# Fix simple mistakes

15 minutes

Sometimes, things go wrong. You might forget to add a new file, or maybe you add a file by mistake. Perhaps you made a spelling error in your latest commit or you committed something you didn't intend to. Perhaps you accidentally deleted a file.

Git lets you make changes fearlessly, because it always offers a way to get back to where you were. You can even change Git's commit history as long as you only change commits that haven't been shared.

*Amend a commit: --amend flag*

In the preceding exercise, you updated the index.html file to modify the path to the style sheet. You should have added the following statement:

HTML

<link rel="stylesheet" href="CSS/site.css">

Suppose you discover that you made an error when you entered the statement. Instead of specifying the folder path as CSS, you entered CS:

HTML

<link rel="stylesheet" href="CS/site.css">

When you refresh the page in your browser, you notice that your CSS style sheet isn't applied. After you investigate, you realize that you entered the path values incorrectly.

So, you update index.html with the correct path to the style sheet. At this point, you could just commit the corrected version of index.html, but instead, you prefer to put it in the same commit as the original. The --amend option to git commit lets you change history (and how often does one get the chance to change history?).

Bash

*git commit --amend --no-edit*

The --no-edit option tells Git to make the change without changing the commit message. You can also use --amend to edit a commit message, to add files that were accidentally left out of the commit, or to remove files that were added by mistake.

 Note

The ability to change history is one of Git's most powerful features. As with most power tools, you must use it carefully. In particular, it's a bad idea to change any commits that have been shared with another developer, or which were published in a shared repository, like GitHub.

Recover a deleted file: git checkout

Imagine that you made a change to a source code file that broke the entire project, so you want to revert to the previous version of that file. Or perhaps you accidentally deleted a file altogether. Git makes it easy to retrieve an earlier version, even if the current version no longer exists. Your best friend in this situation is the [git checkout](https://git-scm.com/docs/git-checkout) command.

git checkout has multiple uses, but in the next exercise, we'll use it to recover a deleted file. git checkout updates files in the working tree to match the version in the index or in the specified tree.

If you've accidentally deleted a file, you can recover it by bringing the version from the index back into the working tree by using this command:

Bash

*git checkout -- <file\_name>*

You can also check out a file from an earlier commit (typically, the head of another branch), but the default is to get the file from the index. The -- in the argument list serves to separate the commit from the list of file paths. It's not strictly needed in this case, but if you had a branch named <file\_name> (perhaps because that's the name of the file being worked on in that branch), -- would prevent Git from getting confused.

Later, you'll learn that you also use checkout to switch branches.

Recover files: *git reset*

You also can delete a file by using git rm. This command deletes the file on your disk, but it also has Git record the file deletion in the index.

So, if you ran this command:

Bash

*git rm index.html*

*git checkout -- index.html*

Git would not happily restore index.html! Instead, you'd get an error like this example:

Output

error: pathspec 'index.html' did not match any file(s) known to git.

To recover index.html, we would have to use a different technique: git reset. You can use git reset to unstage changes.

You could recover index.html by using these two commands:

Bash

*git reset HEAD index.html*

*git checkout -- index.html*

Here, git reset unstages the file deletion from Git. This command brings the file back to the index, but the file is still deleted on disk. You can then restore it to the disk from the index by using git checkout.

Here's another "Aha!" moment for new Git users. Many VCSes make files read-only to ensure that only one person at a time can make changes; users use an unrelated checkout command to get a writable version of the file. They also use checkin for an operation similar to what Git does with a combination of add, commit, and push. This fact occasionally causes confusion when people begin to use Git.

Revert a commit: *git revert*

The last important command to know for fixing mistakes with Git is git revert. git checkout works only in situations where the changes to undo are in the index. After you've committed changes, you need to use a different strategy to undo them. In this case, we can use git revert to revert our previous commit. It works by making another commit that cancels out the first commit.

We can use git revert HEAD to make a commit that's the exact opposite of our last commit, undoing the previous commit while leaving all history intact. The HEAD part of the command just tells Git that we want to "undo" only the last commit.

As an aside, you can also remove the most recent commit by using the git reset command:

Bash

*git reset --hard HEAD^*

Git offers several types of resets. The default is --mixed, which resets the index but not the working tree; it also moves HEAD, if you specify a different commit. The --soft option moves HEAD only, and it leaves both the index and the working tree unchanged. This option leaves all your changes as "changes to be committed", as git status would put it. A --hard reset changes both the index and the working tree to match the specified commit; any changes that you made to tracked files are discarded.