**Backtracking Intro**

When working on a Git project, sometimes we make changes that we want to get rid of. Git offers a few eraser-like features that allow us to undo mistakes during project creation. In this lesson, we'll learn some of these features.

To start out, let's review the basic Git workflow.

**head commit**

In Git, the commit you are currently on is known as the HEAD commit. In many cases, the most recently made commit is the HEAD commit.

To see the HEAD commit, enter:

git show HEAD

The output of this command will display everything the [git log command](https://www.codecademy.com/en/courses/learn-git/lessons/git-workflow/exercises/git-log" \t "_blank) displays for the HEAD commit, plus all the file changes that were committed.

# git checkout

What if you decide to change the ghost's line in the working directory, but then decide you wanted to discard that change?

You could rewrite the line how it was originally, but what if you forgot the exact wording? The command

git checkout HEAD filename

will restore the file in your working directory to look exactly as it did when you last made a commit.

Here, filename again is the actual name of the file. If the file is named **changes.txt**, the command would be

git checkout HEAD changes.txt

# more git add

The **hamlet** repository we are working on contains five files. In Git, it's common to change many files, add those files to the staging area, and commit them to a repository in a single commit.

For example, say you want to change the character "LARRY" to "LAERTES" in the script. The name currently appears in two files. After you change the name in both files, you could add the changed files to the staging area with:

git add filename\_1 filename\_2

Note the word filename above refers to the name of the file you are adding to the staging area, such as **scene-3.txt**.

# git reset I

Great! The files you've added to the staging area belong in the same commit.

What if, before you commit, you accidentally delete an important line from **scene-2.txt**? Unthinkingly, you add **scene-2.txt** to the staging area. The file change is unrelated to the Larry/Laertes swap and you don't want to include it in the commit.

We can unstage that file from the staging area using

git reset HEAD filename

This command resets the file in the staging area to be the same as the HEAD commit. It does not discard file changes from the working directory, it just removes them from the staging area.

# git reset II

Creating a project is like hiking in a forest. Sometimes you take a wrong turn and find yourself lost.

Just like retracing your steps on that hike, Git enables you to rewind to the part before you made the wrong turn. You can do this with:

git reset commit\_SHA

This command works by using the first 7 characters of the SHA of a previous commit. For example, if the SHA of the previous commit is 5d692065cf51a2f50ea8e7b19b5a7ae512f633ba, use:

git reset 5d69206

HEAD is now set to that previous commit.

# 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 rewound the project's history

**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 commit\_SHA: Resets 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