

[< Previous](#)[Next >](#)

Unit 1 / Lesson 2 / Assignment 2

# Git and GitHub Basics

Estimated Time: 1-2 hours

On the path to becoming a developer, there's one more tool you must learn as part of your development workflow: Git. Git is the most popular program used for version control. Git lets you save snapshots of your project as you write it. Each time you save a snapshot, it shows you how the version you just saved differs from the previous version. You can (and will) use Git on any kind of project, whether it's a simple Python script, or a complex web application.



Git on its own is useful, but if you want to work from multiple computers or collaborate with friends, simply saving a snapshot to your machine isn't enough. You will need to save a version to a server in the cloud that you and your co-workers can access. For that, you'll use GitHub, the most popular service for hosting code remotely. After saving a snapshot to Git, you'll "push it" to GitHub and then you'll have a copy stored on their servers. If you then switched to a new computer (or if a collaborator logged in from a different computer), you could "pull" a copy from GitHub, and work from exactly where you left off.

There are a number of ways to use Git, but the most common workflow is to control it using the command line. That's how you'll do it. For the rest of the course, you're going to save every project using Git and GitHub. In this assignment, you'll learn and practice the basics. By the end of the assignment, you'll be able to:

1. Initialize (create) a new Git repository
2. Add and commit (save) files to a repository
3. Create a repository on GitHub for your project
4. Push your commits to the GitHub repository
5. Recover lost work from GitHub

There's way more you can do with Git and GitHub, and more you'll have to learn to work as a developer, but you should become comfortable with these basics during this course before learning more advanced workflows.

## Gitting ready

First, you'll need a GitHub account. Create an account by clicking [this link](#) and following their instructions.

Then you need to save your name and email in Git. You'll only have to do this once. Use the following commands:

```
git config --global user.name "Beyonce Knowles"  
git config --global user.email beyonce@thinkful.com
```

Next, you may optionally set up a list of files which you want Git to ignore. Make a new file (not a directory) in your home directory (~) called *.gitignore\_global* (notice the leading dot). Copy the contents of [this file](#) into *.gitignore\_global*. Then tell Git to use the file as a guide to which files to ignore. Again, this will only need doing once. Use the following command:

```
git config --global core.excludesfile ~/.gitignore_global
```

## Initializing a Git Repository

In the previous assignment we created a *hello\_world* project. So let's initialize a Git repository in that project.

First we need to change directory so we are inside the *hello\_world* project using `cd`. To confirm you've done it correctly, print your working directory (are you still saying 'pwd' out loud?). Then, use the `git init` command to initialize the repository:

```
$ cd ~/workspace/thinkful/projects/hello_world
$ pwd
/home/ubuntu/workspace/thinkful/projects/hello_world
$ git init
Initialized empty Git repository in /home/ubuntu/workspace/thinkful/pr
```

Notice how Git tells you that you created the repository in the *.git* directory. This is a new directory that Git creates where all of the history for the project will be stored. The existence of a *.git* directory is also used to inform Git that we are working within a repository. The dot at the start of the directory name indicates that it is a hidden directory (you won't see it if you open the folder in through a graphical interface).

## "Saving" Files

Take a look at what Git sees when a new repository is initialized. To do that, type `git status` into your command line. This command will soon be your best friend when using Git: it tells you everything you need to know about the current state of the repository. It's similar to `pwd` in that it doesn't cause anything to change; rather it's a sort of check for where you are:

```
$ git status
# On branch master
#
# Initial commit
#
```

```
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README.md
#       hello_world.py
#
nothing added to commit but untracked files present (use "git add" to
```

First it tells us that we are working on the **master branch**. A single repository can have multiple branches, each containing a different version of the code. There are a number of different ways to use branches to help you manage your code. For example one common workflow is to have a stable branch which contains production-ready code, and a development branch which is used as you develop new features. The master branch is the default branch which Git creates for us to work in.

The next important piece of information Git gives when you type 'git status' is the status of the files. Currently you have two "untracked" files. These are files which are inside the repository but are not under version control. To take a step back, when you save a Word document, you only need to do one thing: hit "save." In Git, saving a file requires two steps: telling Git to track the file (what's called, "adding" it), and then saving it (what's called "committing").

Generally you don't want to have untracked files within your repository. So let's use `git add` followed by the directory or file name to tell Git to track any changes we make to the files. Type the following line:

```
$ git add .
```

Now let's see what's changed in our status:

```
$ git status
# On branch master
#
# Initial commit
```

```
#  
# Changes to be committed:  
# (use "git rm --cached <file>..." to unstage)  
#  
#       new file:   README.md  
#       new file:   hello_world.py  
#
```

Now that Git is tracking the two files, you can commit them to store this particular snapshot of your project. Try committing the changes using the `git commit` command:

```
$ git commit -m "Initial commit"
```

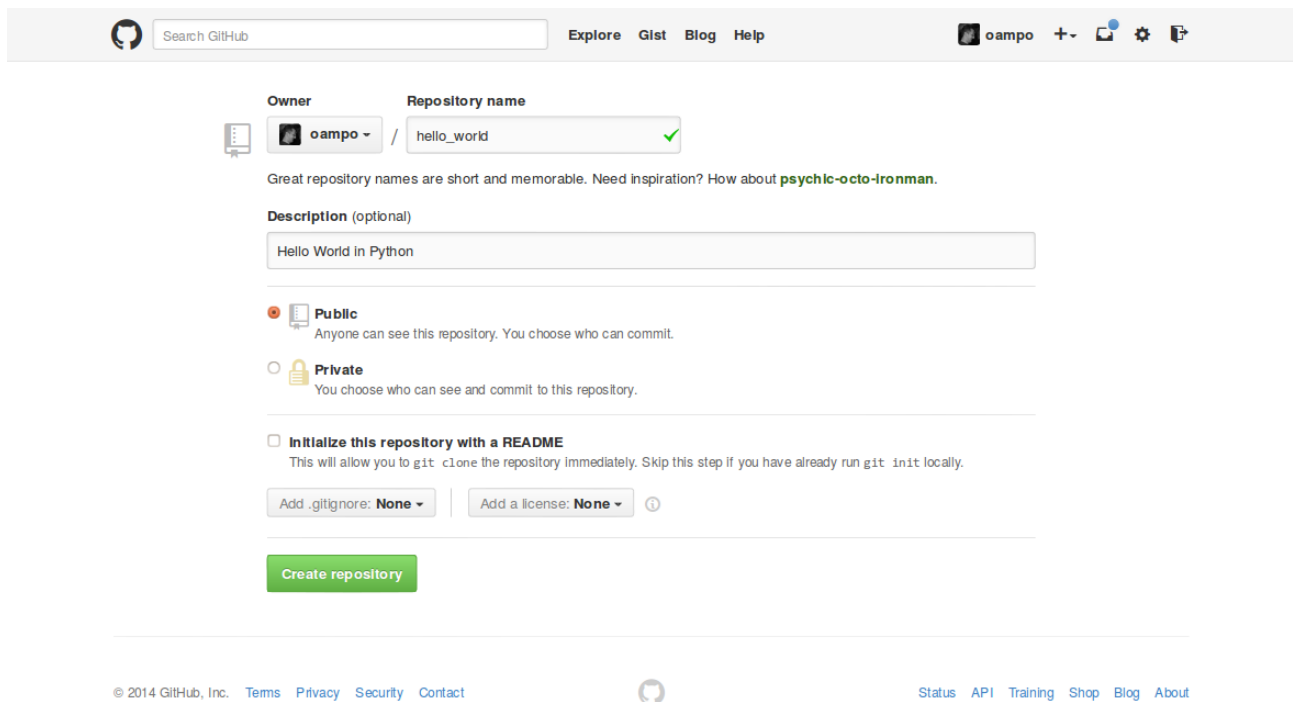
The `-m` option here is used to give a description of what changes you made in the commit. The two files have now been committed successfully. To confirm that everything was committed properly, type `git status` :

```
$ git status  
# On branch master  
nothing to commit, working directory clean
```

When you see 'nothing to commit, working directory clean', you know that all your recent work was committed.

## Saving your Snapshot to GitHub

Now that you have your first Git repo up and running you want to send it to GitHub so your co-workers can start adding to it. Go to the [create new repository page](#), fill in the information for your repository, and hit *Create Repository*:



Owner: oampo / Repository name: hello\_world ✓

Great repository names are short and memorable. Need inspiration? How about **psychic-octo-ironman**.

Description (optional): Hello World In Python

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

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

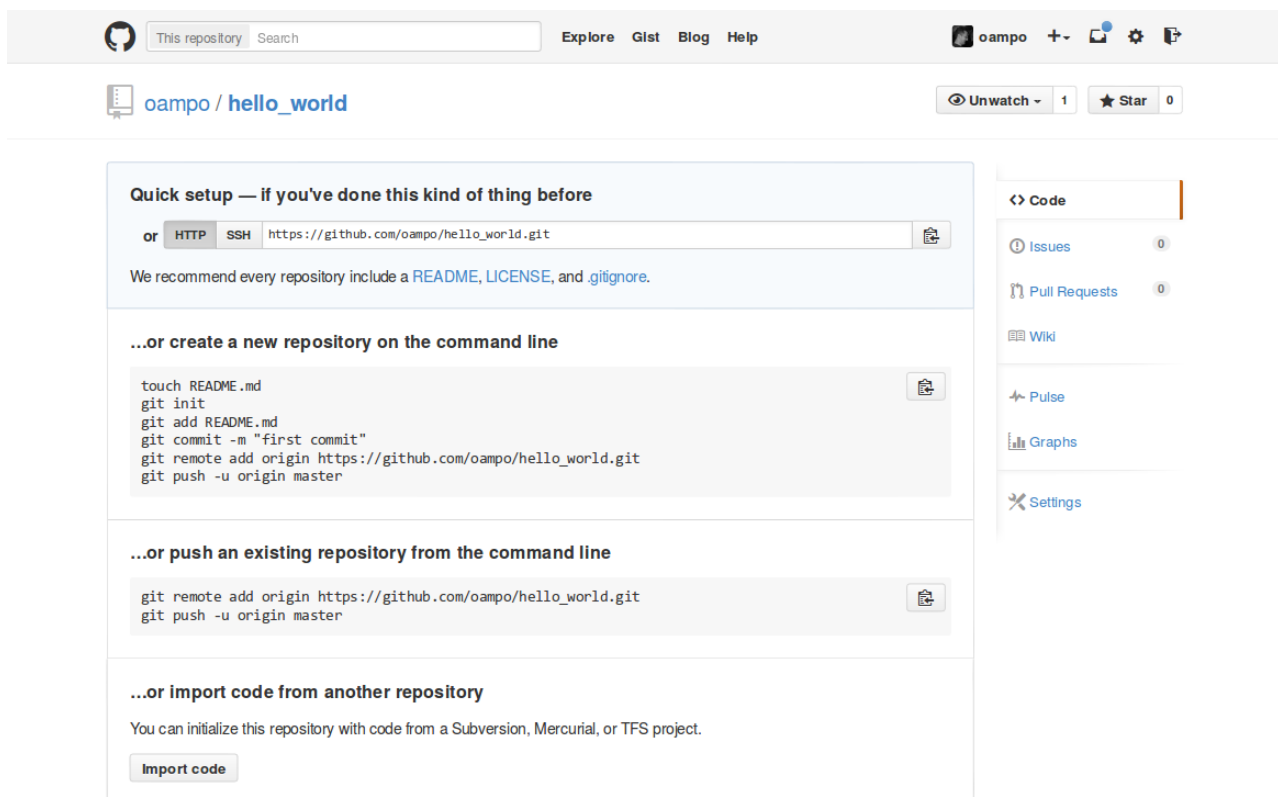
☐ **Initialize this repository with a README**  
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

© 2014 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Contact](#) [Status](#) [API](#) [Training](#) [Shop](#) [Blog](#) [About](#)

When you have created the GitHub repository you will be shown a screen with some instructions for linking your Git repository with the remote GitHub repository:



oampo / hello\_world Unwatch 1 Star 0

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

or **HTTP** **SSH** `https://github.com/oampo/hello_world.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/oampo/hello_world.git
git push -u origin master
```

**...or push an existing repository from the command line**

```
git remote add origin https://github.com/oampo/hello_world.git
git push -u origin master
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

**Import code**

**Code**

- Issues 0
- Pull Requests 0
- Wiki
- Pulse
- Graphs
- Settings

Because you have an existing repository set up on your computer, follow the second set of instructions. First, add the GitHub repository as a remote to

your local Git repository using the `git remote add` command (copy the command from GitHub rather than here):

```
$ git remote add origin https://github.com/oampo/hello_world.git
```

A remote is a copy of your local repository which is held somewhere else (on GitHub's servers, in this case). You can push changes to a remote or pull changes from a remote to make sure that your local and remote repositories stay in sync. You can have more than one remote for a single repository. For example, you may have a remote which belongs to a collaborator so you can pull changes they have made into your local repository. Or you may have a separate remote which you will push to in order to deploy the latest version of your code.

Here we are creating a remote called *origin*. This is the name which is conventionally given to your main remote. You should push changes up to your origin remote on a regular basis so you have an up-to-date backup of your code and its history.

Now, send your changes to the remote repository using the `git push` command.

```
$ git push origin master
Username for 'https://github.com': oampo
Password for 'https://oampo@github.com':
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 282 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/oampo/hello_world.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

This command is doing a couple of things. First, it will create the master branch in the remote repository because it does not currently exist. Then it sends the most recent snapshot of the newly created master branch so the copy on GitHub matches our local copy.

Try visiting your repository on the GitHub website. You should see that your local files and change history are now available to view online.

## Pulling code from GitHub

There's one more basic function of Git you need to know: how to pull from a remote repository so that your version matches the version on a remote server. To do so, you'll use the 'git pull' command, pulling from the master branch. Type the following command:

```
$ git pull origin master  
Already up-to-date.
```

Since the repo you're pulling from is already the same as your local repo, git compares the two and notifies you that your local version is up-to-date. If the remote version were different than your local version, it would overwrite your local version with any updated changes, such as a new feature or bug fix that a co-worker had developed.

## Practicing this workflow: Challenge

This workflow is the same one you should follow with every project: initialize a Git repository on the folder of a specific project (make sure to initialize it in the right place), add your files, commit them, and then push them to GitHub. If it's your first time pushing a project to GitHub, you'll need to visit the



GitHub website, create a repository there, and then set your local repository up to match that with the command provided by GitHub.

A common mantra with Git is: "commit early and often." You want to save your work frequently so you don't lose progress, and so that you can review the history of your progress in logical steps when you're finished (there's a page on GitHub where you can see a history of commits).

Run through this workflow one more time, this time using your FizzBuzz project:

1. Initialize the repository using Git from the command line.
2. Create a repository on GitHub and connect your local repository to it.
3. Add and commit your files.
4. Push them to GitHub.

You are likely to find yourself referring back to the commands for each of these steps. If you are, do the challenge one more time for your notes directory to get comfortable with the workflow.



· [Report a typo or other issue](#)

Completed



Previous

Next

