
KodeGo
Git and Github

What is Git?

- Git is a version control system (VCS) designed to make it easier to have multiple versions of a code base, sometime across multiple developers or teams.
- It allows you to see changes you make to your code and easily revert them.
- Important note: **Git != Github**

What is Github?

- **Github.com** is a website that hosts git repositories on a remote server.
- Hosting repositories on Github facilitates the sharing of codebases among teams by providing a GUI to easily fork or clone repos to a local machine.
- By pushing your repositories to Github, you will pretty much automatically create your own developer portfolio as well!

Confirm that you have git

- Open terminal and run ‘git’
- If you see an error with ‘Command not recognized’, you probably haven’t installed git or there were errors upon installation.

Where to download git?

Just click this link to go download Git - Git (git-scm.com)

Initial configuration of git

$ git config –global user.name - checks information of current user

$ git config –global user.email - checks information of current email address

$ git config –global user.name “Jose Rizal” - setting current user information

$ git config –global user.email “jrizal@sheeesh.com” - setting current email information

KodeGo

Using Git / Github

Connecting git with Github

- In order to prevent having to enter your password each time you push up to Github, you must configure git and Github to recognize Secured Shell (SSH) keys that you generate.
- To check and see if you have any recognized SSH keys active on Github, go to https://github.com/settings/keys
- If you do not see any SSH keys listed, you do not have SSH configured with Github.
- If using Mac OS X, you can also configure your keychain to automatically enter your password via HTTPS.

Connecting git with Github

- From your project directory, run ` git init ` to initialize a git repository.
- Go to Github, and create a new repository with the name of your project.
- Follow the instructions on Github to connect your initialized git repository to the remote server on Github.

Basic git / Github workflow

- From your project repo on Github, navigate to the Issues tab and create a new issue.
- From the command line, use git to create a new branch off of master to make your edits to. To tie the branch to your issue on Github, make sure to include the issue number after the branch name, e.g. branchName # 1
- Stage edits to be committed to your git repository by using ` git add <filename> ` to track the files that you want to add directly or ` git add . ` to add all files at the current directory level that you have worked on.
- Commit changes using ` git commit -m <message> `and be sure to leave a short but descriptive message detailing what the commit will change when merged to the master branch.
- Push changes to save them by using ` git push origin <branch name> `

Basic git / Github workflow

- On Github, create a new pull request for the branch that you have just pushed, and add any clarifying comments that you deem necessary.
- In order to close the issue associated with the pull request, tell Github to do so by adding ` closes <issue number> ` in the comments. (keywords like ‘closed’,‘fix’,‘fixes’,‘resolve’,‘resolves’, and ‘resolved’ also work).
- Github will automatically check for merge conflicts. If there are any, check to see them and resolve them.
- Once everything is up to date, Github will allow you to merge using three options: merge; squash and merge; rebase and merge.

Git branching

- ` git branch ` to view current branches in repo.
- ` git checkout -b <branch name> ` to create a new branch with branch name.
- ` git checkout <branch name> ` without ` -b ` flag to switch between existing branches.

Merging vs Rebasing

- From a conceptual standpoint, git merge and git rebase are used to achieve the same ultimate goal: to integrate changes from one branch to another branch. There are, however, distinct mechanics to both methods.

git merge: pros

- Generally, the easiest option to merge your master branch into your current working feature branch.
- You can ` git checkout feature ` and then ` git merge master ` or your could just do it with one command: ` git merge feature master `.
- Doing this, you create a new merge commit in your feature branch, which is a non-destructive operation that ties the histories of both branches. This preserves the exact history of your project.

KodeGo

Thank you!