Universität Basel

Version Control - RStudio meets Git and GitHub

Jana Jarecki

18 July 2022

Agenda: What today teaches or repeats

Version Control

- What the h*** is version control?
- Difference Git vs. GitHub

Git

- Set up Git (to talk to RStudio)
- What is the `status`, and `add`, what is a `commit`?
- How to go back in time?
- What is a branch and when to use it?

GitHub

- What is a repository?
- How to get code/files from the web?
- How to put code/files online to GitHub?
- What about data security?

Collaborate

- Collaborate through pull requests
- Collaborate through pushing

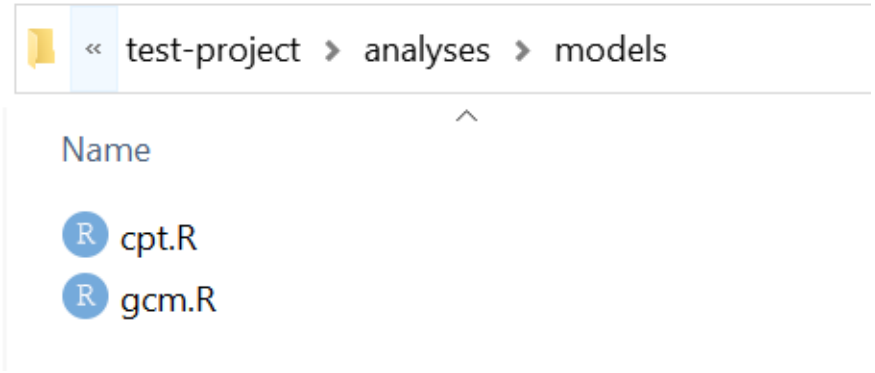
What is Git?

Git is

- A version control program
- Currently becoming the most common version control system
- `Git` enables you to do version control from the command line and RStudio
- What is version control?

No Version Control

Some files



A common workflow

a. `cpt.R` works nicely, yay

b. change `cpt.R` code,
e.g. add a bayesian implementation

c. try it out

d. *"damn I broke something"*

e. oh, I need to explore some predictions from `cpt.R`

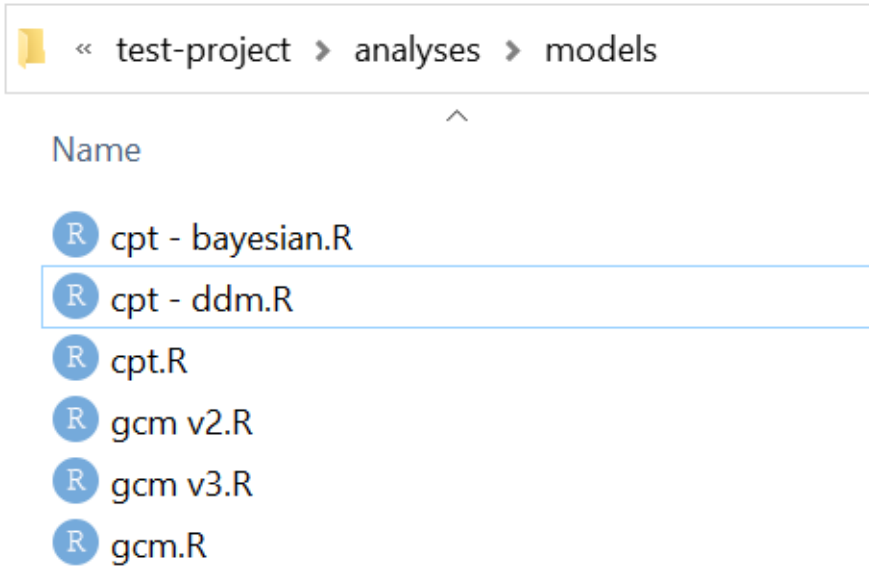
f. ok, delete the new code

g. start at a.

h. rinse and repeat

Version Control by Hand

Versioning is common



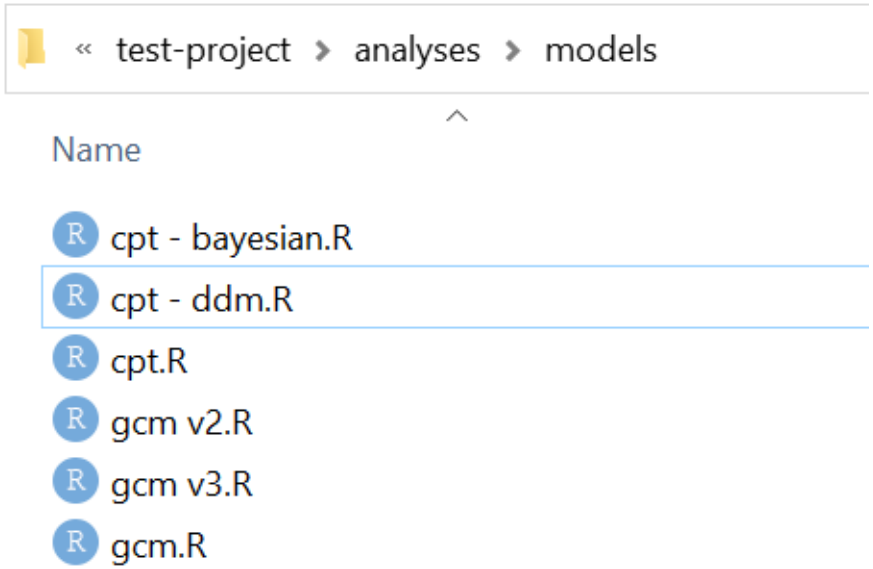
(It's a no no!)

Maybe error-prone

- What do *v1*, *v2* mean?
- Won't make long file names such as "`cpt - bayesian version with rstan`" or "`cpt - trying out integrating reinforcement learning with the model`"
- Going back to a previous version is tedious
- Oh no, I made an error in the original `cpt.R` - now I need to copy-paste the new code into all files

Version Control

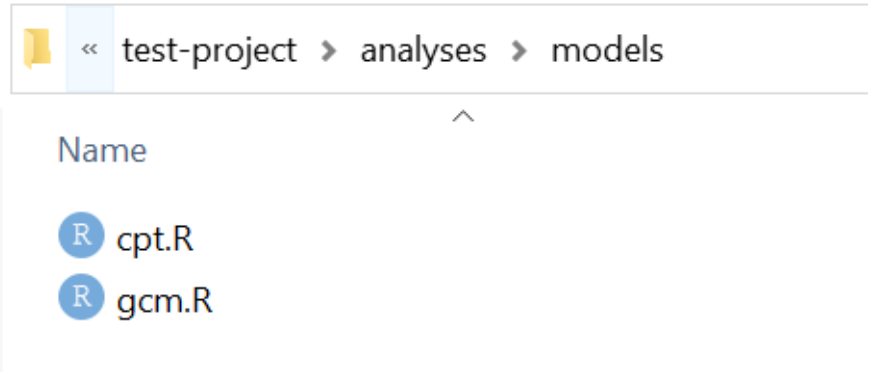
Versioning is common



Note: this goes beyond R scripts, also applies to e.g., experimental programming code

Cleaner: version control program

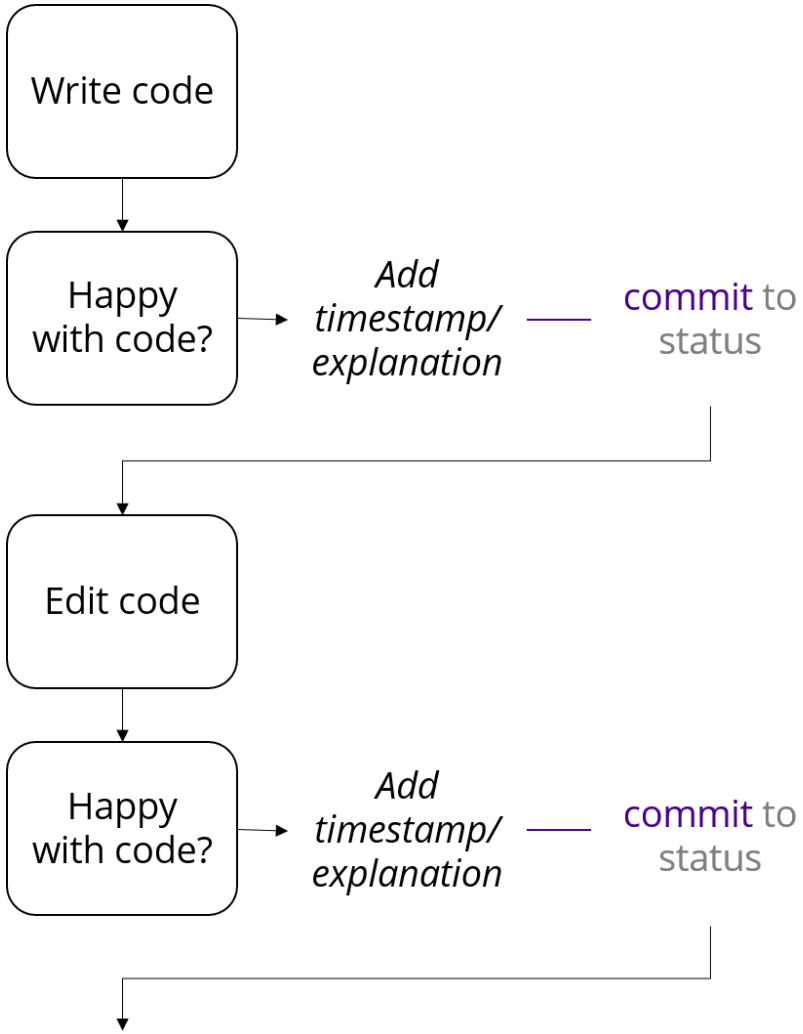
In the foreground



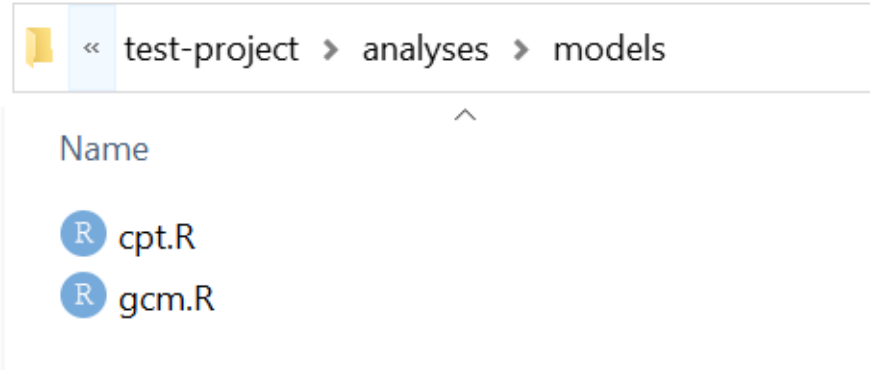
In the background: hidden **timestamp**

```
9929a8c adds ddm extension of models
59e33ba adds bayesian model versions
15e4edc adds cpt and gcm models
```

Git Workflow



In the foreground



In the background: hidden **timestamp**

```
9929a8c adds ddm extension of models
59e33ba adds bayesian model versions
15e4edc adds cpt and gcm models
```

How To Version Control - Example with RStudio

Getting Started - RStudio meets Git & GitHub

Getting Started, Git & GitHub

RStudio, meet Git

- Ensure the Git program is installed e.g., here (Mac/Win/Linux), if not please install it.
- Register a (free) online GitHub account at https://github.com/
- In RStudio run `pacman::p_load(usethis, gitcreds)` to load two packages
  - run `usethis::edit_git_config()` …
  - … which opens a configuration text file, add your name + email, save, close it
  - make a new R project, e.g. following this
  - then run `usethis::use_git()`
  - if the project has the Git repository you see a new tab "Git" next to the "Environment" tap (usually) on the top-right – the project we work in
  - Open the Terminal tab and type `git status` and *add*, *commit* files!

Git/RStudio, meet Github

- In RStudio
  - run `usethis::create_github_token()` …
  - … which opens the browser, "New personal access token". Under *note* name the token "RStudio", click "generate token", and *copy* your token. Tokens looks e.g. like `gxp_6WH1rIJt4b7qqb60trV47F8XPZy2nKI6G` (don't lose it!))
  - run `gitcreds::gitcreds_set()` and paste your *GitHub-username* and the *token* as your password (NOT the GitHub password)

If this instruction does not work

you can also try to follow from 3.1 on here

Using Git: Two Key Steps

We write some code, save our files, test it, until we are happy with some of the code (e.g., a model that does what it should, an experiment that runs smoothly, etc.)

Version conrol the files I am happy with:

a. Tell Git *which* files we're happy with:
   add files to a file list (and e.g., omit some files that we're still working on)

b. Tell Git to make the timestamp/explanation:
   commit (to) these files (with a message such as "cpt model")

Git in the Command Line

Basic workflow

`git status`: check what's the status of my files, which have changes
`git add filenames` (or `add --all`): happy with *these* files (*all* files)
`git commit -m "new model"`: commit to file status, adding timestamp/explanation "new model"

Go back in time

`git log`: view your history (`git log --oneline --graph`)
`git reset ####`: go back in time

- `git reset #### --soft`: timetravel but keep the future!
- `git reset #### --hard`: timetravel and scrap the future

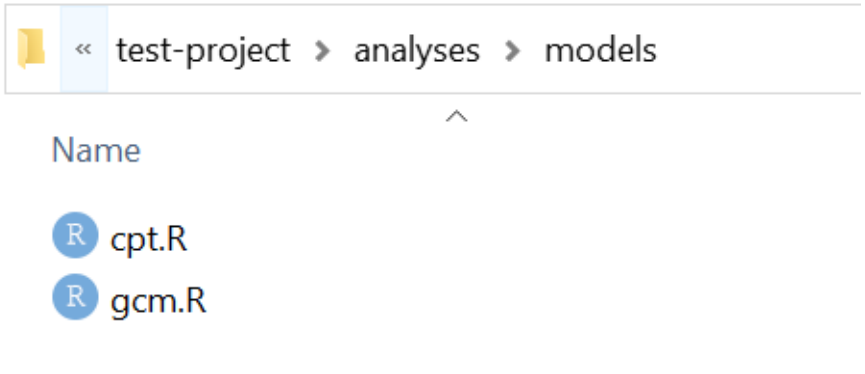
`git commit --amend`: edit the previous commit

Note the notation: `--`

Version Control: Two Possibilities

Versioning in time

In the foreground

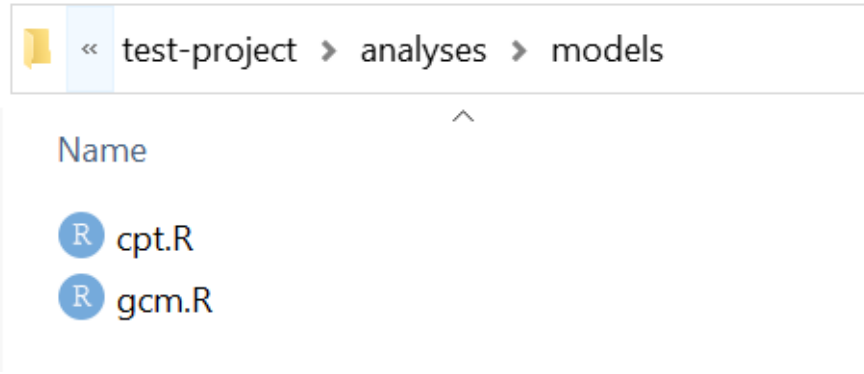


In the background: hidden **timestamp**

```
● 9929a8c adds ddm extension of models
● 59e33ba adds bayesian model versions
● 15e4edc adds cpt and gcm models
```

Versioning in parallel worlds

In the foreground



In the background: hidden **switch**

```
| ● b42e46f (ddm) adds ddm extension of models
|/
| ● 3582fdb (bayesian) adds bayesian model versions
|/
● 15e4edc adds cpt and gcm models
```

Versioning in Parallel Worlds

Basic workflow (just the same don't even look at me)

`git status`: check what's the status of my files
`git add filenames` (or `add --all`): happy with *these* files (*all* files)
`git commit -m "new model"`: commit to file status, adding timestamp/explanation "new model"

Create a parallel universe

`git branch myuniverse`: copy your files in the background
`git checkout myuniverse`: copy your files in the background

GitHub - Take Things Online

Git vs GitHub?

Git

- For **local** version control on my machine
- Think of Git as a **single** computer
- Version control for a folder (called repository)
- Don’t need an internet connection to use Git

GitHub

- For **web-based** sharing and collaborating with others
- Think of Github as a **network** of computers
- Web-based hosting service for Git folders (repositories)
- Need an internet connection to use GitHub
- You can use Git without GitHub, but you cannot use GitHub without Git

From Git to GitHub

name: clone `git clone URL`: first-time get files from GitHub online (called remotes)

Look what we got

`git remote`: see remotes
`git remote rename myname newname`: rename remotes
`git remote get-url myname`: rename remotes

Download and upload

`git pull`: download (pull) the *whole* files from GitHub online
`git push`: upload (push)

- `git push`: upload, complaines if trying to upload local changes to *previous* commits (amended previous commits)
- `git push --force`: upload, can change past commits (careful!)

What about data security?

Data privacy

Problem

- Often I have a raw data version that may contain information that could identify people
- I want to NOT have that stuff online

Solution

- I can tell Git to not track files
- In a file called `.gitignore`
- If you don't see the .gitignore file, turn on viewing of hidden files in your file explorer

Collaboration

Two basic ways

Pushing as collaborator

I can be added as *collaborator* to an online GitHub repository ***R***. Then I can:

- clone ***R*** to my local computer (see this slide)
- add and commit changes on my local computer
- directly push and pull my committed changes to the original online ***R***
- pull new changes the other made in R to my local machine

This works if the others' repository online has added me as collaborator.

Note this means I can accidentally overwrite code in ***R***, but they can always revert to their previous commit if I broke something.

Pull requests

My local changes to a cloned online repository ***R*** can be added to ***R*** by a *pull request*. Pull requests are like knocking on somebody's door and asking "hey, can I add these new edits to your code?" and the other can review and decide.

- I have cloned ***R***, made a *new branch b* (!), and in branch *b* committed edits locally on my machine
- pushed the *new branch b* to the online repository ***R***
- The can create the pull request from the github webpage more info that says "do you want to integrate the branch *b* into your existing branch?"

There is also the option to *fork* a public repository, if I am no collaborator on the repository, and do a pul request which we haven't covered here.

Version Control

More benefits

- All major software companies use version control
- Most industry requires it
- Going back to a stable version of your code (think code of experiments you may want to run!) always possible

In Science …

For instance, in one of our current projects, the reviewers asked us to re-run an experiment in which we had manipulated the available response times and add two more (slower) response time levels. We just jumped to that version of the experiment using Git and changes one number (the respons time) and could re-run it smoothly.