

Lesson 4 - Working Remotely

We're almost done! Phew. We've covered a lot of ground, but there's still another important lesson, and that is embracing what some people call the 4th state of git - the remote repository. There are many ways to do this, and sometimes they all seem sort of similar but different in frustrating ways, i.e. you might think you are following a work-flow that worked for you only to run into some weird unexplained git error.

Important note here - git is huge and can be frustrating - but since it's huge and since it was developed by an unber-nerdy crowd, there are a huge amount of resources out there. My first thing to do if I see an error I don't understand, is to type it into google, and start browsing the answers on StackOverflow. At current count, there are 88,555 questions tagged with `git` on stack overflow. Chances are, the answer is out there!

At any rate, one workflow to connect a local repo to GitHub is as follows (n.b. this is done at the start):

1. on github, add a new repository. I typically add a R themed .gitignore file, and a README
2. copy its url (http or ssh - I prefer ssh) to clone it
3. Navigate to the parent folder, clone the repo
4. At this point you can either `cd` into the repo and start work, or if you want to work in RStudio, simply start a new project with the "Existing Directory" option chose the repo and the git tab will appear.

You can also bypass the cloning from the command line be choosing the option to clone from a repo, but make sure all your ssh keys/ducks are in a row.

Now when you look at the git tab in RStudio, you will see one important difference from before - the Push/Pull tabs are live, i.e. they are not greyed out.

However, what if you already have a local repo that is not connected to GitHub? Sort of like we have here. First type this:

```
git remote -v
```

You should see nothing! How do we connect to github then? Follow these steps:

1. Go to github and make a new empty repo using the same name as the one on your computer (you can use different names, but let's keep it simple for now).
2. Copy the url of the repo - using the ssh option
3. Add the remote locally
4. Verify the connections
5. Push the changes.

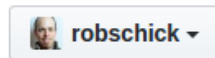
Let's see that:

Once you click on Create Repository, GitHub gives you this *really* helpful page:

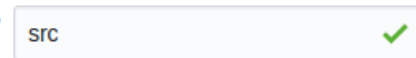
Create a new repository

A repository contains all the files for your project, including the revision history.

Owner





Repository name



Great repository names are short and memorable. Need inspiration? How about **ideal-umbrella**.

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▼

Add a license: **None** ▼



Create repository

Figure 1:

robschick / src

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Quick setup — if you've done this kind of thing before

or **HTTPS** **SSH** `https://github.com/robschick/src.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# src" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/robschick/src.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/robschick/src.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

In our case, we're going to push the existing repo from the command line (note that of course your url will be different):

```
git remote add origin https://github.com/robschick/src.git
```

Once we've added it, let's look again to see if we are talking to github:

```
git remote -v
```

I see this now:

```
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git remote -v
origin https://github.com/robshick/src.git (fetch)
origin https://github.com/robshick/src.git (push)
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$
```

Figure 2:

With it added, then let's push our changes

```
git push -u origin master
```

Wait, why did it fail?

I know I used the right password, but it's failing. Well, you'll note in my enumerated list that I said I prefer using SSH - this is why. If I look at my config file I'll see that in the remote.origin.url it's using https, but I want ssh:

I change it to ssh at the command line:

```
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git push origin master
Username for 'https://github.com': robschick
Password for 'https://robschick@github.com':
remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/robshick/src.git/'
```

Figure 3:

```
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git config --list
user.name=robschick
user.email=robschick@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/robshick/src.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$
```

Figure 4:

```
git remote set-url origin git@github.com:robschick/src.git
```

You can verify that it is right with

```
git remote -v
```

And then happiness ensues:

```
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git push origin master
Warning: Permanently added the RSA host key for IP address '192.30.255.112' to the list of known hosts.
Counting objects: 26, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (20/20), done.
Writing objects: 100% (26/26), 2.66 KiB | 0 bytes/s, done.
Total 26 (delta 7), reused 0 (delta 0)
remote: Resolving deltas: 100% (7/7), done.
To git@github.com:robschick/src.git
 * [new branch]      master -> master
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$
```

Figure 5:

And finally we can see them on GitHub to make sure all is right.

Pushing Local Changes

So now we've pushed up all of the changes we have made to date locally, so they are mirrored on GitHub. One of the biggest advantages of using git, or any distributed version control system, is that you now have multiple points of failure, instead of just one, i.e. you've increased your bus factor and this is always a good thing.

When we work with a remote repo, we now add an extra stage to our local git workflow. From time to time, we need to push our local changes to the local repository. How often you do this is a matter of personal preference. I tend to do it a lot, especially since my code is typically only being used by me. For someone like Hadley Wickham who is writing a lot of influential R code, his feeling is that pushing is essentially publishing. So each time you push, it'd better work!

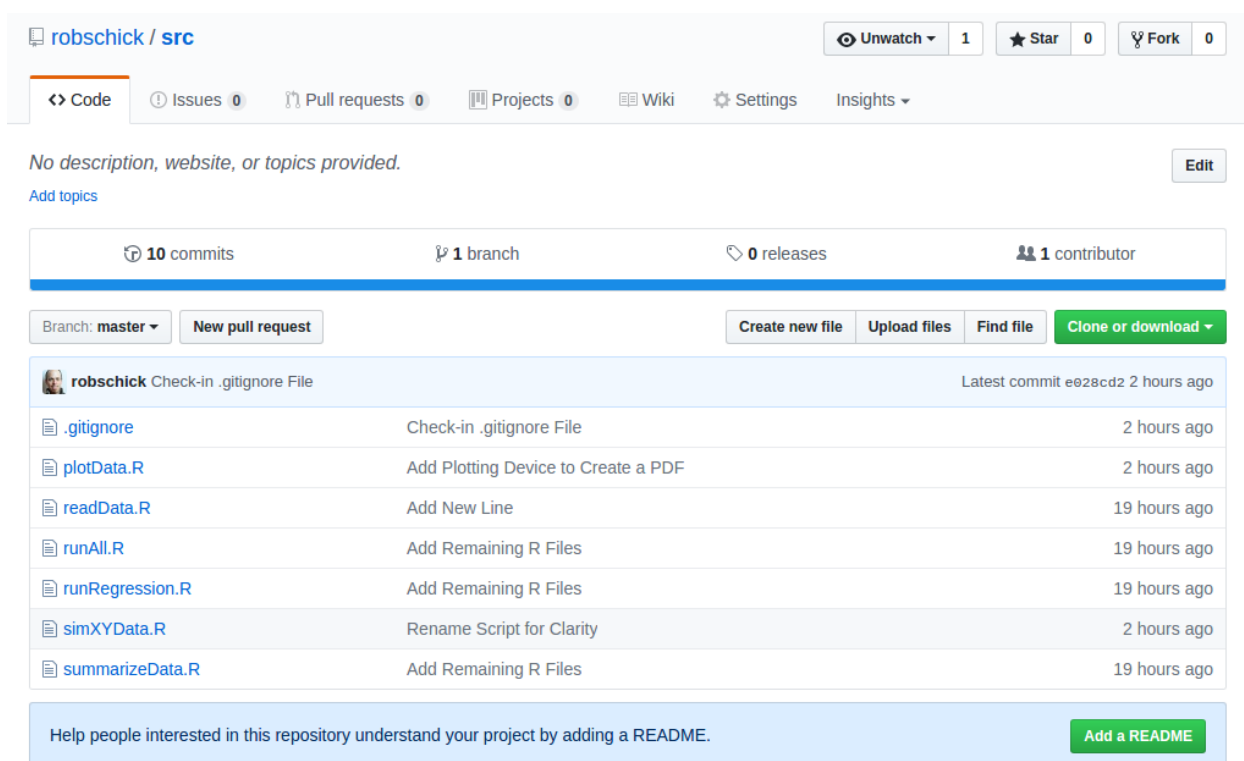


Figure 6:

Go ahead and repeat the cycle - either in RStudio, or the command line. Add some code, add some files, commit them, and push them. Then go and view them on GitHub.

Pulling Changes

One best practice is issue `git pull` before you push. This just ensures that you have the most up to date code before you push. The benefits of this will become apparent in the next section

Merge Conflict

Ah the dreaded merge conflict. These can be a pain in the butt when you first encounter them, but all in all, they are pretty straightforward. As you get more advanced with git, you can configure a graphical merge tool if you want to help with this. They can be great, but for now, we'll just handle this on the command line.

What is a merge conflict? It's essentially what happens when the repository gets out of sync. Let's say you make a change on your local computer, and push it to github, but then you working on a node, and you make a change there - but you haven't yet brought down the most up to date changes from GitHub. When you go to push your changes up to the node, you will get rejected, and then you have to manually resolve the conflict.

We don't have a node here, but we can mimic it well enough with a local repo and GitHub:

1. In your repo, make a change to a file, add it, and commit it, but don't push it to GitHub.
2. Then, navigate to github, and make a change to the same file. Use the pencil icon to edit it directly
3. Add a commit message, and commit the change

4. Navigate back to the command line and issue the `git push` command
5. If you are lucky, all hell will break loose

Here's what I have on GitHub

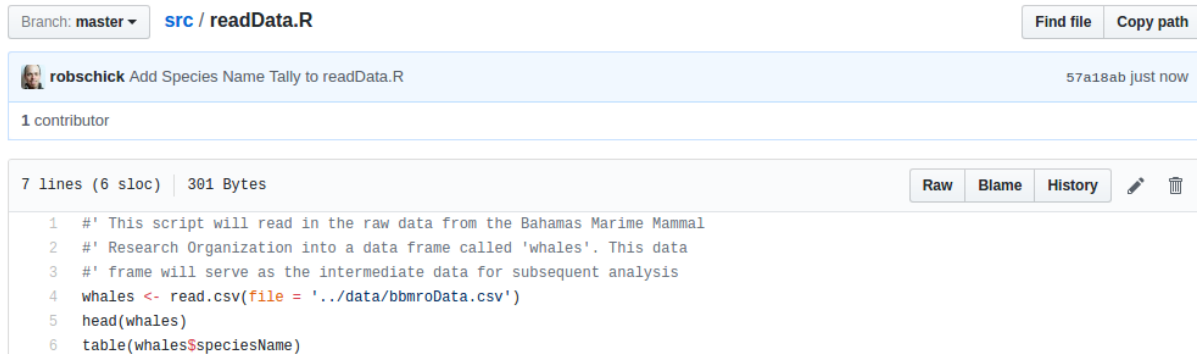


Figure 7:

And what I have locally:

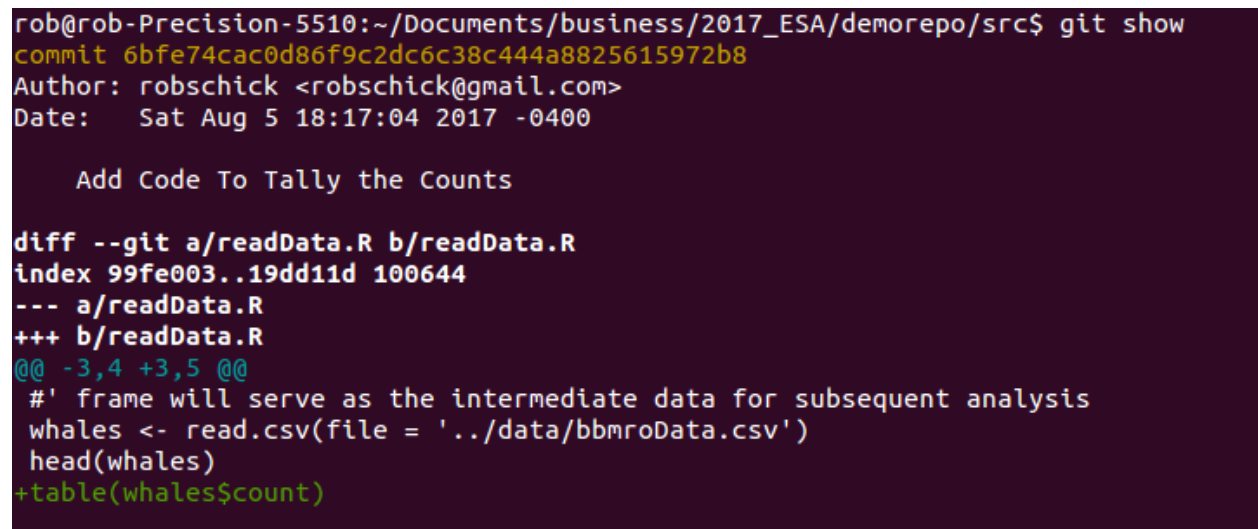


Figure 8:

Ok, let's give it a whirl:

```
git push origin master
```

Sad times, but we can fix the problem. First, let's pull.

```
git pull origin master
```

And here we see the conflict:

```
rob@rob-Precision-5510: ~/Documents/business/2017_ESA/demorepo/src
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git push origin master
To git@github.com:robschick/src.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:robschick/src.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$
```

Figure 9:

```
rob@rob-Precision-5510: ~/Documents/business/2017_ESA/demorepo/src
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git pull origin master
From github.com:robschick/src
* branch          master      -> FETCH_HEAD
Auto-merging readData.R
CONFLICT (content): Merge conflict in readData.R
Automatic merge failed; fix conflicts and then commit the result.
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$
```

Now we have to fix them manually, which we do in a text editor

```
readData.R (~/Documents/business/2017_ESA/demorepo/src) - VIM
''' This script will read in the raw data from the Bahamas Marine Mammal
#' Research Organization into a data frame called 'whales'. This data
#' frame will serve as the intermediate data for subsequent analysis
whales <- read.csv(file = '../data/bbmroData.csv')
head(whales)
<<<<<<< HEAD
table(whales$count)

=====
table(whales$speciesName)
>>>>>>> 57a18ab1334773b2033b00edc2f4de771346366c
~
~
```

Figure 10:

Once it's resolved, then we go through the regular cycle again:

```
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ vi readData.R
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git add readData.R
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git commit -m "Resolve Merge Conflict"
[master 2cf2829] Resolve Merge Conflict with GitHub Code
rob@rob-Precision-5510:~/Documents/business/2017_ESA/demorepo/src$ git push origin master
```