(/learn)

Git Tracks the Submodule’s State

Learn how to track the submodule's state.

We'll cover the following

Get the experimental branch

Since your `alicelib` submodule is a straightforward clone of the remote `alicelib origin`, you have the `master` branch and the `origin`’s `experimental` branch:

```
1   cd alicelib
2   git branch -a -vv
```

Terminal 1

Terminal

Click to Connect...

You are on the `master` branch (current `HEAD` location is indicated with a `*`), which is mapped to `remotes/origin/master`.

> **References Will Be Different**
> The refs (e.g., `ff75b7f`) will be different in your output.

You do not have an `experimental` branch locally. However, if you check out a branch that does *not* exist locally but *does* exist remotely, Git will assume you want to track that remote branch.

```
3   git checkout experimental
4   git branch -a -vv
```

> **More Than One Remote With The Same Branch?**
> If more than one remote has the same name, Git will not perform this matching. In that case, you would have to run the full command that follows.

Alternatively, you could track a completely different branch if you specify it:

```
5   git checkout -b alicemaster --track origin/master
```

This is, assuming it's the `origin`'s `master` branch that you want to track.

Terminal 1

Terminal

Now that you've checked out and tracked the remote `experimental` branch in your submodule, a change has taken place in `bob_repo`. If you return to `bob_repo`'s root folder and run `git diff`, you will see that the subproject commit of `alicelib` has changed:

```
6   cd ..
7   git diff
```

Terminal 1

Terminal

The commit reference now matches the `experimental` reference.

Note that "`bob_repo`" tracks the *specific commit* and not the remote branch. This means that changes to "`alicelib`" in the origin repository are not automatically tracked within `bob_repo`'s submodule. This is a crucial difference with submodules; they are static in that they are pointed at a specific point in the history until you move it on.

You want to commit this change to the submodule:

```
8   git commit -am 'alicelib moved to experimental'
```

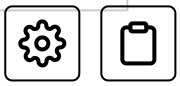
Terminal 1

Terminal

Alice Makes a Change

Alice now spots a bug in her experimental branch that she wants to fix:

```
9   cd ../alicelib
10  git checkout experimental
11  echo 'D' >> file1
12  git commit -am 'D - a fix added'
```



Terminal 1

Terminal

There is a mismatch between `alicelib`'s `experimental` branch and `bob_repo`'s `experimental` branch.

```
13  cd ../bob_repo/alicelib
14  git status
```

`git status` reports that `bob_repo`'s `alicelib` is up-to-date with `origin/experimental`. Remember that `origin/experimental` is the locally-stored representation of `alicelib`'s `experimental` branch. Since you have not contacted `alicelib` to see if there are any updates, this is still the case.

To get the latest changes, you can perform a fetch and merge or save time by running a `git pull`, which does both:

```
15  git pull
```

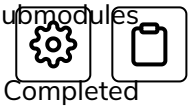
Terminal 1

Terminal

Warning!
Generally, I would advise against editing repositories that are checked out as submodules until you are more experienced with Git. You may quickly find yourself in a `detached HEAD` state and confused about what you've done.

The 'git submodule' Command

Cloning a Project With Submodules



Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/git-tracks-the-submodule’s-state__git-submodules__learn-git-the-hard-way)