**Basic Git Workflow**

**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
  + 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

**How to Backtrack**

git reset review

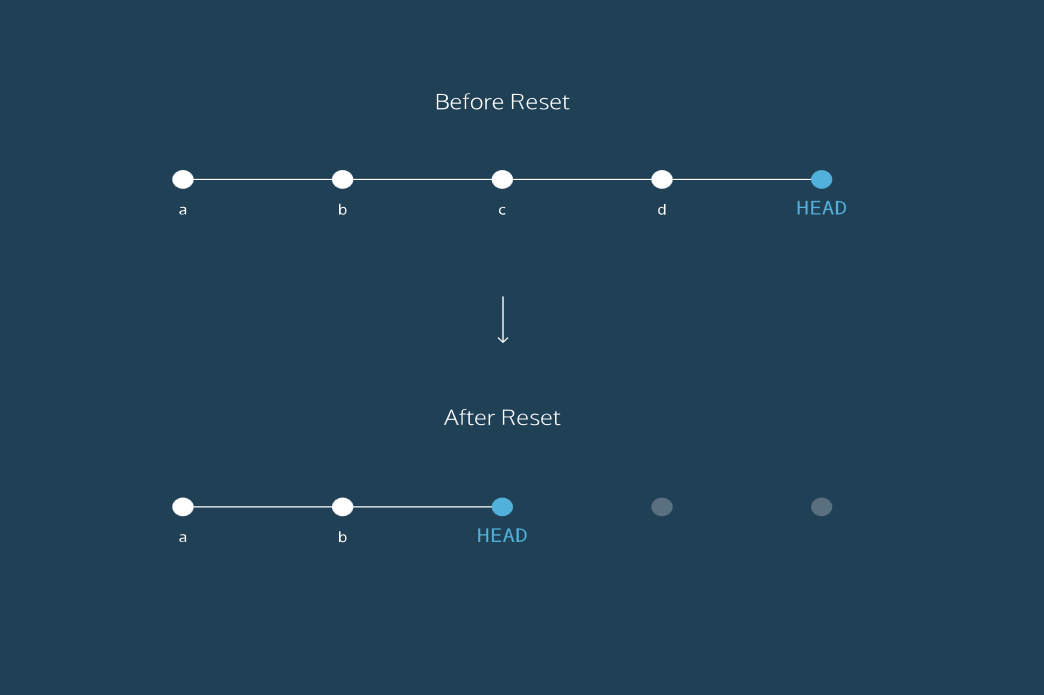
To better understand git reset commit\_SHA, notice the diagram on the right. Each circle represents a commit.

Before reset:

* HEAD is at the *most recent commit*

After resetting:

* HEAD goes to a *previously made commit* of your choice
* The gray commits are no longer part of your project
* You have in essence *rewinded* the project's history



**How to Backtrack**

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.
* 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.

Additionally, you learned a way to add multiple files to the staging area with a single command:

**git add filename\_1 filename\_2**

**Git Branching**

**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

**Overview**

So far, we've learned how to work on Git as a single user. Git offers a suite of collaboration tools to make working with others on a project easier.

Imagine that you're a science teacher, developing some quizzes with Sally, another teacher in the school. You are using Git to manage the project.

In order to collaborate, you and Sally need:

* A complete replica of the project on your own computers
* A way to keep track of and review each other's work
* Access to a definitive project version

You can accomplish all of this by using *remotes*. A remote is a shared Git repository that allows multiple collaborators to work on the same Git project from different locations. Collaborators work on the project independently, and merge changes together when they are ready to do so.

**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.