Comic Credit: Randall Munroe, xkcd.com

Lab 3: Version Control

CS 2112 Fall 2020

September 21 / 23, 2020

Why Version Control?

You're emailing your project back and forth with your partner. An hour before the deadline, you and your partner both find different bugs and work feverishly to correct them. When you try to submit, you find that you have two different versions of the code, and you don't have enough time to figure out who changed what, how to merge them together into one final project, and what, if any, bugs were introduced along the way!

Download Git

If you are comfortable with the command line, Git / Git Bash is a powerful tool that gives you full control over your repository: https://git-scm.com/downloads

If you prefer a graphical interface, GitHub Desktop is an application developed by GitHub to streamline the process of working with their hosted repositories. It is sufficient for almost all use-cases covered in this class: https://desktop.github.com

Configure Git (Terminal)

If you're using git on the terminal, enter the following two commands to set up your name and email:

```
1 git config --global user.name "<Your Name>"
2 git config --global user.email <netid>@cornell.edu
```

Configure GitHub Desktop

If you're using GitHub Desktop, on first launch, choose "Sign in to GitHub Enterprise Server"
Then enter `github.coecis.cornell.edu` as the server and login with your netid (without `@cornell.edu`) and password.
Finally, enter your name and Cornell email on the next screen.

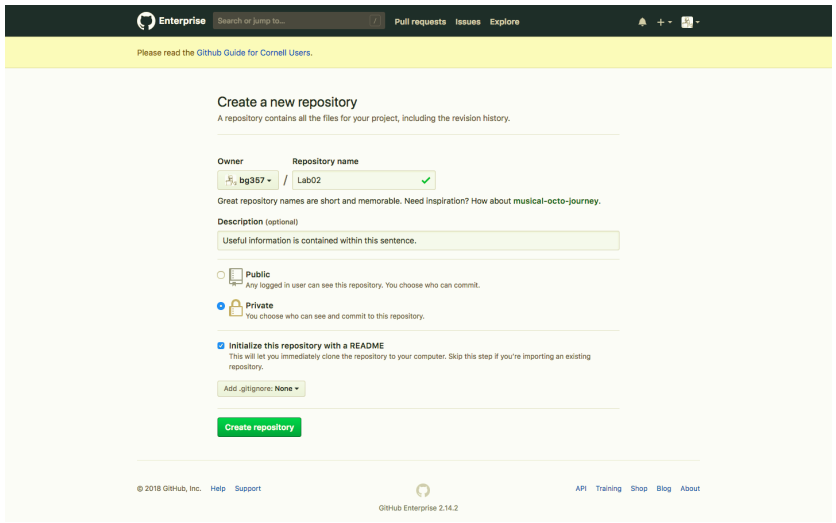
Repository

Git calls a project a "repository".
Go to https://github.coecis.cornell.edu and login with your netid to get started.

Note: for future assignments in this class, we will be assigning you a repository.

Creating the Repository

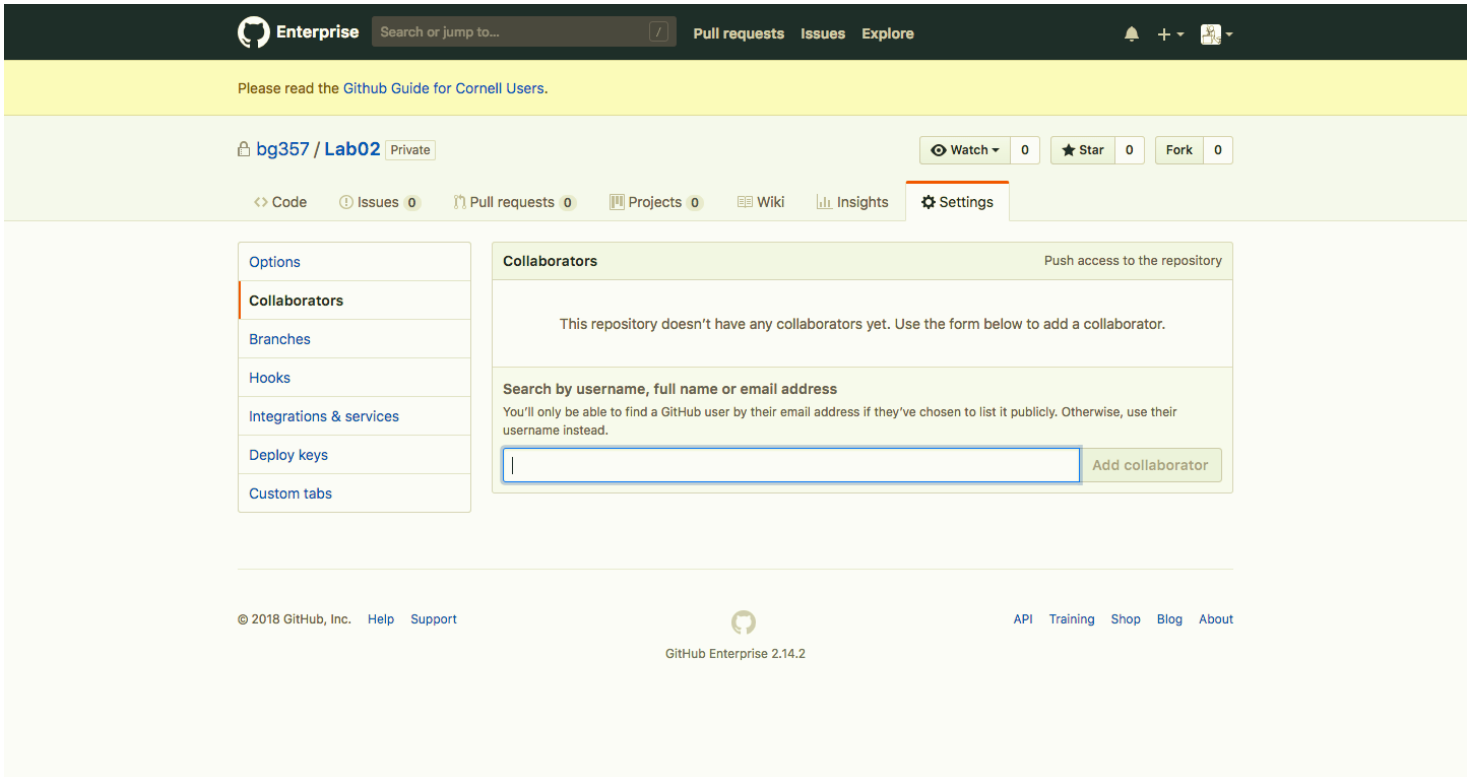
Click the plus sign in the top right to make a new repository. Enter a repository name, choose "Private" for privacy, and check "Initialize with a README."



VERY IMPORTANT: You MUST remember to make your repositories private. Academic integrity is enforced very strictly at Cornell and you do not want to open yourself to potential liability.

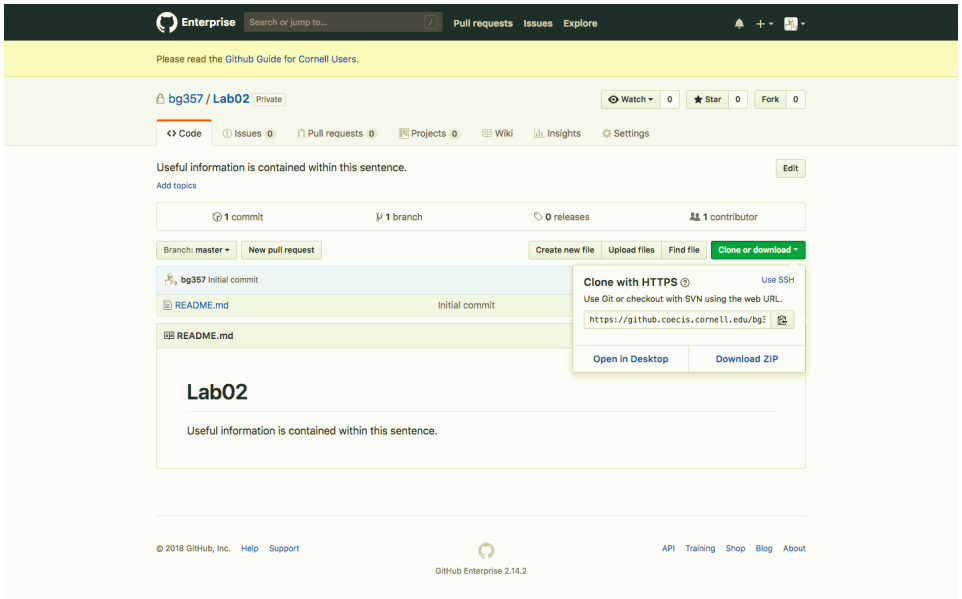
Adding Collaborators

From the repository's main page, choose "Settings" → "Collaborators" and enter their Cornell emails.



Cloning a Repo

From the repository's main page, click the green "Clone or Download" button.
For GitHub Desktop users, choose "Open in Desktop". For terminal users, copy the provided link.



Cloning a Repo

From GitHub Desktop, click the "Choose" button to select where to clone your repo. On the terminal, `cd` to the directory you want and then run `git clone <PASTE>` (where `<PASTE>` is where you paste the link from the last slide).
For class projects, clone into your Eclipse workspace. For this lab, clone to your Desktop.

The Workflow

The three main steps:

1) Commit - Confirm the changes you've made
2) Push - Sync your changes to the server
3) Pull - Sync down changes from the server

1) Commit

To demonstrate, create a file with some text in it.
On terminal, run `git add <filename>` followed by
`git commit -m "<description>"`.
On GitHub Desktop, type the commit description into the Commit Message box and click Commit.

2) Push

Get your changes onto the remote server:
On terminal, run `git push`.
On GitHub Desktop, click the "Push origin" button in the top right.

3) Pull

From the other computer:
On terminal, run `git pull`.
On GitHub Desktop, click the "Fetch" button in the top right twice - once to fetch, again to pull.

Automatic Merge

If two people edit different files or different parts of the same file, the second person to push will fail. Instead, they will first need to pull the changes.
Git will automatically merge the changes from both users by creating a new "Merge Commit."
Then, push again. This time, it should go through.

Merge Conflicts

If two people edit the same parts of the same file, then Git will not be able to automatically merge and you will have a merge conflict. Open the conflicting file and you will see something like this:

```
<<<<<<< HEAD
Your changes
=======
Partner changes
>>>>>>> 123456789
```

Git is marking which parts of the file conflict. Just delete what you don't want, keep what you do, and then run `git commit` again (GitHub Desktop will prompt you automatically).

What is Version Control?

Version control allows multiple people to work on a project simultaneously by keeping versioned copies of each file in your project for each edit that you make. This makes it easy to:

- Revert files back to a previous state if you make a mistake.
- Look over any changes made by you or your collaborators.
- Recover any files if they are lost.
- Merge changes between multiple people's contributions

Types of Version Control

Centralized Version Control
One central repository stored on a server. Users can check out portions of the repository to their local machine for development.
Examples include Subversion and Perforce.

Distributed Version Control
Every user keeps a full copy of the repository. Changes can be pushed from any repo to any other, though usually, a single repository (usually hosted on a server, like GitHub) is designated as the main one.
Examples include Git and Mercurial.

Version Control Software

Several widely used version control systems exist. Some of the most popular free ones are Git, Mercurial, and Subversion. Perforce is sometimes used in industry.
We will use Git, which is freely available, powerful, and rapidly becoming the de facto standard.
Subversion is a bit simpler to use but less powerful. Mercurial is very similar to Git, though less popular. Eclipse has some support for projects that use Git, but it's generally preferred to use separate Git tools.

Git Log

GitHub Desktop shows you a running chronological log of all the commits in your repository and the changes made.
On terminal, you can run `git log` to see the same information.

Revert

You can revert a bad commit to undo its changes.
Note that this is an undo, not a rewind. A revert does not rewind a repository to its previous state. Instead, it creates a new commit that does the opposite of the commit you're reverting.
On GitHub Desktop, right click the commit and choose Revert. On terminal, run `git revert <hash>`. See the next slide to find the hash.

Commit Hash

Each commit is identified by a unique hash (or any unambiguous prefix of the hash).
On terminal, `git log` lists the hash next to the word "commit."
On GitHub Desktop, selecting a commit in the History pane opens it on the right. The hash prefix can be found under the commit title, to the right of the author.

Git Ignore

- Files and folders to not add to version control
  - eg: IDE temporary files (.settings folder)
- .gitignore File
- Goes inside a directory; applies recursively to all subdirectories
- Supports wildcards (*)

The Local Repository

Your local git project contains 3 sections:

- **Working Directory** - This is where you will look at, modify, and add files to your project.
- **Staging Area/Index** - When you add a file (`git add` ⟨`file`⟩ from the command line, or Add to Index in Eclipse), the file is added to the staging area. This is where you prepare the files that you would like to eventually commit to your repository.
- **Repository** - When you commit the changes made to your files (`git commit` by command line, Commit in Eclipse), changes made to the files in your staging area are added to the repository as new versions of those files. You must specify a commit message with each commit to let others know what you have changed.

The Local Repository

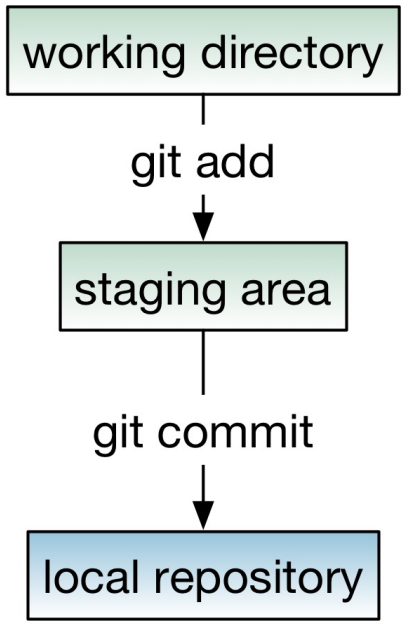
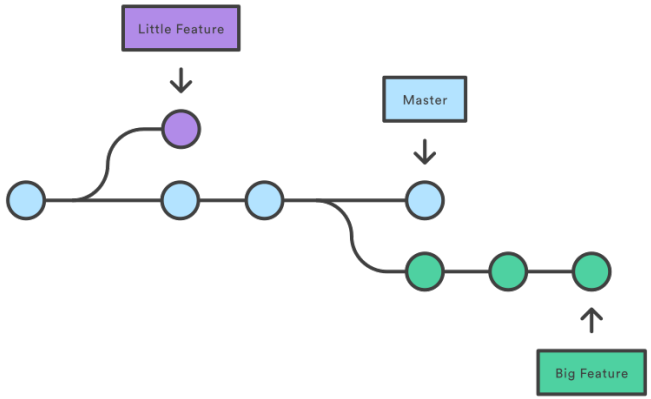


Figure: Local Repository Workflow

On GitHub Desktop, changes are automatically staged by default. You can unstage a change by unchecking the file under Changes.

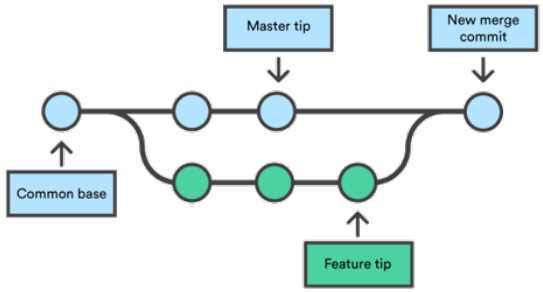
Branches

Git technically maintains your repository as a tree. You can create a new branch to isolate new features or experimental work.
On terminal, create a new branch with `git checkout -b <name>` and move between branches with `git checkout <branch>`.
On GitHub Desktop, click "Current branch" to open the branch menu, and click a branch to switch or choose "New branch" to make a new one.



Merging

You can merge a branch back into another (such as your main master branch).
First, move to the destination branch.
From terminal, run `git merge <branch>`. From Desktop, under History, type the name of the branch and click Merge.



Stash

Temporarily store changes for later.
On terminal, `git stash`.
On Desktop, when trying to change branches, you will be asked if you want to stash.

Blame

Check the last person to edit a particular line in a file.
On terminal, `git blame <file>`.
Alternatively, go to the GitHub website, find your file in the repo, and click the "Blame" button.

Diff

Run `git diff <commit1> <commit2>` to get the difference between two commits. Use `HEAD` to represent the current commit.

```
dhcp-rhodes-252:lab02 benjamingillott$ git diff 09157c44b7cefd41967941d57d453b68df33463e HEAD
diff --git a/README.md b/README.md
index 2bcfca7..4f2ce3f 100644
--- a/README.md
+++ b/README.md
@@ -2,4 +2,4 @@
 Useful information is contained within this sentence.

 This line was written by B.
-THis line added by mx57
+Testing
dhcp-rhodes-252:lab02 benjamingillott$ git diff 09157c44b7cefd41967941d57d453b68df33463e HEAD > git.txt
```

Further Reading

Official documentation: https://www.git-scm.com/docs

BitBucket's illustrated tutorial:
https://www.atlassian.com/git/tutorials

GitHub cheatsheet: https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf

Command Line

The following slides were not shown in lab,
but are provided here as a reference.

The shell: a lower-level interface

A *shell* is a command-line interface to your computer's operating system. Less pretty than the GUI interface, more functional.

- Windows: PowerShell, WSL (Windows Subsystem for Linux), Cygwin (Unix on Windows), or cmd
- Unix (Mac OS X, Linux): sh (bash), (t)csh

Shell commands

A shell command consists of a program name followed by some optiona command-line arguments. Spaces separate the command and the arguments from each other.
Examples:

- "rm" ⟨filename⟩ (Windows: "del" ⟨filename⟩) : remove a file
- "ls" ⟨directory⟩ (Windows: "dir" ⟨directory⟩): list contents of directory
- "echo" ⟨message⟩ : print the message to standard output
- "cat" ⟨filename⟩ (Windows: "type" ⟨filename⟩) : print the contents of the file.
- "cd" ⟨directory⟩ : change the current directory (shell built-in command)

Command context

Every command is executed with some context provided by the operating system:

- The *current directory* in which the command runs. All files accesses are relative to this directory. Note: folders are known as *directories* at the operating system level.
- A set of *environment variables* that can be looked up by the running program.[1]
- Standard input and output devices. Normally standard input reads from the shell's input and output goes to the shell. However, it is possible to override these. The syntax "$>$ ⟨filename⟩" *redirects* output to the specified file.

[1] The ShellShock vulnerability exploits a bug in how the "bash" shell handles environment variables.

Pathnames and directories

- Files and directories are specified by *pathnames*: names separated by slashes (Unix, Cygwin) or backslashes (Windows).
- Pathnames starting with a slash (backslash) are *absolute* and start from the root of the file system.
- The special directory "." means the current directory, and ".." means the parent directory of the current directory.
  - To go up a directory: "cd .."
  - To go to the root directory: "cd /"
  - To list the current directory: "ls .", or just "ls".