Teaching Survival Git

Enough Git to Survive as a Data Scientist

Rob Donald

Saturday 31 August 2024

Contents

Introduction

My notes on using git as a data scientist. In other words you *know and accept* you should be using git but man is it confusing and almost painful. ¯_(ツ)_/¯

I've been a software developer and data scientist for more than 30 years and I've used all sorts of version control systems (started with Apollo DSEE, look that one up). I find git confusing.

Moving from SVN to Git

- https://smartbear.com/blog/migrating-from-subversion-to-git-lessons-learned/
- https://dev.to/fpuffer/what-makes-companies-switch-from-svn-to-git-3687
- https://ohshitgit.com/

Git setup in general

- https://happygitwithr.com/
- https://happygitwithr.com/git-intro.html

GitHub Access Tokens

Github now requires you to use personal access tokens (PATs). These are a bit of a pain but described by Jenny Bryan here:

- https://happygitwithr.com/https-pat

Lightbulb Moments

Switching back and forth between main and branches

This is from the world famous Jenny Bryan [1]

- https://happygitwithr.com/git-branches.html Section 22.2

So you *can* do it and yes git **stash** is a way but she recommends the same idea I had read somewhere else about *always* doing a commit even if it is just with the comment 'WIP xxx'. But she then goes on to explain how you can get rid of that temporary commit using terminal (bash) commands like:

```
$ git checkout issue-5
$ git reset HEAD^
```

[1] If you don't know who she is look at this link https://www.rstudio.com/speakers/jenny-bryan/ and follow her on twitter: @JennyBryan.

Merge conflicts

- https://happygitwithr.com/git-branches.html Section 22.4

Daily routine

```
# fetch the changes from the remote
$ git fetch origin

# show commit logs of changes
$ git log master..origin/master

# show diffs of changes
$ git diff master..origin/master

# apply the changes by merge..
$ git merge origin/master

# .. or just pull the changes
$ git pull
```

I Just Want To Rename A File (git mv)

Hold on to your hat.

1. Make sure everything is commited i.e. you don't have any work in progress.
2. Use the `git mv` command
   - % git mv -v old-name.R new-name.R
   - Using `git mv` rather than just `mv` means the file is automatically 'staged' for you. All you need to do now is '`commit` it with a suitable message.
3. Do a git status to double check
   - % git status
4. Do the commit
   - % git commit -m"Renamed file"
   - (*you could of course use the RStudio Git menus for this bit*)

Marking Something as Significant (git tag)

If you want to be able to refer to a commit without using the odd looking SHA1 you need to use git tag. You do this ***after*** the commit which you want to mark.

```
$ git tag -a v1.4 -m "my version 1.4"
```

Don't put quotes round the -a bit

- https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag

Then once you have done that you need to `push` the tag up to the remote.

```
$ git push --tags
```

If you just want to see what tags you have

```
$ git tag -l
```

Which only give you the tag 'ref name'

What you probably wanted was this

```
$ git tag --format "%(refname:short) | %(taggerdate:iso) [%(contents:subject)]"
  --sort=-taggerdate:iso
```

Do the above all on one line. How painful is that ¯_(ツ)_/¯.

Tagging A Specific Commit (with the correct date)

Sometime you may want to clean things up by putting a tag on an *older* commit and make it fit in with some naming scheme you have come up with. This link and bash code snippet will do that *importantly making the tag use the correct date of the older commit*.

See this link

- https://stackoverflow.com/questions/4404172/how-to-tag-an-older-commit-in-git/21759466#21759466

```
# Set the HEAD to the old commit that we want to tag
git checkout 9fceb02

# temporarily set the date to the date of the HEAD commit, and add the tag
GIT_COMMITTER_DATE="$(git show --format=%aD | head -1)" \  <--- NOTE The line continuation
git tag -a v1.2 -m"v1.2"

# I.E the above two lines are a SINGLE line at the command prompt.
# TWO commands will not work.

# push to origin
git push origin --tags

# set HEAD back to whatever you want it to be
git checkout master
```

How Do I Compare Two PDF Output Files

Well you would think this is easy eh? But the problem is the fact that it is a .pdf file i.e. ***not*** a text file. So what you need to do is get the older version's commit SHA1 and then extract that particular commit to somewhere different. Note this command below is extracting the *whole folder structure* at the point in time from your SHA1.

From this link:

- https://stackoverflow.com/questions/11018411/how-do-i-export-a-specific-commit-with-git-archive

```
$ git archive --format zip --output ./CompareResults/Cmp-01.zip 4081d45d
```

Using the above command you have the whole repo as at SHA1 `4081d45d` in the zip file `Cmp-01.zip`

You can of course just pull out a specific file using this command:

```
$ git archive --format zip --output ./CompareResults/SingleFile.zip 4081d45d Prospec_Stg_1_N69_ELvsDP.p
```

Getting Info

Sometimes you just wish you could ask a simple question ¯_(ツ)_/¯.

Order the branches by time and give me some context

```
$ git branch -vv --sort=-committerdate
```

Find out what branch a commit came from

This command also shows you what branches it is in

```
$ git branch --contains ec0b406e -vv --sort=-committerdate
```

Where the `ec0b406e` is the SHA1 hash you are interested in. The output is reverse time ordered so look for the bottom one to see where it started out in life.

Is my file being tracked by git?

The simple answer is:

```
$ git ls-files
```

As you'll probably expect by now you can get an uber nerd answer as well, see this:

- https://stackoverflow.com/questions/15606955/how-can-i-make-git-show-a-list-of-the-files-that-are-being-tracked/15606998

Tag Info

- https://stackoverflow.com/questions/13208734/get-the-time-and-date-of-git-tags