ILLINOIS

Introduction to Git II

Lecture 4

Dirk Eddelbuettel

STAT 430: Data Science Programming Methods (Fall 2019)

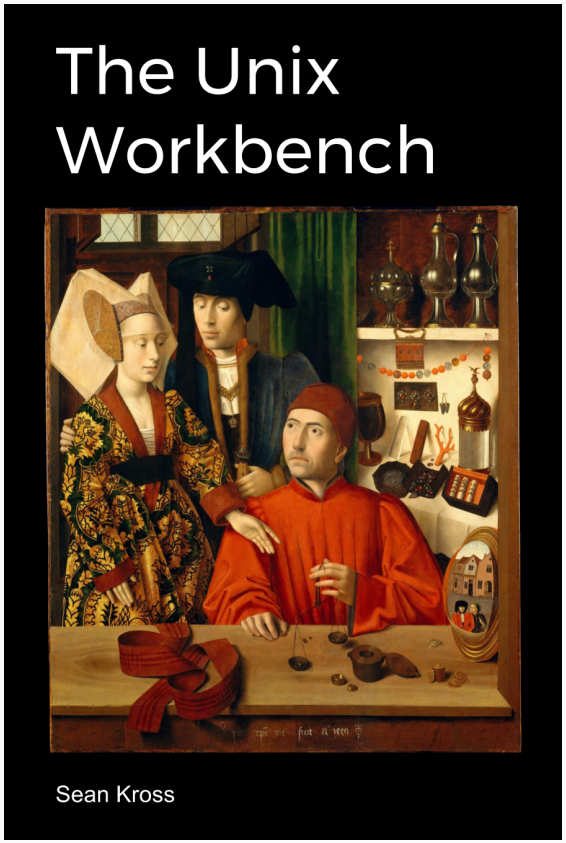
Department of Statistics, University of Illinois

`git`: Previous lecture

- We learned about several `git` commands
- We operated in a linear fashion
- We saw the ability to undo changes via `revert` or `reset`

`git`: This lecture

- branches
  - from simple 'will this work'
  - to separating alternative approaches
- remoting
- pull requests with yourself
- github, social coding, development



Please read Sections 6.5 and 6.6 from Chapter 6 on "Git and GitHub" at https://seankross.com/the-unix-workbench/

As before, not every line or paragraph is needed but there is decent overlap with what we cover here.

Source: https://xkcd.com/1296/

A key concept for `git`

- Branches existed in earlier version control systems
- But they made it “more work” and were more cumbersome
- In `git` they are one of the (if not *the*) best features
- Think of branches as “alternate views” of files and directories
- Branches are “lightweight”: easy so setup, to discard, to switch
- We switch between branches … but changes appear in place (!!)
- This sounds weird so just hold on!

Another useful setting

- Switching between branches can make you forget which branch you are in
- Useful to have the shell prompt show it
- I use a short snippet I also posted on the course site
- You can copy and paste into a file `nameoffile` and do either
  - `source nameoffile`
  - `. nameoffile`
- Or you can add it to you `~/.bashrc`

ILLINOIS

Discussed last lecture

I have the following in ~/.gitconfig :

```
[color]
    ui = true
[alias]
    st = status
    ci = commit
    co = checkout
    # the following is one long line, the \ was added to show the line break
    ls = log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset \
            %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit
    ll = log --pretty=format:\"%C(yellow)%h%Cred%d %Creset%s%Cblue [%cn]\" --decorate --numstat
    hist = log --graph --decorate --pretty=oneline --abbrev-commit
    pu = pull --all --prune
## https://dev.to/devcamilla/why-git-alias-575h
## https://dev.to/megamattmiller/the-git-aliases-that-get-me-to-friday-1cmj
    nb = checkout -b
    fp = fetch -p
```

Also see https://stat430.com/resources/snippets/

Let's start

- In a directory like the scratch directory we used last time
- Do this `git branch --all`
- You should see only one: `master` with a `*` indicating it is active
- Now do
  - `git branch some_test` followed by
  - `git branch --all` and then
  - `git checkout some_test`
- The shell prompt should change
- `git branch --all` should show a difference
- As an aside: a shortcut is `git checkout -b some_test`
- This creates the branch *and* checks out into it

Let's add content

- In the new branch, add a new file, say `third_file.txt`:
  - `touch third_file.txt # or edit with content`
  - `git add third_file.txt`
  - `git commit third_file.txt -m'adding file 3'`
  - `git status` should be clean
- Do `ls` in the directory
- Do `git branch --all` to confirm
- Then: `git checkout master` (or just `git checkout -` to get back)
- What is different when you do `ls` ?

So what do branches do?

- They allow us to 'depart' from a know state
- Make a number of changes we can test
- Here it was `file_three.txt`
- It could also (and often is) changes in a file
- At this point such an "experiment" either
  - works out and we integrate it ("merge")
  - or we discard it by discarding the branch
- So that "the mainline" (ie `master`) can either
  - take the change
  - or continue

Merge

- The easiest `merge` for us is directly on `master`

```
edd@rob:/tmp/scratch(master)$ git merge some_test
Updating 6c8a22d..d245d00
Fast-forward
 third_file.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 third_file.txt
edd@rob:/tmp/scratch(master)$
```

- Do `ls` now in master
- Do `git log` (or `git ls` if you have the alias defined)
- What changed?

Review the content around branching here.

At http://rogerdudler.github.io/git-guide/

Optional

- Try https://learngitbranching.js.org/
- This is pretty good
  - and just that little bit advanced enough to keep you on your toes
  - try the first few exercises
  - keep in mind that it is simplified
  - ie `git commit` without actual file change
- It uses the common graphical approach from many tutorials
- You can stop at `merge` or `rebase`

Remotes

Git Hosting and extra User-Interface Goodies

- GitHub is very *very* popular
- Generally free to use for open source or school projects
- Now lets you have private repos too (not visible to world)
- Same for server installation you may want to run 'at work'
- U of Illinois has some local servers
- GitHub now owned by Microsoft, that worries some: wait and see
- Alternates are provided by GitLab, BitBucket and other sites
- You can also run your own server
- In general:
  - people tend to confuse `git` and `GitHub`: not the same
  - think of `R` and `RStudio`: also not the same

GitHub Hosting

- But just how RStudio puts *a lot* of polish around R
- So does GitHub for using `git`
- We will aim to learn `git` as a *protocol* and a *tool*
- By using GitHub but keeping in mind what may be different
- First tasks:
  - Create a GitHub account
  - Create a repo
  - Make sure you understand how to authenticate: `https` or `ssh`
- In that remote repo create a file or two.

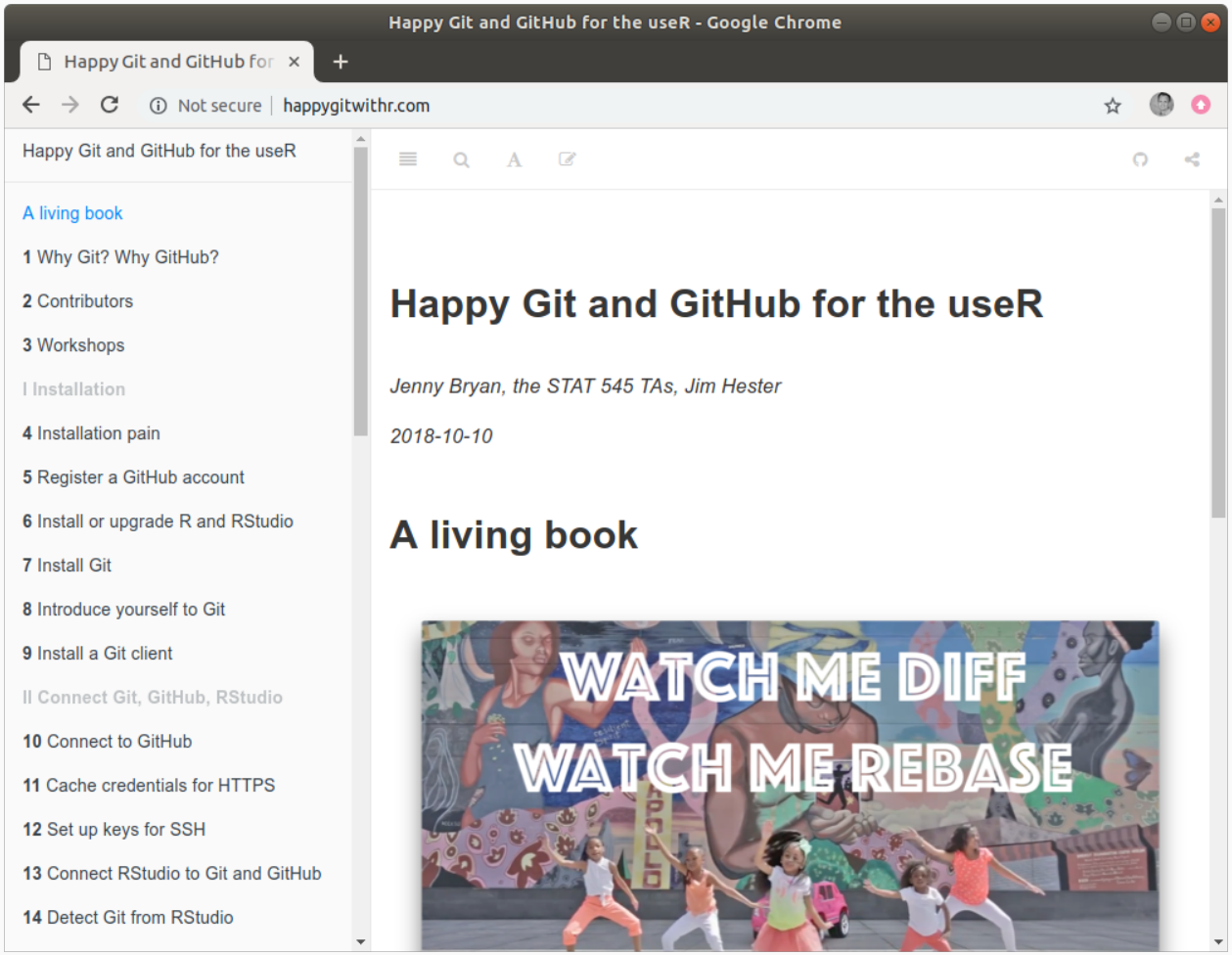
GitHub cloning

- Once you have a remote repo (yours, or another you want to work with)
- Cloning lets you take a 'read-only' copy of someone else's repo
- Cloning one of *your* own lets you write to your own
- Press the greep button, copy the address.
- Paste it to the command-line into
  - `git clone ...that address here...`
  - this will bring the remote repo here

GitHub forking

- Forking is related to cloning
- With one big difference:
  - a *distinct* copy of the repo is created at your remote
  - so “the paths diverged”: you can now do work independently
- This is often the first step to collaboration and “pull requests”:
  - first fork to create that copy
  - then clone the fork from your remote to your local machine
  - you could then push back to your remote
  - and possibly initiate a pull request from there
- Outside of our scope – but you should know the terminology

Other Git reading



Please read Sections 16, 17, 18 on working with remote projects at GitHub.

At http://happygitwithr.com/

Pull requests

- *The* bread and butter of contributing to other projects
- Also one of the cleanest ways to maintain your own project
- Setup:
  - You have local repo
  - It also exists at a remote site like GitHub (or GitLab)
  - We will now work *locally* and then ...
  - ... integrate it at the remote end

Pull requests

- Given the local and remote repo, do:
  - Create a (local) branch
  - Change a file or two. Commit two change.
  - Repeat until done.
  - Push the local branch
- The remote will now be visually distinguished at the remote end

Pull requests

- GitHub offers a button 'Create Pull Request'. *Do that.*
- It should state that the pull request ("PR") is between a branch and the `master` branch.
- We can now describe the pull request, inspect its `diff` etc.
- Once sent, we (as the owner of the repo) get alerted
- At this point we can review, comment, request changes...
- If we accept, the pull request is "merged".
- We can then pull the updated `master` branch

Merge conflicts

- Sometimes two people work on the same file at the same time
- Or sometimes you make changes in your paper on your laptop and on another machine
- Consequence: the `merge` can not process cleanly as the underlying computing of differences (cf `diff`) and applying them (cf `patch`)

Worked merge conflict

- To learn the mechanics better, let's create one
- So in a repository with
  - a local checkout, ie local files, and
  - a remote, ie at GitHub
- Change a file twice in a way that is not reconcilable:
  - e.g. change the "quick brown fox" to blue and orange
- The try to merge

Resolve a merge conflict

- The conflicted file(s) will have sections delineated by
  - `<<<<` and `>>>>`
  - you manually edit and accept one (correct or better) version
- Then do `git add` and `git commit`
- And that resolves it!

About Git(Hub)

- `git` is not the easiest revision control system
- `git` is arguably overly complicated with “too many options”
- But it works really well … and “has won” the war
- Similarly, GitHub bet on adding bells and whistles
- And that won: both Google and Microsoft had competing hosting systems
- Which they folded and they use GitHub now, Microsoft even bought it
- Familiarity with this workflow is really important in today’s world
- We will be working with this for the rest of the course.

Source: https://xkcd.com/1597/

Not an entirely uncommon workflow.

We all had our “git moments” and have our “git stories”...

Keep yourself a README or cheatsheet, write down commands that worked, reference material that was helpful ... and come back.

I still have mine...

Markdown

Next lecture

- Markdown
- Used extensively *with git*
- Used extensively *at GitHub*