

Git Level 1 - Hello World and Local Repositories

Copyright

Adapted from the SUAV git tutorial series and licensed to HYL.

Contents in brief

In this tutorial we'll cover the following concepts and commands

- Version control
- Git repositories
- Commits
- git init
- git add
- git commit
- git log

Introduction

Welcome to our bare-bones introduction to git. If you've been around software and worked on a couple projects you might have heard this term tossed around. Through a couple basic examples I want to get you starting to understand what git is all about and some of the major concepts.

First of all we'll start with version control.. what is it? Why do we need it? Then we'll move on to some specific git commands to help you on your projects.. I'll introduce you to github and then we'll talk about some of the more advanced techniques for continuous integration.

Note about master vs main

Recently github has changed the default branch name from "master" to "main". Thus some commands may not work exactly as written; you might need to substitute master for main or vice-versa.

What is version control?

Right off the bat we have a word that I think warrants some explaining. Version control is a somewhat nebulous concept, but for our purposes I'll define it as "the act of monitoring and recording changes to our codebase".

Why git?

Let's start with a story.

Once upon a time there was a group of bright ENEL 300 students working on a coding project. However, they had a problem; there were five of them, so how could they all work on the code at the same time together? Well, you say, I have no problem working on something like google drive with my friends. Why can't we use that for code? And you might have a good idea because that's exactly what this group did.

However, google drive is obviously not made for sharing code like this. Consider some of the following issues:

- What if someone breaks the code with a change? Code is not like a word document where we can quickly fix a typo. A typo could easily break everyone's code and make some people very mad
- How can I quickly get the latest changes from my teammates? Will I have to download the entire google drive every time I want to test something out?
- What if I want to take a look at older versions of the code? Google drive actually does allow this but it's not really designed for this

In addition, working on a code base is different from working on a shared document because:

- We typically want some sort of approval process before changes make it into the codebase
- We want to keep track of why we made certain changes

Instead, the vast majority of the software industry uses tools like git which are specifically designed for editing code.

Our First Commit

Let's start from the very beginning. What is a repository? So if we continue with the google drive analogy, say you are working on a school project with a bunch of files. You might put those in a folder. Now say you were working on a software project. We'll put all our files in a folder and then we call this folder a "repository".

Just like we can have files on our google drive and files on our desktop, we can also have a git repository on our own computer -- we call this a local repository -- or on some server where we can share the code with others -- we call this a remote repository. We'll start by making a local repository to work with and then later we'll see how we can share our code to a remote repository. Now go ahead and follow the install for your machine.

Install

Windows Install

Download and install the git bash from <https://git-scm.com/download/win>. You can accept the defaults for the installation.

Mac or Linux Install

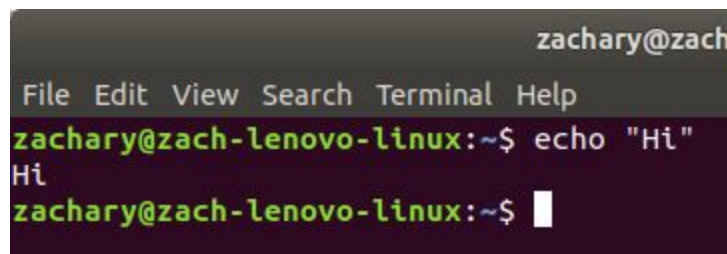
You should already have git installed! If not you might need to either

- Install XCode (Mac)
- Open a terminal and type `sudo apt-get install git` (Linux)

A Little About Command Line Interfaces

For this tutorial we will be using the git "command line interface". This means you will be communicating with the computer using text (as opposed to pressing buttons). Command line interfaces are very common in the software world so it's a good skill to learn. Most command line apps will be run in a "terminal". On Linux and Mac this means that we open the program called terminal and we're ready to talk to the computer! Windows has something called a "shell", but this is not as popular as the Mac and Linux terminal.

How does a terminal work? We type the name of the program we want to run followed by any "arguments". See the example below

A screenshot of a Linux terminal window. The title bar at the top right says "zachary@zach". Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the prompt "zachary@zach-lenovo-linux:~\$" followed by the command "echo "Hi"". The output "Hi" is displayed on the next line. The prompt "zachary@zach-lenovo-linux:~\$" is shown again on the third line, followed by a white cursor block.

```
zachary@zach-lenovo-linux:~$ echo "Hi"  
Hi  
zachary@zach-lenovo-linux:~$
```

In this case I have called the "echo" command with the argument "Hi". It just echoes back at me.

Command Line Basics - Navigation

Start by opening up a terminal! For Linux and Mac just find the program “terminal”. For windows you’ll want to open file explorer and right click. Then hit “git bash here”.

When we first open up our terminal we will usually be in our “home” folder (on windows you’ll be in whatever folder you were in when you hit “git bash here”). This is kind of like being inside your home folder in finder or your file manager. We can see what directory we are in by typing “pwd” which stands for “present working directory”.

```
zachary@zach-lenovo-linux:~$ pwd
/home/zachary
```

To see all the files in the directory type “ls”

```
zachary@zach-lenovo-linux:~$ ls
apmplanner2  Desktop                                Music
AppSource    Documents                             Pictures
Arduino      Downloads                             Programs
Audio        DyCalibResult                         Public
cces         eclipse-workspace                     snap
Code         forge-1.12.2-14.23.5.2847-installer.jar.log Templates
Datasets     mapscache                             tmp
deb          MATLABDesktopCreateError.log         Videos
zachary@zach-lenovo-linux:~$
```

We can also change to any directory we want with the “cd” command which stands for “change directory”. For example, below I changed to my Desktop folder.

```
zachary@zach-lenovo-linux:~$ cd Desktop/
zachary@zach-lenovo-linux:~/Desktop$
```

Another important command is “cd ..”. This will change to the parent directory of our present working directory. In this case I’m changing from my Desktop to my home directory (the little squiggle means home, in this case it would be /home/zachary since that is my home directory).

```
zachary@zach-lenovo-linux:~/Desktop$ cd ..
zachary@zach-lenovo-linux:~$
```

Aside about relative paths

Note that since Desktop is within my current directory, I don't need to specify the whole path (/home/zachary/Desktop). There are actually two ways to refer to any folder: absolute path and relative path. In this case "Desktop" is the relative path and /home/zachary/Desktop is the absolute path. If I changed to /home, then the absolute path would be unchanged (/home/zachary/Desktop), but the relative path would now be zachary/Desktop.

Making our first git repository

Since we're making a new project let's make a new folder. You can do this by navigating to wherever you want to put your project and using "mkdir"

```
File Edit View Search Terminal Help
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials$ mkdir git_tutorial_hyl
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials$
```

Now let's change into our folder

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials$ cd git_tutorial_hyl/
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```

To tell git that this folder is going to be a git repository we type "git init"

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git init
Initialized empty Git repository in /home/zachary/Desktop/HYL/Tutorials/git_tutorial_hyl/.git/
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```

So what's actually going on here? Well git is actually creating a new hidden folder (see below) inside our "git_tutorial" folder that will keep track of all the changes we make to our code! We can see this folder by using the special "-a" option with ls. This "flag" tells the "ls" command to list all the folders, including the "hidden folders".

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ ls -a
.  ..  .git
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```

A couple notes on hidden folders

Hidden folders are any folders that begin with a period. They're typically used by programs (like git) and not directly accessed by users. You'll also notice that we have the folders "." and ".."

listed. The “.” directory stands for our current directory and “..” as you’ll recall from before is our parent. If you’re curious you can change into our .git folder and take a look around.

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ cd .git
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl/.git$ ls
branches  config  description  HEAD  hooks  info  objects  refs
```

Hello World

Time to write our first hello world program, for this I’ll use python since it’s very simple and you might have experience. Go ahead and make a file called hello.py in your git_tutorial folder. You can open this with any sort of text editor... notepad, VS Code etc. Personally I prefer VS Code.

```
helloworld.py
1 print("Hello HYL!")
```

So there’s my python program! You can go ahead and run that however you usually run your python programs. If you can’t figure out how to actually run this program you can skip that step, you’ll still be able to do the rest of the tutorial.

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ code .
C zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ python3 helloworld.py
Hello HYL!
C zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```

And look at that, it runs fine! Now remember, git is a version control system so it will help us keep track of the work we have done. But it’s not automatic, I need to make something very important called a “commit”.

Commits

Here’s another git command for you: “git status”. If we run this command in our git terminal, it will show us the status of all the files in the directory. In my case, I get the following

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git status
On branch master

No commits yet

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

        helloworld.py

nothing added to commit but untracked files present (use "git add" to track)
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```

You'll notice that "helloworld.py" is red. Why? It's because we haven't added our changes to the project yet. This allows us to play around and test out our changes without having git track them.

Since we've tested our file and we're happy we can go ahead and add the hello_world.py file to our project with "git add helloworld.py".

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git add helloworld.py
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git status
On branch master

No commits yet

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

        new file:   helloworld.py
```

A "git status" shows us that git sees a new file has been added. However we still have to "commit" the change. This is important to note. Adding a change in git is a two step process, first we have to "add" the file (also called staging) and then we have to "commit" the change.

Let's go ahead and do that with a "git commit".

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git commit
```

At the next step it might ask you to specify your name and email. That's just so we can keep track of who made which change. Follow the instructions that git gives you. E.g. if your name was Steve Jobs it would be something like

```
git config --global user.name "Steve Jobs"
git config --global user.email "steve.jobs@apple.com"
```


Now we can really go ahead with our commit. Type “git commit” again and oh no... what’s this screen?!

```

GNU nano 2.9.3 /home/zachary/Desktop/HYL/Tutorials/git tutorial hyl/.git
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   helloworld.py
#

```

This right here is just a text editor. Before Graphical User Interfaces, we had command line text editors like Vim, Nano and Emacs. All that git wants is for you to write a “commit message” to go along with your commit. Since it’s a command line tool it can’t pop open a dialogue box or anything. It has to use these old text editors.

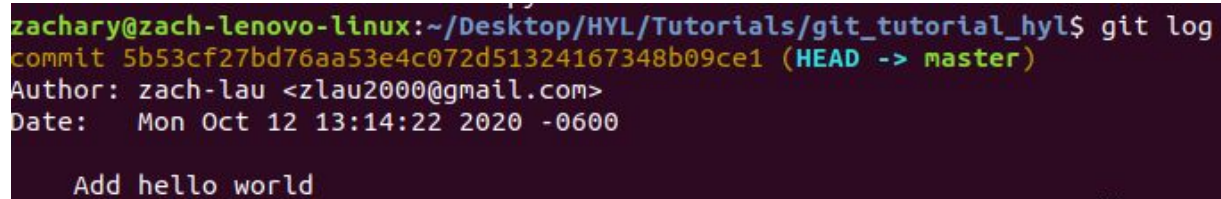
Now go ahead and follow the instructions for your editor. If you're using vim you're going to need to hit "i" to start typing, type your message, and then hit "escape" then type ":wq" and ENTER to finish. For nano just type your message then hit "CTRL+o" then "ENTER" then "CTRL+x". By the way, that "^" sign means to hit the control key. Your message should be succinct but also specific.

```
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git commit
[master (root-commit) 5b53cf2] Add hello world
1 file changed, 1 insertion(+)
create mode 100644 helloworld.py
zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$
```


And ta-da... we've made our first commit! But what is a commit? Well it's just a single unit of change. Any kind of modern software development is done in this kind of incremental fashion--make a change, write a message to explain why you made that change, rinse and repeat. This helps us to keep a very clear track of why each change was made and easily rollback to an older version if we break something.

The Log

Here's another command to try out.. "git log"

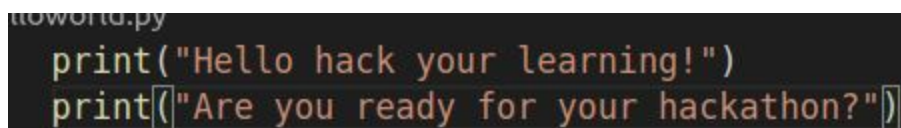
A terminal window with a dark purple background. The prompt is 'zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl\$'. The command 'git log' has been executed, showing the following output: 'commit 5b53cf27bd76aa53e4c072d51324167348b09ce1 (HEAD -> master)', 'Author: zach-lau <zlau2000@gmail.com>', 'Date: Mon Oct 12 13:14:22 2020 -0600', and 'Add hello world'.

This just shows a record of all the changes that we have made. There's something special I'd like to point out in this image... at the beginning see that it says "commit 5b53cf..."? This is called the commit "hash" and it's just a number associated with the change we just made. You'll see people use it often to refer to a specific commit.

The other thing you might be wondering about is the (HEAD -> master) thing. That has to do with branches... more on that later. Anyways, why don't we go ahead and make a few more commits just to practice.

Fast forward a few minutes...

So now my code looks like this

A code editor snippet with a dark background. The first line is 'roworld.py' in a light blue font. Below it are two lines of Python code: 'print("Hello hack your learning!")' and 'print(["Are you ready for your hackathon?"])' in a light orange font.

And my log looks like this

```

zachary@zach-lenovo-linux:~/Desktop/HYL/Tutorials/git_tutorial_hyl$ git log
commit 05171834825f3a1769289979ff3a3762a62eec43 (HEAD -> master)
Author: zach-lau <zlau2000@gmail.com>
Date:   Mon Oct 12 13:16:42 2020 -0600

    Add 'are you ready for your hackathon' line

commit 1209042a786cf2e7da254213bc1bdde5355c016c
Author: zach-lau <zlau2000@gmail.com>
Date:   Mon Oct 12 13:16:11 2020 -0600

    Change HYL to hack your learning

commit 5b53cf27bd76aa53e4c072d51324167348b09ce1
Author: zach-lau <zlau2000@gmail.com>
Date:   Mon Oct 12 13:14:22 2020 -0600

    Add hello world

```

See if you can reproduce my changes.

Review

In this tutorial we've taken the first steps towards using probably the most important tool you will need throughout your software career. You now know how to

- Create a repository
- Make a change and commit it
- Review your changes with "git log"

Give yourself a pat on the back and move along to Git Level 2 where we'll learn how to share our code with the world on GitHub.

Commands

git init	Create a git repository
git add <file>	Stage a file
git commit	Commit a file
git log	Show a history of commmits
git --help	Show the help menu