Reproducible research: Goals, Guidelines and Git

May 2019

Boriana Pratt
Office of Population Research (OPR)
Princeton University

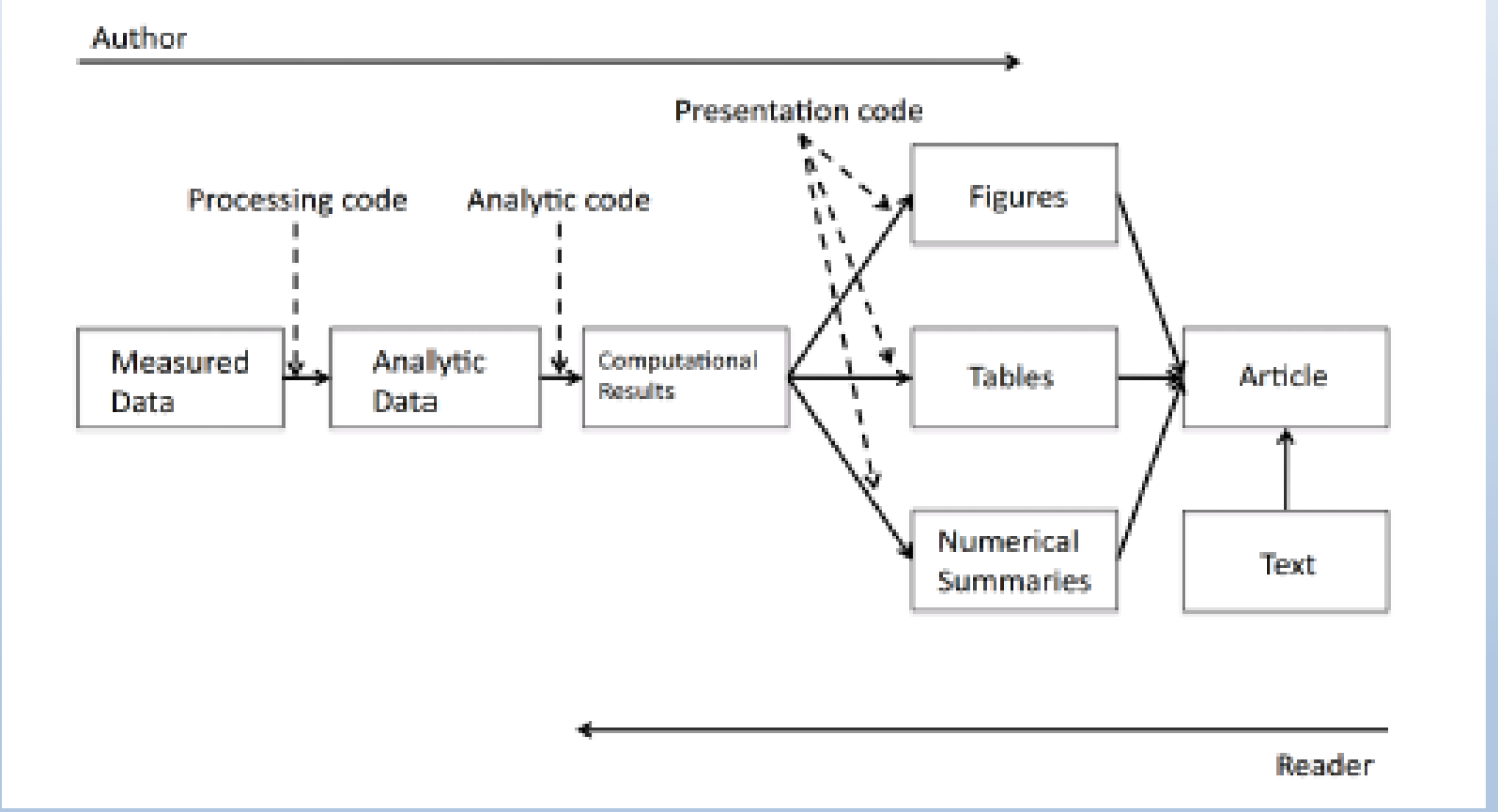
Reproducible Research

- Goals (for open and reproducible research)
- Guidelines
- Git (+Github)

Reproducible Research – example of current practice

https://www.youtube.com/watch?v=66oNv_DJuPc

Research Pipeline



Above picture by: Roger Peng (http://www.biostat.jhsph.edu/~rpeng/research.html)

Reproducible Research

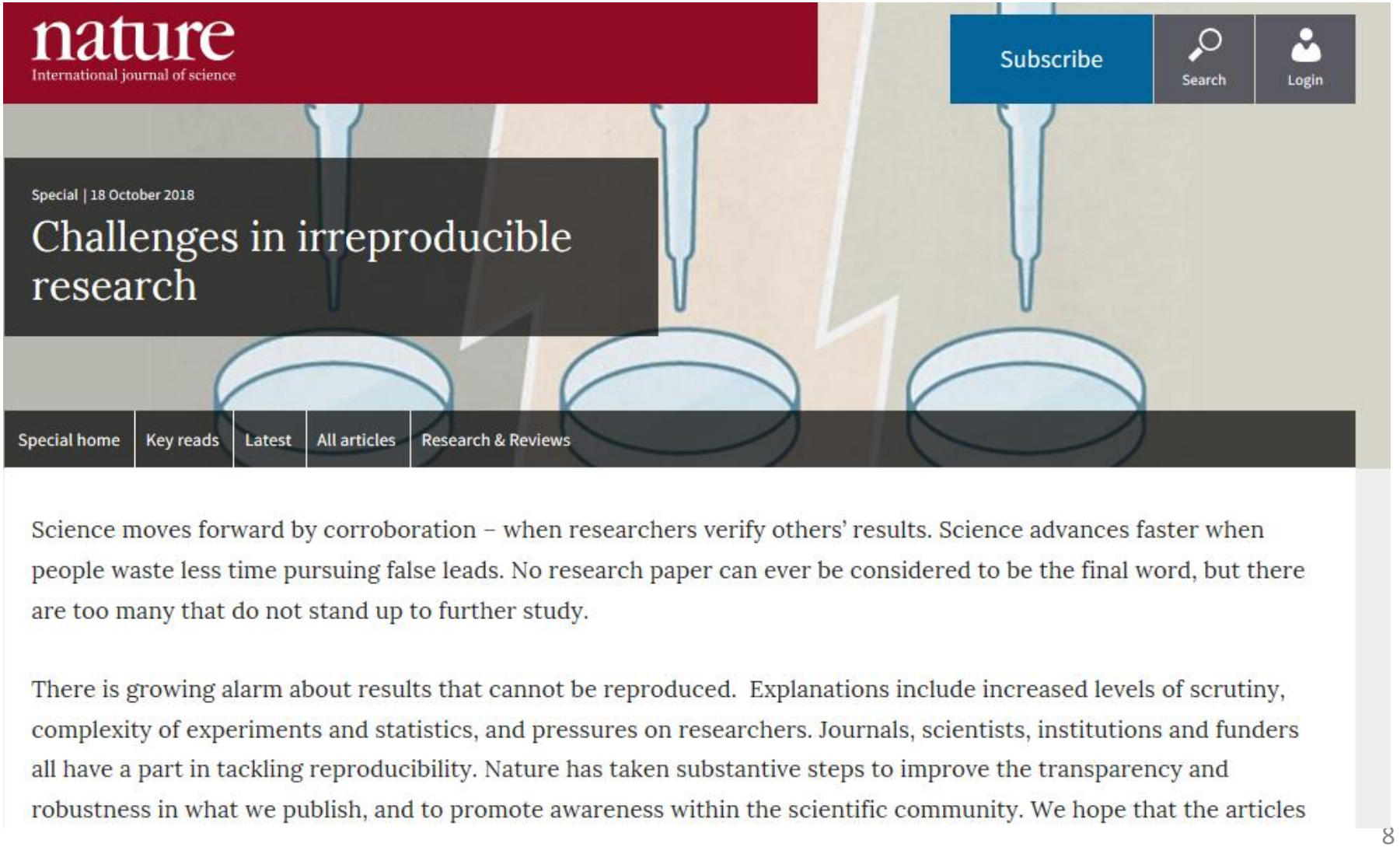
"An article about computational results is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result." –Claerbout and Karrenbach, 1992, "Electronic Documents Give Reproducible Research a New Meaning"

- paper is available
- code is available
- data is available

- paper is available
- code is available
- data is available

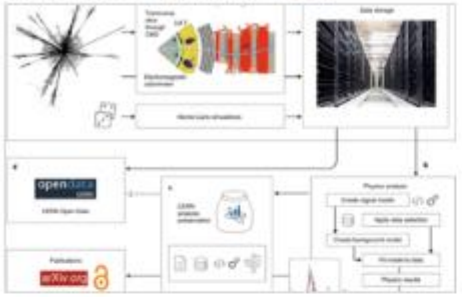
Your research is considered reproducible if someone with access to your raw data and your code, and your environment (hardware and software) can generate your results (tables and figures).

You can find what the scientific community says about reproducibility here:
https://www.nature.com/collections/prbfkwmwvz/#/



Key reads

Nature Physics | Perspective | OPEN



Open is not enough

The solutions adopted by the high-energy physics community to foster reproducible research are examples of best practices that could be embraced more widely. This first experience suggests that reproducibility requires going beyond openness.

Xiaoli Chen, Sünje Dallmeier-Tiessen [...] & Sebastian Neubert

Nature | World View

Before reproducibility must come preproducibility

Instead of arguing about whether results hold up, let's push to provide enough information for others to repeat the experiments, says Philip Stark.

Philip B. Stark

Nature | Editorial

Checklists work to improve science

Nature authors say a reproducibility checklist is a step in the right direction, but more needs to be done.

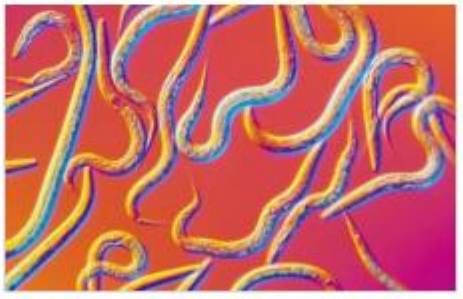
Nature | World View

Give every paper a read for reproducibility

I was hired to ferret out errors and establish routines that promote rigorous research, says Catherine Winchester.

Catherine Winchester

Nature News | Comment



A long journey to reproducible results

Replicating our work took four years and 100,000 worms but brought surprising discoveries, explain Gordon J. Lithgow, Monica Driscoll and Patrick Phillips.

Gordon J. Lithgow, Monica Driscoll & Patrick Phillips

Nature News | Editorial

Reality check on reproducibility

A survey of *Nature* readers revealed a high level of concern about the problem of irreproducible results. Researchers, funders and journals need to work together to make research more reliable.

Considerations:

- being able to reproduce own results at a later date
- manage changes to data, analysis and results
- satisfy journal requirements

“Institutionalizing Transparency”, Jeremy Freese and Molly King, SOCIUS, 2018, vol4: 1-7, DOI: 10.1177/2378023117739216

Every piece of code you write has multiple audiences – the computer, you, your coauthor, or someone wishing to reproduce your writings.

Goal: to be able to reproduce results from start to finish.

- Use script for everything you do
  - download data via script
  - Don’t hand edit files
  - Analysis should be in scripts, same for data cleaning
  - Set and save the seed (in R also run sessionInfo())

See Karl Broman’s page: http://kbroman.org/steps2rr/

- Organize data, code, and dependencies
  - Encapsulate everything
  - Separate raw data from derived data
  - Separate data from code
  - Use relative paths
  - Write readme files (document everything!)

See: http://kbroman.org/steps2rr/

- Automate the process
  - Using make (makefile)
- Turn scripts into reproducible reports
  - Include documentation (why you did this)
- Use version control

See: http://kbroman.org/steps2rr/

Recommendations for reproducible research:

1. **Encapsulate the full project into one directory.**
2. **Document everything and use code as documentation.**
3. **Make figures, tables, and statistics the results of scripts.**
4. **Write code that uses relative paths.**
5. **Always Set your seed.**
6. **Release your code and data.**

From (Reproducible Research Using RMarkdown and Git through RStudio, Marian Schmidt):
https://rpubs.com/marschmi/105639

Recommendations for reproducible research:

1. **Encapsulate the full project into one directory.**
2. **Document everything and use code as documentation.**
3. **Make figures, tables, and statistics the results of scripts.**
4. **Write code that uses relative paths.**
5. **Always Set your seed.**
6. **Release your code and data.**

The code should turn raw/original data into final results.

Resources:

- Best Practices for Scientific Computing by Greg Wilson et al., 2014

- "Bioinformatics Data Skills" book by Vince Buffalo (and it's github page)

- "Institutionalizing Transparency", Jeremy Freese and Molly King, SOCIUS, 2018, vol4: 1-7

- Nature Journal on reproducible research - https://www.nature.com/collections/prbfkwmwvz/#/

Recommended Data Repositories:
https://www.nature.com/sdata/policies/repositories

Summary of best practices for scientific research:

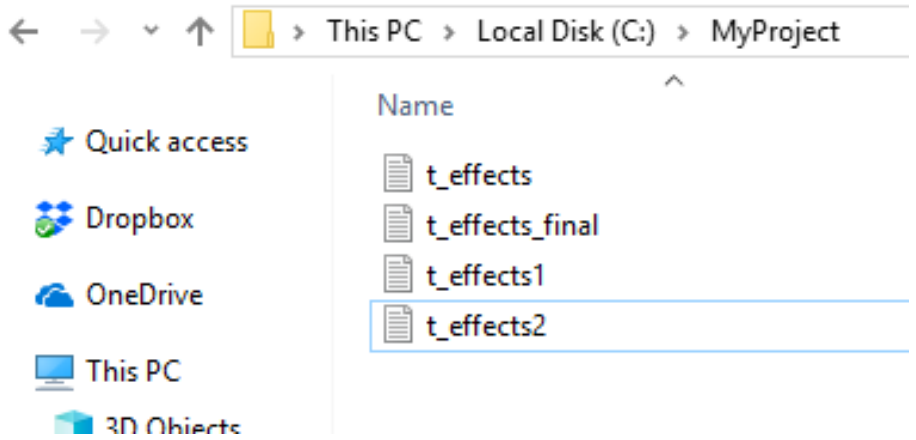
1. Write programs for people, not computers.
2. Let the computer do the work.
3. Make incremental changes.
4. Don’t repeat yourself.
5. Plan for mistakes.
6. Optimize software only after it works correctly.
7. Document design and purpose, not mechanics.
8. Collaborate.*

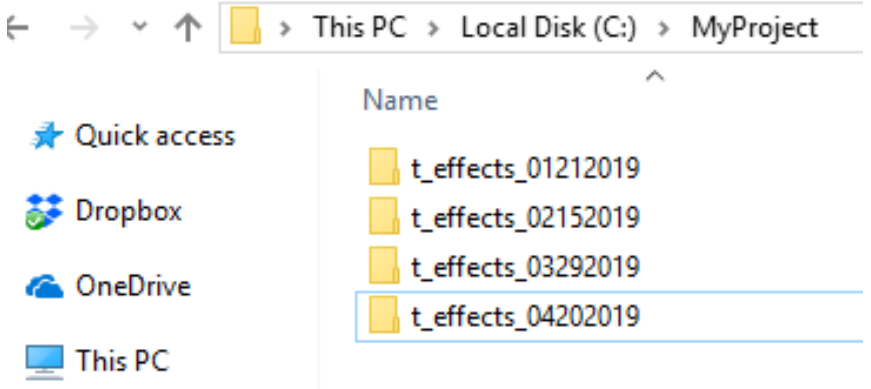
Find more here: https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

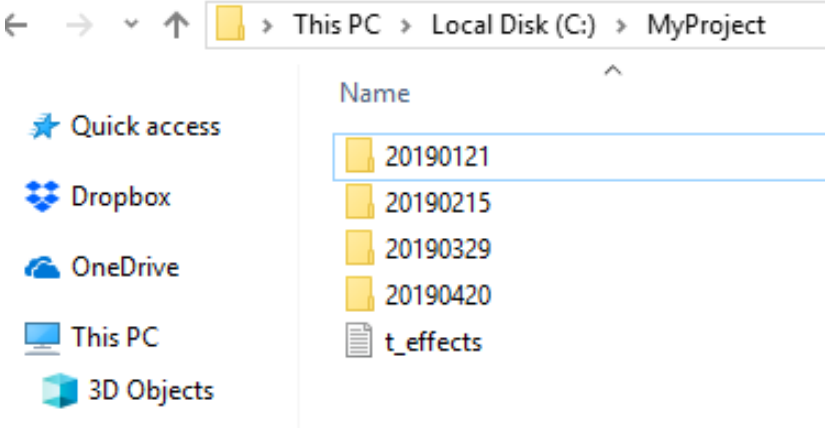
How to keep track of changes in the code?

Version Control

Version Control



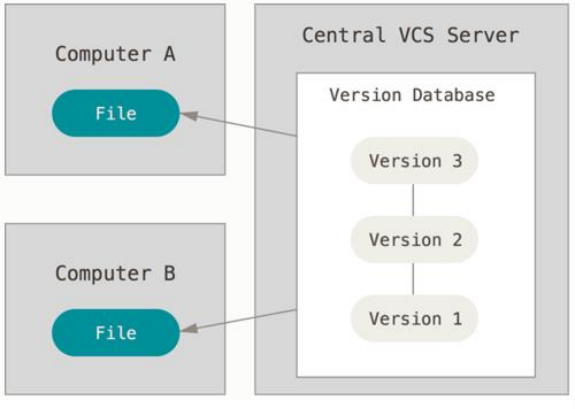




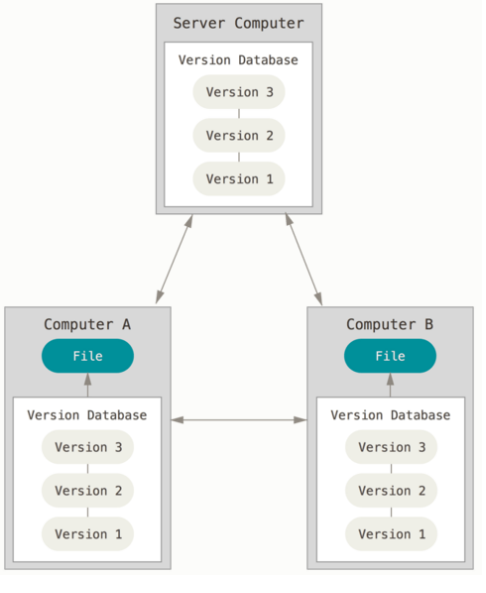
OR You could use a version control system…

Version control systems come in two types:

- Centralized version control



- Distributed version control



Git is a distributed version control system.

Git is:

- open source program for tracking changes in text files
- version control system, a tool to manage your source code history

GitHub is:

- hosting service for Git repositories

At the heart of **GitHub** is an open source version control system (VCS) called ***Git***. **Git** is responsible for everything GitHub-related that happens locally on your computer.

To download: https://desktop.github.com/ or https://git-scm.com/downloads

How **Git** works:

stores a snapshot of the code file – called ‘commit’.

Stores the file as well as metadata (who did the commit, date, notes).

- Git stores everything under the (hidden) .git directory
- Everything in Git is check-summed before it is stored (by 40-character SHA-1 tag)
- Git is a distributed system – everyone has their own local copy of the whole repository

For more check this website: https://git-scm.com/

Git takes snapshots of your files at a given moment, to which you can revert later.

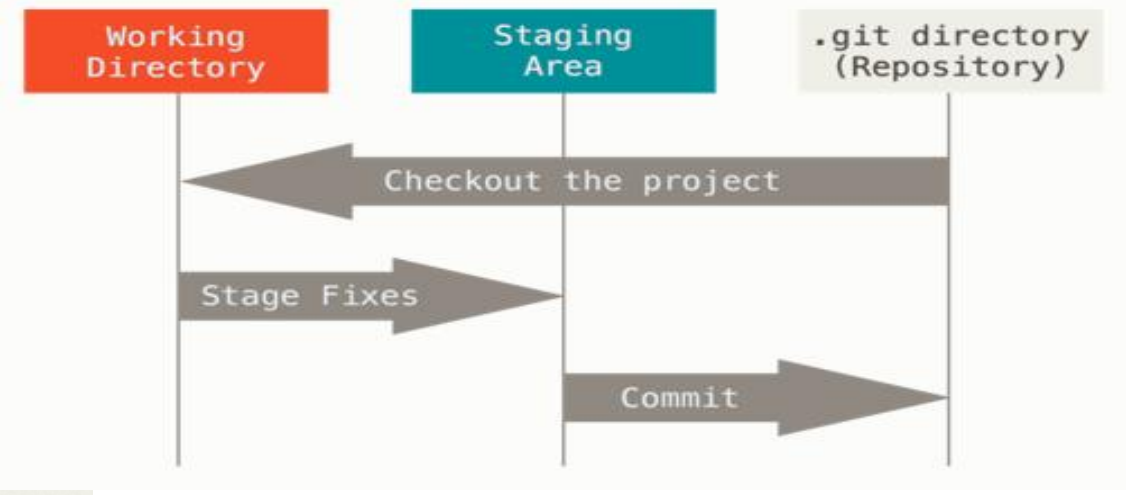
The files in your directory can be either:

- tracked (were in the last snapshot)
- untracked

The tracked files in your directory can be either:

- unmodified
- modified
- staged

Only ‘staged’ files can be ‘commit’-ed (meaning taken snapshot of).



Git is a command line tool, which works locally (on your machine).

- open the cmd.exe (or terminal) window
- set name and address for all commits:

```
>git config --global user.name "Me"
>git config --global user.email me@example.edu
```

Question:
What do you expect the command '`git config --global user.name`' to give?

To start, run:

>**git init** - initialize a directory to be ‘tracked’

>**git help**

>**git config** - set up

>**git config --help** – to see what options are available

git --everything-is-local

Before a file that is new or changed has a snapshot taken, it has to be staged (cashed).

To stage files:

>**git add** "***file_name***"	- stage one file
>**git add css/**	- stage a directory (and subdirectories)
>**git add –all**	- stage all files in the current directory

To commit (take snapshot):

>**git commit –m** "***note***"	- commit what is in the staging area

When you do actions in Git, nearly all of them only *add* data to the Git database.

git --everything-is-local

Let's try it…

Initial commit:

- Create a project directory
- Create an R markdown file (ex_mkdown.Rmd)

```
git init
git add ex_mkdown.Rmd
git commit –m "first commit"
git status
git log
```

To check repository status and history:

>**git status** - shows if there are not committed changes to any files

>**git diff** - shows differences between last snapshot and the current file

>**git log** - see all commits to date

git --everything-is-local

Undoing

```
>git commit --amend
```

If you commit too early and forget a file, for example:

```
>git commit –m “first commit”

>git add forgotten_file
>git commit --amend
```

Let's make a change to the file - add a line.

> **git diff** – to see what has changed

Commit the change.

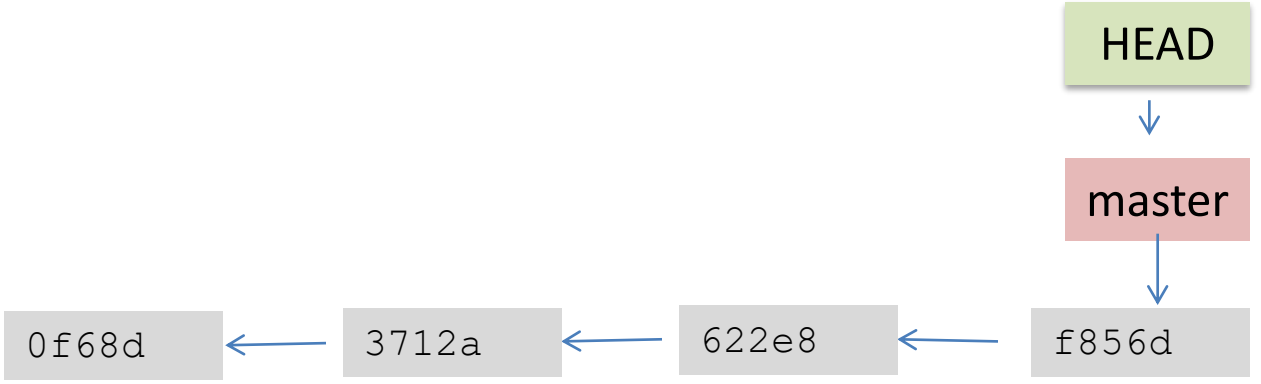
Now let's add another file to the folder and commit it...

Let's look at the history/log…

```
>git log --oneline --decorate
f856d83 (HEAD -> master) minor change
622e8ef third commit
3712a38 adding a line
0f68d35 first commit
```

```
>git log --oneline --decorate
f856d83 (HEAD -> master) minor change
622e8ef third commit
3712a38 adding a line
0f68d35 first commit
```

How Git works:

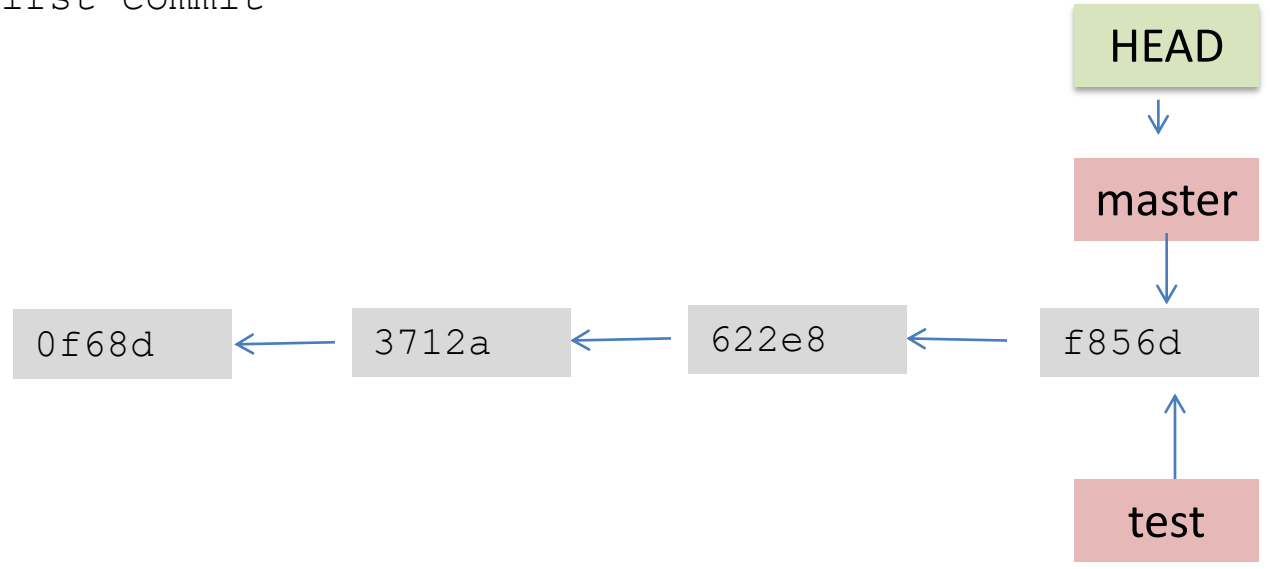


each commit points to its parent
HEAD is a special pointer (points to where you are currently in the tree of commits)

Make a branch:

```
>git branch test
```

```
>git log --oneline --decorate
f856d83 (HEAD -> master, test) minor change
622e8ef third commit
3712a38 adding a line
0f68d35 first commit
```

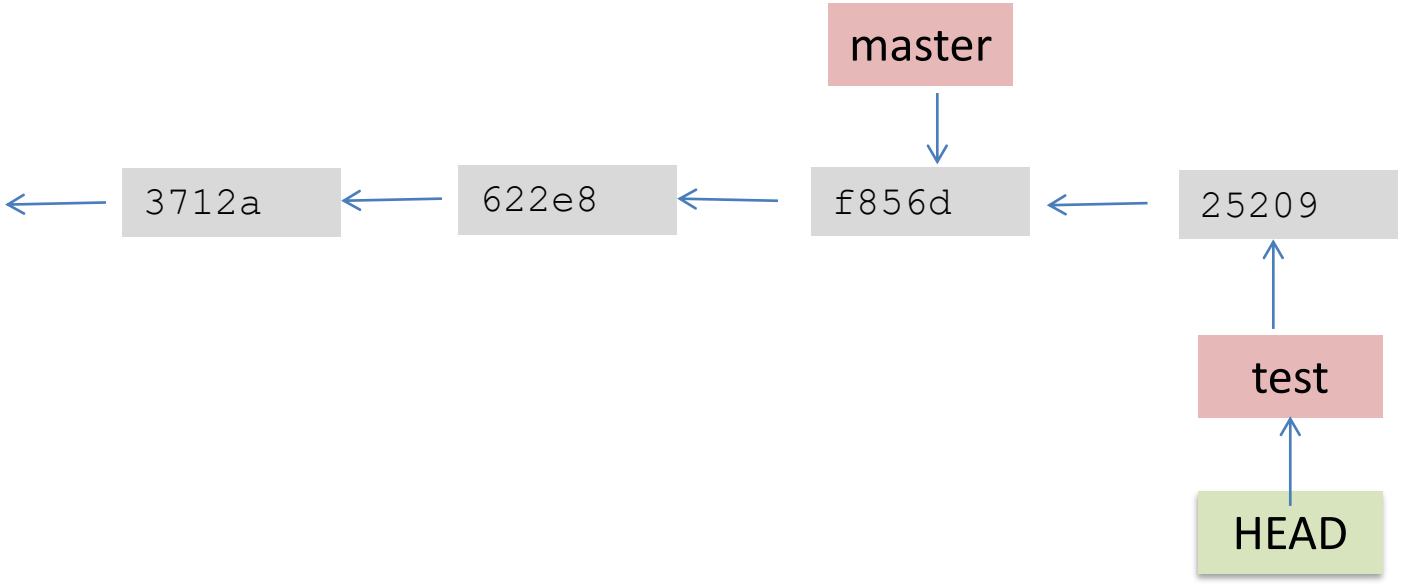


Checkout a branch:

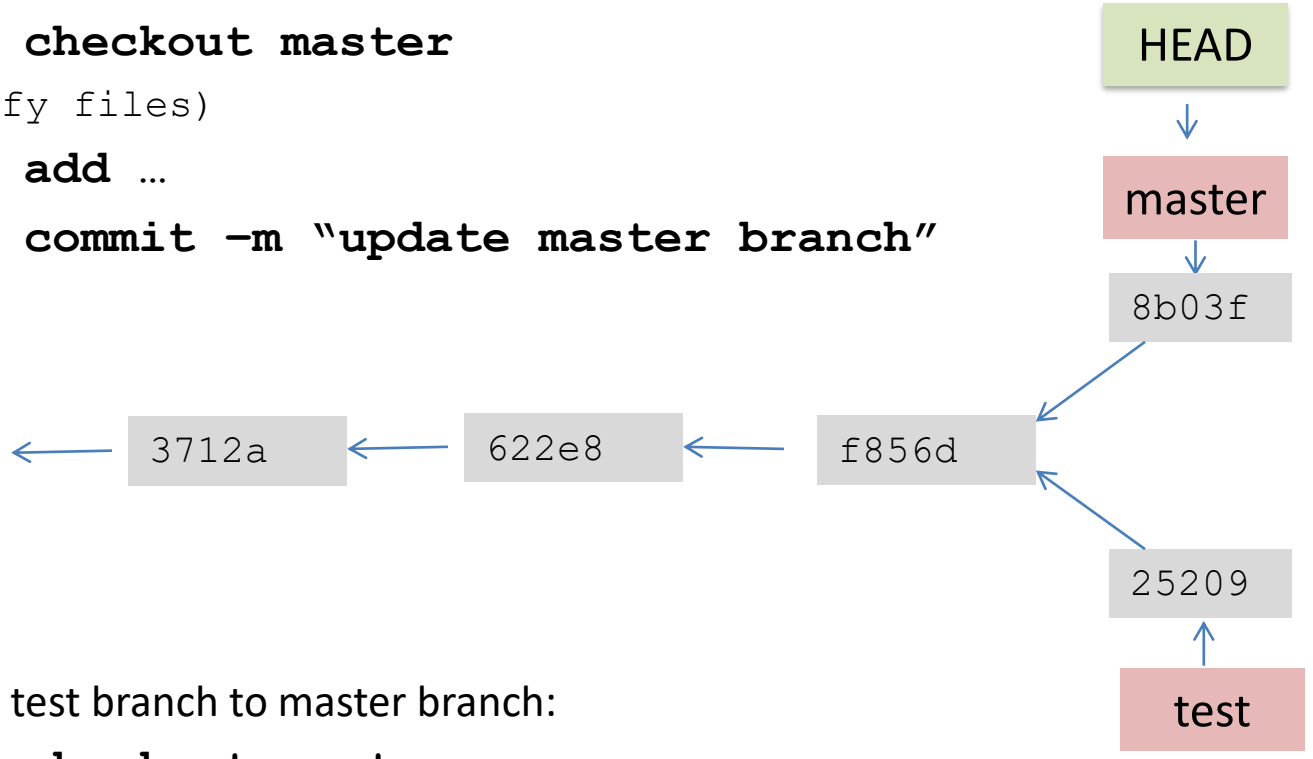
```
>git checkout test
```

Proceed with changes and commit:

```
>git add *.*
>git commit
```



```
>git checkout master
(modify files)
>git add …
>git commit –m "update master branch"
```

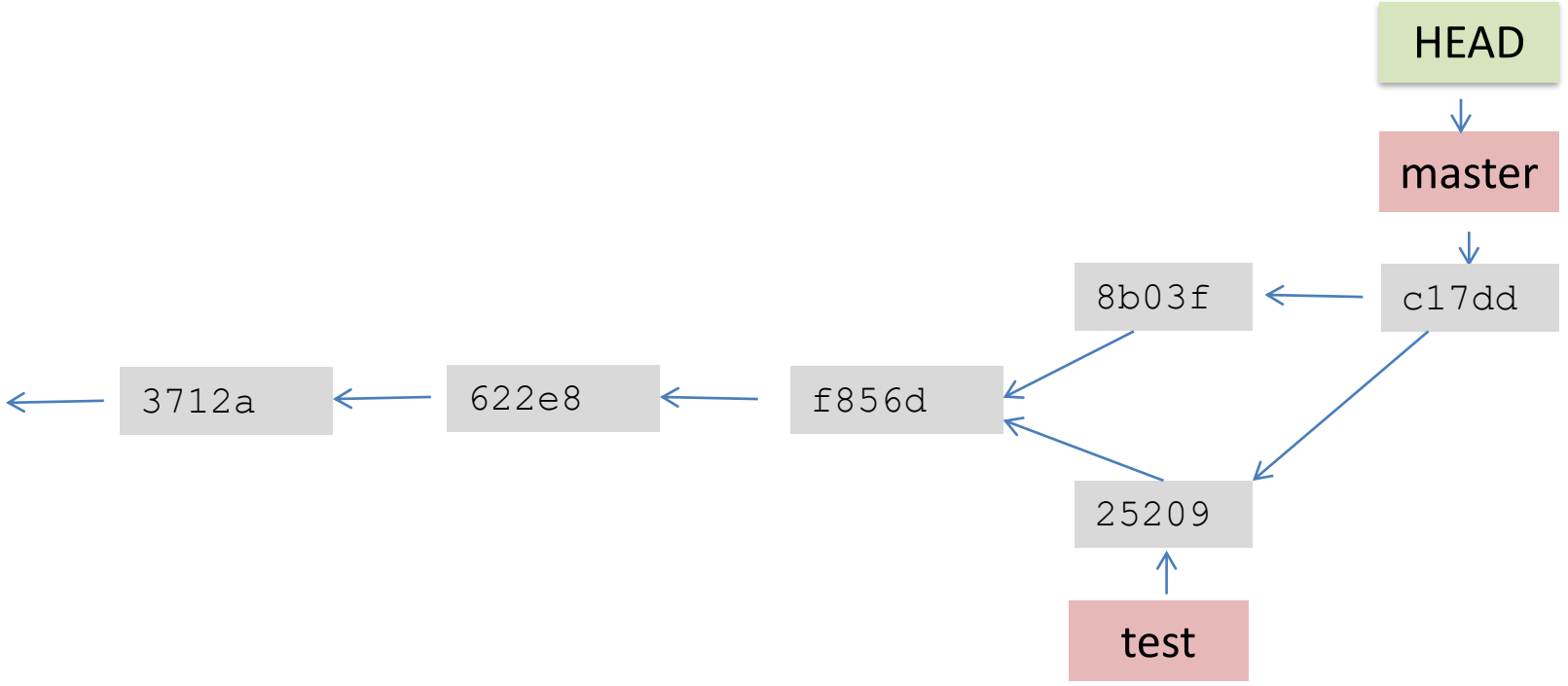


Merge test branch to master branch:

```
>git checkout master
```

```
>git merge test
Auto-merging ex_mkdown.Rmd
CONFLICT (content):  Merge conflict in ex_mkdown.Rmd
Automatic merge failed; fix conflicts and then commit the result.
```

When you get a “conflict” message you have to manually edit the conflicting parts (remove <<<<<<<, ======= and >>>>>>>) , then stage the file and commit.



Unstaging a staged file:

```
>git reset HEAD file_to_unstage
```

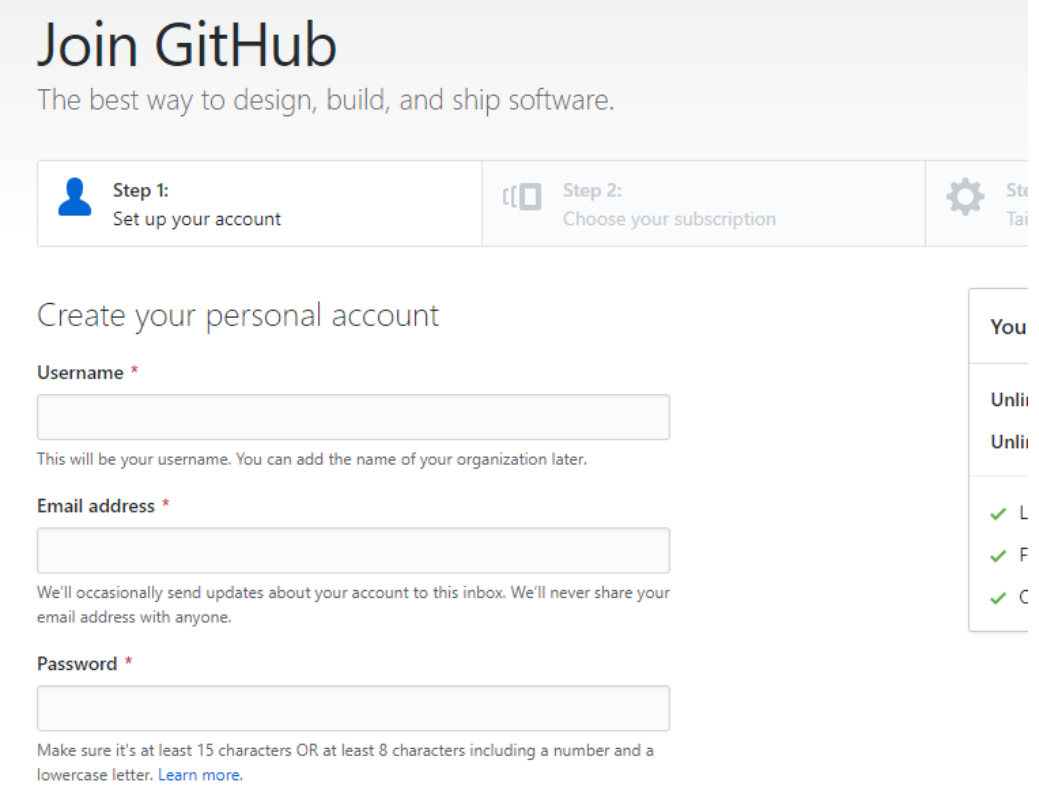
Undo changes to a file (modified it and would like to revert to how it was at the last commit):

```
>git checkout -- file_name
```

GitHub (https://github.com/about) is:

- a repository hosting service
- launched in 2008
- a place to share your code (like a 'Dropbox' for code)

You can sign up through this website: https://github.com/join



Let's look at: https://github.com/tidyverse/ggplot2

tidyverse / ggplot2

Watch 329 | Star 3,795 | Fork 1,425

Code | Issues 157 | Pull requests 25 | Wiki | Insights

An implementation of the Grammar of Graphics in R https://ggplot2.tidyverse.org

r | visualisation | data-visualisation

4,289 commits | 14 branches | 26 releases | 184 contributors | GPL-2.0

Branch: master | New pull request | Find File | Clone or download

clauswilke Improved release notes (#3324) ... Latest commit 291cd7e 2 days ago

.github	Move CODE_OF_CONDUCT.md to the main directory (#2973)	6 months ago
R	prefix lm() and loess() with stats:: (#3306)	4 days ago
data-raw	Update economics data (#2962)	2 months ago
data	Update economics data (#2962)	2 months ago
icons	Tweak icons	a year ago
inst	Correct url link in citation (#3077)	4 months ago
man	prefix lm() and loess() with stats:: (#3306)	4 days ago
pkgdown/favicon	Use retina logo and generate favicons	6 months ago
revdep	revdeps on 2019_03_27 (#3206)	2 months ago

After you create an account on GitHub, you could/should create a repository.

Cloning

A public repository on GitHub can easily be cloned…

either on line – through the button 

Or through Git command on your computer, for example:

```
>git clone https://github.com/ASvyatkovskiy/PrincetonPy
```

This creates a copy of the repo on your local machine.

Pushing to a GitHub repository

After you make changes to files locally, when you want to put these changes in your Github repository, you would need to 'commit' and then 'push'.

First set up your GitHub repository

```
>git remote add origin https://github.com/...
```

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

Will ask for username and password…

General form:

```
>git push <remote> <branch>
```

Fetching (files from a repository):

Getting files from GitHub (/remote repository) to your local computer:

```
>git fetch origin
```

If you have a remote branch named ‘smallfix’ in your ‘origin’ repository (origin/smallfix), then you can
1) either merge it to you’re your current working branch with:

```
>git merge origin/smallfix
```

Or

2) get a local branch called ‘smallfix’ run:

```
>git checkout -b smallfix origin/smallfix
```

Forking a GitHub repository

A ***fork*** is a copy of a repository (a clone on the server side). Forking is done through GitHub.

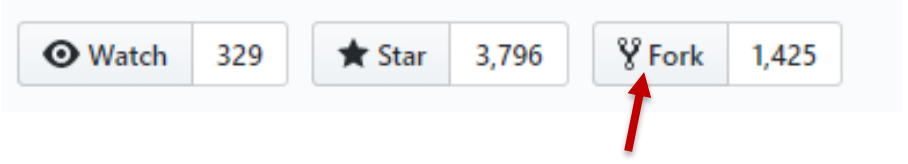
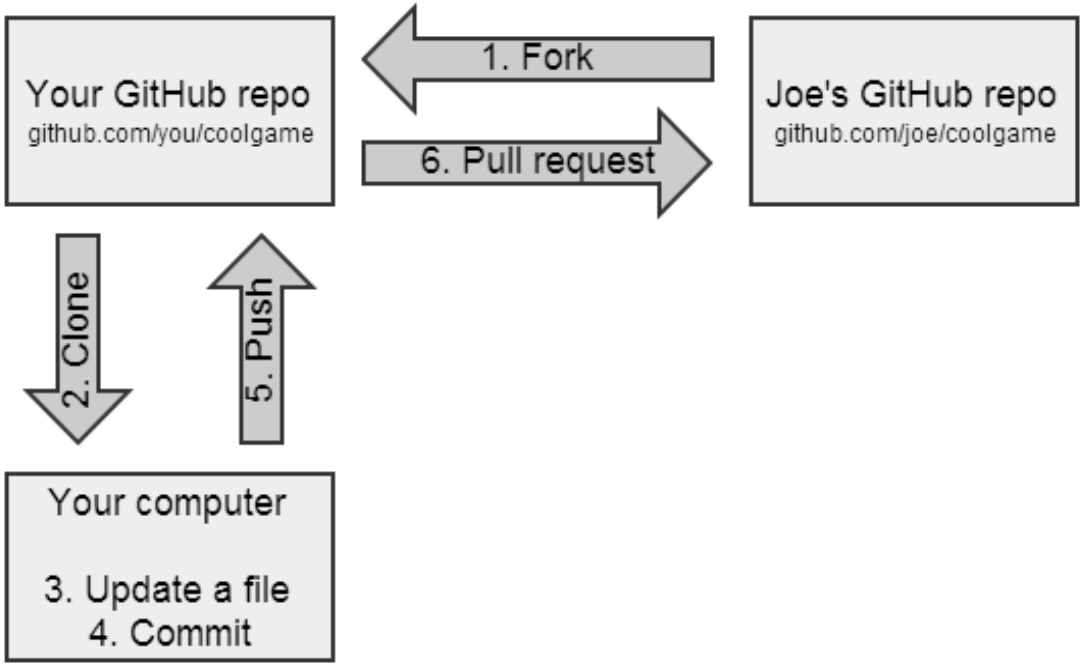
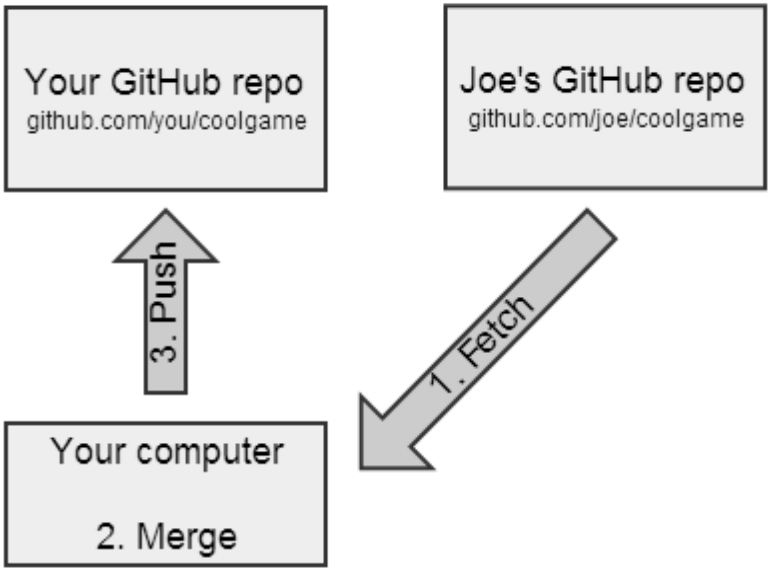


Diagram from Kevin Markham's simple guide to forks :



Syncing a fork

1. Fetch changes
2. Merge to your local repo
3. Push the updates to your GitHub repo (optional)

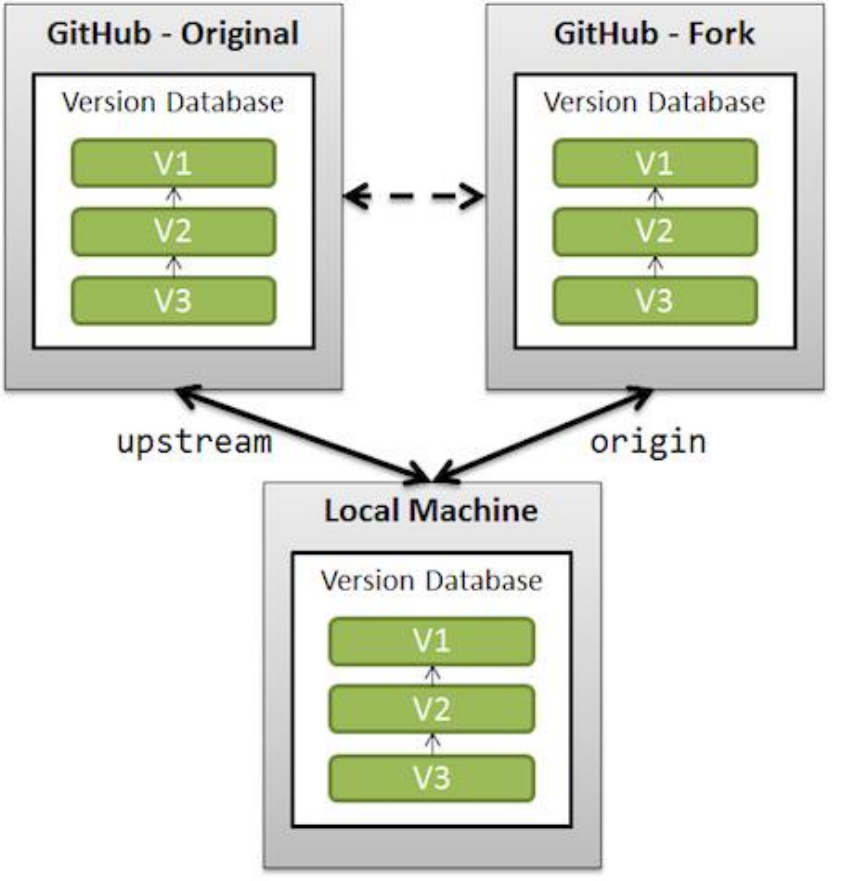


Github

When a repo is **cloned**, it has a default remote called **origin** that points to your fork on GitHub, not the original repo it was forked from.

To keep track of the original repo, you need to add another remote named **upstream**.

```
>git remote add upstream https://...
```



Pull from a GitHub repository

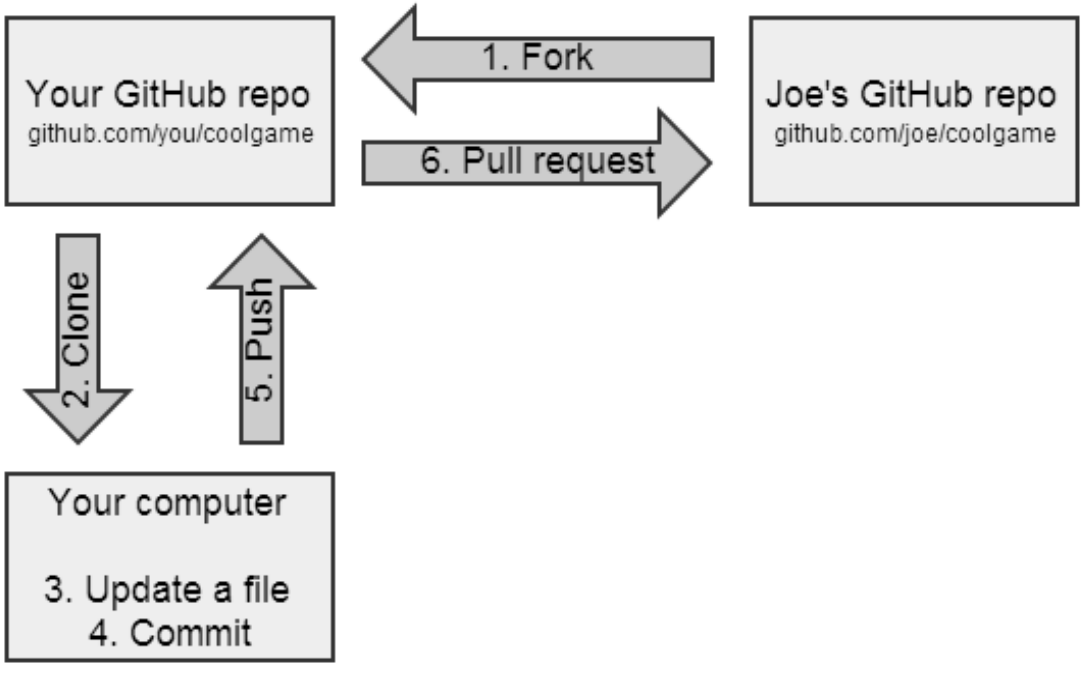
```
>git pull
```

If pulling from your GitHub repository, `pull` does both `fetch` and `merge`.

If you would like someone to pull code from your repository you have to create a pull request.

To create pull request :

1. Create a fork (clone the repo under your account).
2. Clone to your local computer.
3. Create a branch or update files and commit changes.
4. Push changes to own GitHub repo.
5. Send the original project account a PR.



To accept pull request:

1. Review code and documentation.
2. If it's OK, pull from the collaborator's GitHub repo onto local master.
3. After merge push back up to the project's GitHub repo.

Git/GitHub resources

2nd Edition (2014)

- Scott Chacon and Ben Straub's (open source) book "Pro Git" https://git-scm.com/book/en/v2
- Kevin Markham's (Data School) youtube videos on "Version control with Git and GitHub" http://www.dataschool.io/git-and-github-videos-for-beginners/
- Karl Broman's tutorial "git/github guide" http://kbroman.org/github_tutorial/

Git documentation page https://git-scm.com/doc

GitHub guides https://guides.github.com/

Checklist

1. Was as much as possible done by the computer?
2. Was any file hand-edited, or any part of the analysis done by hand?
3. Is everything documented, including the software environment?
4. Was a version control system used?
5. Have we saved any output that we cannot reconstruct from original data and the code?
6. How far back in the analysis pipeline can we go before our results are no longer automatically reproducible?

Question:

It's a good idea to call sessionInfo() in your R code? (Y/N)