Track Changes with Git

Objectives

- Discuss what a Version Control System is
- Describe the purpose of a working directory, staging area, and repository are
- Initialize a git repository with `git init`
- Check the status of changed files in a git repository with `git status`
- Stage new and changed files with `git add`
- Commit staged files to the git repository with `git commit`

Version Control System

Version control is a class of tools that programmers use to manage software projects. It allows you to track changes you make to files on your machine. This is helpful for when you screw things up! And you will. 😉 And that's ok. Version control allows developers to revert back to a specific time and place in your code. Sort of like a reset button.

Version control allows developers to:

- Keep track of changes to files over time
- View previous states of your project
- Return to a previous state of your project
- Manage changes to files from multiple people
- Make changes without worrying about stability
- Keep files together as a group

There are many flavors of VCS:

- Git
- Mercurial
- SVN
- Perforce
- TFS
- etc.

What is Git?

Git is a free and open source software for version control. While there are many different version control systems, git is incredibly popular and powerful. Many companies use git, and if you understand git it will be easy to learn another version control paradigm.

Any files tracked by git typically go through 3 stages:

1. Unstaged

    - These changes will not be committed in the next commit
2. Staged
    - These changes will be committed in the next commit
3. Committed
    - Changes committed in the last commit

What is GitHub?

Git and GitHub are NOT the same thing. Github is a web based service that hosts repositories on a server and allows developers to easily collaborate. Github acts as a remote backup service for git repositories. Once we've **pushed** to a**remote** such as GitHub, we know our code is safe. Even if our hard drives die. And if GitHub goes down, we can still work on our distributed repos offline.

A Metaphor: Git is a Rocketship, Github is Mars

🚀

Space Version	Git Version
Package	Unstaged change
Package on Launchpad	Staged change
Package in Rocketship	Commited change
Launch	Push
Launchpad	Staging area
Rocketship	Git repo
Mars	GitHub

Let's say you want to deliver some packages to Mars with a rocketship. *You want to push changes to GitHub from your git repo*

1. Create some packages. *Make some changes to your files*
2. Choose which packages to place on the launchpad. `git add` *the changed files you want to push*
3. Put the packages on the launchpad into the rocketship. `git commit`
    - Any packages left off the launchpad and not in the rocketship will not be sent to Mars. *Any changes not staged with* `git add` *will not be committed and will not be pushed to GitHub*
4. Repeat the create packages, move to launchpad, and pack rocketship steps for any additional packages you want to send. *Change files,* `git add`*,* `git commit`
5. Set the rocketship coordinates for Mars. `git remote add origin git@github.com:nasa/marooned-astronaut.git`
    - We'll reuse our rocketship, so you only need to do this once per rocket!

6. When the rocketship is sufficiently loaded, we want to launch the rocketship to Mars. `git push -u origin master`
7. Astronaut on Mars will recieve your rocketship and be happy with their new packages. *Check your GitHub repo to make sure the changes were pushed*

Basic Git Commands

There are 4 main commands for `git`

- init
- status
- add
- commit

With these 4 commands you can create a repo and start versioning your project.

`git init`

git manual

Initialize a new git repo in the current directory with:

```
$ git init
```

You can verify git was intialized by checking if a `.git` folder was created with `ls -a`.

`git status`

git manual

When in a git repository, you can type `git status` to see any staged or unstaged changes pending.

In your git repository:

```
$ git status
```

Example Output:

```
$ git status
On branch g15

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
01_vcs.md
02_basic_git.md
03_github.md
04_github_clone.md
05_github_workflow.md
```

`git add`

git manual

If you have any files that are brand new to the repo or have been changed, you can tell git to start tracking it with:

```
$ git add <file>
```

To add all new files and changes in a directory:

```
$ git add .
```

After adding a file or change to the repo, try running `git status` again to make sure it got staged.

`git commit`

git manual

After staging files with `git add`, you can now commit the changes to save the current state of the project as a snapshot in time.

```
$ git commit -m "I fixed all of the bugs. :)"
```

This will create a commit in git that will be a snapshot of what the project currently is.

`git push`

git manual

If you are using GitHub or collaborating with another git repo, you can push any new commits to your default remote with:

```
$ git push
```

If you have your remote pointing to GitHub, you should now be able to see any changes on your GitHub page.

Resources

- VCS on Wikipedia

- Git SCM Manual
- Pro Git book
- Tower Learn Version Control with Git book
- Try Git