

Git Introduction for 6.270

What is git?

Git is a version control system – it allows you to save snapshots (or “versions”) of your code as you work. That means you can easily restore a working version in case you accidentally screw something up. And trust us, at some point this month you *will* completely bork your code.

Git also helps you share your work with your teammates – instead of sending emails back and forth with attachments like “NewestRealCode-useThisVersion.zip” or “SeriouslyUseThisVersion.zip”, you can simply “push” your changes to the “cloud” (aka a team-shared central repository hosted on Athena) and your teammates “pull” the changes to their own computer. If each of you has made changes to different parts of the code, git will automatically merge the changes when you “pull,” giving you the latest version of the code.

Throughout the course of the month, the organizers may make some tweaks to joyos in order to fix bugs or add some features – git allows us to make changes which you simply “pull” into your own code.

The most important thing to remember about git is that it's only useful if you take the time to use it properly. *If you don't regularly commit and push your changes when the code is working*, git won't be able to help you when your computer crashes or you delete your code by accident the night before the competition.

Side note: Why Git?

You might wonder, “why don't we just use email/flash-drives/Dropbox to share files? It seems a lot simpler.” This is a good question – there are several reasons that we use git:

- 1) Git makes it easy to grab the newest code from the organizers, without worrying about losing your changes
- 2) You will almost certainly use a version control system (VCS) like git in industry, and it helps prevent losing work if used properly
- 3) Git handles simultaneous edits cleanly when multiple people work on the same files

Installing git

If you already have git installed on your computer, skip ahead to “Get Joyos” below.

Windows

Get msysgit (<http://code.google.com/p/msysgit/downloads/list?q=full+installer+official+git/>)

Select the following options if prompted during installation:

“Use Git Bash only”

“Use OpenSSH”

“Checkout Windows-style, commit Unix-style line endings”

Mac OS X

Use the installer at <http://code.google.com/p/git-osx-installer/downloads/list?can=3>

Linux

```
sudo apt-get install git-core
```

Get Joyos

Windows

msysgit comes with a program called “Git Bash” which emulates a Linux terminal – by using Git Bash you should be able to follow the Mac/Linux instructions below.

Mac/Linux

Open a terminal window.

If you're not familiar with using a terminal/console, see [Appendix A](#).

Make sure your git username is set correctly (replacing the highlighted portions accordingly):

```
git config --global user.name "Firstname Lastname"  
git config --global user.email "your_email@mit.edu"
```

Navigate to your home directory, and create a folder for 6.270 files:

```
cd ~  
mkdir 6.270  
cd 6.270
```

Now we set up a local clone of your team's repository:

```
git clone your_athena@linerva.mit.edu:/afs/athena/course/6/6.270/git/team0 joyos
```

Git will say:

```
Cloning into joyos...
```

You will likely also see:

```
The authenticity of host 'linerva.mit.edu (18.181.0.232)' can't be established.  
RSA key fingerprint is 30:2e:20:bb:bf:84:70:4e:da:a7:33:bc:e8:20:c1:60.  
Are you sure you want to continue connecting (yes/no)?
```

Type yes and press [Enter] (you can ignore the warning on the following line - “Warning: Permanently added 'linerva.mit.edu' (RSA) to the list of known hosts.”)

Git will ask for your athena password - **nothing will show up as you type:**

```
Password:
```

You'll probably see a warning that the repository is empty – this is ok.

```
warning: You appear to have cloned an empty repository.
```

cd into the repository and configure it to know about the central joyos repository hosted on GitHub:

```
cd joyos
git remote add joyos git://github.com/sixtwhoseventy/joyos.git
```

Configure an editor for commit messages. We suggest 'open' on Mac (opens in TextEdit), 'gedit' on Linux, . You can use a more advanced editor like vim or emacs if you want.

Mac:

```
git config core.editor open
```

Linux:

```
git config core.editor gedit
```

Windows:

```
git config core.editor notepad
```

If you are the first person on your team to set up git, do the following:

Pull the latest version of joyos:

```
git pull joyos master
```

You should see output that looks like:

```
remote: Counting objects: 2636, done.
remote: Compressing objects: 100% (832/832), done.
remote: Total 2636 (delta 1794), reused 2579 (delta 1764)
Receiving objects: 100% (2636/2636), 4.99 MiB | 1.29 MiB/s, done.
Resolving deltas: 100% (1794/1794), done.
```

Push this to your team repository:

```
git push --all
```

```
Password:
Counting objects: 2737, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (838/838), done.
Writing objects: 100% (2737/2737), 5.55 MiB, done.
Total 2737 (delta 1863), reused 2737 (delta 1863)
To sbezek@linerva.mit.edu:/afs/athena/course/6/6.270/git/team0
* [new branch]      master -> master
```

Understanding Git

Your 6.270 code folder is known as a working copy – this is a local copy of the code that you can make changes to. Furthermore, there's a notion of **tracked files** - not all files within the working copy are tracked by git. For example, the compiled output files (*.o files) are setup to be ignored by git. You can track a file by using “git add *filename*”

When you have a version of your code that forms a logical milestone (e.g. you finally got your

navigation code working), you can **commit** those changes, which permanently saves a snapshot of the tracked files, along with a description you supply (known as a “commit message”). Unlike SVN (another version control system), git commits to a *local repository*, not a remote one. This means you don't even need an internet connection to save a checkpoint in your code. When you have one or more commits that you want to share with your teammates, you can **push** those changes to your team's remote repository (which is stored on Athena). Your teammates will **pull** those changes into their own *local repositories*, which **merges** your changes into their working copy of the code.

Some useful commands:

Command	Description
<code>git status</code>	Displays the status of your git working copy – useful to see which files you've changed and check to make sure all files are being tracked
<code>git add file1 file2</code>	Tells git to track file1 and file2
<code>git commit -a</code>	Commits all modified files – will open a text editor to type a descriptive commit message
<code>git push</code>	Push local committed changes to your team's shared repository
<code>git pull</code>	Pull teammate's changes from the shared repository and merge them into your own working copy (your local changes must be committed or else git will complain when you pull)
<code>git diff</code>	Show the difference between the latest committed version and the current version of your working copy – useful to see what you've changed since your last commit
<code>git log</code>	Show a log of past commit messages
<code>git show commit_id</code>	Show the changes made in the specified commit (the id will be a hash value like “33b80ab204897ca4e683fa6214fbc3edaofd24e3”, although you can usually just use the first few characters to uniquely identify it – e.g. “33b80”)

Restoring an old version from git

While the procedure isn't extremely difficult, it's nonetheless pretty easy to accidentally mess up your repository while trying to restore an old version. We suggest you see an organizer/TA to help with this process (they won't bite, I promise)

Git tutorial (recommended if you've never used a VCS before)

Make sure you've followed the directions above to get a copy of the joyos source code.

To demonstrate how version control works, each team member should create a new text file with their athena name (e.g. "sbezek.txt") in their 6.270 code folder.

```
cd ~/6.270/joyos/  
touch sbezek.txt
```

Now we can check the status of our git repository:

```
git status
```

Notice that your new file is listed under "Untracked files":

```
# On branch master  
# Untracked files:  
#   (use "git add <file>..." to include in what will be committed)  
#  
#   sbezek.txt
```

Now edit the file and write something fun inside:

On a Mac	On Linux:
open sbezek.txt	gedit sbezek.txt

We have to let git know that we want to track this new file:

```
git add sbezek.txt
```

Commit those changes with a descriptive commit message:

```
git commit -a -m "Added a fun little file"
```

```
[master 4ddc8c3] Added a fun little file  
1 files changed, 2 insertions(+), 0 deletions(-)  
create mode 100644 sbezek.txt
```

Push those changes to your team's repo (you will be prompted to enter your *Athena* password)

```
git push
```

If your teammates were faster than you, you may get a message that looks like:

```
To sbezek@linerva.mit.edu:/afs/athena/course/6/6.270/git/team0  
! [rejected]      master -> master (non-fast-forward)  
error: failed to push some refs to  
'sbezek@linerva.mit.edu:/afs/athena/course/6/6.270/git/team0'  
To prevent you from losing history, non-fast-forward updates were rejected  
Merge the remote changes (e.g. 'git pull') before pushing again.  See the
```

```
'Note about fast-forwards' section of 'git push --help' for details.
```

In this case you need to pull your teammates' latest changes before you can push your own

```
git pull  
git push
```

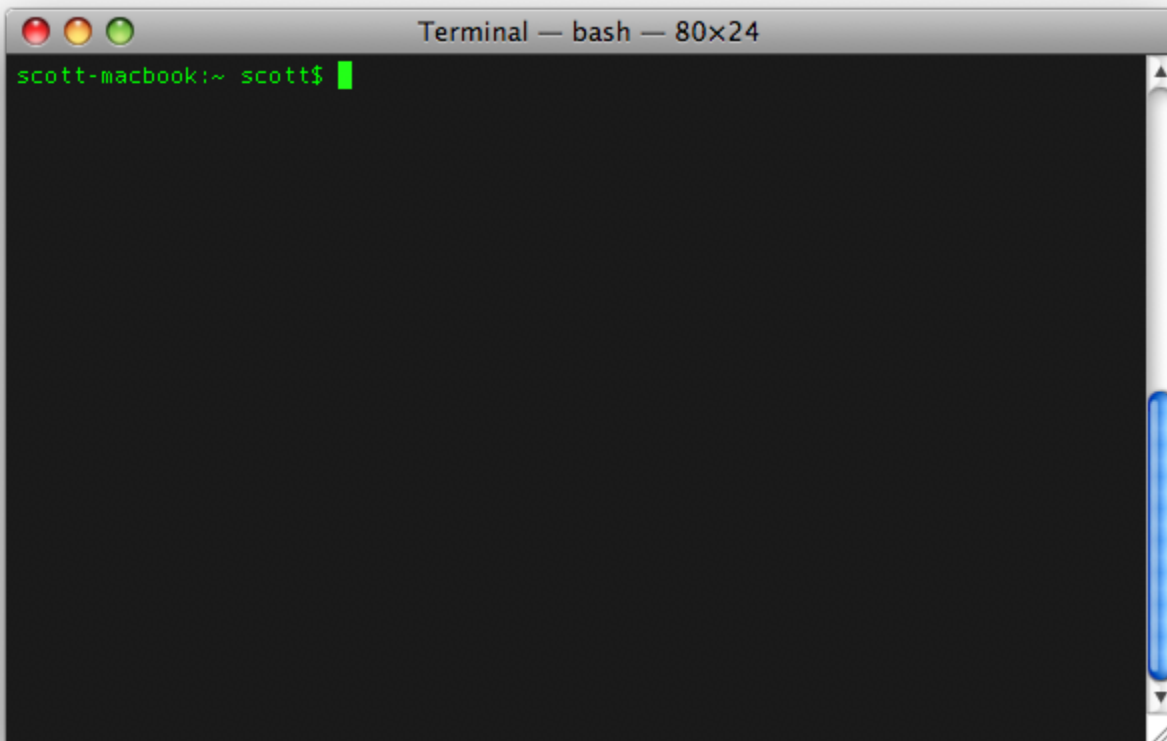
After everyone has committed and pushed their changes, make sure you pull the latest version

```
git pull
```

Appendix A: Using Terminal

The Terminal window allows you to run commands and programs on your computer in a completely text-based environment. The basic principle of the terminal is that you are sitting in a directory (the “present working directory” or “pwd”), from which you can navigate to other directories, access files, and run programs.

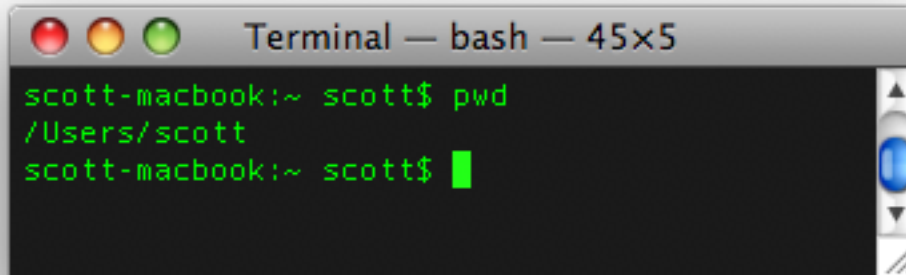
When you open Terminal, you'll see something like this:



There are a couple things to note:

- “scott-macbook” is the name of my computer
- the text after the colon is the directory that you are currently in – here “~” represents your home directory, which is “/Users/scott” on Mac or “/home/scott” on Linux.
- “scott” is my user name
- “\$” is the prompt – it indicates you can enter a command to run

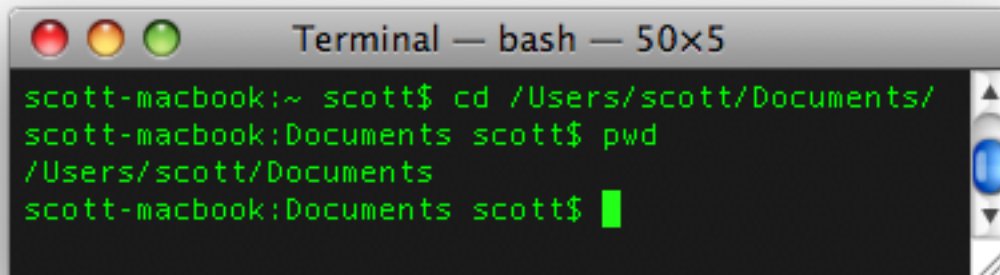
If you ever get lost in Terminal, you can type “pwd” and press [Enter]:

A terminal window titled "Terminal — bash — 45x5" with a dark background and green text. The prompt "scott-macbook:~ scott\$" is followed by the command "pwd". The output is "/Users/scott". The prompt "scott-macbook:~ scott\$" is followed by a green cursor bar.

```
scott-macbook:~ scott$ pwd
/Users/scott
scott-macbook:~ scott$ █
```

The current directory is printed on the next line (“/Users/scott”), and another command prompt appears below it (“scott-macbook:~ scott\$”).

To navigate to another directory, use the “cd” command (cd stands for “change directory”) followed by the directory you want to switch to. For example, we can navigate to my Documents folder by running “cd /Users/scott/Documents”:

A terminal window titled "Terminal — bash — 50x5" with a dark background and green text. The prompt "scott-macbook:~ scott\$" is followed by the command "cd /Users/scott/Documents/". The prompt "scott-macbook:Documents scott\$" is followed by the command "pwd". The output is "/Users/scott/Documents". The prompt "scott-macbook:Documents scott\$" is followed by a green cursor bar.

```
scott-macbook:~ scott$ cd /Users/scott/Documents/
scott-macbook:Documents scott$ pwd
/Users/scott/Documents
scott-macbook:Documents scott$ █
```

There are 2 ways you can type a path:

- **Absolute paths** begin with a “/” which indicates your system's root directory. E.g. /Users/scott/Documents
- **Relative paths** are specified *relative* to the current directory. E.g. if I'm currently in /Users/scott, I can navigate to /Users/scott/Documents simply by typing “cd Documents”

You can also use the special relative path “..” (two full stops) to indicate the parent directory. For example, if I'm at “/Users/scott/Documents”, I can type “cd ..” to navigate to the parent directory “/Users/scott/”. You can use this multiple times: from “/Users/scott/Documents/” I can type “cd ../../” to navigate to “/Users/”

Helpful hint: when typing a path, you can press [Tab] to auto-complete. For example, after typing “Doc” you can press [Tab] to complete it to “Documents”. If there are multiple files or

directories that begin with what you've already typed, you can press [Tab][Tab] to show a list of all possibilities. E.g. typing “Do[Tab][Tab]” will show “Documents/ Downloads/” as the 2 options.

