Working with remotes

In this section you will:

- add a remote repository to your local clone
- fetch remote information
- checkout a remote branch
- merge an upstream branch

Managing remotes

Your repository on GitHub is the **remote** for the clone on your local machine. By default, your clone refers to that remote as **origin**. At the moment, it's the only remote you have:

```
$ git remote
origin

$ git remote show origin
    * remote origin
      Fetch URL: git@github.com:evildmp/afraid-to-commit.git
      Push  URL: git@github.com:evildmp/afraid-to-commit.git
      HEAD branch: master
      Remote branches:
        amend-my-name tracked
        master        tracked
      Local branch configured for 'git pull':
        master merges with remote master
  Local refs configured for 'git push':
    amend-my-name pushes to amend-my-name (up to date)
    master        pushes to master        (up to date)
```

It's very useful for Git to know about other remote repositories too. For example, at the end of the previous section, we considered a conflict between your GitHub fork and and the upstream GitHub repository. The only way to fix that is locally, on the command line, and by being able to refer to both those remotes.

Now you *can* refer to a remote using its full address:

> https://github.com/evildmp/afraid-to-commit

But just as your remote is called **origin**, you can give any remote a more memorable name. Your own **origin** has an upstream repository (mine); it's a convention to name that **upstream**.

Add a remote

```
git remote add upstream git@github.com:evildmp/afraid-to-commit.git
```

Fetch remote data

The remote's there, but you don't yet have any information about it. You need to **fetch** that:

```
git fetch upstream
```

This means: get the latest information about the branches on **upstream**.

List remote branches

`git branch` shows your local branches.

To see a list of them all, including the remote branches:

```
git branch -a
```

Checkout a remote branch

You'll have seen from `git branch -a` that there's a branch called *additional-branch* on the **upstream** repository.

You can check this out:

```
git checkout -b additional-branch upstream/additional-branch
```

This means: create and switch to a new local branch called *additional-branch*, based on branch *additional-branch* of the remote **upstream**.

Managing *master* on the commandline

Until now, you have only updated your *master* on GitHub using GitHub itself. Sometimes it will be much more convenient to do it from your commandline. There are various ways to do it, but here's one:

```
git checkout master # switch back to the master branch
git fetch upstream # update information about the remote
git merge upstream/master # merge the changes referred to by
upstream/master
```

`git status` will tell you that your local master is ahead of your *master* at **origin**.

Push your changes to master:

```
git push origin master
```

And now your *master* on GitHub contains everything my *master* does; it's up-to-date and clean.