**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.

**Instructions**

**1.**

You are in a Git project titled **hamlet-prince-of-denmark**. In the code editor, you’ll be working on **scene-5.txt**. Here, Hamlet encounters the ghost of his father. Add this text to the file:

Ghost:

My hour is almost come,

When I to sulphurous and tormenting flames

Must render up myself.

**2.**

From the terminal, add **scene-5.txt** to the staging area.

[Here’s a hint](https://www.codecademy.com/en/courses/learn-git/lessons/git-workflow/exercises/git-add?action=lesson_resume) on how to do it.

**3.**

Commit the changes to the repository with a good commit message.

[Here’s a hint](https://www.codecademy.com/en/courses/learn-git/lessons/git-workflow/exercises/git-commit?action=lesson_resume) on how to do it.

**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) displays for the HEAD commit, plus all the file changes that were committed.

**Instructions**

**1.**

Enter the command to show the HEAD commit.

Notice the output. The ghost’s most recently added line is in green text.

**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

**Instructions**

**1.**

Change the ghost’s words in some way. Here’s a fun suggestion:

Ghost:

My hour is almost come,

When I to sulphurous and tormenting balloons

Must render up myself.

**2.**

From the terminal, use git diff to see the difference between **scene-5.txt** as it appears in the working directory vs. how it appears in your last commit.

You may need to press q on your keyboard to restore the terminal.

**3.**

Use the new Git command to restore the file in your working directory to look as it did when you last made a commit.

Notice that the changes you made to the ghost’s line have been discarded.

**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**.

**Instructions**

**1.**

The code editor is open to **scene-3.txt** and **scene-7.txt**. In **scene-3.txt**, everywhere you see the name “LARRY” change it to “LAERTES.”

**2.**

Now change all instances of “LARRY” to “LAERTES” in **scene-7.txt**.

**3.**

Add the files to the staging area together using a single git command.

# 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.

**Instructions**

**1.**

To try out the new command, let’s make a mistake on purpose!

The code editor is open to **scene-2.txt**. Delete any line from the file and click Run.

**2.**

From the terminal, add **scene-2.txt** to the Git staging area.

**3.**

Now check the status of the Git project.

In the output, notice **scene-2.txt** under “Changes to be committed”.

**4.**

Use the new Git command to unstage **scene-2.txt** from the staging area.

Notice in the output, “Unstaged changes after reset”:

M scene-2.txt

* M is short for “modification”

**5.**

Now that changes made to **scene-2.txt** have been booted out of the staging area, you’re ready to commit. From the terminal, make a commit to save the Larry/Laertes name swap in **hamlet**.

# 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.

**Instructions**

**1.**

From the terminal, print out your Git commit log.

**Note**: If your cursor gets stuck in “git log” mode in the terminal, press “q” on your keyboard to escape.

**2.**

From the terminal, enter the command to reset to a previous commit, using the first 7 characters of one of the past commit SHAs in your Git log.

Next, print the Git commit log again.

Notice anything interesting? The commits that came after the one you reset to are gone. The HEAD commit has been reassigned. You just changed history.

# git reset review

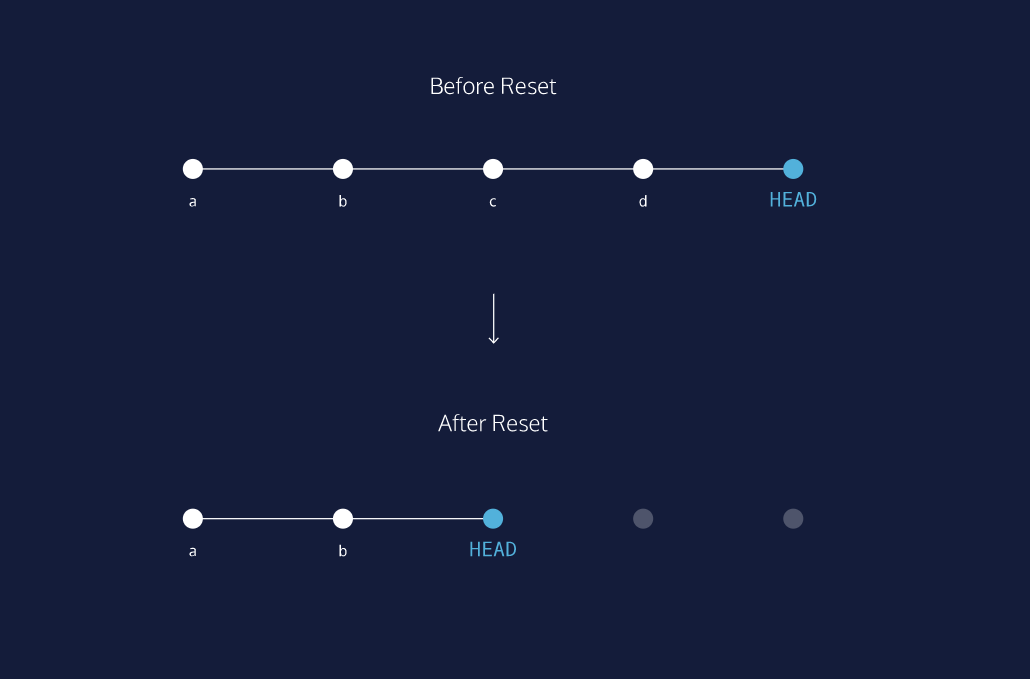
To better understand git reset commit\_SHA, notice the diagram on the right. Each circle represents a commit.

**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