Adrenal Paper Code

Ignacio del Valle

19/03/2020

Document adapted from https://happygitwithr.com/

Part 1 Install or upgrade R and RStudio

1. Confirm R installation. Update if necessary

```
 R.version.string
[1] "R version 3.6.3 (2020-02-29)"
```

2. Install RStudio Desktop (upgrade to latest Preview version):

   Download it here: https://www.rstudio.com/products/rstudio/download/preview/

   Unless you have a specific reason to prefer the released version, try the Preview. RStudio is fairly conservative with official releases, so the Preview version is used by many people for their daily work.

   The Preview version updates much more frequently (and in smaller increments) than the released version. This is something you might update once **every month** or so.

3. Update R packages:

```
update.packages(ask = FALSE, checkBuilt = TRUE)
```

Part 2 Install and configure Git

1. Confirm installation:

```
which git
## /usr/bin/git
git --version
## git version 2.17.1
```

2. Set Git parameters if needed:

   http://swcarpentry.github.io/git-novice/02-setup/

```
git config --global user.name 'Ignacio del Valle'
git config --global user.email 'ignacio.delvalle.torres@gmail.com'
git config --global --list
## user.name=Ignacio del Valle
## user.email=ignacio.delvalle.torres@gmail.com
## diff.tool=p4merge
## core.editor=code --wait
## difftool.p4merge.path=/home/nacho/src/p4v-2017.3.1590419/bin/p4merge
```

```
## difftool.prompt=false
## merge.tool=p4merge
## mergetool.p4merge.path=/home/nacho/src/p4v-2017.3.1590419/bin/p4merge
## mergetool.prompt=false
## alias.hist=log --oneline --graph --decorate --all
```

3. Install a Git client

   Git is really just a collection of individual commands you execute in the shell.

   This interface is not appealing for everyone. Some may prefer to do Git operations via a client with a graphical interface.

   There are several clients- I am using **GitKraken** (works on Windows, macOS and Linux)

Part 3 Testing Git and GitHub installation

3.1 Make a repo on GitHub

Go to https://github.com and make sure you are logged in.

Click green "New repository" button. Or, if you are on your own profile page, click on "Repositories", then click the green "New" button.

How to fill this in:

- Repository name: `myrepo` (or whatever you wish, we'll delete this soon anyway).
- Description: "testing my setup" (or whatever, but some text is good for the README).
- Public.
- YES Initialize this repository with a README.

For everything else, just accept the default.

Click big green button "Create repository."

Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button.

3.2 Clone the repo to your local computer

Go to the shell.

Clone `myrepo` from GitHub to your computer. This URL should have **your GitHub username** and the name of **your practice repo**.

```
(base) nacho@xps:~/Desktop$ git clone https://github.com/idelvalle/myrepo.git
Cloning into 'myrepo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

Get information on its connection to GitHub:

```
(base) nacho@xps:~/Desktop$ cd myrepo/
(base) nacho@xps:~/Desktop/myrepo$ ls
README.md
(base) nacho@xps:~/Desktop/myrepo$ head README.md
```

```
# myrepo
testing my setup
(base) nacho@xps:~/Desktop/myrepo$ git remote show origin
* remote origin
  Fetch URL: https://github.com/idelvalle/myrepo.git
  Push  URL: https://github.com/idelvalle/myrepo.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up-to-date)
```

3.3 Make a local change, commit, and push

Add a line to README and verify that Git notices the change:

```
(base) nacho@xps:~/Desktop/myrepo$ echo "A line I wrote on my local computer" >> README.md
(base) nacho@xps:~/Desktop/myrepo$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Stage ("add") and commit this change and push to your remote repo on GitHub. If you're a new GitHub user, you will be challenged for your GitHub username and password. Provide them!

```
(base) nacho@xps:~/Desktop/myrepo$ git add -A # stages all changes
(base) nacho@xps:~/Desktop/myrepo$ git commit -m "A commit from my local computer"
[master 6ab2dfd] A commit from my local computer
 1 file changed, 1 insertion(+)
(base) nacho@xps:~/Desktop/myrepo$ git push
Username for 'https://github.com': idelvalle
Password for 'https://idelvalle@github.com':
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/idelvalle/myrepo.git
   fc20cc0..6ab2dfd  master -> master
```

The `-m "blah blah blah"` piece is very important! Git requires a commit message for every commit, so if you forget the `-m` flag, Git will prompt you for a commit message anyway.

Using the `>>` (append) operator is the easiest way to append text to a filefrom the computer

3.4 Confirm the local change propagated to the GitHub remote

Go back to the browser. I assume we're still viewing your new GitHub repo.

Refresh.

You should see the new "A line I wrote on my local computer" in the README.

If you click on "commits," you should see one with the message "A commit from my local computer."

3.5 Clean up

a) Local

When you're ready to clean up, you can delete the local repo any way you like. It's just a regular directory on your computer.

Here's how to do that in the shell, if current working directory is `myrepo`:

```
cd ..
rm -rf myrepo/ # -f ignore nonexistent files and arguments, never prompt
```

b) GitHub

In the browser, go to your repo's landing page on GitHub. Click on "Settings".

Scroll down, click on "delete repository," and do as it asks.

3.6 SSH keys

You should swap out your SSH keys periodically. Something like once a year.

Don't do weird gymnastics in order to have only one key pair, re-used over multiple computers. You should probably have one key per computer. It is normal to associate multiple public keys with your GitHub account. For example, one public key for each computer you connect with.

3.6.1 Do you already have keys?

From RStudio

Go to **Tools > Global Options... > Git/SVN**. If you see something like `~/.ssh/id_rsa` in the SSH RSA Key box, you definitely have existing keys.

From the Shell

List existing keys:

```
(base) nacho@xps:~/Desktop$ ls -al ~/.ssh/
total 32
drwx------  2 nacho nacho 4096 Jan 31 20:50 .
drwxr-xr-x 81 nacho nacho 4096 Mar 18 12:18 ..
-rw-rw-r--  1 nacho nacho  277 Jan 31 20:50 config
-rw-------  1 nacho nacho 3326 May 20  2018 id_rsa
-rw-r--r--  1 nacho nacho  759 May 20  2018 id_rsa.pub
-rw-r--r--  1 nacho nacho 5288 Mar 18 17:14 known_hosts
-rw-rw-r--  1 nacho nacho 1024 Apr 18  2018 .known_hosts.swp
```

If you are told `~/.ssh/` doesn't exist, you don't have SSH keys.

If you see a pair of files like `id_rsa.pub` and `id_rsa`, you have a key pair already.

3.6.2 Create an SSH key pair

From RStudio

Go to **Tools > Global Options... > Git/SVN > Create RSA Key...**.

Click "Create" and RStudio will generate an SSH key pair, stored in the files `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`.

From the Shell

Create the key pair like so, but substitute a comment that means something to you, especially if you'll have multiple SSH keys in your life. Consider the email associated with your GitHub account or the name of your computer.

```
ssh-keygen -t rsa -b 4096 -C "COMMENT"
```

Accept the proposal to save the key in the default location (press Enter).

You have the option to protect the key with a passphrase. It is optional, but also a best practice.

3.6.3 Add key to ssh-agent

Things get a little OS-specific around here. When in doubt, consult GitHub's instructions for SSH, which is kept current for Mac, Windows, and Linux.

Mac OS

Make sure ssh-agent is enabled:

```
eval "$(ssh-agent -s)"
# Agent pid 95727
```

Add your key. If you set a passphrase, you'll be challenged for it here. Give it. The `-K` option stores your passphrase in the keychain.

```
$ ssh-add -K ~/.ssh/id_rsa
```

f you're using a passphrase AND on macOS Sierra 10.12.2 and higher, you need to do one more thing. Create a file `~/.ssh/config` with these contents:

```
Host *
 AddKeysToAgent yes
 UseKeychain yes
```

This should store your passphrase *persistently* in the keychain. Otherwise, you will have to enter it every time you log in.

Linux

In a shell, make sure ssh-agent is running:

```
(base) nacho@xps:~/Desktop$ eval "$(ssh-agent -s)"
Agent pid 28406
```

Add your key.

```
ssh-add ~/.ssh/id_rsa
```

3.6.4 Provide public key to GitHub

RStudio to clipboard

Go to *Tools > Global Options... > Git/SVN*. If your key pair has the usual name, `id_rsa.pub` and `id_rsa`, RStudio will see it and offer to **View public key**.

Do that and accept the offer to copy to your clipboard. If your key pair is named differently, use another method.

Shell to clipboard

Copy the public key onto your clipboard.

For example, open `~/.ssh/id_rsa.pub` in an editor and copy the contents to your clipboard.

Or do one of the following at the command line:

- Mac OS: `pbcopy < ~/.ssh/id_rsa.pub`
- Linux: `xclip -sel clip < ~/.ssh/id_rsa.pub`

On GitHub

Make sure you're signed into GitHub.

Click on your profile pic in upper right corner and go *Settings*, then *SSH and GPG keys*. Click "New SSH key". Paste your public key in the "Key" box. Give it an informative title, presumably related to the comment you used above, during key creation. For example, you might use `2020-xps` to record the year and computer. Click "Add SSH key".

To confirm everything is correct you can use `ssh -T git@github.com` to test your connection to GitHub.

Part 4 Connect RStudio to Git and GitHub

4.1 Create a repo on GitHub

Go to https://github.com and make sure you are logged in.

Click the green "New repository" button. Or, if you are on your own profile page, click on "Repositories", then click the green "New" button.

How to fill this in:

- Repository name: `myrepo` (or whatever you wish, we'll delete this soon anyway).
- Description: "testing my setup" (or whatever, but some text is good for the README).
- Public.
- YES Initialize this repository with a README.

For everything else, just accept the default.

Click the big green button "Create repository."

Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button.

4.2 Clone the new GitHub repository to your computer via RStudio

In RStudio, start a new Project:

- File > New Project > Version Control > Git

  In "Repository URL", paste the URL of your new GitHub repository. It will be something like this

  `https://github.com/idelvalle/myrepo.git`
- Accept the default project directory name, e.g. `myrepo`, which coincides with the GitHub repo name.
- Check "Open in new session".
- Click "Create Project".

You should find yourself in a new local RStudio Project that represents the new test repo we just created on GitHub. This should download the `README.md` file from GitHub. Look in RStudio's file browser pane for the `README.md` file

4.3 Make local changes, save, commit

From RStudio, modify the `README.md` file, e.g., by adding the line "This is a line from RStudio". Save your changes.

Commit these changes to your local repo. How?

From RStudio:

- Click the "Git" tab in upper right pane.
- Check "Staged" box for `README.md`.
- If you're not already in the Git pop-up, click "Commit".
- Type a message in "Commit message", such as "Commit from RStudio".
- Click "Commit".

4.4 Push your local changes online to GitHub

Click the green "Push" button to send your local changes to GitHub. If you are challenged for username and password, provide them (but see below). You should see some message along these lines.

```
>>> /usr/bin/git push origin HEAD:refs/heads/master
To https://github.com/idelvalle/myrepo.git
   e8bfa2f..065c8b4  HEAD -> master
```

4.5 Confirm the local change propagated to the GitHub remote

Go back to the browser. I assume we're still viewing your new GitHub repo.

Refresh.

You should see the new "This is a line from RStudio" in the README.

If you click on "commits", you should see one with the message "Commit from RStudio".

Part 5 New Project-GitHub first

5.1 Make a repo on GitHub

Do this once per new project.**

Go to https://github.com and make sure you are logged in.

Click green "New repository" button. Or, if you are on your own profile page, click on "Repositories", then click the green "New" button.

- Repository name: `myrepo` (or whatever you wish)
- Public
- YES Initialize this repository with a README

Click the big green button "Create repository."

Copy the HTTPS clone URL to your clipboard via the green "Clone or Download" button. Or copy the SSH URL if you chose to set up SSH keys.

5.1 New RStudio Project via git clone

In RStudio, start a new Project:

- *File > New Project > Version Control > Git.* In the "repository URL" paste the URL of your new GitHub repository. It will be something like this `https://github.com/idelvalle/Adrenal.git`.
- Be intentional about where you create this Project.
- Suggest you "Open in new session".
- Click "Create Project" to create a new directory, which will be all of these things:
  - a directory or "folder" on your computer
  - a Git repository, linked to a remote GitHub repository
  - an RStudio Project
- **In the absence of other constraints, I suggest that all of your R projects have exactly this set-up.**

This should download the `README.md` file that we created on GitHub in the previous step. Look in RStudio's file browser pane for the `README.md` file.

There's a big advantage to the "GitHub first, then RStudio" workflow: the remote GitHub repo is added as a remote for your local repo and your local `master` branch is now tracking `master` on GitHub. This is a technical but important point about Git. The practical implication is that you are now set up to push and pull. No need to fanny around setting up Git remotes and tracking branches on the command line.

5.2 Make local changes, save, commit

Do this every time you finish a valuable chunk of work, probably many times a day.

From RStudio, modify the `README.md` file, e.g., by adding the line "This is a line from RStudio". Save your changes.

Commit these changes to your local repo. How?

- Click the "Git" tab in upper right pane
- Check "Staged" box for any files whose existence or modifications you want to commit.
  - To see more detail on what's changed in file since the last commit, click on "Diff" for a Git pop-up
- If you're not already in the Git pop-up, click "Commit"
- Type a message in "Commit message", such as "Commit from RStudio".

- Click “Commit”

5.3 Push your local changes to GitHub

Do this a few times a day, but possibly less often than you commit.

You have new work in your local Git repository, but the changes are not online yet.

This will seem counterintuitive, but first let’s stop and pull from GitHub.

Why? Establish this habit for the future! If you make changes to the repo in the browser or from another machine or (one day) a collaborator has pushed, you will be happier if you pull those changes in before you attempt to push.

Click the blue “Pull” button in the “Git” tab in RStudio. I doubt anything will happen, i.e. you’ll get the message “Already up-to-date.” This is just to establish a habit.

Click the green “Push” button to send your local changes to GitHub. You should see some message along these lines.

```
>>> /usr/bin/git push origin HEAD:refs/heads/master
To https://github.com/idelvalle/Adrenal.git
   306e851..0e36055  HEAD -> master
```

5.4 Confirm the local change propagated to the GitHub remote

Go back to the browser. I assume we’re still viewing your new GitHub repo.

Refresh.

You should see the new “Created on March 2020” in the README.

If you click on “commits,” you should see one with the message “Included creation date from RStudio”.

5.5 Make a change on GitHub

Click on README.md in the file listing on GitHub.

In the upper right corner, click on the pencil for “Edit this file”.

Add a line to this file, such as “Created on March 2020.”

Edit the commit message in “Commit changes” or accept the default.

Click the big green button “Commit changes.”

5.6 Pull from GitHub

Back in RStudio locally . . .

Click the blue Pull button.

Look at README.md again. You should now see the new line there.

5.7 The end

Now just ... repeat. Do work somewhere. Commit it. Push it or pull it* depending on where you did it, but get local and remote "synced up". Repeat.

* Note that in general (and especially in future when collaborating with other developers) you will usually need to pull changes from the remote (GitHub) before pushing the local changes you have made. For this reason, it's a good idea to try and get into the habit of pulling before you attempt to push.

Part 6 Analysis

6.1 FASTQC

Run fastqc for all of the samples:

```
find . -name "*.fastq.gz" | parallel fastqc -o {//}/ {} # Run in same folder where files are located
```

6.2 STAR Index Generation

STAR index generated in myriad (sequences are 43 bp long) using:

```
#!/bin/bash -l
#$ -S /bin/bash
#$ -l h_rt=02:00:00
#$ -l mem=124G
#$ -N star_index
#$ -pe smp 12
#$ -wd /home/sejjide/Scratch/genome/star_43bp

module load star/2.5.2a


STAR --runMode genomeGenerate --runThreadN 24 --genomeDir ~/Scratch/genome/star_43bp \
--genomeFastaFiles  ~/Scratch/genome/Homo_sapiens.GRCh38.dna.primary_assembly.fa \
--sjdbGTFfile ~/Scratch/genome/Homo_sapiens.GRCh38.86.gtf --sjdbOverhang 42
```

From the STAR manual:

2.2.1 Which chromosomes/scaffolds/patches to include?

It is strongly recommended to include major chromosomes (e.g., for human chr1-22,chrX,chrY,chrM,) as well as un-placed and un-localized scaffolds. Typically, un-placed/un-localized scaffolds add just a few MegaBases to the genome length, however, a substantial number of reads may map to ribosomal RNA (rRNA) repeats on these scaffolds. These reads would be reported as unmapped if the scaffolds are not included in the genome, or, even worse, may be aligned to wrong loci on the chromosomes. **Generally, patches and alternative haplotypes should not be included in the genome.** Examples of acceptable genome sequence files: • ENSEMBL: files marked with .dna.primary.assembly, such as:

ftp://ftp.ensembl.org/ pub/release-77/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz

6.3 STAR Alignment

fastq files aligned in myriad using the following bash script:

```
#!/bin/bash -l
#$ -S /bin/bash
#$ -l h_rt=16:00:00
#$ -l mem=124G
#$ -N star
#$ -pe smp 12
#$ -wd /home/sejjide/Scratch/paper

module load star/2.5.2a

for i in $(ls *.fastq.gz | rev | cut -c 13- | rev | uniq);
do
STAR --genomeDir ~/Scratch/genome/star_43bp/ \
--sjdbGTFfile ~/Scratch/genome/Homo_sapiens.GRCh38.86.gtf \
--sjdbOverhang 42 --readFilesIn ${i}_R1.fastq.gz ${i}_R2.fastq.gz \
--readFilesCommand zcat --outSAMtype BAM SortedByCoordinate \
--limitBAMsortRAM 12000000000 --runThreadN 12 --twopassMode Basic \
--outFileNamePrefix ${i%.*} ;
done;
```

```
# Rename files
rename -v 's/Aligned\.sortedByCoord\.out//' *.bam
```

6.4 Counting reads using featureCounts