A ‘pre-commit’ Hook

Learn about the "pre-commit hook."

We'll cover the following

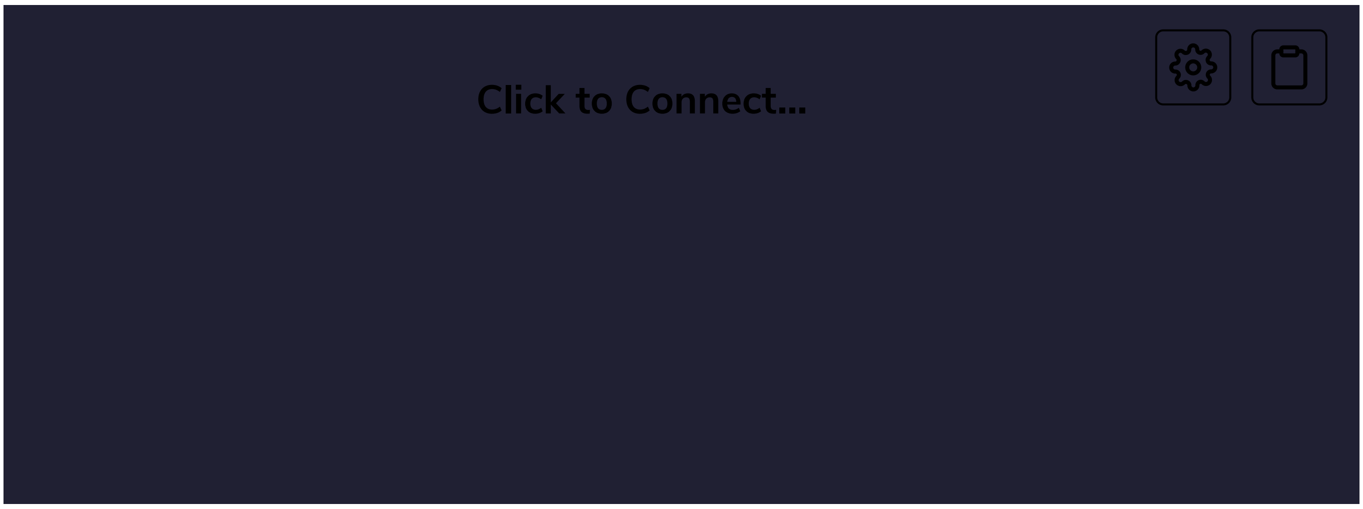
Create repositories

To understand Git hooks properly, you’re going to create a bare repository with nothing in it and then clone from that bare repo. You looked at bare repositories earlier in this (https://www.educative.io/collection/page/10370001/6075350136651776/6425606638534656) (advanced) part.

```
mkdir lgthw_hooks
cd lgthw_hooks
mkdir git_origin
cd git_origin
git init --bare
cd ..
git clone git_origin git_clone
```

Terminal 1

Terminal



You have two repositories: `git_origin` (which is the bare repository you will push to) and `git_clone` (which is the repository you will work in). You can think of them as part of a client-server Git workflow where users treat the `git_origin` folder as the server and clones as the client.

Next, add some content to the cloned repository, and push it:

```
cd git_clone
echo 'first commit' > file1
git add file1
git commit -m 'adding file1'
git push
```

Nothing surprising should have happened there. The content was added, committed, and pushed to the origin repo.

Adding a “pre-commit” hook

Now imagine that you’ve set a rule for yourself that you shouldn’t work on weekends. To try and enforce this, you can use a Git hook in your clone.

Add a second change, and take a look at the `.git/hooks` folder:

```
echo 'second change in clone' >> file1
ls .git/hooks
```

In the `.git/hooks` folder, there are various examples of scripts that can be run at various points in the Git content lifecycle. If you want to, take a look at them now to see what they might do, but this can be somewhat bewildering.

Terminal 1

Terminal

What you’re going to do now is create a script that is run before any commit is accepted into your local Git repository:

```
cat > .git/hooks/pre-commit << EOF
> echo NO WORKING AT WEEKENDS!
> exit 1
> EOF
chmod +x .git/hooks/pre-commit
```

You have created a `pre-commit` script in the `hooks` folder of the repository’s local `.git` folder and made it executable. The script prints a message about not working on weekends and `exit`s with a code of `1`, which is a generic error code in a shell script (`exit 0` would mean “OK”).

See what happens when you try to commit:

```
git commit -am 'Second change'
```

Terminal 1

Terminal

You should have seen that the commit did not work. If you’re still not sure whether it got in, run a log command in the terminal above and check that the diff is still there:

```
git log
git diff
```

This should confirm that no commit has taken place.

To show a reverse example that lets the commit through, replace the script with this content:

```
cat > .git/hooks/pre-commit << EOF
> echo OK
> exit 0
> EOF
```

Terminal 1

Terminal

This time you’ve added an OK message and exited with a `0` (success) code rather than a `1` for error.

Now your commit should work, and you should see an OK message as you commit.

```
git commit -am 'Second change'
```

Back

Introduction: Git Hooks

Next

A More Sophisticated Example

✓ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/a-‘pre-commit’-hook__git-hooks__learn-git-the-hard-way)