Git Notes

Generalizations

You have now been introduced to the fundamental Git workflow. You learned a lot! Let's take a moment to generalize:

* Git is the industry-standard version control system for web developers
* Use Git commands to help keep track of changes made to a project:
  + git init creates a new Git repository
  + git status inspects the contents of the working directory and staging area
  + git add adds files from the working directory to the staging area
    - Doing two files at once git add filename\_1 filename\_2
    - Adding all of the files in the directory at once git add .
  + git diff shows the difference between the working directory and the staging area
  + git commit permanently stores file changes from the staging area in the repository
  + git log shows a list of all previous commits

generalizations

Congratulations! You've learned three different ways to backtrack in Git. You can use these skills to undo changes made to your Git project.

Let's take a moment to review the new commands:

* git checkout HEAD filename: Discards changes in the working directory.
  + This actually takes your changes off the file – PRETTY SWEET
  + Shortcut – git checkout -- filename
* git reset HEAD filename: Unstages file changes in the staging area.
* git reset SHA: Can be used to reset to a previous commit in your commit history.
  + You have to type the first 7 characters of that 40 character GIT Name

generalizations

Let's take a moment to review the main concepts and commands from the lesson before moving on.

* Git *branching* allows users to experiment with different versions of a project by checking out separate *branches* to work on.

The following commands are useful in the Git branch workflow.

* git branch: Lists all a Git project's branches.
* git branch branch\_name: Creates a new branch.
* git checkout branch\_name: Used to switch from one branch to another.
* git merge branch\_name: Used to join file changes from one branch to another.
* git branch -d branch\_name: Deletes the branch specified.

**GIT TEAMWORK**

git clone

Sally has created the remote repository, **science-quizzes** in the directory **curriculum**, which teachers on the school's shared network have access to. In order to get your own replica of **science-quizzes**, you'll need to *clone* it with:

git clone remote\_location clone\_name

In this command:

* remote\_location tells Git where to go to find the remote. This could be a web address, or a filepath, such as:

/Users/teachers/Documents/some-remote

* clone\_name is the name you give to the directory in which Git will clone the repository.

**GIT TEAMWORK**

generalizations

Congratulations, you now know enough to start collaborating on Git projects! Let's review.

* A *remote* is a Git repository that lives *outside* your Git project folder. Remotes can live on the web, on a shared network or even in a separate folder on your local computer.
* The *Git Collaborative Workflow* are steps that enable smooth project development when multiple collaborators are working on the same Git project.

We also learned the following commands

* git clone: Creates a local copy of a remote.
* git remote -v: Lists a Git project's remotes.
* git fetch: Fetches work from the remote into the local copy.
* git merge origin/master: Merges origin/master into your local branch.
* git push origin <branch\_name>: Pushes a local branch to the origin remote.

Git projects are usually managed on Github, a website that hosts Git projects for millions of users. With Github you can access your projects from anywhere in the world by using the basic workflow you learned here.