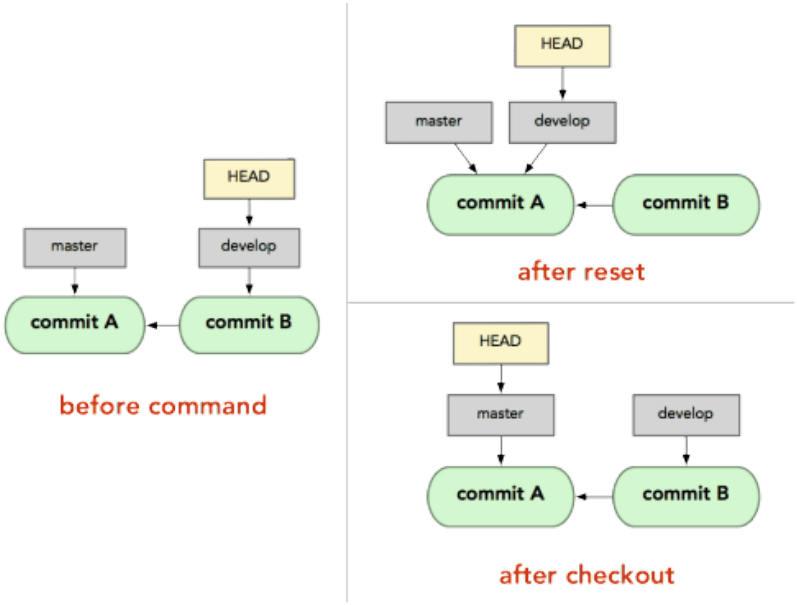
Introduction

As you get more comfortable adding and committing changes using git, you may sometimes accidentally commit something you didn't mean to, or you may want to remove some file that's in the working directory before it's been staged. We'll show you some techniques to undo your changes in this chapter. Please be warned that some of these commands are not reversible, so use them with caution!



checkout

If you want to remove files from the working directory (before they have been staged) you can use `git checkout NAME_OF_FILE`. Be careful with this - you can not undo this command!

Here's a quick example. Create a new git repository, then add and commit a blank file called `first.txt`. Once you've committed the file, `echo hello > first.txt` to add some text to the file. If you check `git status` now, you'll see that `first.txt` is not staged for commit. If you decide that you don't like the change you just made to the file, you can type `git checkout -- first.txt`. If you `cat first.txt` you'll see the file is empty again!

clean

If you are dealing with an untracked or unmerged file, you cannot use `git checkout` to remove it from the working directory. You must use `git clean -df` to remove these files. Be careful with this

- you can not undo this command either! (Curious about the `-df` flags? Run `man git-clean` to learn more about this command!)

`git rm --cached`

We've seen what to do if we have something in the working directory that we want to remove. But what if we accidentally add something to the staging area and want to move it back to the working directory? To do this, you can type `git rm --cached NAME_OF_FILE`. If you need to remove a folder pass the `-r` flag to `git rm --cached`. If you want to move all of your files in the staging area to the working area you can type `git rm -r --cached .`. If you want remove your files from the staging area **AND** the working directory, you can type `git reset --hard HEAD`, but be careful - this can not be undone!

Undoing commits with `reset`

We've seen how to remove things from the working directory, and how to remove things from the staging area back into the working directory. But sometimes we end up *committing* things we do not want to be remembered. To undo commits we can use the `git reset` command. There are 3 flags we can pass to this:

`git reset --soft COMMIT_SHA` - moves the files committed back to the staging area

`git reset --mixed COMMIT_SHA` - moves the files committed back to the working directory (if you use `git reset` without a flag, the default will be `--mixed`)

`git reset --hard COMMIT_SHA` - undoes the entire commit (**dangerous**!!!)

What's the `COMMIT_SHA`, you ask? You may have noticed that every commit has a unique identifier, called a sha, which identifies that commit. If you type `git log --oneline`, you'll see your list of commit messages along with the first seven characters of the commit sha. This is what you should pass into each of these commands.

Using reset will not change the commit that you switch to but any commits that have come after it. So if we have 4 commits:

```
a808698     Fourth commit
ca0bbb4     Third commit
5ffcac5     Second commit
ac49968     First commit
```

And I want to move the last two commits to the staging area:

`git reset --soft 5ffcac5`, this will move whatever files we had in the Fourth commit and Third commit back to the staging area.

Your Turn

1. Create a folder called `destruction`.
2. `cd` into that folder.
3. Initialize an empty git repository.
4. Create a file called `done.txt`.
5. Remove that file from the working directory (remember you can not use `git checkout`).
6. Create a file called `stage_me.txt`.
7. Add `stage_me.txt` file to the staging area.
8. Move `stage_me.txt` file from the staging area to the working directory.
9. Add `stage_me.txt` file to the staging area.
10. Remove `stage_me.txt` from the staging area **and** the working directory.
11. Create a new file called `commit_me.txt`.
12. Add `commit_me.txt` to the staging area.
13. Commit with the message "adding commit_me.txt".
14. Create another file called `second.txt`.
15. Add `second.txt` to the staging area.
16. Commit with the message "adding second.txt".
17. Check out your previous commits using `git log --oneline` to see the unique identifier or SHA for each of your commits.
18. Using `git reset`, undo the previous commit and move your changes back to the **working directory**.
19. Add `second.txt` again.
20. Commit with the message "Trying to commit again".
21. Using `git reset` undo the previous commit and move your changes back to the **staging area**.
22. Commit with the message "Trying to commit again and again".
23. Using `git reset` undo the previous commit so that any changes are **not** part of the working directory.
24. Pat yourself on the back! You just went through a pretty complex git workflow!

Well done! Play around some more with these commands as they will be essential when dealing with larger files, branches, and merges.

Previous next