

Using git and Github notes

Prof Bill - Mar 2020

My goal here is super simple: Introduce you to git and Github.

These are two of the most important tools available for any coder today!

We won't be going into any great depth at all. I just want you to put some/all of your CSC 210 code up into the cloud using git/Github. The next step is yours.

thanks...yow, bill

Sections:

- I. Most important resources (links)
- II. Overview
- III. Download and signup
- IV. Using git + Github
- V. Etc

I. Most important resources (links)

Pretty much everything is online. Imagine that.

- ❖ **git** - the official site, git-scm.com
- ❖ git documentation hub, git-scm.com/doc
 - **Pro Git** is a free book are the site, git-scm.com/book/en/v2
 - Their 4 intro **videos** take less than 25 mins to watch, git-scm.com/videos
- ❖ **Download** git, git-scm.com/downloads

- ❖ **Github** - the official site, github.com
 - **Help** on Github, help.github.com/en/github
 - Github "learning paths", lab.github.com/githubtraining/paths
- ❖ **Download** Github...nope, let's just use their website

- ❖ Etc
 - History of git, en.wikipedia.org/wiki/Git#History and git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git
 - History of Github, github.com/about and en.wikipedia.org/wiki/GitHub#History
 - Git is open source and written in C; unsurprisingly, its code resides in a git repo on Github, github.com/git/git
 - Don't forget our favorite Noctrl CSC alum: Github co-founder PJ Hyett, github.com/pjhyett
 - The best git cheat sheet I've found (so far), github.github.com/training-kit/downloads/github-git-cheat-sheet/

II. Overview

What is **git**?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- git-scm.com

The official git site is git-scm.org. What is **SCM**?

~~SCM = Supply Chain Management...but not for nerds!~~

SCM = Source Control Management, aka version control

...the management of changes to documents, computer programs, large web sites, and other collections of information

- en.wikipedia.org/wiki/Version_control

What is **Github**?

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 40 million developers.

- github.com

/ OK, some of that is marketing blather. But it's largely true, imho. Github is HUGE and important and does more than just store some git repos. */*

What parts of git and Github are we **covering**?

- ❖ The most basic git commands: init, add, commit, push, pull, checkout.
- ❖ On Github, we'll create a Github repo and use it for Program #3.

What are we **skipping**?

- ❖ Ugh - tons. We are just barely skimming the surface.

- ❖ The biggies off the top of my head: collaboration, branches, merging, etc.

What's **the point**?

- ❖ I want you to have public repos of your school code and other work, if you want. Then, at a company or interview or in the kitchen with your folks, you show others your work (code) in a browser tab.
- ❖ We're skipping important stuff because of time limits. But my hope though is that this introduction will help you explore more about git and Github later if you want.

III. Download and signup

Download git. Signup at Github.

III.i Download git

Here's the link. Go!

git-scm.com/downloads

Run this command from the command prompt to verify that you're cool.

```
$ git --version
```

Tell git your name and email.

```
$ git config --global user.name "[name]"  
$ git config --global user.email "[email address]"
```

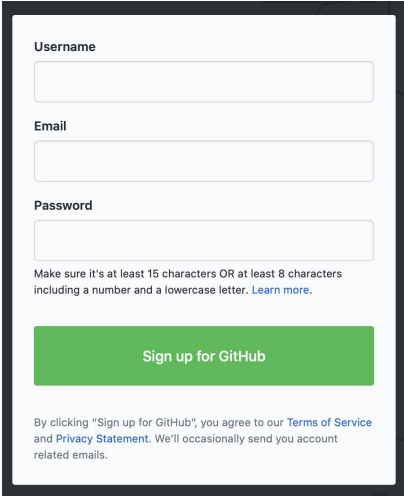
III.ii Signup at Github

Here's the link. Go! (again)

github.com

Signup choices can be stressful, my two cents:

- + I prefer a professional **username**; don't get too cute. Mine is williamt777, fwiw. Your real/full name is part of your profile
- + I use my real **email**, not noctrl. You can have multiple emails (like noctrl) and select one as primary
- + Dear fellow nerd - I sure *hope* you have a **password** algorithm or manager by now (cough)
- + Github will let you edit profile stuff later. They do warn that changing your username may be painful, however.



The image shows a screenshot of the GitHub signup form. It contains three input fields: 'Username', 'Email', and 'Password'. Below the 'Password' field, there is a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)'. At the bottom of the form is a green button labeled 'Sign up for GitHub'. Below the button, there is a small disclaimer: 'By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.'

IV. Using git + Github

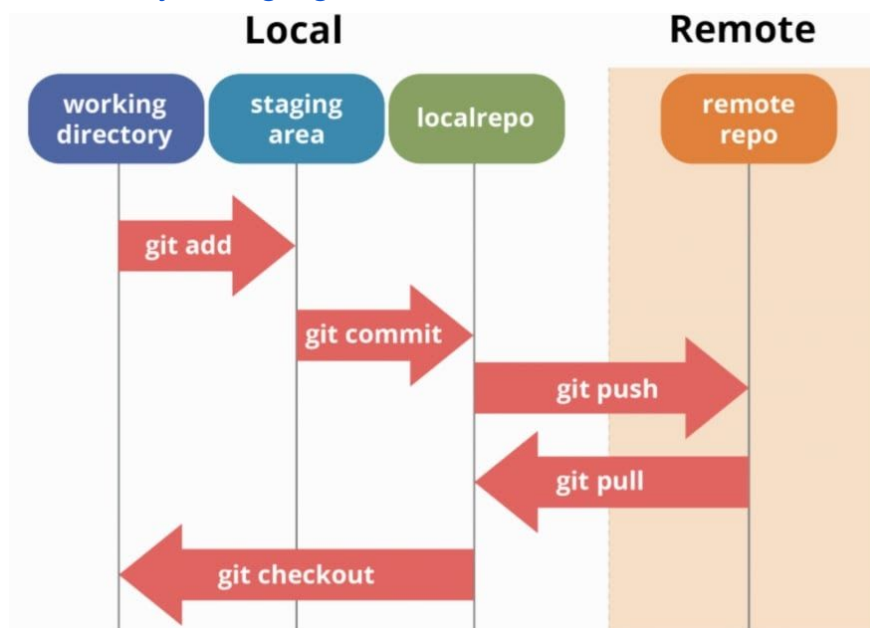
This section is the nitty-gritty. Get your command line ready. (smile)

IV.i. Workflow

This is often called "workflow".

I really like this blog post and diagram.

dev.to/mollynem/git-github--workflow-fundamentals-5496



Some details:

- "Local" is your machine and "Remote" is Github.
- Your workaday commands are: **git add** and **git commit**
- To save your work, **git push** it to Github.
- To get the latest, greatest from Github onto your local machine, do **git pull**.
- If you're using branches, use **git checkout**.

IV.ii. Brand new work, your first repo

Let's create your "first repo" and call it **csc210-hello**.

Step 1: Create a **new repo** at Github

- Goto github.com. Make sure you're logged in.
- Select the + sign in the upper right, then "New repository"
- Repo name: Try something like "csc210-hello"
- Other options: No README or gitignore or whatever...you can add those later

Step 2: Do local git setup. I follow the handy Github instructions (below), pretty much:

```
$ mkdir csc210-hello # new folder
$ cd csc210-hello
$ echo "# Hello, CSC 210" > README.md
$ git init
$ git add -A # all files is better
$ git commit -m "my first commit"

# copy-paste this command from Github, not mine
$ git remote add origin
https://github.com/your-username/csc210-hello.git
$ git push -u origin master
```

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

```
echo "# csc210-hello" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/williamt777/csc210-hello.git
git push -u origin master
```

Step 3: Check your repo on Github

- Go back to Github and click on your repo link. You should see your README.

Step 4: Try a normal workflow.

```
# make a small edit to your README.md
# Add a couple lines: "This is my first repo", "blah blah"
$ git add -A
$ git status    # for fun!
$ git commit -m "tiny edit to README"

# tell Github about this, push!
$ git push origin master
```

Refresh your Github repo to see your README changes.

Note:

- **README.md** is special in Github. It's automatically displayed at the bottom of your repo page; other files are not.
- Your repo is public. Just like mine: github.com/williamt777/csc210-hello
- Notice that you can visit my repo, and **git clone** to get your own copy.

IV.iii. An existing project

If you already have code, then the initial setup is slightly different than when starting from scratch. I put the "existing project" case first because it seems more common.

I'll mostly use the steps in Github's overview:

Adding an existing project to GitHub using the command line
[help.github.com/en/github/importing-your-projects-to-github/adding-an-existing-pr
oject-to-github-using-the-command-line](https://help.github.com/en/github/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line)

I call my example csc210-program3. That's a project I have already started.

Step 1: Create a repo on Github. Same as before. Repo name: **csc210-program3**.

Step 2: Init git

Goto to your existing folder and initialize git.

```
$ cd csc210-program3
$ git init
```

Step 3: add, commit

```
$ git add -A
$ git commit -m "short, helpful message"
```

Step 4: push up to your Github repo

```
# At your new Github repo, copy the csc210-program3.git link
$ git remote add origin <repo URL/csc210-program3.git>
$ $ git remote -v # verify

$ git push -u origin master
```

Check your repo on Github. That's it! Once you're setup, the workflow is the same as above: edit, git add, git commit, git push.

IV.iv Final usage points

If you crash and burn on a small program like this, you can always **start over**:

- Delete your Github repo
- Create a new repo, folder, copy your code, and do git init again

Danger, Will Robinson! You **never** want to push sensitive information up to Github, especially a public repo! Here's Github's warning:

Warning: Never `git add`, `commit`, or `push` sensitive information to a remote repository. Sensitive information can include, but is not limited to:

- Passwords
- SSH keys
- [AWS access keys](#)
- API keys
- Credit card numbers
- PIN numbers

For more information, see "[Removing sensitive data from a repository.](#)"

help.github.com/en/github/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line

To prevent this, use the **.gitignore file** to exclude specific files from git oversight.

- This is overkill: help.github.com/en/github/using-git/ignoring-files
- Just create the .gitignore file in your folder. Then list files and folders you want ignored, one per line.
- google .gitignore examples

V. Etc

/* blather, mostly */

V.i Your job interview

Imagine the near future...

Mr. Acme (interviewer): Hello, <your name>.

You: Hello, Mr. Acme.

Acme: So, tell me a little about yourself.

You: Well, I love git and Github. Most of my Noctrl school work and projects are available on Github. Want to see? (open your laptop)

Acme: You're hired! (smiles and shakes your elbow)

OK, maybe a little over-dramatic, but still...having your Github repos available when interviewing is like a musician having youtube videos. You are showing: 1) you know cool stuff like git + Github, 2) you do **extra**, and 3) you are an owner.

V.ii Back in the day

Before git and Github, back in the day, source control was simpler but very restrictive:

- Packages/modules had owners. Those people, alone, changed their code. If you needed something, then you requested they do it.
- Files were locked. To make changes, you "checkout" a file, which would lock the file and prohibit other people from changing the code. When your work is done, you "checkin" the file, unlocking it for others.

That was then. Today, the impact of the git + Github combo has been dramatic:

- ❑ It has led to an explosion in **open source software**. Anyone can now contribute to an open source repo. Or start their own. Or fork a new version of some open source project.
- ❑ Development work is more **distributed** than ever. People are able to work at home or away from some central office. Companies can create private repos on

Github which strictly restrict access. Also, git is so efficient that even the largest projects (like Windows, a very interesting [link](#)) are now managed using git.

V.iii Git is mean

First things first...git is a revolution and a revelation. It's transformed how software is developed here in 2020.

Git isn't always very "friendly".

- The user model is awkward for many people. I "checkout"...and git does what?!? And staging...can you explain that again? The git learning curve is pretty steep.
- Do the wrong thing in git, and you can really hurt yourself. There's no undo for many operations. Google "git horror stories"...to find blog posts like this one, "The Git Haters Guide, www.evan.org/Fun/Git/index.htm. (ha)
- Merge is a wonderful idea. Many people, working on the same code, and then you merge. Huzzah! Well, in practical terms, it's often a tremendous and complex headache.

OK, enough already! Git and Github aren't going anywhere, and they're tremendous tools. The guys who run large projects or open source behemoths are able to do amazing things with git and Github. So, it's up to us (and every other coder) to learn it, love it, live it. (ha again)

V.iv Collaboration, branches

Branches are probably the most important feature we've skipped. It allows multiple coders to work on different parts of a repo at one time. When you push a branch, the repo owner decides if/when your branch is merged.

You can google and get tons more information. This is a turbo-summary.

```
# Step 1: set to master branch, pull latest code from Github
$ git checkout master
$ git pull origin master

# Step 2: create a new branch
$ git checkout -b <new-branch-name>
```

```
# Step 3: do your normal workflow of {edit, add, commit}
# Step 4: when you're done, push your branch up to Github
git push origin <new-branch-name>

# Step 5: after branch is merged, back to master, pull to
update
$ git checkout master
$ git pull origin master
```

V.v More on Github

Github offers so much functionality!

- Education pack - Github offers students gobs of free goodies..."to give students free access to the best developer tools in one place so they can learn by doing", education.github.com/pack
- Issues - Track bugs/enhancements/ideas right in Github. Very nice!
- Using a gui - Github desktop is their gui for much of what I have described here, desktop.github.com. There are many git guis out there. And editors like VS Code have git operations built into them. So, the command line is not the only way.
- Github Pages - create and host a website (username.github.io) at Github (free), pages.github.com; you can grab a domain (tiny \$\$\$) and use it at Github Pages as well

V.vi That's a wrap

I'll wrap with a practical question. What's the best organization for your Github account?

The best answer (I think): It's mostly up to you.

In CSC 210, it seems you have two options:

- A repo for each program, or
- One repo for the whole class (so each program is a folder in some csc210 folder)

Since no one is using or relying on your code, you can change your mind. So, try a repo for each program, and then put them all in one folder later if you want.

two cents...yow, bill