An Introduction to Git and GitHub!

Sierra OBryan (she/her)
Senior Consultant, Atomic Robot

A little bit about me!

THOMAS MORE UNIVERSITY

OHIO STATE

CHASE

5/3

Kroger

P&G

A little bit about me!

Took CS101 as a senior
(and cried about it)

Took computational physics

First job! YAY!
Finally learned git

What are we doing today?

- We’re going to talk about the basics of git
  - How and what of using git
  - Labs using command line and GUI
  - Branching, committing and merging, oh my!
- AND we’re going to talk about the basics of GitHub
  - Creating a repo
  - Features

Git vs. Github??

What the heck ???

VCS:
Version Control System

Version Control System

- Any VCS should have at minimum the following:
  - Complete history of changes for all files
  - Branching and merging by authorized users
  - Traceability
- Does Software Development Require VCS?
  - Strictly speaking – no (but you'll regret it!)

Version Control Systems

Local

The simplest form where a database keeps all the changes to files under revision control. Changes are kept as patch sets and you can then add up all the patches needed to re-create a file at any point in time.

Centralized

A single repository contains all history and each user gets their own working copy of the latest snapshot of the code. You need to commit your changes in the repository and others can only see your changes by updating their working copy.

Example: Subversion

Distributed

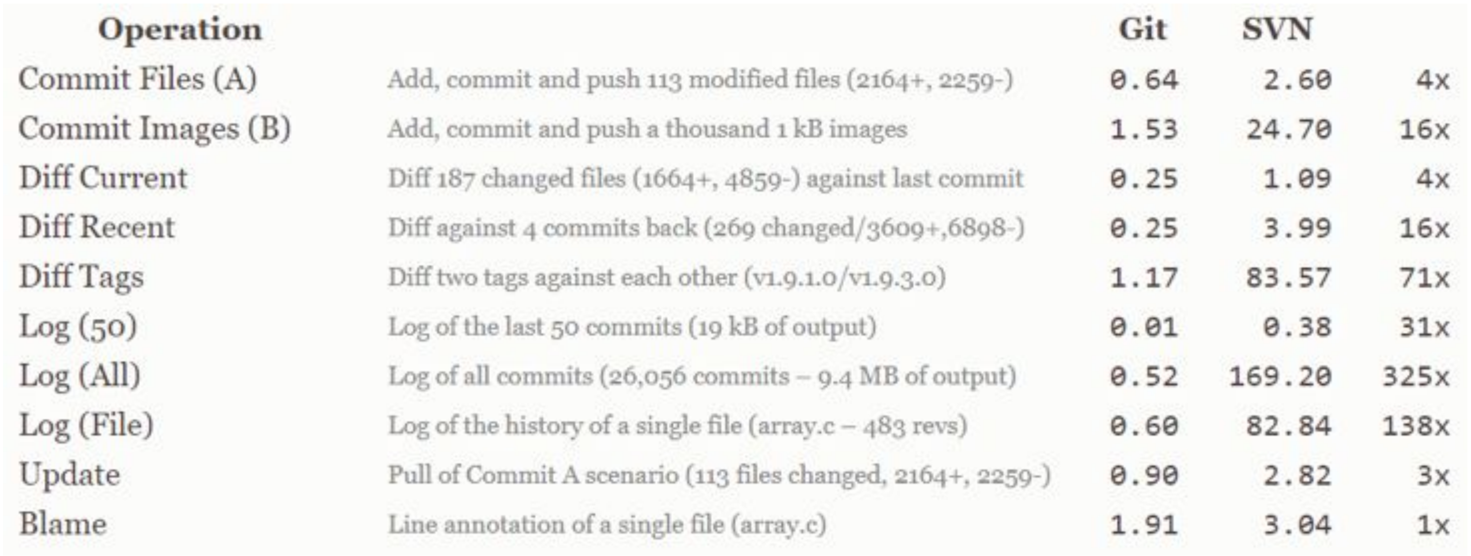
Multiple repositories where each user has their own repository that contains all history in addition to their own working copy. You must commit and push your changes in order to make them visible on the central repository.

Example: Git

Industry Standard!

Why is Git the industry standard?

- Works offline
- Branching is easy
- Merging is easy
- Full history is kept in each repository copy
- It’s fast… really fast!



Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push a thousand 1 kB images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19 kB of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits – 9.4 MB of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c – 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x

Some Useful Terms

Repo(sitory)

A project's folder. A repo contains all of the project files and revision history. It can be either public or private

Checkout

Switches between different versions of the code, and generally is used with branches

Commit

A revision to the repo. Git creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of the the changes

Branch

A parallel version of a repository, generally used to change code in an encapsulated environment

Status

Shows the state of your working directory and helps you see all the files which are untracked by Git, staged or unstaged.

Pull Request

Proposed changes to a repo submitted by a user. It can be accepted or rejected. You are asking to have your changes “pulled into” the repo

Git – Clone vs Fork

Clone

A copy of a repo locally on your computer. When you make a clone, you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online. The repo you cloned is still connected to the remote version so that you can push your local changes to the remote to keep them synced when you're online.

Fork

A personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository. You can also open a pull request in the upstream repository and keep your fork synced with the latest changes since both repositories are still connected.

More Useful Terms

Add

Adds the change(s) in the working directory to the staging area. However the changes are only recorded once they are committed.

Push

To send your committed changes to a remote repository on github. You “push” them to the repo

Fetch

Retrieves the latest meta-data info from the original but doesn’t do any file transferring. Basically checking to see if there are any changes available

Merge

Taking the changes from one branch (in the same repository or from a fork), and applying them into another

Pull

Fetch and download content from a remote repository and update the local repository to match. It’s a combination of two other commands, fetch followed by merge

Lab 1 – Part 1 – Setup git

1. Install Visual Studio Code
2. Download git (if not already done)
3. Open git-bash (MacOS users should use terminal instead)
4. Set config variables:
   a. git config --global user.name "My Name"
   b. git config --global user.email me@example.com

Let's verify it!

This can be verified by

1. git config --list
2. git config -l
3. git config --get user.name (together with)
4. git config --get user.email

Lab 1 - Part 2 - Github

1. Create a repository in github
    a. Repository will be initialized by github
2. Clone repository
3. Create branch
4. Make change
5. Commit change
6. Push
7. Merge branch

Lab 1 - Part 2 - Github

1. Navigate to https://github.com
2. Create an account (if needed)
   a. Complete all fields
   b. Verify email
3. If you have an account, sign in

Github – Create a Repo

1. Click the “Create Repository” button on the left side of the screen
2. In the Repository Name field, enter *yourusername*.github.io
3. Leave the public setting enabled
4. Click Create Repository button
5. Repo created!

Github – Clone your Repo

1. Click the copy button next to the repo name
2. Open your terminal
3. Navigate to a location where you prefer to store your code, or create a new folder
4. Type “git clone the-repo-name-i-just-copied” and enter
5. Change directory to the new repo

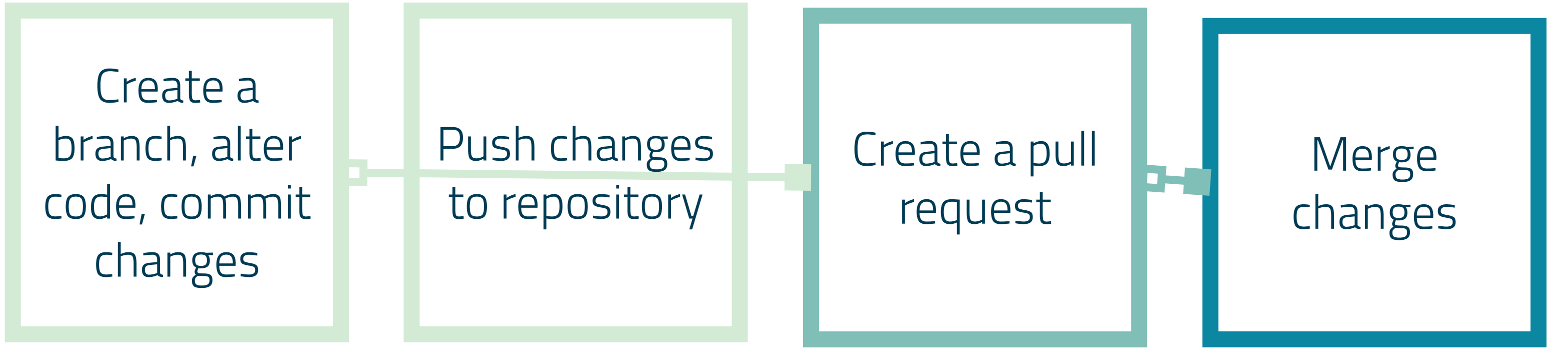
Github – Update your Repo

1. Create a basic html file:
   a. echo "Hello World" > index.html
2. Type “git status” - notice the new untracked file
3. Type “git add .“
4. Type “git status” - notice the file is now labeled as a new file

Github – Commit to Repo

1. Type “git commit -m “initial commit”
2. Type “git status” - nothing to commit, working tree clean
3. Type git push -u origin main
4. Your new file has now been pushed to your repo in github and branch 'main' set up to track remote branch 'main' from 'origin'.
5. Navigate to https://yourusername.github.io.

Let's Practice!


Create a branch, alter code, commit changes
Push changes to repository
Create a pull request
Merge changes

Create a Branch

Navigate to your source code directory

Type ‘git branch’ to verify you are on the main branch

Type ‘git branch my-new-branch-name’

Type ‘git checkout my-new-branch-name’

Make Code Changes and Commit

Open the code in your favorite editor

Make some code changes to your branch - add a photo, change the words, fancy it up!

When done, in your terminal type ‘git status’

Then add the file changes with ‘git add .’

Then verify with ‘git status’

Then commit with ‘git commit -m “my commit message"

Push Changes and Create a Pull Request

Type ‘git push origin my-new-branch-name’

Go to your repo in Github

Click on ‘Pull Requests’

Click on ‘New Pull Request’

Using the compare dropdown, select your branch

Verify the files are as you expect

Click ‘Create Pull Request’

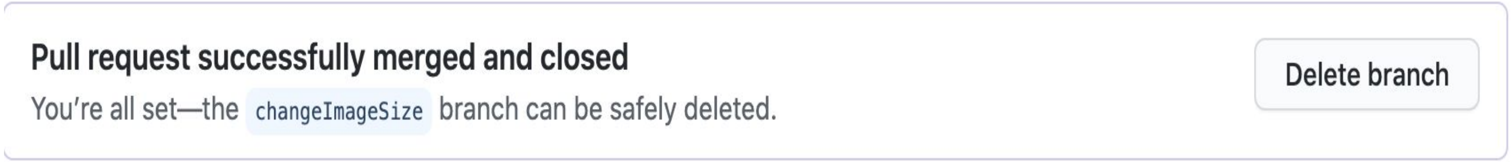
Merge Pull Request

In your repo in Github,click on 'Pull Requests'

Locate the pull request you just created and select it

Click "merge pull request" button, and "confirm merge"

You should see a message indicating that your changes have been successfully merged


Pull request successfully merged and closed

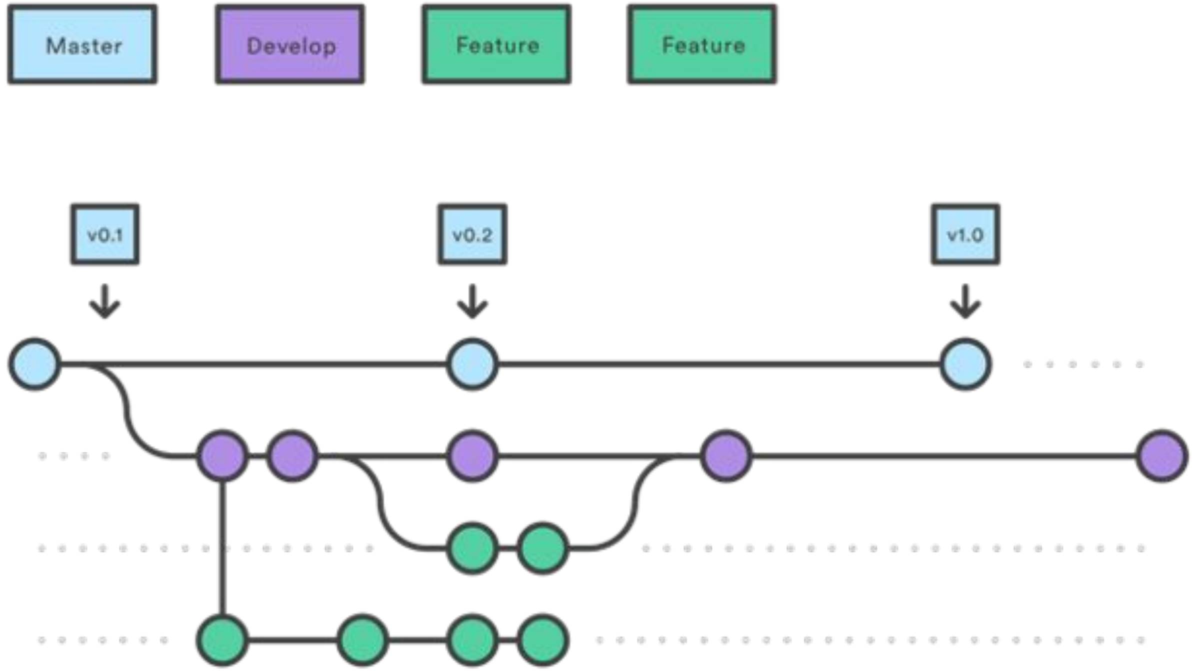
You're all set—the `changeImageSize` branch can be safely deleted.

Command Line isn't for everyone (and isn't the best for everything) ¯_(ツ)_/¯

And that's okay! There are a lot of applications you can download to visualize the process

We'll use GitHub desktop

Gitflow



Conflict is inevitable

- Unless you are working on a project by yourself, it will likely be almost impossible to avoid merge conflicts
- Managing conflicts can be challenging
- Different strategies exist, but our next lab will be the most likely resolution strategy
- Rebasing is another strategy we'll discuss post-lab

Lab 2

1. For this lab, we'll use GitHub Desktop
2. Use the repository we created as our starting point
3. Create a new branch, the name is unimportant
4. Switch back to main
    a. (GitHub Desktop switches to the newly created branch by default)

Lab 2

5. Update the README.md file
6. Commit changes
7. Switch to your new branch
8. Update the README.md file on the same line
9. Commit changes
10. Switch to main branch
11. Attempt to merge your new branch into master
12. Resolve conflict

Other things

1. Rebasing
2. Commit History
3. Pull Requests
4. Settings
5. Additional Resources Found at - http://github.zach-martin.com/resources.html
6. Slides: https://github.com/sierraobryan/intro-to-git

THANKS!

Any questions?

You can find me at
sierra@atomicrobot.com