**ConsigliereRobertGitTutorial-05-31-2019**

**Part 3:**

**Question:**

What is GitHub? When was it created? Why? By who? What similar platforms exist? Why would you use such a platform?

**Answer:**

GitHub is a Git based repository service that also provides a web based GUI which manages project source code, keeps track of project history, and also facilitates effective collaboration for projects.

GitHub was created in 2005 by Linus Torvalds in response to the break down of the relationship between the Linux developer community and BitKeeper (DVCS provider to Linux). Originally, BitKeeper’s product was free-of-charge to the Linux developer community, but this was revoked as a result of the deterioration of their relationship. Linus began developing a tool to replace BitKeeper’s DVCS which eventually evolved and matured into what it is today.

Some comparable platforms include Bitbucket, Beanstalk, SourceForge, and AWS CodeCommit.

The benefits to using a Distributed Version Control System, or DVCS, is that project files/repository site and full history are fully mirrored which means if any server malfunctions/dies, any of the client repositories can be copied up to the server to effectively restore that version. Each clone can be considered a full backup of project data. DVCS tools also allow for simultaneous and non-linear collaboration between different groups of people for the same exact project. Some other benefits include providing an environment for documentation, showcasing your projects, and tracking changes.

**Part 4:**

**Topic:**

Learn Git Branching tutorial.

**Progress/Work/Notes:**

Git Commit:

A commit is a recording of a snapshot of the files in your local directory.

*$ git commit*

Git Branches:

Branches are lightweight pointers to a specific commit. A single branch instance says “I want to include the work of this commit and all parent commits.”.

*$ git branch newImage*

// this creates a reference to a commit (newImage is the branch referring to the commit, c1).

*$ git checkout newImage; git commit*

*$ git checkout -b [yourbranchname] // shortcut to create new branch & jump to it*

// this allows us to “jump” from one branch to another branch.

// changes will be recorded on the new branch, newImage.

Branches & Merging:

Merging branches integrates two independent branches into a single branch.

*$ git merge [branchname]*

Git Rebase:

Rebasing is a second way of combining work; takes a set of commits, copies them, and puts/”bases” them down somewhere else.

This essentially creates the appearance of sequential feature development rather than the separate features being created in parallel.

*$ git rebase [otherbranchname] // combines ‘otherbranchname’ with checked out branch name*

Moving Around in Git:

HEAD is the symbolic name for the currently checked out commit. HEAD points to a branch name (any changes to this branch will be visible through HEAD).

*$ git checkout C1; git checkout master; git commit; git checkout C2*

HEAD is underneath usually underneath master.

Detaching HEAD means attaching it to a commit instead of a branch (detach from master and attach to C1)

*$ git checkout C1*

Relative Refs:

*$ git log // allows you to see the hashes for each commit*

Relative refs are shortened/related to the actual hash strings.

* ‘^’ move upwards one commit at a time.
* ‘~<num>’ move upwards a number of times.

*$ git checkout master^ // first parent of the master*

*$ git checkout master^^ // grandparent of the master*

*$ git checkout C3; git checkout HEAD^; git checkout HEAD\* // continue moving up the Git tree*

Relative Refs #2:

*$ git checkout HEAD~4 // checks out and moves HEAD to the 4th gen parents of current HEAD*

Branch forcing is when you move (by force) the master branch to another branch in the tree.

*$ git branch -f master HEAD~3 // moves master to 3 parents behind HEAD*

Reversing Branches in Git:

There are two ways to reverse changes in Git (git reset and git revert).

*$ git reset HEAD~1 // moves a branch ref backwards in time to an older commit (rewriting history)*

*// good for local branches (doesn’t work for remote branches)*

*$ git revert HEAD // reverse changes and SHARE those changes with others*

*// pushes C2 up and creates a new branch/commit called C2’*

*// C2’ reverses the commit of C2*

Cherry-Pick Intro:

*$ git cherry-pick <Commit1> <Commit2> <…> // copy a series of commits below current location (HEAD).*

*$ git cherry-pick C2 C4 // copy C2 and C4 and plop them below HEAD*

Git Interactive Rebase:

Rebasing is good for when you don’t know which commits you want to copy/cherry-pick.

*$ git rebase -i HEAD~4 // opens up interactive rebase window allowing you to reorder/omit*

Locally Stacked Commits/Grabbing Just One Commit:

Completed.

Juggling Commits:

*$ git rebase -i // re-order commits so the one we want to change is on top*

*$ git commit –amend // make slight modification*

*$ git rebase -i // re-order commits back to original order*

Juggling Commits 2:

Juggling Commits EXCEPT only use git cherry-pick (not git rebase -i)

Git Tags:

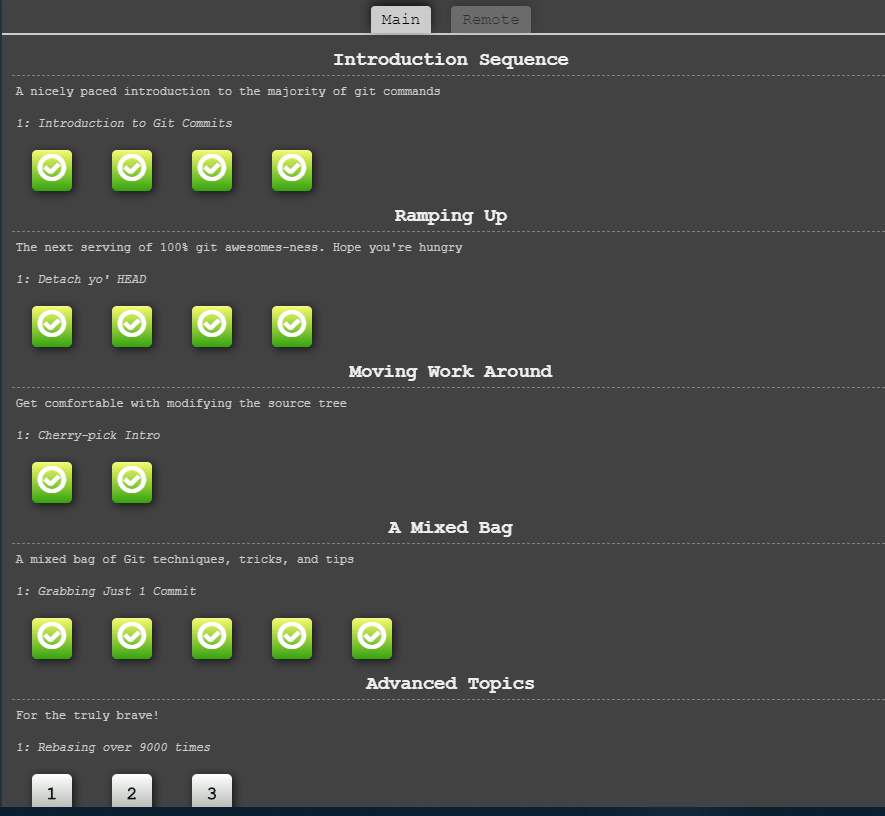
Tags exist as anchors in the commit tree that designate certain spots (they DON’T move as more commits are created).

*$ git tag v1 C1 // make tag at C1 and call it ‘v1’*

Git Describe:

*$ git describe <ref> // DESCRIBES where you are relative to the closest anchor/tag*

Output = anchor name – number of commits away - hash



**Part 5:**

**Define:**

* Repository
  + A file location where you store all the files related to your project. Local is on your system, and remote is outside your system.
* Commit
  + A lightweight snapshot of all files in your local directory.
* Push
  + Used to upload local repository content to a remote repository location.
* Branch
  + A pointer/reference to a specific commit.
* Fork
  + Creates a copy of a project so that you can make changes without affecting the original project.
* Merge
  + Combines/merges two independent branches into a single branch.
* Clone
  + Creates a copy of a target repository.
* Pull
  + Fetches and then downloads files/data from a remote repository – updates local repository with that content.
* Pull request
  + Tells others about all of the changes that you’ve pushed to a specific branch in a Git repository.

**Part 7:**

**Task:**

Modify README.md file within the repository located at:

<https://github.com/paceuniversity/courses>

**Steps:**

1. $ git clone <https://github.com/paceuniversity/courses.git>
2. $ cd ~/Desktop/github/courses
3. $ echo Consigliere, Robert June 1, 2019 3:11pm >> README.md
4. $ git status
5. $ git add README.md
6. $ git commit -m "Modified README.md which should now display my name, date, and time"
7. $ git status

Modify README.md file within the repository located at:

<https://github.com/paceuniversity/courses>

**Steps:**

1. $ git clone <https://github.com/paceuniversity/courses.git>
2. $ cd ~/Desktop/github/courses
3. $ git checkout -b revert-39-patch-2
4. $ echo Consigliere, Robert June 1, 2019 3:11pm >> README.md
5. $ git status
6. $ git add README.md
7. $ git commit -m "Modified README.md which should now display my name, date, and time"
8. git push -u origin revert-39-patch-2