $WORKON_HOME/hello ~/Sites/hello
Setting project for hello to /Users/rich/Sites/hello
```

To start working on your project again, just run the `virtualenvwrapper` `workon` command. Run the Bash `pwd` command to show that you are in fact in the project directory.

```
$ workon hello
(hello)$ pwd
/Users/rich/Sites/hello
```

Additionally you can use the `cdproject` command to snap back to your project directory if you ever move away from it.

Now that your project is set up, you can deactivate your virtual environment.

It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

```
(hello)$ deactivate
```

Exit your Bash session cleanly.

```
$ exit
logout

[Process completed]
```

Note: If for whatever reason you'd like to remove your virtual environment and your Django project:

```
$ rmvirtualenv hello
$ rm -rf ~/Sites/hello/
```

Congratulations on your first steps to becoming a Django web developer!

Please visit FriendlyDjango.org for more tutorials, including [Friendly Photos](#), which is the next step in how to create a Django web application!