LIGHTHOUSE LABS

Git Workflow Cheatsheet

Start your Git repo

On your local machine (vagrant box):

```
mkdir myproject
cd myproject
git init
touch README.md
git add README.md
git commit -m "Initial commit"
```

GitHub actions

Log into GitHub
On the right-hand side, click the New repository button
Name your Repository
Click 'Create Repository'
On the next page, click on the button next to the SSH Clone URL to copy to clipboard

Connect local to remote

```
git remote add origin <paste SSH Clone URL>
git remote -v
git push -u origin master
```

Make a branch

```
git checkout -b mynewbranch
```

Merging branches

```
git checkout master
git merge --no-ff mynewbranch
git branch -d mynewbranch
git push origin master
```

LIGHTHOUSE LABS

Git Glossary

`mkdir myproject`	Creates a new directory on your vagrant box to put all the files into for your project. Replace ‘myproject’ with a simple but descriptive name for your project.
`cd myproject`	Change directory to your new project
`git init`	Initialize that new directory to be a git repository.
`touch README.md`	Create an empty file in that directory named README.md * It is recommended to go back and write up the description of your project here
`git add README.md`	Add this file into your git repository. Git will begin tracking changes to this file
`git commit -m “Initial commit”`	Commit the changes you have made to your repository into git’s system, and add the comment that it is your initial commit. * ALL commits require a message/comment
`git remote add origin` **`url`**	Register with your local git repository that you will be distributing all changes to a remote repository
`git remote -v`	Verify that your local git repository can speak/sync with the remote one
`git push -u origin master`	Push all changes to your master branch up to your remote repository. * The -u flag means that git will use this remote/local combination when you type ‘git push’ going forward (until you specify a new combination)

`git checkout -b branchname`	This creates a new branch and switches to it, in one command. This is equivalent to typing:
	`git branch branchname`
	`git checkout branchname`

`git checkout master`	Switch back to master branch

`git merge --no-ff branch`	Merge all of the differences between branch and master into the master branch. This brings all the work you have done in your branch into the master branch * The --no-ff flag tells git to maintain the full record of al the changes you made, including creating a new branch and all the work that went into that branch
`git branch -d branchname`	This deletes the branch. The record of the branch is there, it is no longer an active branch that can be checked out and committed to.
`git push origin master`	Push all the changes you have made to your code by merging your branch to the remote repository
`git pull`	Retrieve a copy of all changes that have been pushed to your remote repository **AND** merge them into your code.
`git clone` **`<url>`**	Clone an existing repository from a remote source * By default, this creates a new directory with the name of the repository, taken from the .git file in the URL
`git clone` **`<url> <dir>`**	Clone an existing repository from a remote source **AND** specify the directory into which it should go
`git diff`	Show differences between your working copy and the index
`git diff --cached`	Show differences between your working copy and the last commit

The following are git commands that are useful, but may not be used in your everyday workflow:

`git fetch`	Retrieve a copy of all changes that have been pushed to your remote repository. Does **NOT** merge into your code.
`git tag -a 1.0`	Done after a merge, for the purposes of ‘tagging’ a milestone, such as a new version release. **OPTIONAL**

LIGHTHOUSE LABS

Key Git Tips

- Commit often! Make ‘atomic’ changes. Each time you add a file, or make significant changes to a file, commit your work. This gives you a log of all the changes you make, which helps to not only track where any bugs were created but also document your development process.

- Use descriptive messages when committing. Don’t have messages that say “added code” or “blah” or something equally non-descriptive. Commit messages should document the changes you have made since your last commit. The more atomic your commits are, the easier your commit messages are to write.

  Example: “Added create method to Contact class, no error checking yet.”

- You can branch off of branches. Here is a common workflow:

  ```
  git checkout -b develop
  git checkout -b myFeature develop
  <do work>
  git commit -am “Implemented new feature myFeature”
  git checkout develop
  git merge --no-ff myFeature
  ```

  This allows you to have a develop branch separate from your master, and then each feature you add to your app/program gets its own branch.

- There are many ways to manage git branches, stashes, tags, and repositories. This is a minimum workflow. Find the workflow that works for you, or adopt one outlined by someone you respect. Some companies will enforce a particular workflow or standard for how commits, branches, and merges are handled. Be flexible.

- Remember that git is a tool to help you be a better developer, not a draconian taskmaster whose whims you must fulfill. Your tools should always be approached as a partnership, not a negotiation.