COURSEWARE

Professional Skills
Agile Fundamentals
Jira
Git
- Introduction to Source Control
- Basics
- Cloning
- Forking
- Branching
- Merging
- Reverting
- GitHub Pull Requests
- GitHub Reviews
- GitHub Actions

Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React

Reverting

Contents

Overview

Reverting in Git is a forward-moving way of undoing changes on a repository.

Reviewing the history of a repository

The best way to view commit history is by using the following command:

```
git log
```

This command will show you the history of all commits for the current branch, with the output including: date, commit message and the SHA-1 identifying hash.

There are also additional flags that make this command easier to use, such as:

```
git log --oneline
```

Using `--oneline` flag simplifies the output into one line per commit.

```
git log --branches=*
```

By using the `--branches=*` flag, we are making Git return the history of all commits for all the branches in that repository.

To get the data for a specific branch, the log command would need to be used like this:

```
# git log [BRANCH_NAME]
git log main
```

Reverting to a previous commit with revert

Let's assume that your `git log --oneline` looks similar to this:

```
875f31e (HEAD -> main) fourth commit
483856a third commit
2dd011d second commit
bcabb84 first commit
```

If we execute:

`git revert HEAD`

Git will create a new commit, which will do the opposite of the previous commit (for example, if you added a piece of code you didn't need, the revert would create a commit deleting this piece of code). You can also use revert to go back to a specific SHA-1 (`483856a` for example).

The history will still contain the fourth commit, but it's changes will be undone. Using `revert` allows us to use the same branch and is considered the better solution for reverting.

Reverting with reset

Instead of using `git revert` in the situation above, we could have used:

`git reset --hard 483856a`

This command will return the state to the selected commit (`483856a third commit`).

The difference between this and `git revert` is that now the Git history will no longer contain the fourth commit, and work would continue as if the `fourth commit` never happened.

However, not having the commit reflected in the commit history can cause complications when working with a shared remote repository.

If the reset happened to a commit that is already shared with others, and we tried to push some changes afterwards,
Git would throw an error; this is because it would think that our local Git history isn't up to date. In these scenarios, it's more appropriate to use the `revert` strategy.

Using Revert to go Back to the Latest Commit

If you have made a lot of changes and just want to back to the latest commit that was made then you can use `reset` without specifying a SHA1:

```
git reset --hard
```

Tutorial

1. Create a folder called "tmp"
2. Initialise the folder as a git repository
3. Create a new file called `test.txt`
4. Stage the file and commit, with an explicit commit message
5. Repeat the previous two steps (using a different file name each time) until you have done 5 commits
6. Now check the git log history for the branch you are on (try out the additional flags for viewing the git log history)
7. Use the `git revert HEAD` command to 'undo' the last commit you did
8. Use the `ls` command to see what happened
9. Use the `reset` command to go back to your first commit
10. Use the `ls` command to see what happened
11. Check the history for the branch you're working on.

Exercises

There are no exercises for this module.