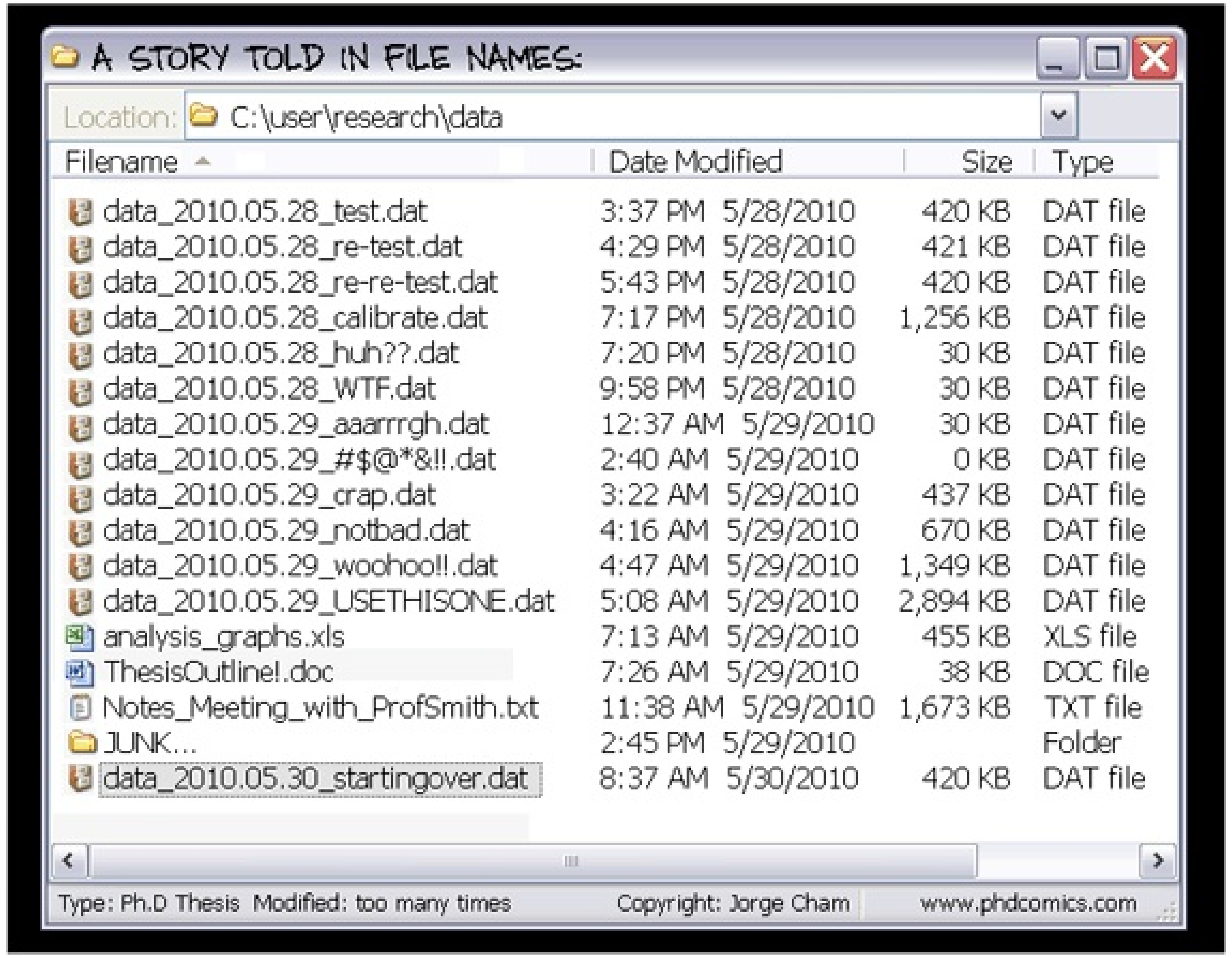
Git for psychologists

Sander

Retreat 2017 @Vielsalm



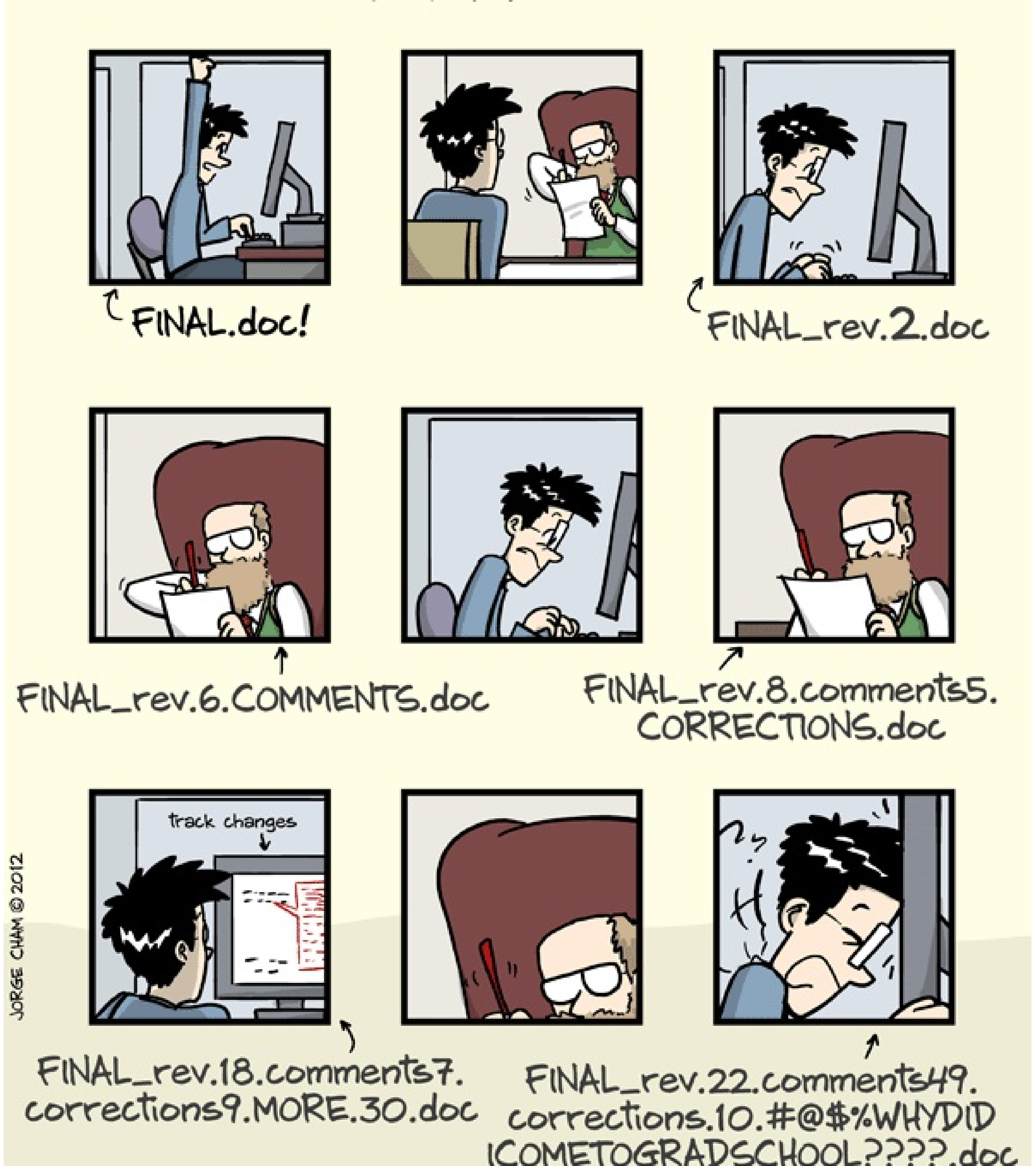
A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

A tragedy told in file names


FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
JORGE CHAM © 2012
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc

For reference

Tutorials:

- Version Control with Git for scientists (*Excellent*)
- Essential skills for reproducible research computing
- Git for Scientists: A Tutorial

Publications:

- Ram, K. 2013. Git can facilitate greater reproducibility and increased transparency in science. Source Code for Biology and Medicine 8:7.
- Mascarelli, A. 2014. Research tools: Jump off the page. Nature jobs 507: 523-525.

What's Git?

- Think: revision history/track changes/undo for entire folders

What's Git?

- Think: revision history/track changes/undo for entire folders
- aka: "version control system"

What's Git?

- Think: revision history/track changes/undo for entire folders
- aka: "version control system"
- Stream of snapshots, can be played back (the story of your project)

What's Git?

- Think: revision history/track changes/undo for entire folders
- aka: "version control system"
- Stream of snapshots, can be played back (the story of your project)
- Keeping record of what has changed (when & who)

What's Git?

- Think: revision history/track changes/undo for entire folders
- aka: "version control system"
- Stream of snapshots, can be played back (the story of your project)
- Keeping record of what has changed (when & who)
- Useful for the lone coder, excellent for collaboration

Plain text is king

- Developed for software development: code

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vi
```

Plain text is king

- Developed for software development: code
- But also: data, analysis code, reports, manuscripts

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vi
```

Plain text is king

- Developed for software development: code
- But also: data, analysis code, reports, manuscripts
- Works on anything that is plain text (.csv,.py,.html,.md,…)

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est laborum.
Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
quae ab illo inventore veritatis et quasi architecto beatae vi
```

Plain text is king

- Developed for software development: code
- But also: data, analysis code, reports, manuscripts
- Works on anything that is plain text (.csv,.py,.html,.md,...)
- Tools do not work on docx, image, pdf files (pandoc)

Installing Git

- Download here (Mac & Win)

- Open a command line (Rstudio -> Git -> More -> shell)
- Set up your git email by typing *git config --global user.email YOUR_EMAIL*
- Set up your git name by typing *git config --global user.name YOUR_NAME*
- Create a Github account. Use the same email you used to configure git.

Git cracking

Several ways to use git:

- Command line
- GUI (e.g. gitkraken, video tutorial)
- On Github

Git cracking

Several ways to use git:

- Command line
- GUI (e.g. gitkraken, video tutorial)
- On Github
- **inside Rstudio** (Most accessible)

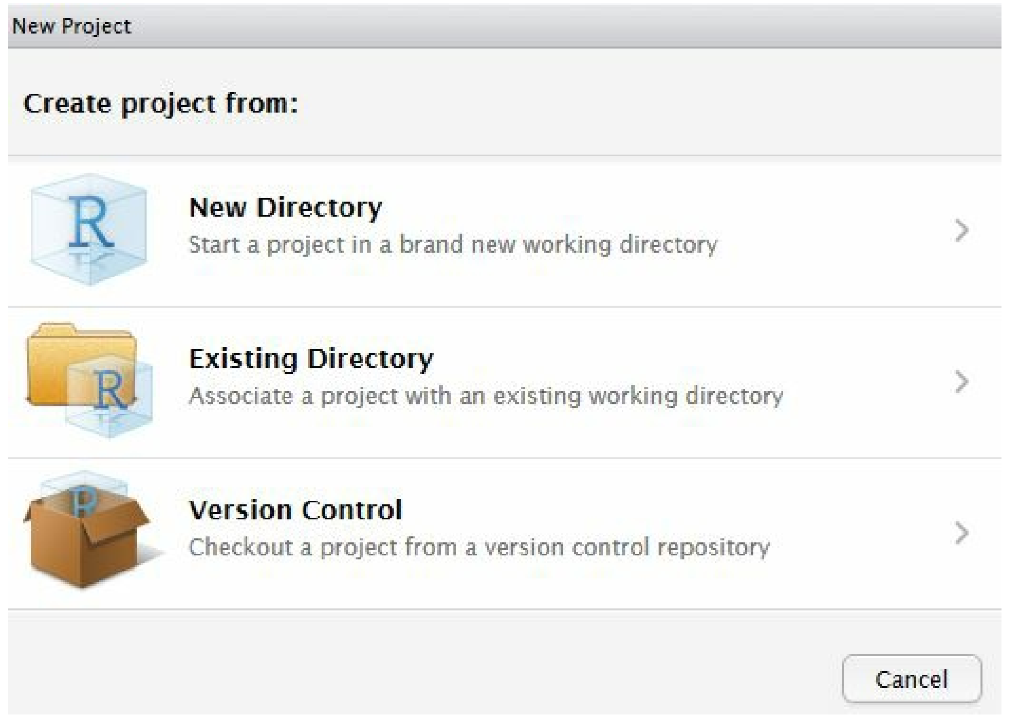
Initialize your project

- Rstudio -> New project -> from new directory -> tick: "create git repository"

or:

- Github -> Repositories -> New
  - then in Rstudio: New project -> from version control-> repository url (**Recommended**)

(*or*: In project folder: Command: **git init**)


New Project

Create project from:

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Cancel

Your new repository ("repo")

Automatically creates:

- *.git*-folder where your version history lives

Your new repository ("repo")

Automatically creates:

- *.git*-folder where your version history lives
- *.gitignore*-file lists filetypes that should be ignored (eg backup files created by our editor or intermediate files created during data analysis)

Your new repository ("repo")

Automatically creates:

- .*git*-folder where your version history lives
- .*gitignore*-file lists filetypes that should be ignored (eg backup files created by our editor or intermediate files created during data analysis)
- Optional: a *README* file (through github)

Your new repository ("repo")

Automatically creates:

- *.git*-folder where your version history lives
- *.gitignore*-file lists filetypes that should be ignored (eg backup files created by our editor or intermediate files created during data analysis)
- Optional: a *README* file (through github)
- Optional: a *LICENSE* file (through github)

Your git workflow

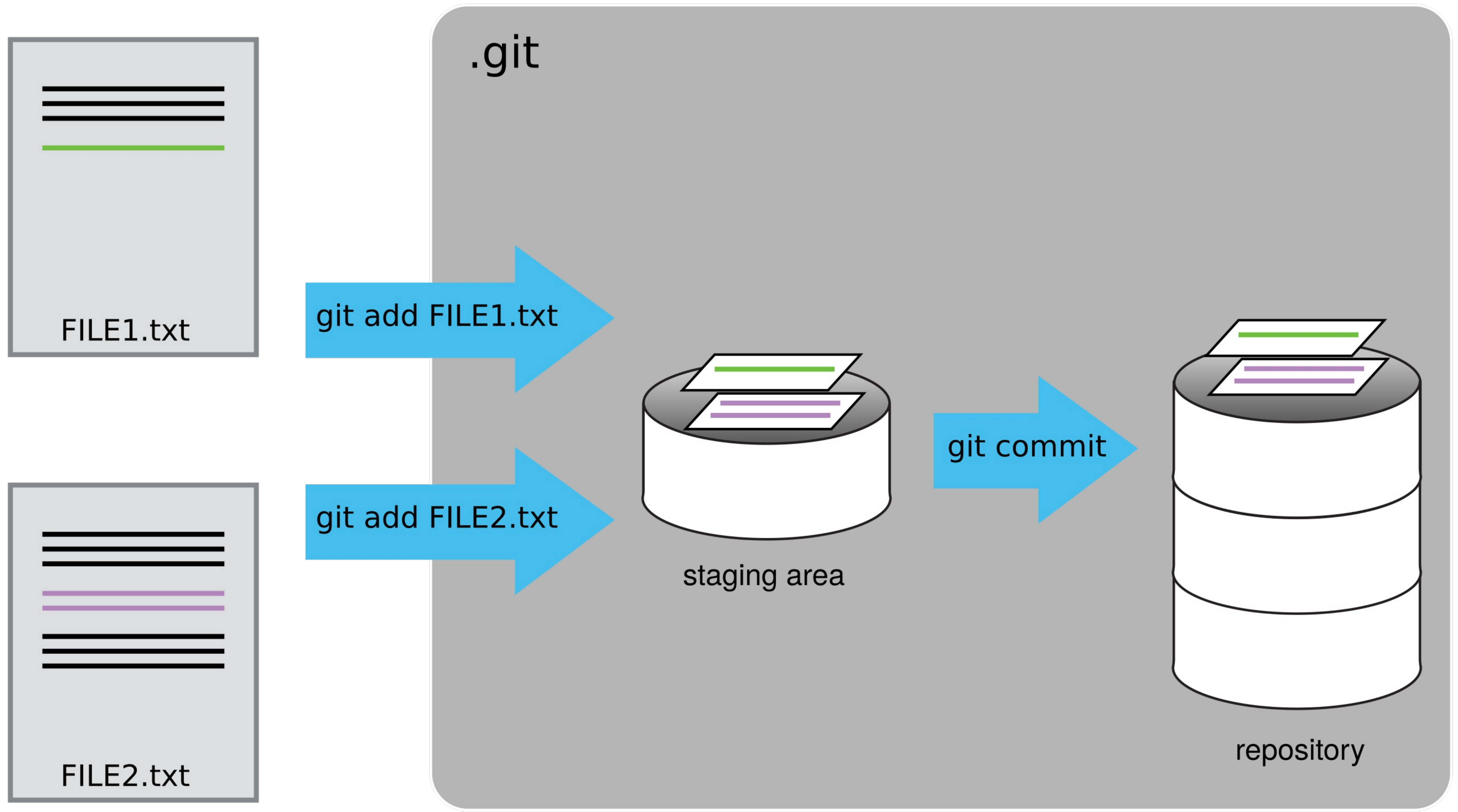
First you code, gather data, write, analyze, plot,…

Then you add files (**"staging"**): *determines what will go into the snapshot.*

- Rstudio: git tab -> Check boxes
- *git add myfile.txt*

Commitment needed

- Rstudio: git tab $\longrightarrow$ Commit
- *git commit -m "my commit message"*



Commitment needed

- Commit messages:

Not required, but strongly recommended. Be informative (like code comments, focus on the why)

Commitment needed

- Commit messages:

Not required, but strongly recommended. Be informative (like code comments, focus on the why)

- Why two steps? (Not: *git commit -a*)

Git insists we add files explicitly before committing to allow us to commit our changes in stages and capture changes in logical portions rather than only large batches.

When to commit?

- At least daily (if you have worked on the project)
- But better: more frequently (logical chunks):
    - I finished building this functionality or fixed this bug,
    - I added this section/analysis/figure (cf. sensible commit message).
    - I finished testing a (batch of) participant(s).
    - When going from pilot to actual, etc.

Understanding diff & status, & log

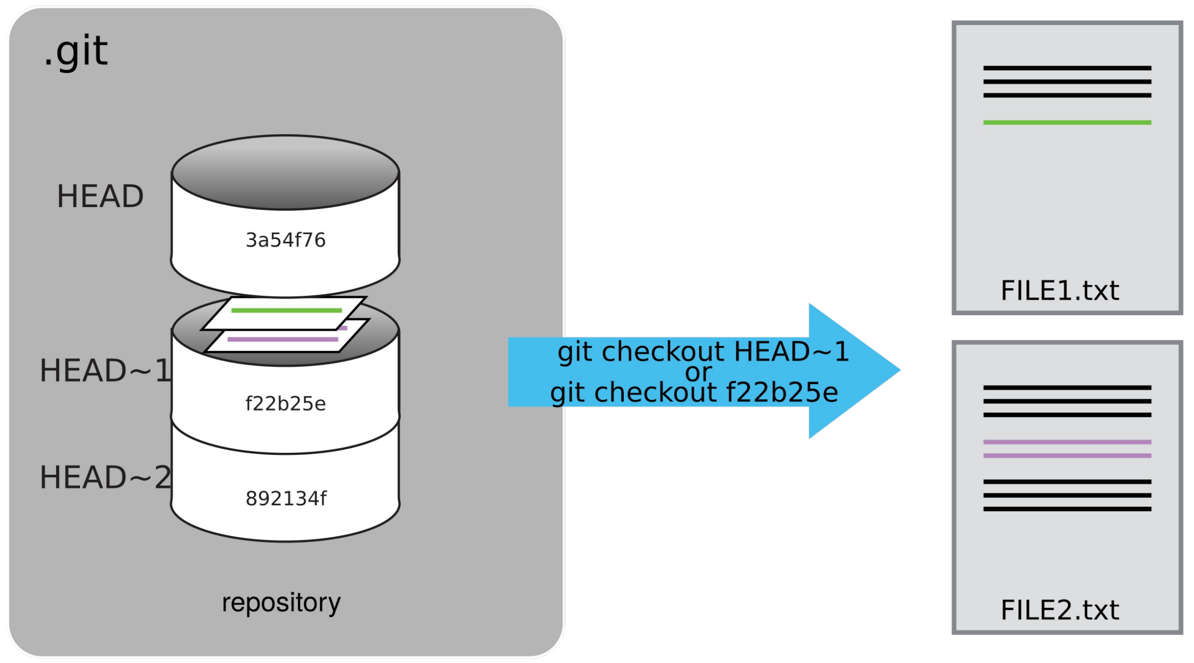
- Rstudio: git tab $\rightarrow$ diff

Command line:

- *git status* (where are you in your project)
- *git log* (overview of all commits)
- *git diff* (changes made to a repo since the most recent commit)
- *git diff 47f748 09633c myfile.py* (changes between two commits for myfile)

Restoring files aka time travel!

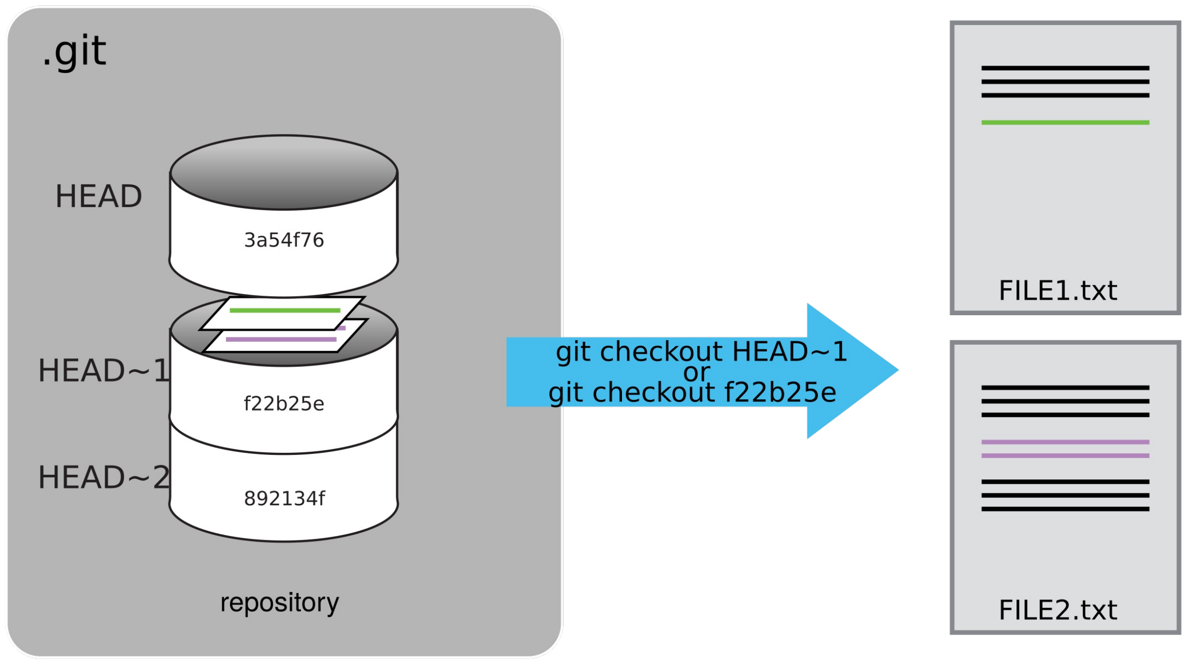
Sometimes we want to restore a previous version of a file entirely.



Restoring files aka time travel!

Sometimes we want to restore a previous version of a file entirely.

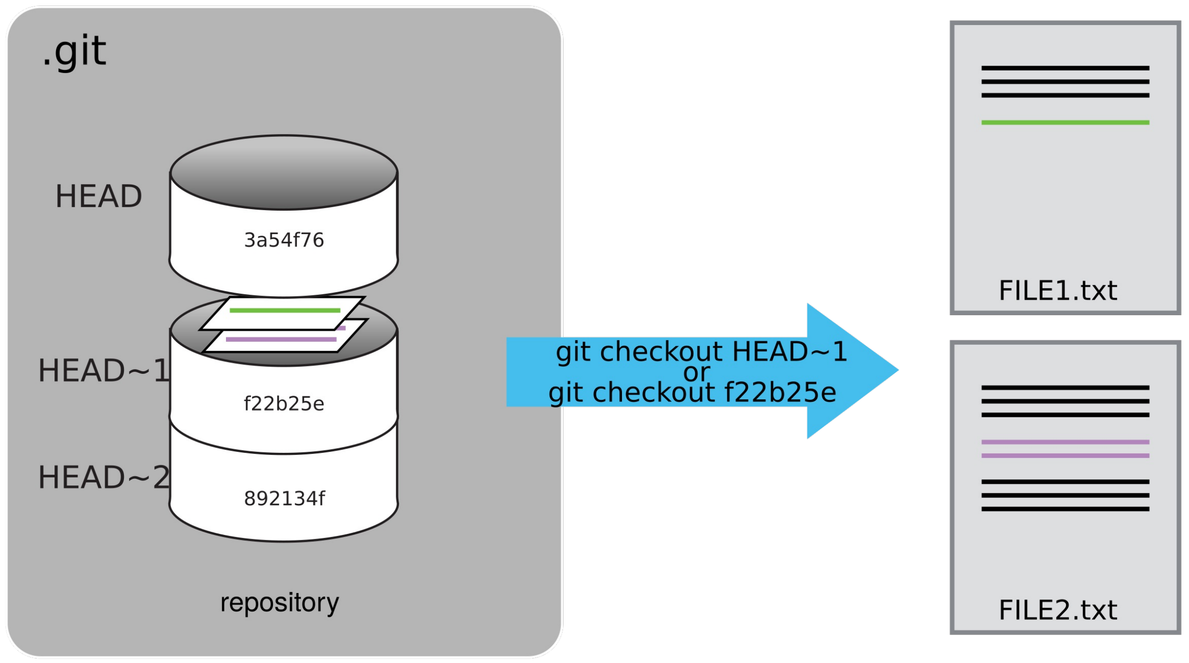
- *git log --oneline*



Restoring files aka time travel!

Sometimes we want to restore a previous version of a file entirely.

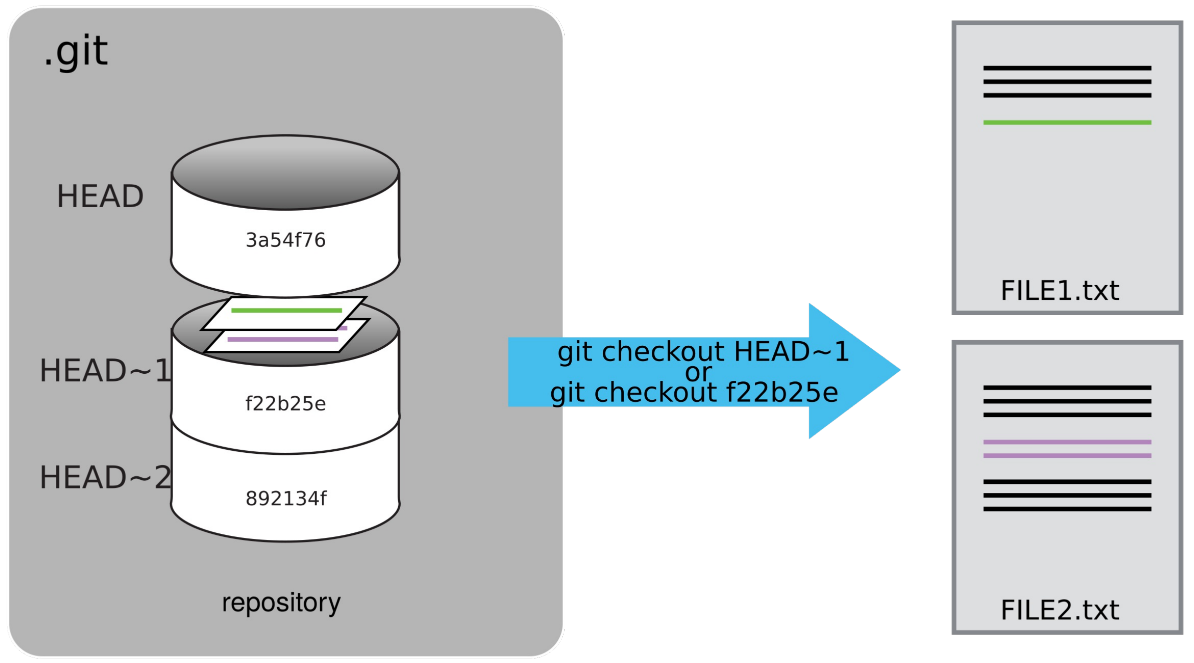
- *git log --oneline*
- *git checkout 47f748 word_count.py* (use commit nb of the state before the change we're trying to undo)

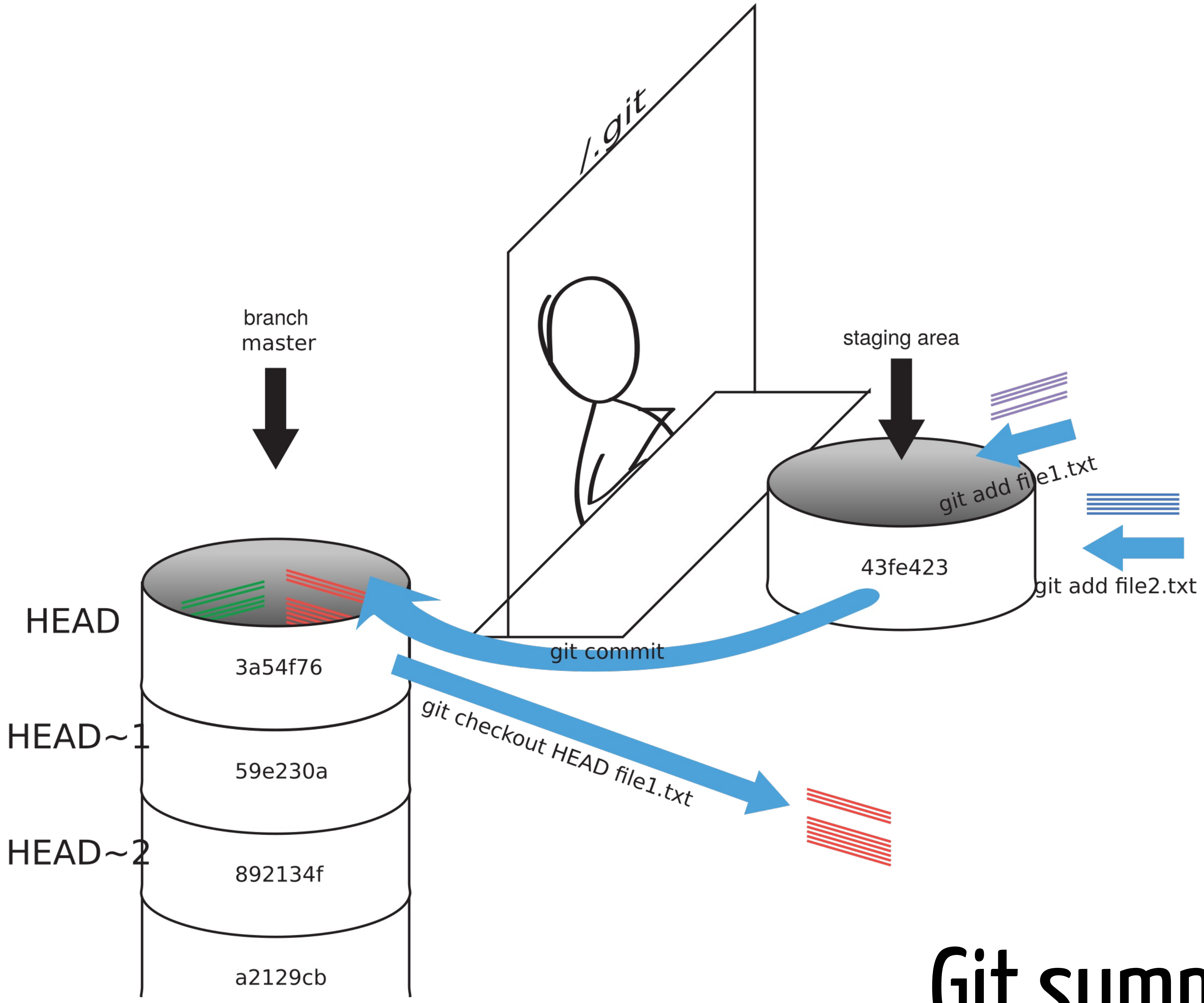


Restoring files aka time travel!

Sometimes we want to restore a previous version of a file entirely.

- *git log --oneline*
- *git checkout 47f748 word_count.py* (use commit nb of the state before the change we're trying to undo)
- Commit to finalize the restore: Rather than eliminating the commits that were made, git creates a new commit that changes everything back, ie you can change your mind again. (git never forgets/failsafe)




/.git
branch master
staging area
HEAD
HEAD~1
HEAD~2
3a54f76
59e230a
892134f
a2129cb
43fe423
git add file1.txt
git add file2.txt
git commit
git checkout HEAD file1.txt

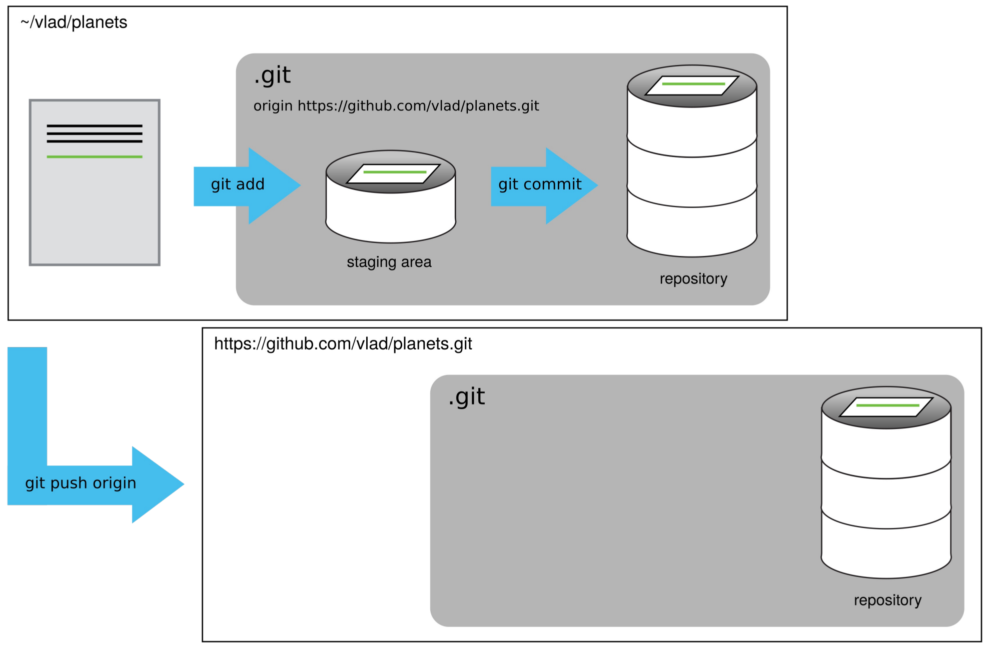
Git summary

Push it & pull it

- No automatic sync for collaborating (for code constant syncing in real time would be a disaster)
- Rstudio: git tab $\longrightarrow$ push (after commiting locally)
- When collaborating, ie changes have may been pushed by someone else: Rstudio: git tab $\longrightarrow$ pull (before working locally)

Command line:

- *git remote add origin https://github.com/try-git/try_git.git*
- *git push origin master*
- *git pull origin master*



Conflicts & merging

But what happens if we make a commit in both places?

- Try it: change the README on github repo (*btw: you can edit files directly in Github*). Push will be rejected $\longrightarrow$ do a pull & auto-merge first. However: If change in same line/file:

```
[...]
From https://github.com/vlad/planets
 branch            master     -> FETCH_HEAD
Auto-merging mars.txt
  CONFLICT (content): Merge conflict in mars.txt
  Automatic merge failed; fix conflicts and then commit the result.
[...]
```

Conflicts & merging

But what happens if we make a commit in both places?

- Try it: change the README on github repo (*btw: you can edit files directly in Github*). Push will be rejected → do a pull & auto-merge first. However: If change in same line/file:

```
[...]
From https://github.com/vlad/planets
 branch            master     -> FETCH_HEAD
Auto-merging mars.txt
  CONFLICT (content): Merge conflict in mars.txt
  Automatic merge failed; fix conflicts and then commit the result.
[...]
```

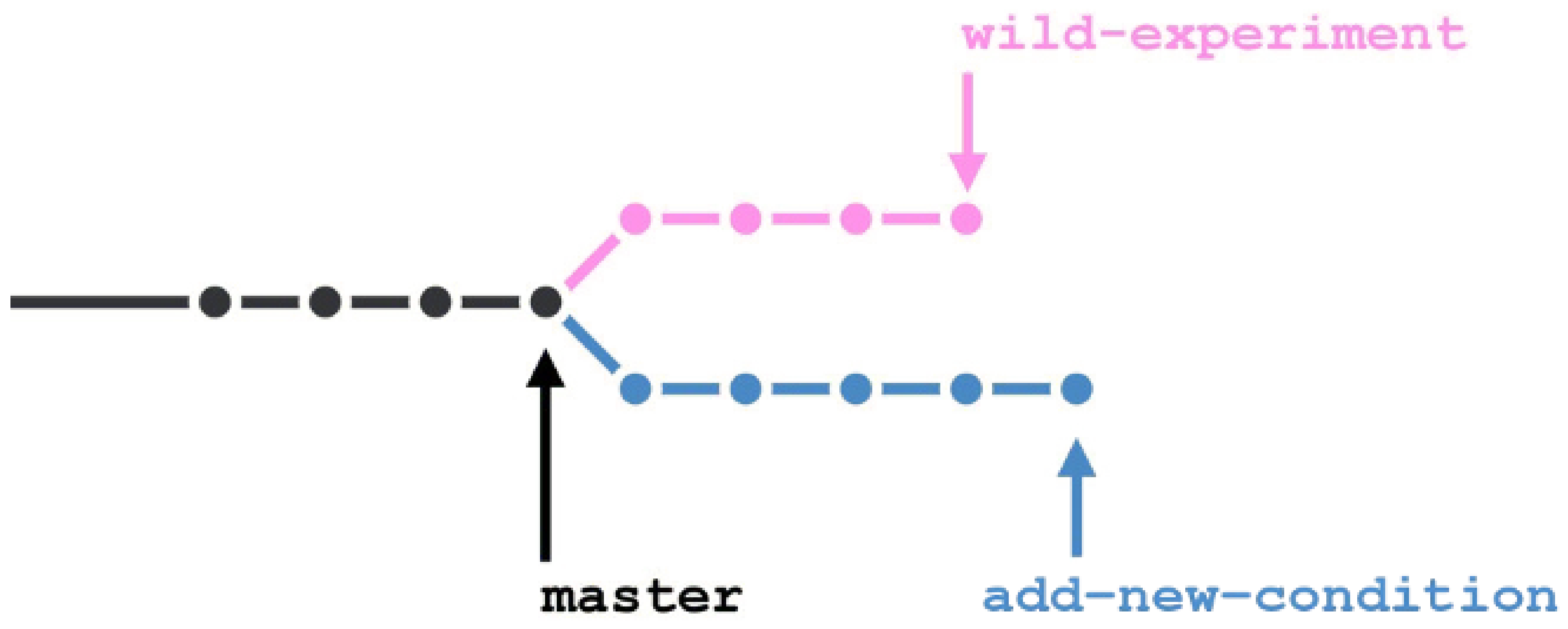
```
[...] The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
  <<<<<<< HEAD
We added a different line in the other copy
  =======
This line added to Wolfman's copy
  >>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d
[...]
```

Cloning, forking, & branching

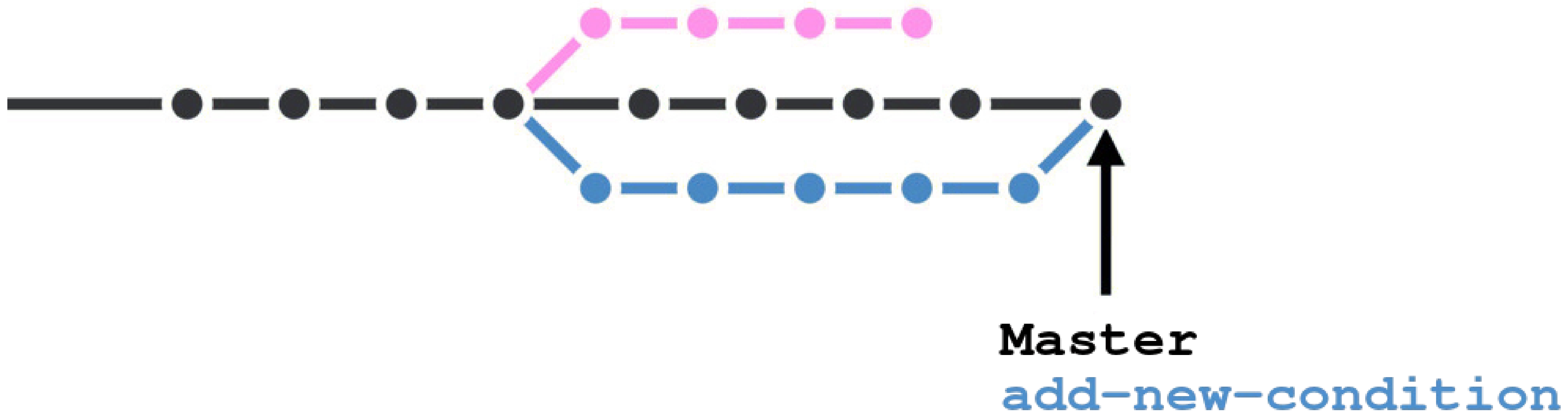
- Clone: download local copy of existing repo (cannot push to origin)
- Fork (example): copy of existing repo under your account. Keeps the link between projects $\rightarrow$ *pull requests* on github

Cloning, forking, & branching

- Clone: download local copy of existing repo (cannot push to origin)
- Fork (example): copy of existing repo under your account. Keeps the link between projects $\longrightarrow$ *pull requests* on github
- Branching as a local fork to work on a new feature, a new version, to experiment with variants of your code. Can be merged again with the *master* branch (or removed). Can only be used with command line.



Branching



What to use it for?

- All code:
    - R or python analysis: R Markdown files and Jupyter notebooks will appear in human-readable, pretty format on github.
    - Your experiment code

What to use it for?

- All code:
  - R or python analysis: R Markdown files and Jupyter notebooks will appear in human-readable, pretty format on github.
  - Your experiment code
- You can use it to backup/share your (behavioral) data and stimuli too. But size limits for non-text files (for very large stimuli or data files, use figshare, Zenodo, OSF).

What to use it for?

- All code:
    - R or python analysis: R Markdown files and Jupyter notebooks will appear in human-readable, pretty format on github.
    - Your experiment code
- You can use it to backup/share your (behavioral) data and stimuli too. But size limits for non-text files (for very large stimuli or data files, use figshare, Zenodo, OSF).
- What about manuscripts? Only if in markdown/latex (but no good solution for citations, track changes)

What to use it for?

- All code:
  - R or python analysis: R Markdown files and Jupyter notebooks will appear in human-readable, pretty format on github.
  - Your experiment code
- You can use it to backup/share your (behavioral) data and stimuli too. But size limits for non-text files (for very large stimuli or data files, use figshare, Zenodo, OSF).
- What about manuscripts? Only if in markdown/latex (but no good solution for citations, track changes)
- Private github repos means you choose when to make your repo/project public (holds for github and osf)

More advantages

- Integrity of each commit is checked (safety against file corruption) with checksums (hash values of content=names in git db)
- Backup on another computer (safer & better ordered than dropbox, cf code conflicts)
- Collaboration (one common repo on github): better conflict management
- Commits helps you tell the story of your project to other people (reproducibility)
- Time-stamped (pre)registration of state of steps in your project (if you commit, of course)

Further tools that connect with git(hub)

- Github-OSF integration:
    - Two-way sync: edit in OSF, in github (or locally).
    - Storage **remains** in github

Further tools that connect with git(hub)

- Github-OSF integration:
    - Two-way sync: edit in OSF, in github (or locally).
    - Storage **remains** in github
- Persistent copy needed? Zenodo or figshare can archive your github repo (with citable DOI).

Further tools that connect with git(hub)

- Github-OSF integration:
    - Two-way sync: edit in OSF, in github (or locally).
    - Storage **remains** in github
- Persistent copy needed? Zenodo or figshare can archive your github repo (with citable DOI).
- Github watching & forking (take existing project/experiment as basis to start yourself; also on OSF)
- Reputation system (for some jobs in the private sector, gtihub account is de facto cv/portfolio)
- GestaltReVision repos

THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.

Thanks!

Don't panic: Your code is good enough ;-)