Using Git with GitHub

elias julian marko garcia
GitHub.com/ejmg/github-demo

February 6, 2017

Outline

Version control is your friend

FOR ALL THAT IS HOLY, YOU SHOULD ALREADY BE USING VERSION CONTROL

- You never know what might happen to your stuff
- If your laptop was destroyed in the next ten minutes, would you be able to recover by tomorrow?
- More extreme: what if your apartment, dorm, etc., caught fire?

Version control is a no brainer

This equally applies to all other work that you have, whether it is class papers or family photos or code.

- It should be agreeable that you need *something*
- Your code is no exception

Version control is more than just a backup

It allows you to do many things:

1. Track and organize your work
   - 1.1 This means being able to switch between versions and revert regressive changes
     - 1.1.1 You can have a main branch, a development branch, a bleeding edge branch, patch branches, and n^{th} feature branches all simultaneously in existence
   - 1.2 It becomes a (near) one stop hub for planning and organizing your project
     - 1.2.1 Especially so with services like github
     - 1.2.2 This is even for non-code/programming related things, i.e. academic research
2. It allows you to show your work to others
   - 2.1 this is a huge plus for employers
   - 2.2 shows your ability (specialization and breadth of knowledge)
   - 2.3 . . . your commitment (how long have you been programming?)
   - 2.4 . . . your creativity (interesting, funny, useful projects)

Version control is more than just a backup cont.

It allows you to do many things:

1. Track and organize your work
2. It allows you to show your work to others
3. Contribute to FOSS
   - 3.1 Linux's kernal has been on GitHub for years
   - 3.2 Google, Facebook, and etc. put many of their projects online and open for use, contribution
   - 3.3 V.C. has been a key part to this process, before GitHub and especially after
4. Cloud based solutions, like GitHub, ensure you stuff is always safe
   - 4.1 Or, at least reasonably safer than you, a non corporate entity with non-corporate backup mechanisms, can guarantee.

There is no excuse not to use version control

Git is available for all platforms, as is services like GitHub

If you are really concerned about sharing your code for x reason, still no excuses

- GitHub has a student account program that gives you the paid benefits of GitHub for free
  - i.e. private repositories that no one else can see unless you invite them to see it
  - tons of other benefits: free domain hosting, free hosting credits, payment processing waiver, database access, email infrastructure credits, free access to the unreal game engine, build integration tools, and a few more

"I don't need version control"

People can't know I'm a bad programmer if I don't use version control

12:14 PM - 6 Feb 2017

Preface

This is a hands on tutorial

- I am not a git professional, but neither are most programmers
- If you are a project manager, things obviously change
- Follow along if you can with your machine, if not, try to write notes
  - Will go over your head otherwise because that is just how git is

Assumptions

You have the following installed for your system:

1. Git
2. Python 3.5
3. Some kind of text editor (won't cover)
4. Some kind of shell (won't cover)
5. GitHub account (won't cover)

1. Install Git on your machine (Reference)

Linux

Tons of ways to do it, tar method works on all. Use system package manager when possible, just a lot easier.

```
# for ubuntu
sudo apt install git #16.04
sudo apt-get install git #14.04
```

Mac

1. Binary Installer via Git
2. Macports

   ```
   sudo port install git +svn +doc +bash_completion +gitweb
   ```

3. Homebrew

   ```
   brew install git
   ```

1. Install Git on your machine (Reference)

Windows

- Personally, I have zero experience with this. That said, Git says it's easy using the msysGit

2. Install Python on your machine (Reference)

Linux and Mac

- Again, tons of ways
- Personally recommend Pyenv, fantastic Python version and virtualenv manager

Windows

- Hahaha
- Anaconda is a very good package that basically does what Pyenv does
  - Anaconda also works for Linux and Mac

Make a project

cd into a project directory, commonly is `/projects/`

- create a directory `test` and cd into it

  ```
  mkdir test && cd test
  ```

- execute the following to create a git project:

  ```
  git init
  ```

Make a local project cont 2

All Git(hub) projects typically have 2 files regardless of the project

1. README.md (or .org)
2. LICENSE.md (or .org)
   - for those unfamiliar with one or both extensions, .md is the Markdown language and .org is an org-mode file

Let's do a README.md since this is only a test repository

```
echo "#Hello, World! This is a test repository!" > README.md
```

- test if you did it right with `cat README.md`

Now make a test program:

```
echo "print(\"Hello, world!\")" > demo.py
```

Investigate your repository

Let's see what is in the repo:

```
ls -a
```

- you should now see all the files in your repo, including dot files
  - dot files are always configuration files. for git, .gitignore and .git are very important
- `.git` is the entire history and record for all branches, remotes of your repository
- `.gitignore` is a file that conveniently tells git to ignore certain files and directories
  - you don't have one by default but typically will create one in the process of a project
  - never put executables, machine specific configurations, and other garbage files/output on git/GitHub

Put your repository online

Go to github, and click on the "+" button on the top right, next to your user image

- name your repo the name you gave the directory (for consistency and simplicity) and don't put in any other info
- click create and take the https (or ssh if setup) url that is given
- return to terminal and do the following:

```
git remote add origin <url> # adds a remote you named origin
git remote -v # checks remote
```

Basic commands

Try these out:

- `git status`
  - shows the current modified state of your current branch
  - i.e., if you haven't changed anything, it'll say so, vice versa
- `git add .`
  - add all changes shown by previous command. 'git add <file/directory>' will limit to only x chosen
- `git commit -m "commit message"`
  - commit the previously uncommitted logical block of edits to the working branch
- `git push <remote> <branch>`
  - push your local edits, which are all commits, to the branch specified

Basic commands cont 2

- `git branch`
  - shows the existing branches, with an ‘*‘ next the one you currently are on
- `git remote`
  - shows the existing remotes, of which you should have two, origin (your fork master) and upstream (original master)
- `git log`
  - shows log of commits, each with the commit comment and the commit id #
  - it opens using vi by default, enter ‘q‘ to exit
- `git checkout <branch>`
  - switches you to the branch chosen. be careful not to choose a head ref like HEAD, master/origin, etc
- `git revert <revert id>`
  - reverts to previous commit mentioned while preserving git history

Basic commands cont 3

- `git stash <file/directory>`
  - if switching branches with uncommitted work, must stash temporarily.
- `git diff <branch>`
  - will show the diff between your working edits and the branch you chose to compare.
- `git fetch <remote>`
  - updates the remote reference with your local copy of it. Does not change anything.
- `git merge <remote>`
  - joins the commit history of your branch with the remote (combines the relevant code changes)
- `git pull <remote> <branch>`
  - fetches and then merges the changes of the selected remote into the local branch

Fork and Clone a repository!

Go to my demo repository, github-demo

For any repository, you will see a fork button on the top right

- What this is doing is making a branch of the repo that is yours and yours alone
- You can edit it, change it, and modify it and it does not effect the original master branch
  - Your original fork, however, is considered *your* master branch

Clone your fork of the `github-demo` repo

- get the clone url and cd into your project directory
- execute the following and cd into the resulting directory:
  `git clone <your fork url/git here>`

Fork and Clone a repository cont

Add an upstream remote

- go back to the original repo and get the https link from it and do the following in the project directory:

  `git remote add upstream https://github.com/ejmg/github-demo.git`
- We now have an additional `remote` that your local git project can point to
  - Remotes are the repository that you can pull and commit to
  - Think of them as the ‘central hubs‘ of your code. Everyone commits to them and pulls from them

Making and working on a branch

For every edit, change, fix, update, whatever: make a branch for it

- admittedly, not as important when you are the only person working on something
- if you don't do it then, at least you have a commit history, backup, etc
- however, you are missing out on productivity the larger your project is
  - think of every time you had an assignment where you had some fundamental change you had to make in the code base
  - branching your work avoids commenting out huge segments of code just to recomment it in for testing, messing with param args, etc.
  - it *modularizes* your code edits into logical and coherent blocks themselves

Making and working on a branch cont 2

Find something to fix

- look at date method, hello method, main driver

Make a branch to fix it

- name your branch your school email#/something to easily identify you
- normally should be very shorty and descriptive

```
# switch to master branch
git checkout master
# fetches updates from the original repo and
# updates our local copy of it with merge
git pull upstream master
# pushes our updated master (copy of the original, remember)
# to our fork's version to update
git push origin master
# creates a new branch off of our updated master named myedit-identifier
# and checks it out (switches to it)
git checkout -b myedit-identifier
```

Making and working on a branch cont 3

After someone has a edit or two, let's do pull request:

```
# -u tells git that origin is the upstream of the branch
git push -u origin <name of your branch>
```

- what this does is pushes your new branch (and its edits) to your master on your fork.
- switch back to GitHub, congrats, you have a new branch and can make a pull request!
  - This is possible through terminal but is honestly a pain.
- depending on what gets merged when, there could be conflicts.
  - will have to be resolved before merging into the master branch of the original repo

References and Further Reading (no particular order)

- The beginner's guide to contributing to a GitHub project
- Git Commit vs Push
- Pull vs Fetch
- Git-Diff (this is Git's documentation website, highly recommend)
- Magit, super awesome emacs minor mode integration for git