Week 4 : MACHINE LEARNING OPERATIONS

Week 4
Undoing Changes

Week 4 Undoing Changes

- We’ve explored how to create repositories, how to push and pull code from GitHub, how to use commits with Git, and how to work with branches.
- But what happens when we make a mistake or wish to undo an action?
- Or what if we wish to explore some historical commits?

Week 4 Undoing Changes

- Today we're focused on how to undo actions related to git commands and explore historical commits.
- We'll discuss:
    - **git checkout** and **Detached HEAD**
    - **git restore**
    - **git reset**
    - **git revert**

Week 4 Undoing Changes

- Keep in mind that you don’t use these commands as often as the other commands we’ve learned so far, but they are still important actions to know!

Let’s get started!

Week 4
Git Checkout

Week 4 Undoing Changes

- **git checkout**
  - This is actually a very versatile command, so versatile in fact, that developers complained it was used for too many different actions, thus new git commands were created, such as **git switch**.

Week 4 Undoing Changes

- **git checkout**
  - A "checkout" is the act of switching between different versions of a target entity.
  - The **git checkout** command can operate on three distinct entities: files, commits, and branches.

Week 4 Undoing Changes

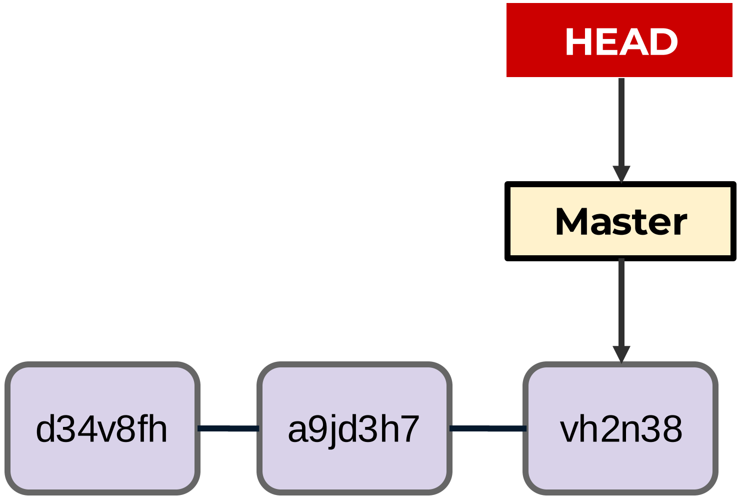
- **git checkout**
  - For example, you could use **git checkout branch_name** instead of **git switch branch_name** to checkout a new branch.
  - Unlike **git switch** however, recall checkout can operate on commits, meaning we can “checkout” historical commits.

Week 4 Undoing Changes

- **git checkout**
  - We can check out a particular commit by specifying its hash, we can get hashes from the **git log** command and we can also see the abbreviated hash using:
    - **git log --oneline**
  - Then we can provide the has as:
    - **git checkout ########**

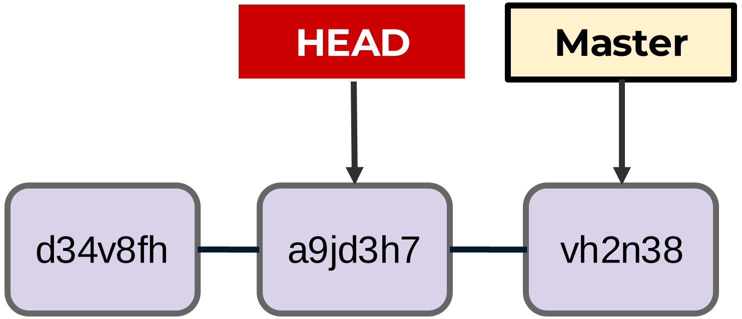
Week 4 Undoing Changes

- Typically our HEAD points to the branch which points to the latest commit.



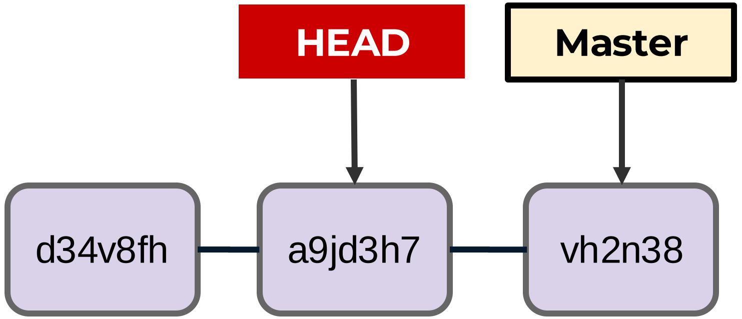
Week 4 Undoing Changes

- Upon calling **git checkout a9jd3h7** we detach the HEAD to a previous commit



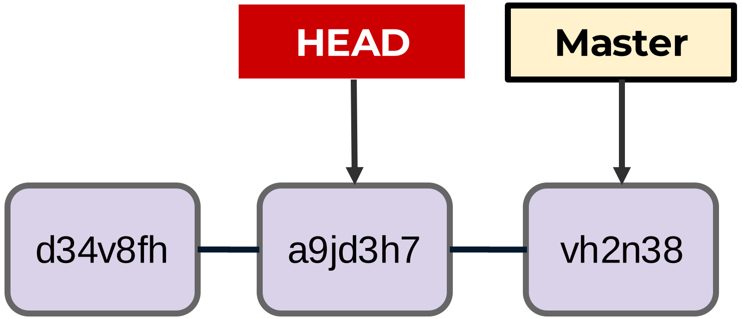
Week 4 Undoing Changes

- You can think of this as traveling back in history to what your code looked like when you ran this commit.



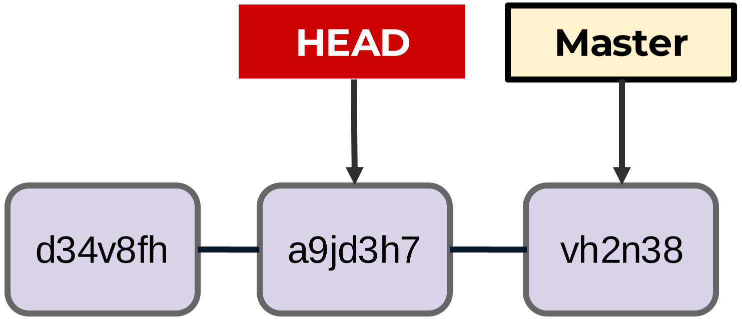
Week 4 Undoing Changes

- This command does **not** undo previous work, you are simply exploring the historical commit.



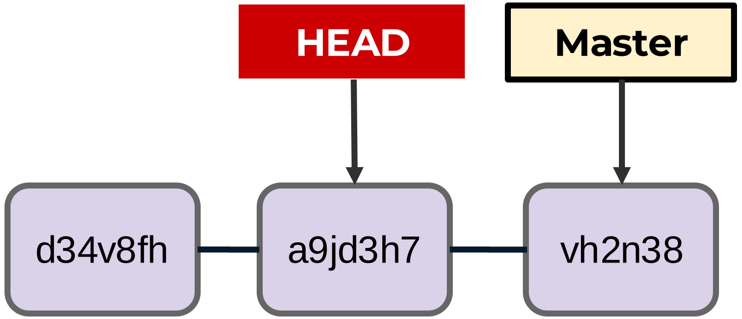
Week 4 Undoing Changes

- If you started making changes here, they won’t be preserved since HEAD is not pointing at a branch reference.



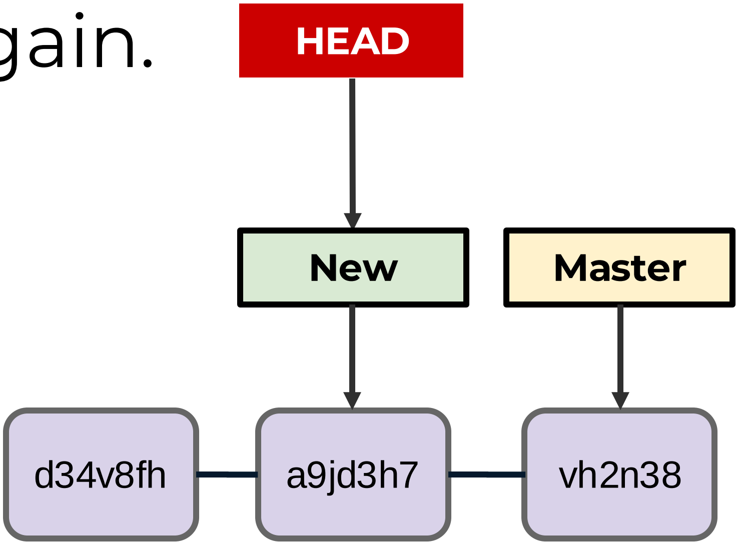
Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



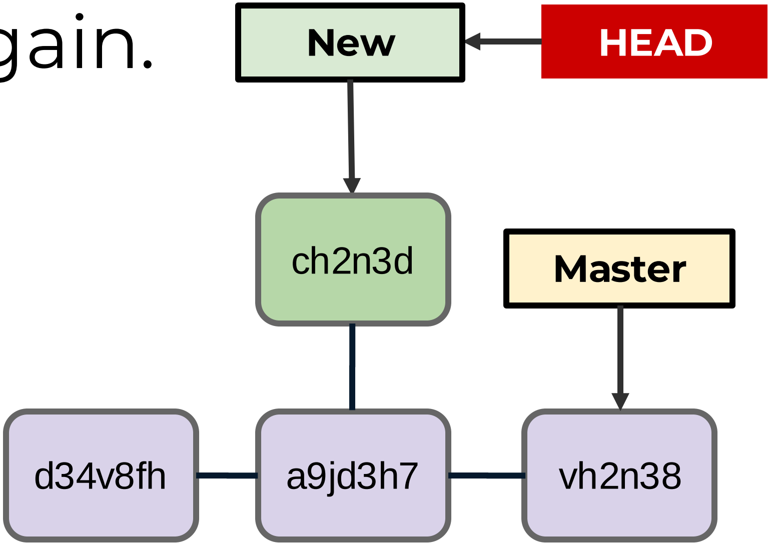
Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



Week 4 Undoing Changes

- Let’s explore these commands in practice!

Week 4
Git Restore

Week 4 Undoing Changes

- Imagine you’ve been working on a file that is already part of a commit.
- You know the code was working at the point of the last commit, however now the code is totally broken, and you just can’t seem to fix it.
- You’ve written too much code to just run Ctrl+Z! How to fix this?

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**
  - ***Warning:***
    - You can not undo a git restore command, since your changes were not committed!

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**
  - ***Warning:***
    - Think of this command as an ultimate “Ctrl+Z” restoring files to their previous commit.

Week 4 Undoing Changes

- **Git restore**
  - Technically speaking **git restore** will restore the file back to the HEAD, which typically we have pointing to the most recent commit in the branch.

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits from the HEAD to go back to:
    - **git restore --source HEAD~N file.txt**

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits prior from the HEAD to go back to:
    - **git restore --source HEAD$\boxed{\text{~N}}$ file.txt**

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits prior from the HEAD to go back to:
    - **git restore --source HEAD~N file.txt**

Week 4 Undoing Changes

- **Git restore**
  - Finally, git restore also allows us to unstage files that we had already added to the staging area using **git add**.
  - We can do this with:
    - **git restore --staged filename**

Week 4 Undoing Changes

- Let's explore this command in practice:
    - **git restore filename**
    - **git restore --source HEAD~N filename**
    - **git restore --staged filename**

Week 4
Git Reset

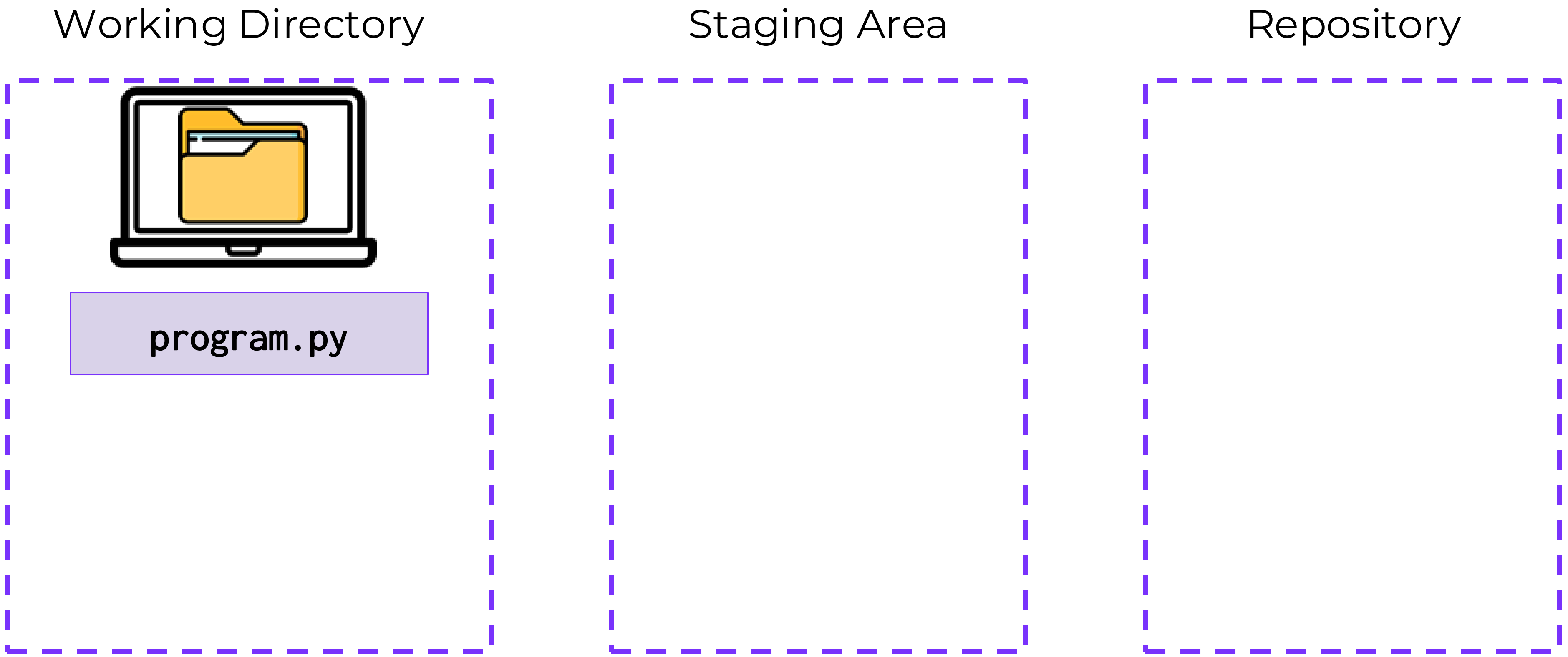
Week 4 Undoing Changes

- Git reset allows us to remove commits and “reset” the branch.
- There are two main types of **git reset** calls:
  - **git reset ########**
    - Removes commits in front of the specific hash called, files unchanged.
  - **git reset ######## --hard**
    - Removes commits *and* the changes in the files.

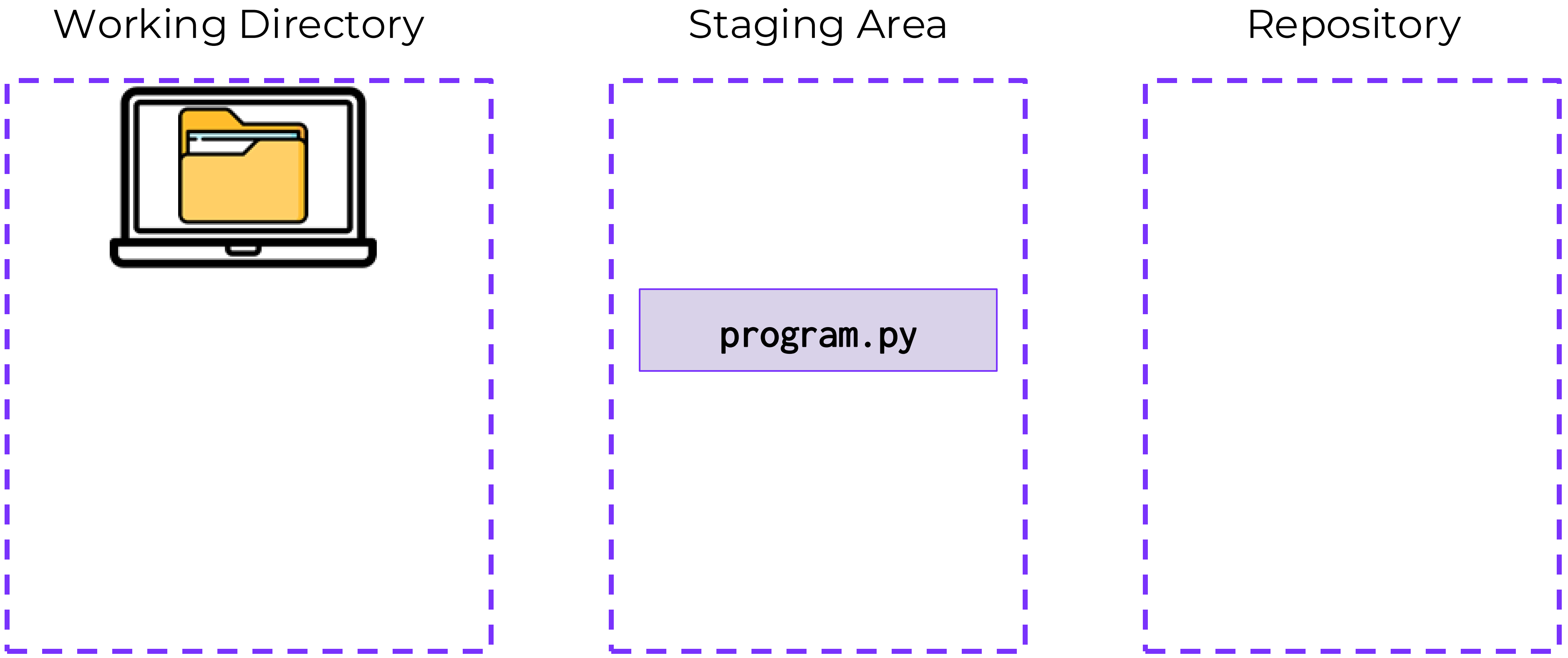
Week 4 Undoing Changes

- To fully understand this, let’s recall our discussions about working directory, staging area, and repository.

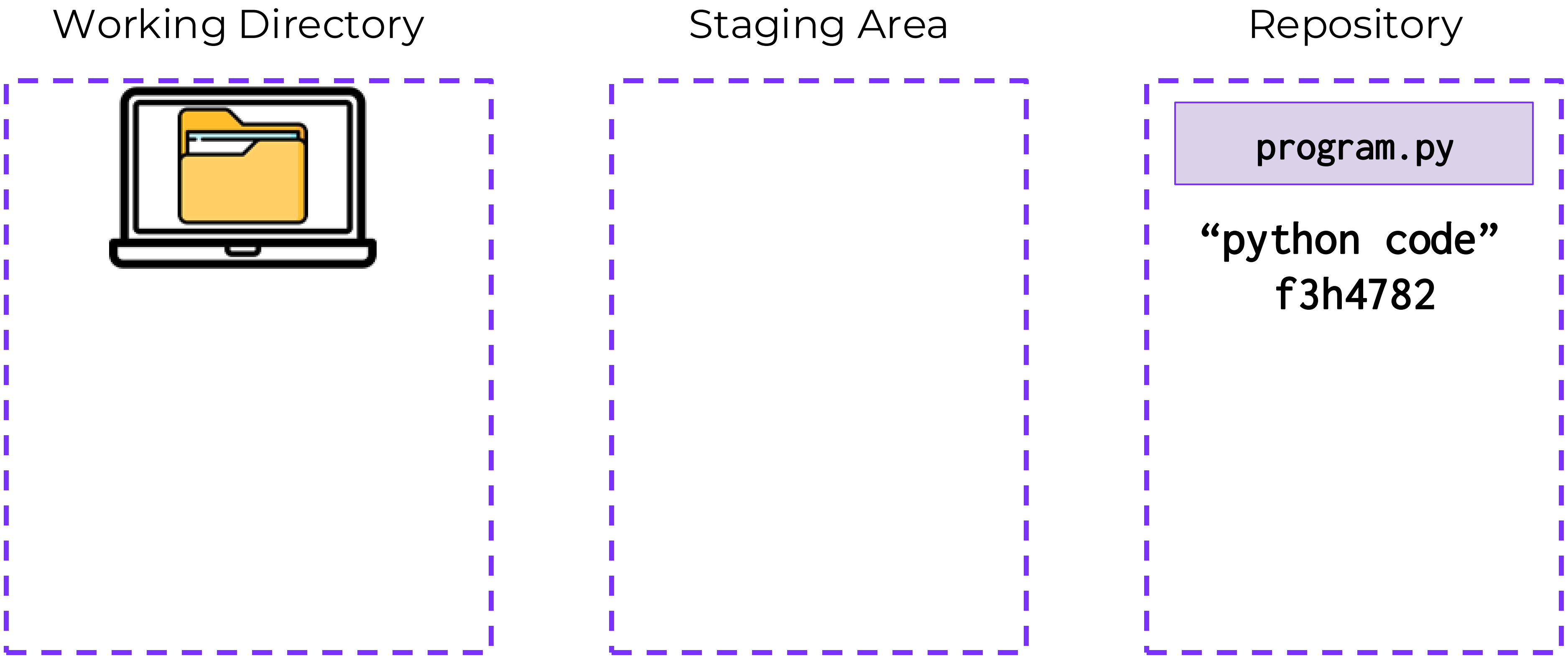
Week 4 Undoing Changes



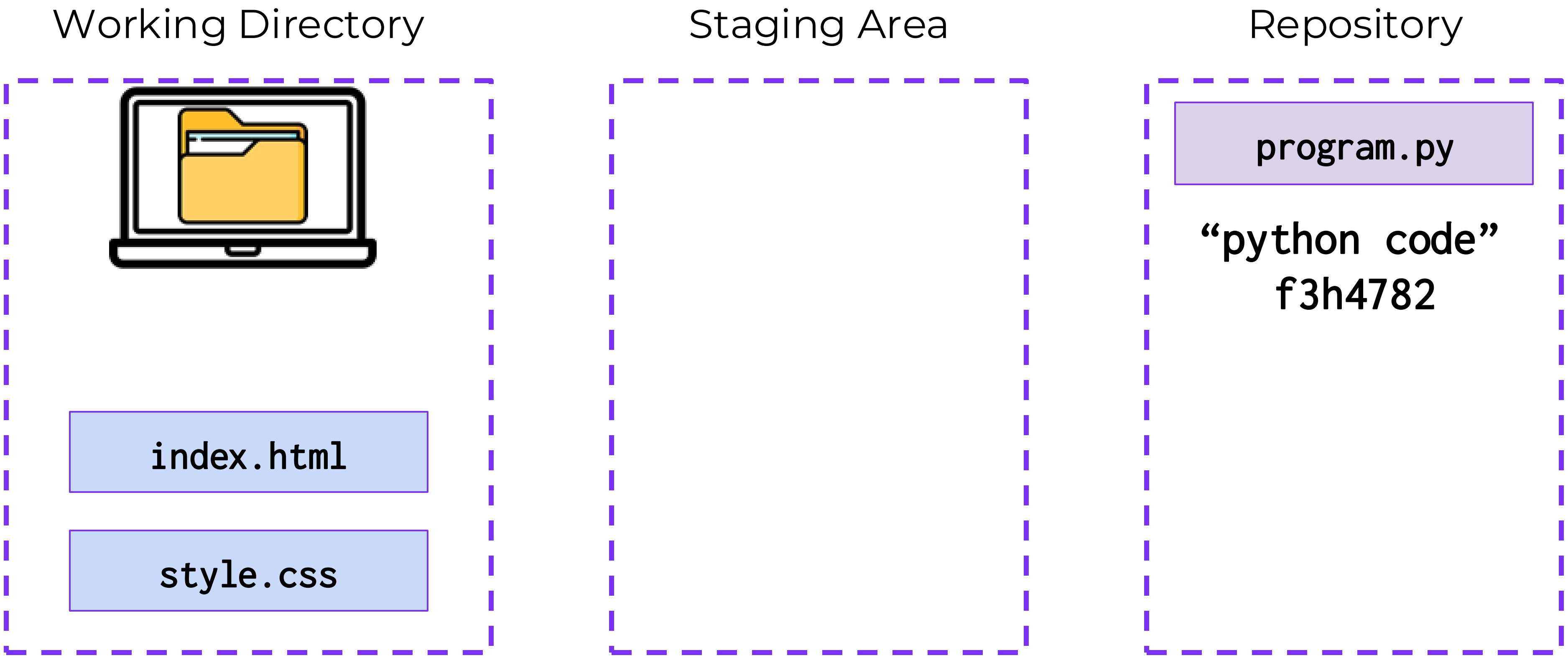
Week 4 Undoing Changes



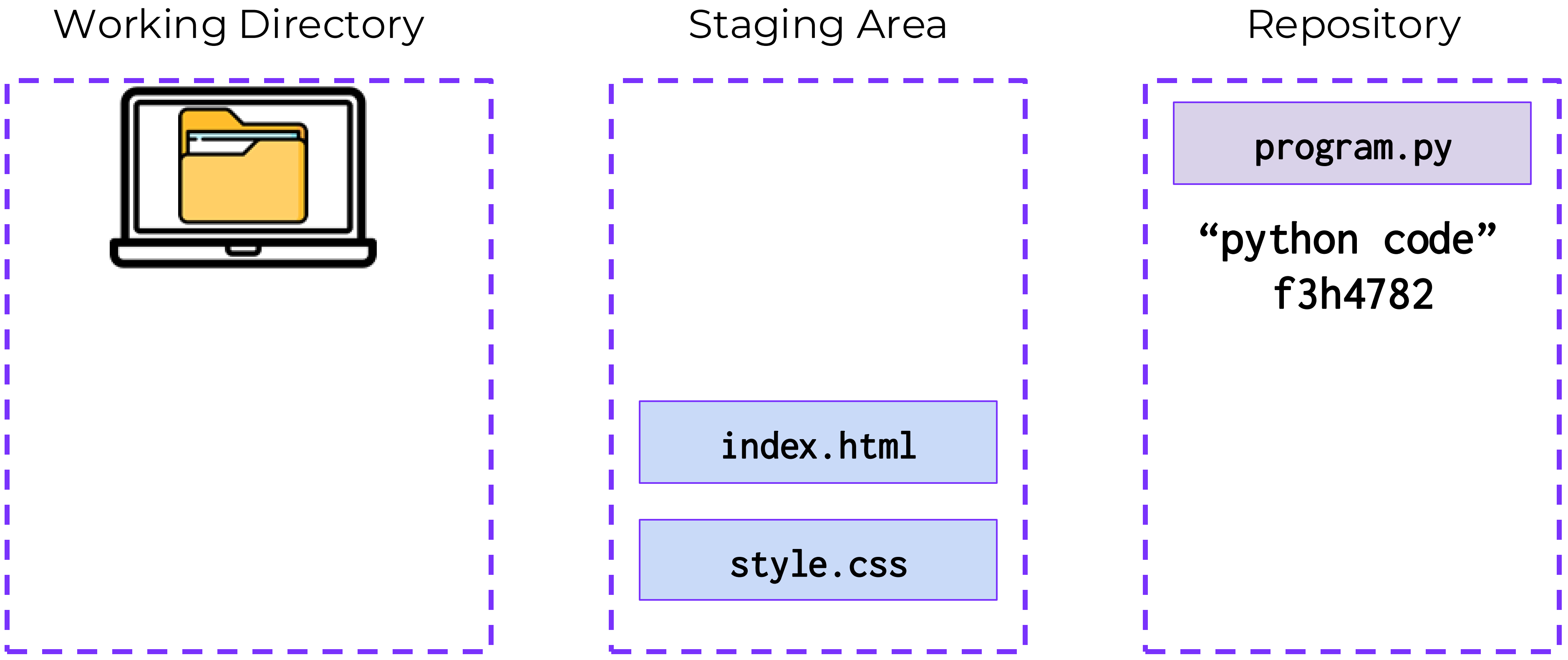
Week 4 Undoing Changes



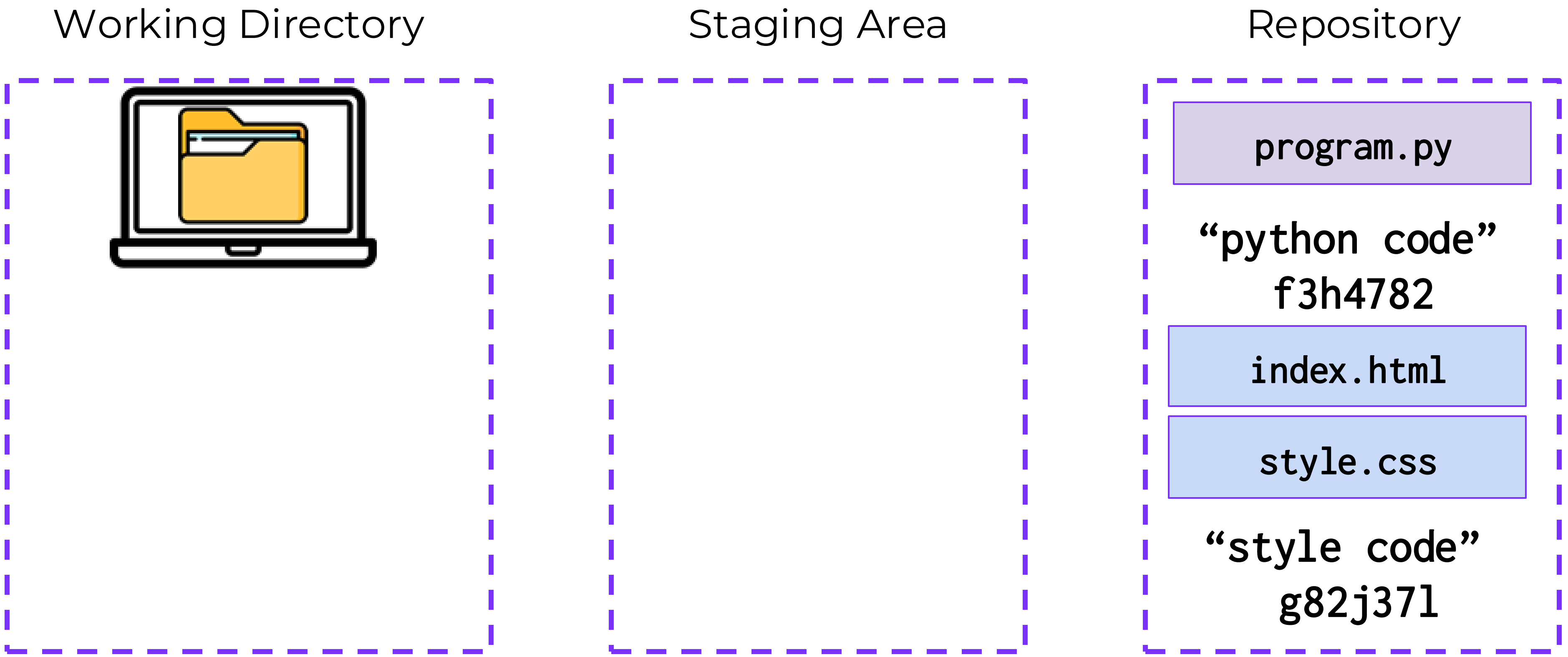
Week 4 Undoing Changes



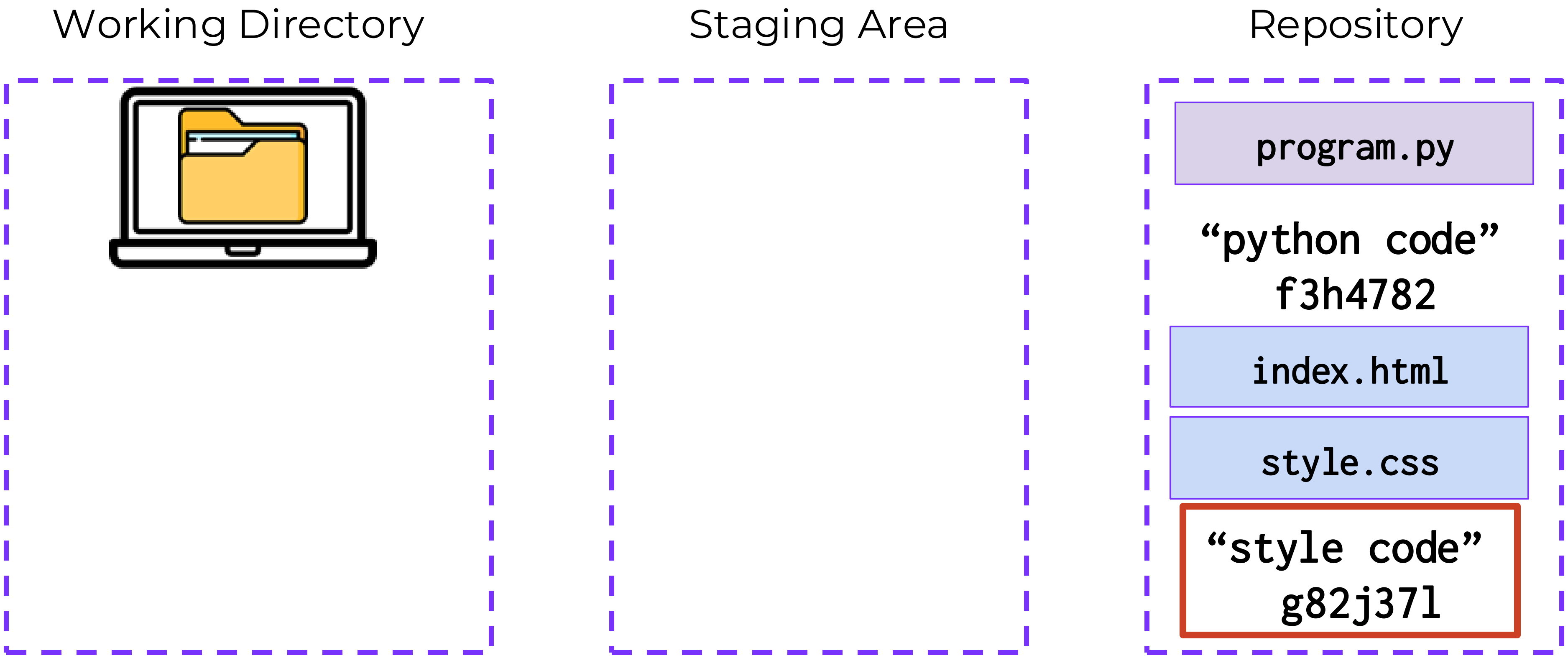
Week 4 Undoing Changes



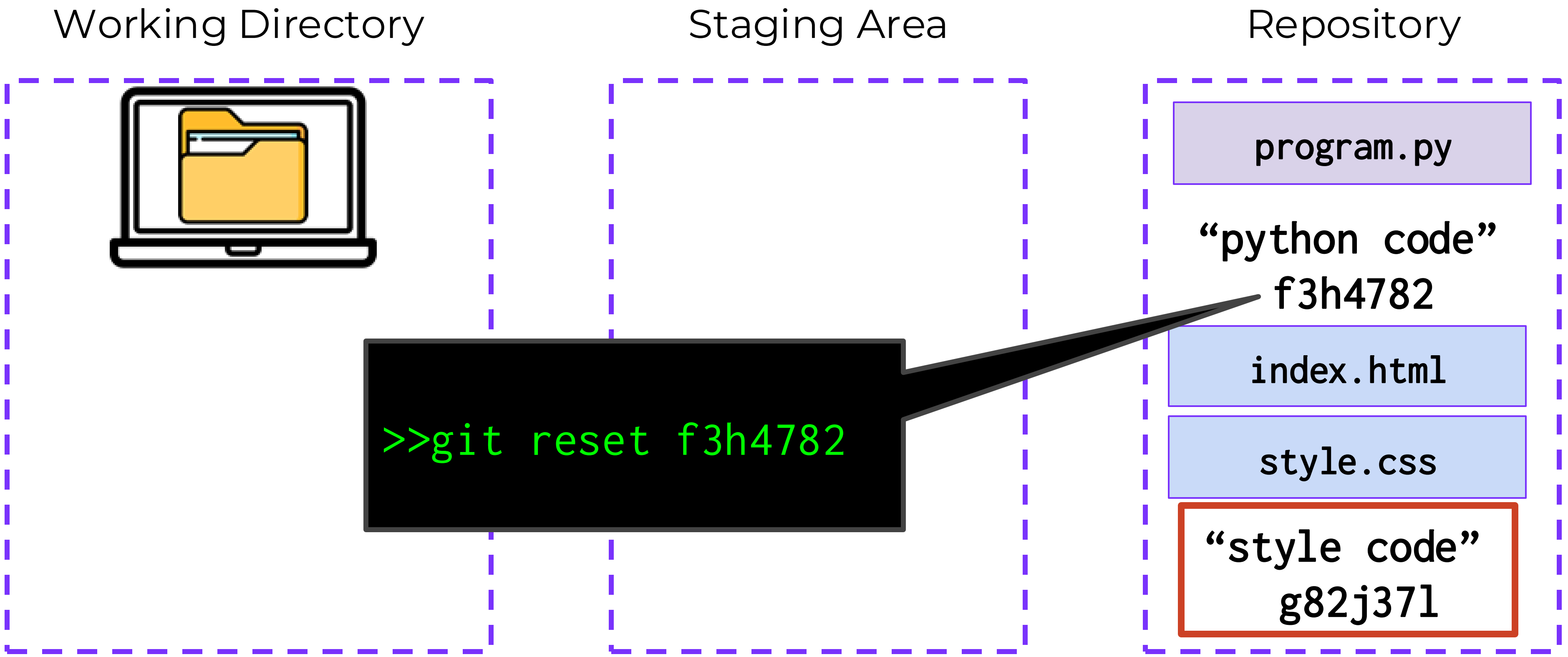
Week 4 Undoing Changes



Week 4 Undoing Changes

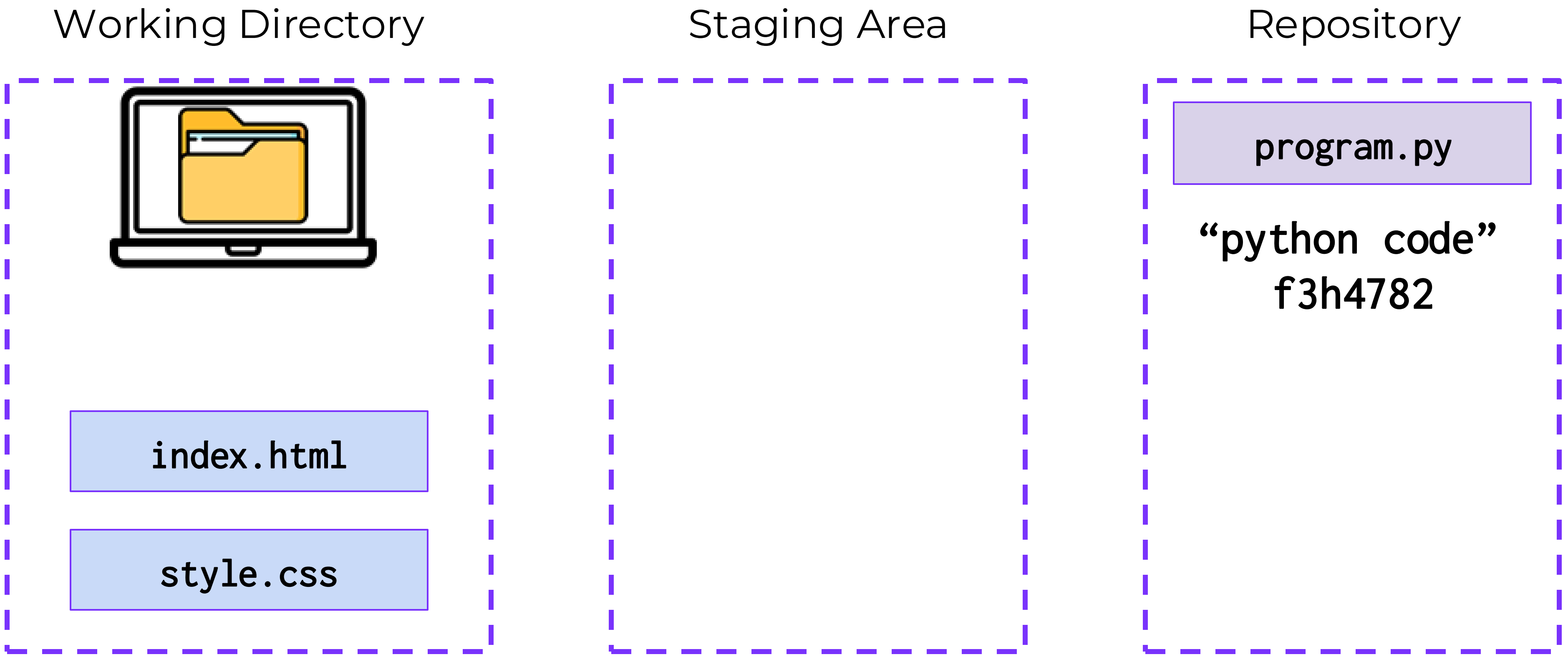


Week 4 Undoing Changes

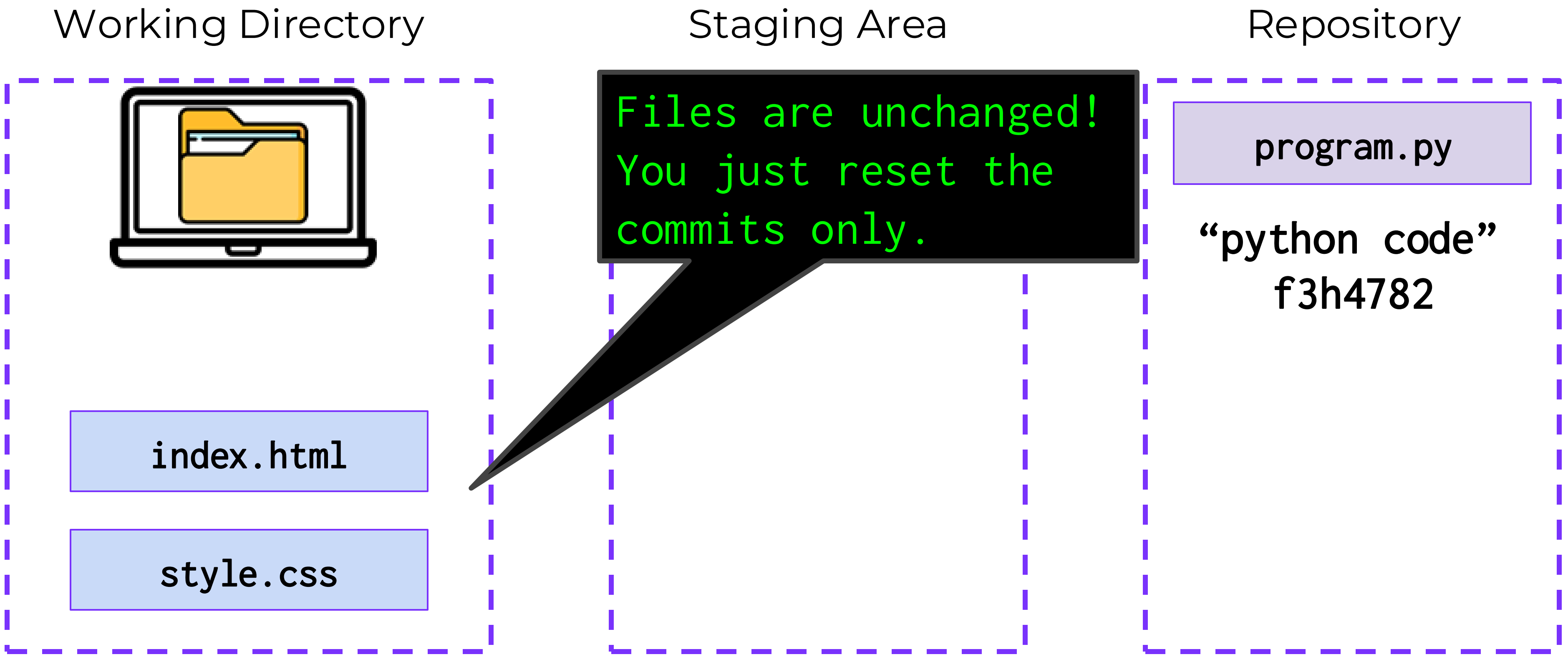


```
>>git reset f3h4782
```

Week 4 Undoing Changes



Week 4 Undoing Changes



Files are unchanged! You just reset the commits only.

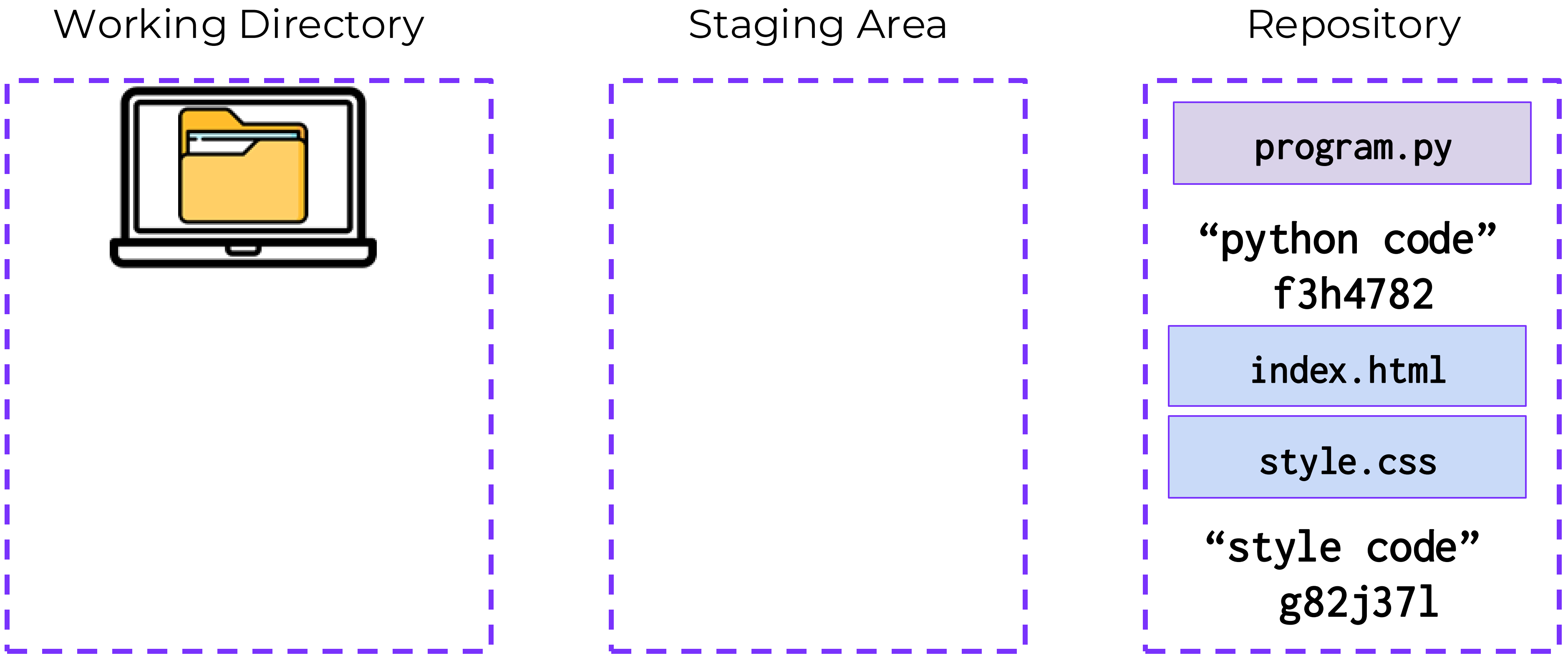
Week 4 Undoing Changes

- This means when you do **git reset** you won’t notice a change in the files themselves, you just reset the commits.
- This is useful if you accidentally committed to the wrong branch (for example, maybe you forgot to run **git switch** right after creating a new branch, accidentally committing to the original branch).

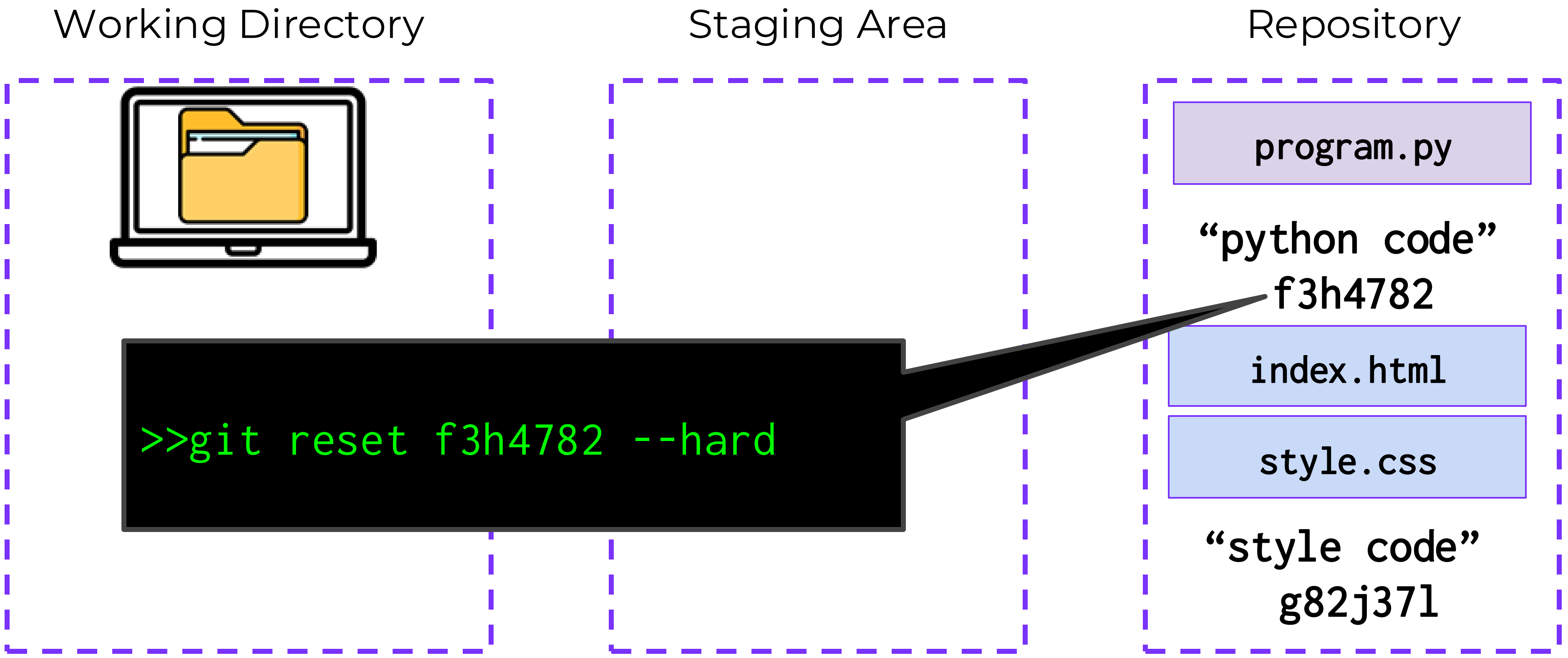
Week 4 Undoing Changes

- What if you do want the files to change?
- In the case where you just want to undo everything, including changes and have the branch files look like they did at a previous commit, you add the flag **--hard**.
- For example:
  - **git reset f3h4782 --hard**

Week 4 Undoing Changes

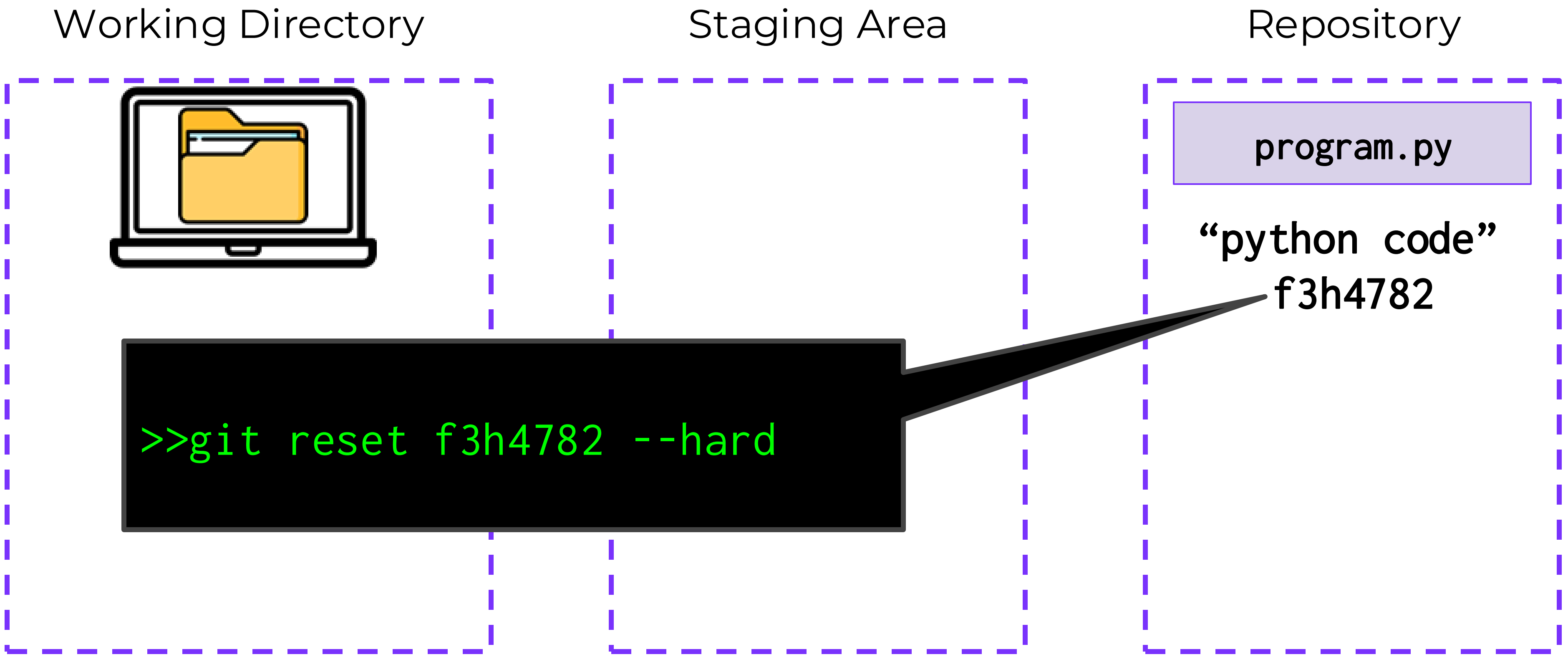


Week 4 Undoing Changes



```
>>git reset f3h4782 --hard
```

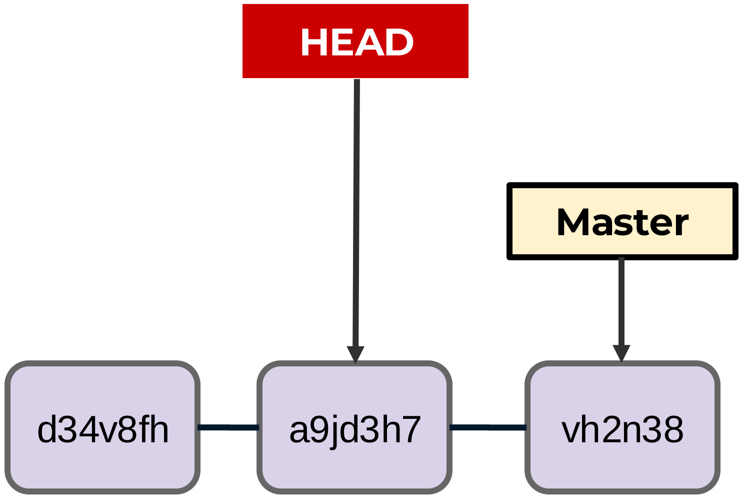
Week 4 Undoing Changes



```
>>git reset f3h4782 --hard
```

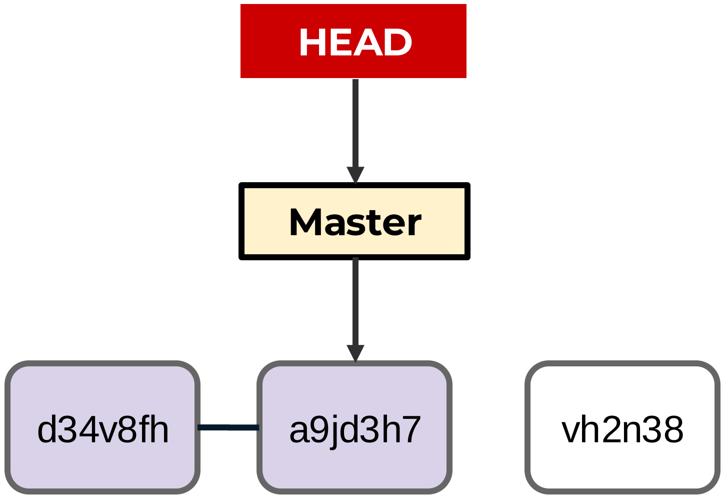
Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



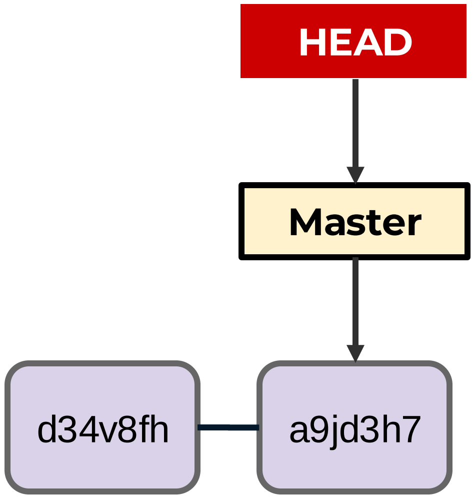
Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



Week 4 Undoing Changes

- *Can you undo a git reset --hard?*
    - Technically you can try to recover a commit before Git does its garbage collection, however you should operate under the assumption that a --hard reset is not recoverable.

Week 4 Undoing Changes

- Let's explore examples of **git reset**!

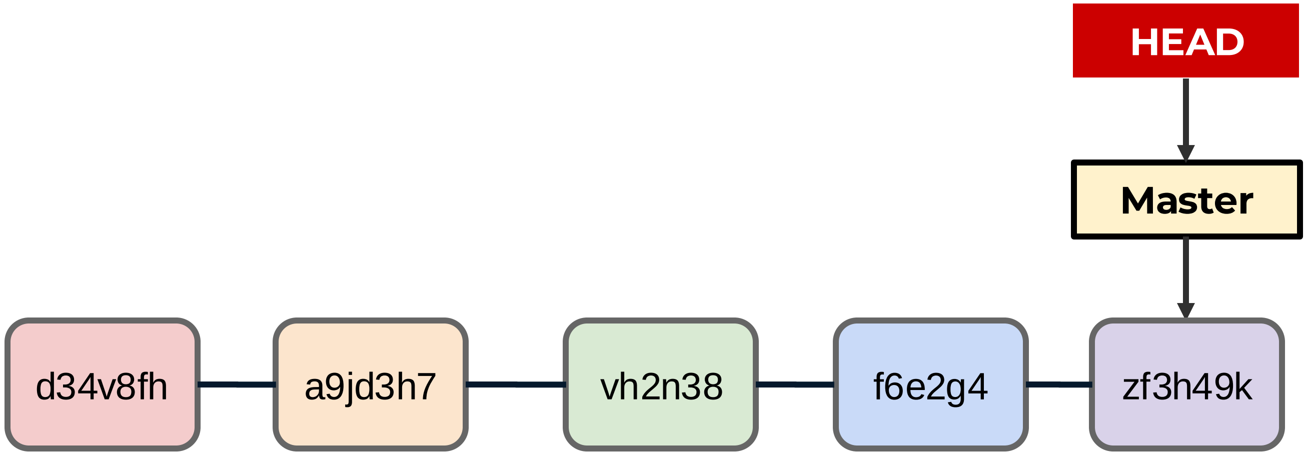
Week 4
Git Revert

Week 4 Undoing Changes

- Let’s explore the last command for undoing changes, **git revert**.
- The **git revert** command will create a new commit that undoes work from previous commits, but keeps those commits in the branch.

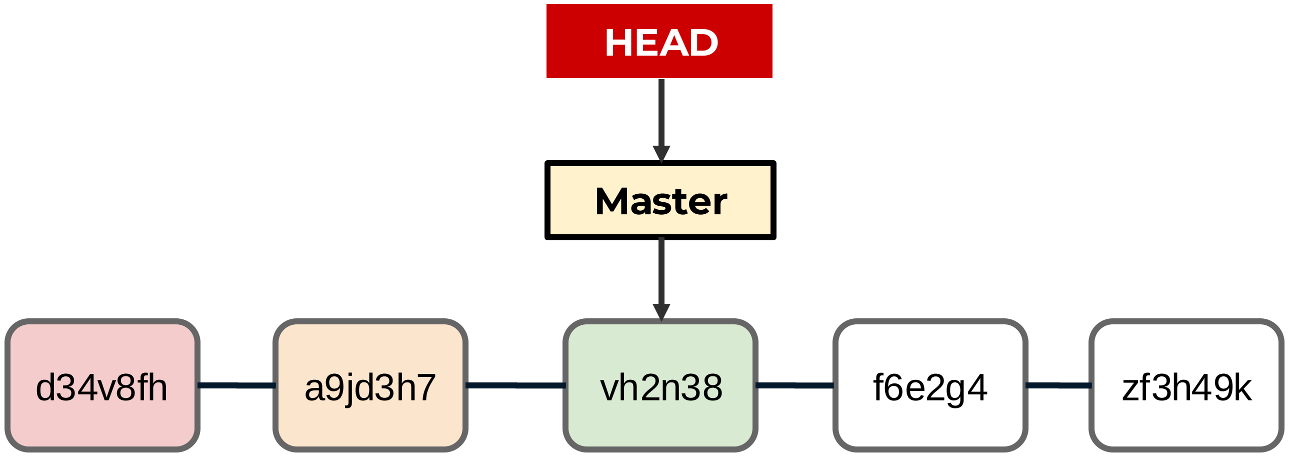
Week 4 Undoing Changes

- Let’s review a **git reset** first.
- A **git reset** goes back and removes the commits (and changes if its **--hard**)



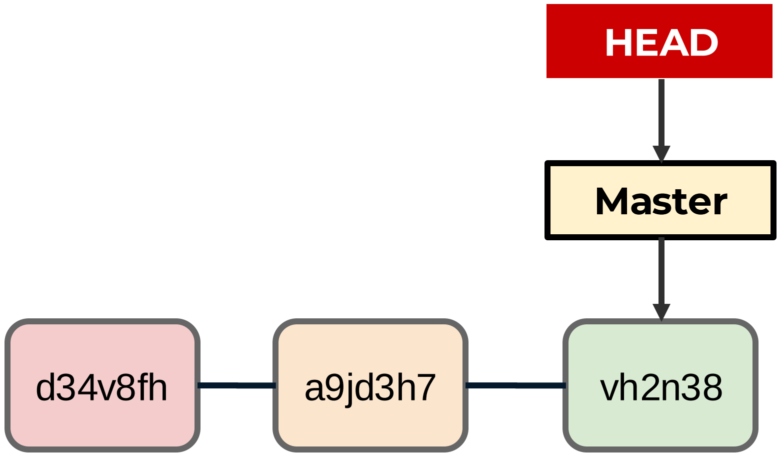
Week 4 Undoing Changes

- Let’s review a **git reset** first.
- A **git reset** goes back and removes the commits (and changes if its **--hard**)



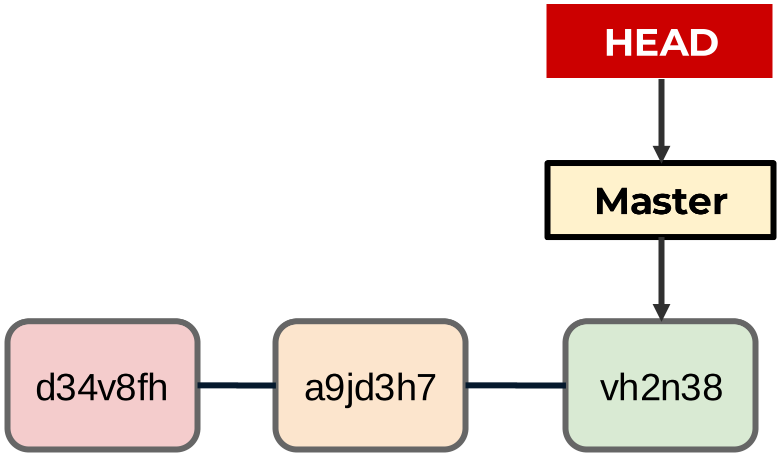
Week 4 Undoing Changes

- Let’s review a **git reset** first.
- A **git reset** goes back and removes the commits (and changes if its **--hard**)



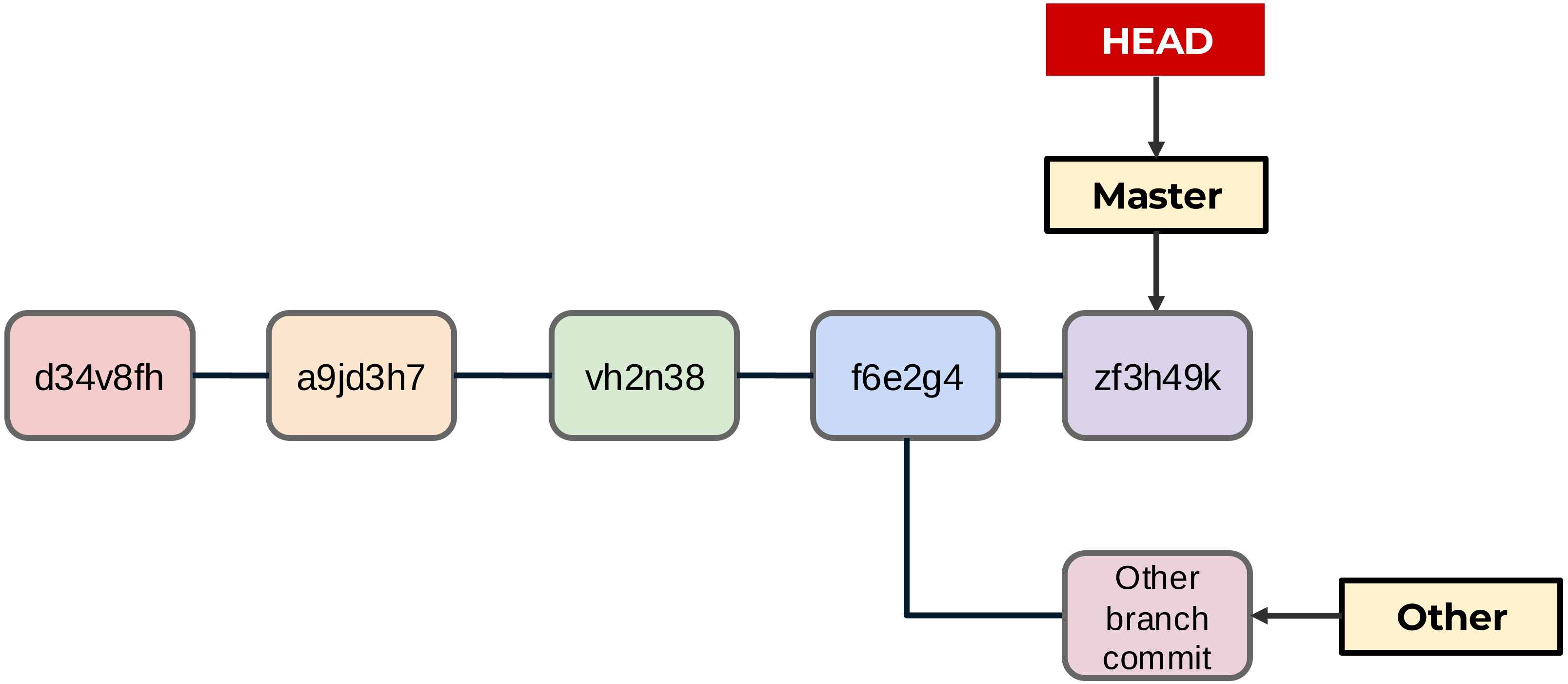
Week 4 Undoing Changes

- Why could this be an issue?
- You can lose **shared history**!



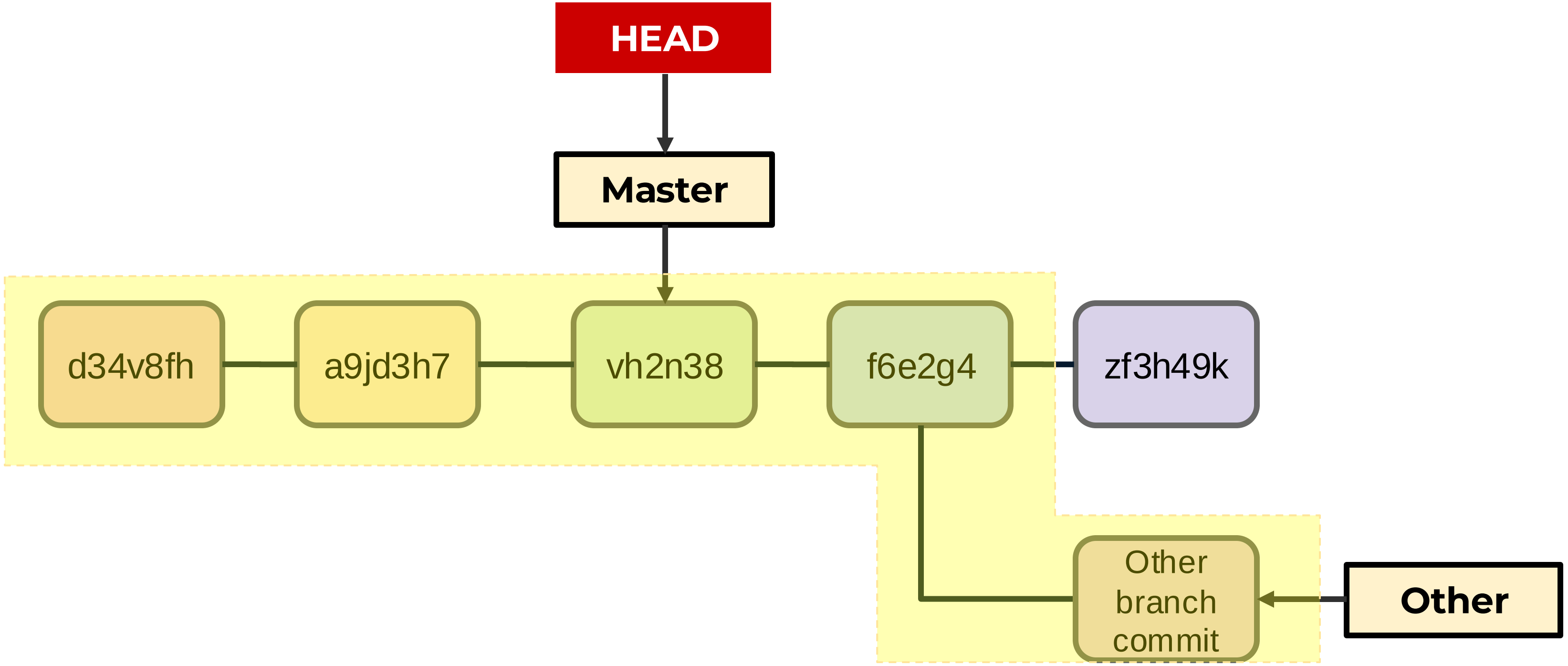
Week 4 Undoing Changes

- Why could this be an issue?
- You can lose **shared history**!



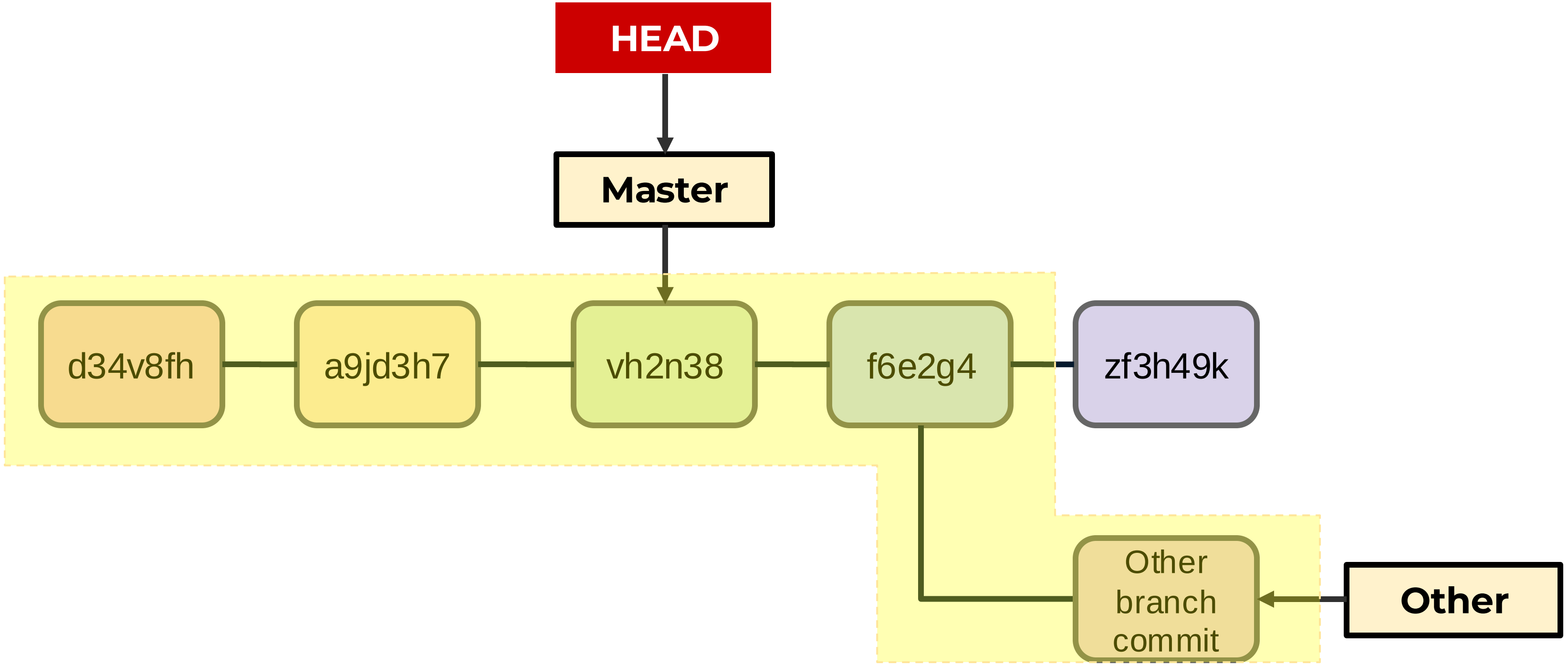
Week 4 Undoing Changes

- Why could this be an issue?
- You can lose **shared history**!



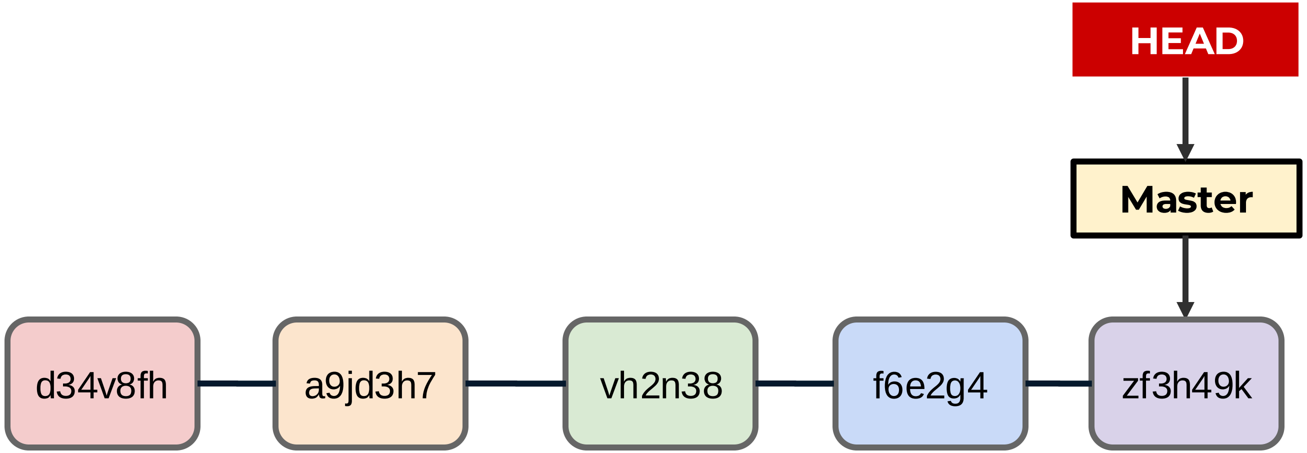
Week 4 Undoing Changes

- This makes a merge of the branches harder!



Week 4 Undoing Changes

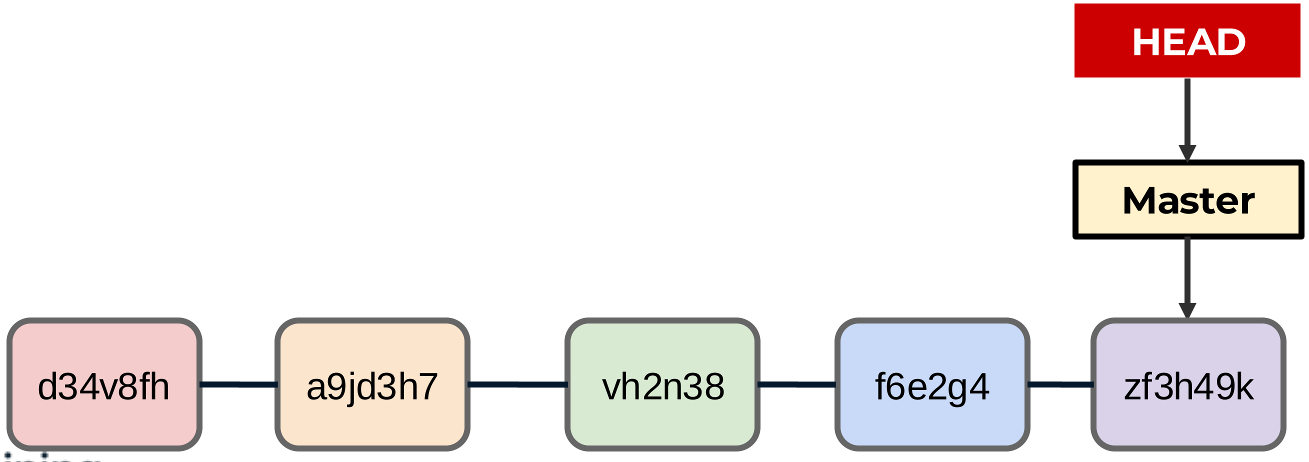
- A **git revert** creates a new commit that matches the historical state of a previous commit.



Week 4 Undoing Changes

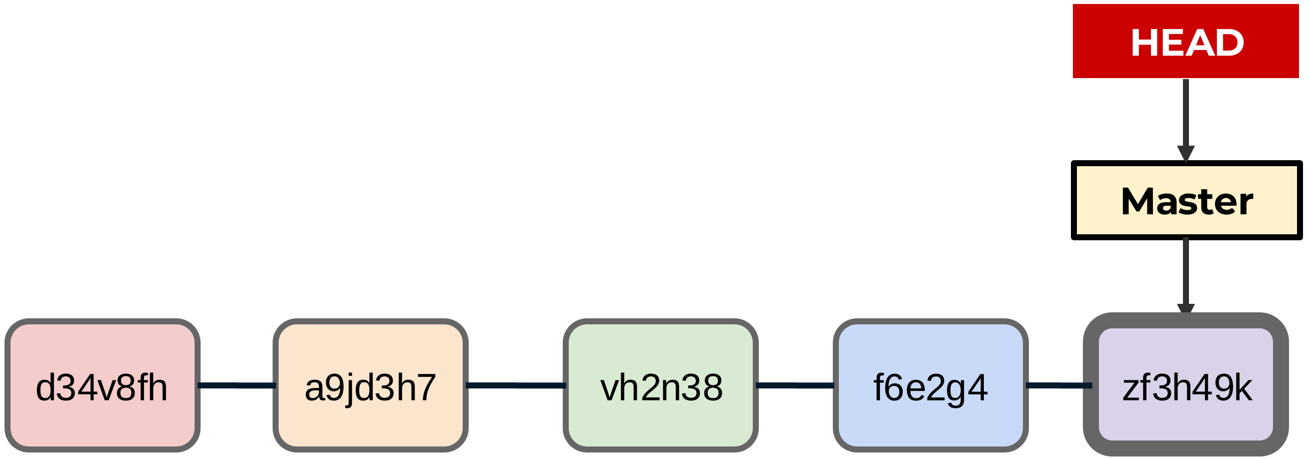
- **Git revert** doesn’t change the project history, which makes it a “safe” operation for commits that have already been published to a shared repository.

- A **git revert** creates a new commit that matches the historical state of a previous commit.



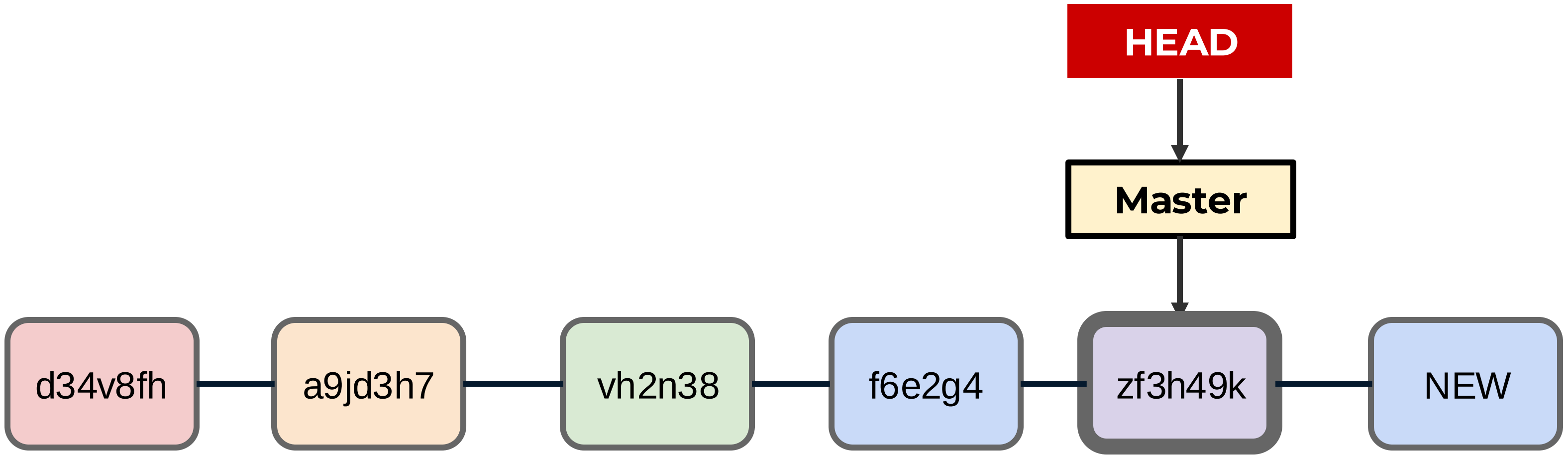
Week 4 Undoing Changes

- A **git revert** creates a new commit that matches the historical state of a previous commit.



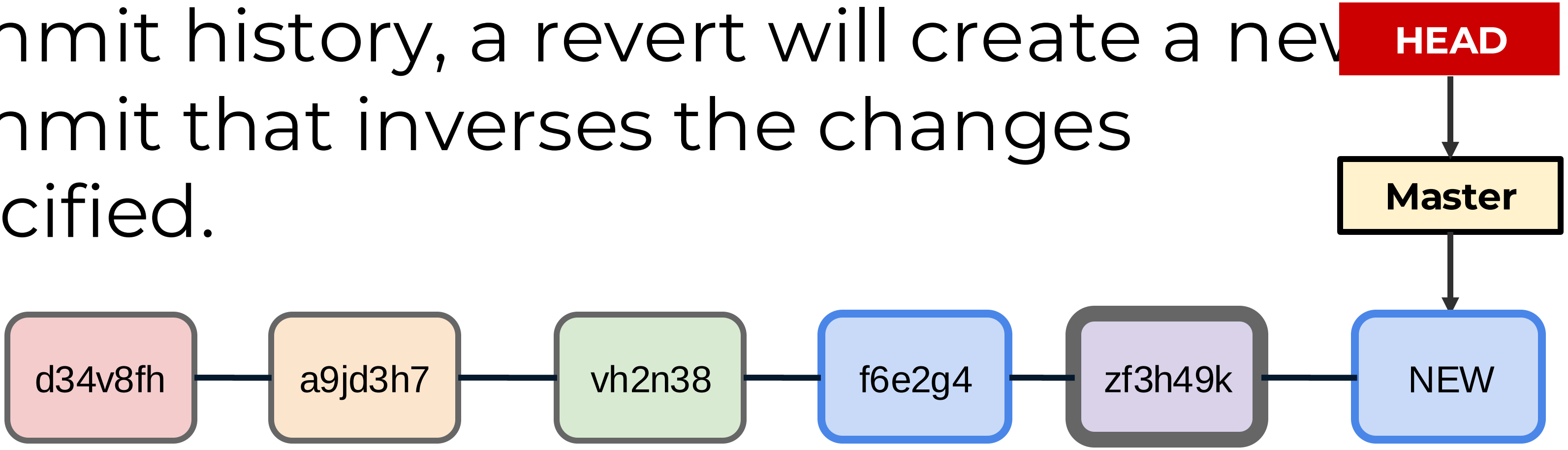
Week 4 Undoing Changes

- A **git revert** creates a new commit that matches the historical state of a previous commit.



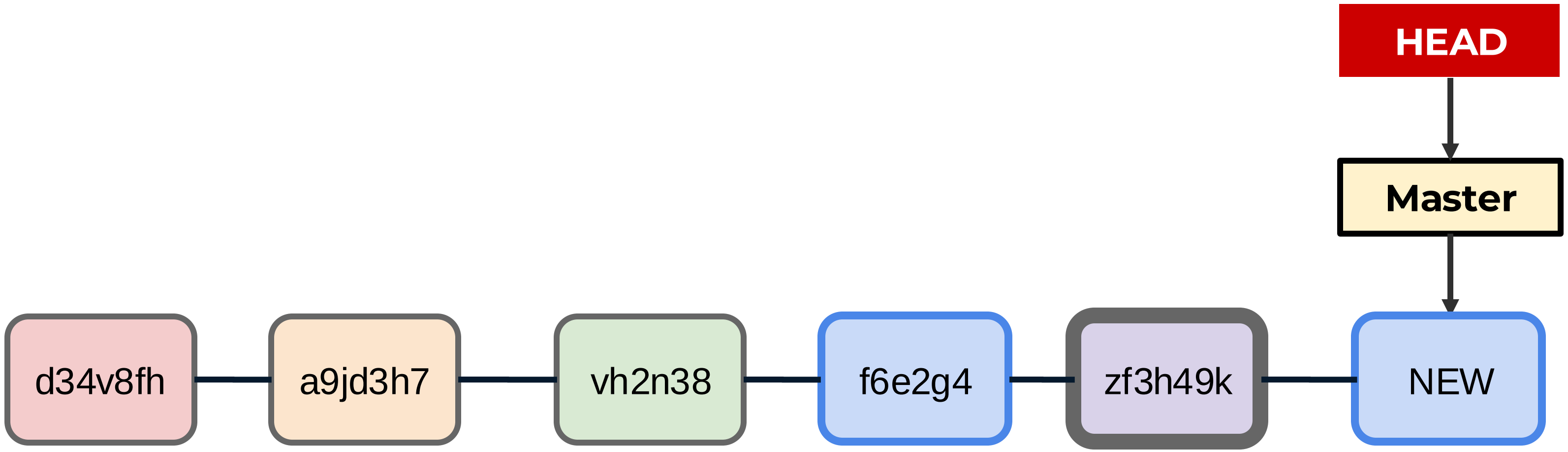
Week 4 Undoing Changes

- The **git revert** command is a forward-moving undo operation that offers a safe method of undoing changes. Instead of deleting or orphaning commits in the commit history, a revert will create a nev commit that inverses the changes specified.



Week 4 Undoing Changes

- **Git revert** is a safer alternative to git reset in regards to losing work.


HEAD
Master
d34v8fh
a9jd3h7
vh2n38
f6e2g4
zf3h49k
NEW