

Version control and data curation with Git and GitHub

Johannes Englisch
englisch@shh.mpg.de

Department of Linguistic and Cultural Evolution
Max Planck Institute for the Science of Human History

MPI-SHH SUMMER SCHOOL 2021

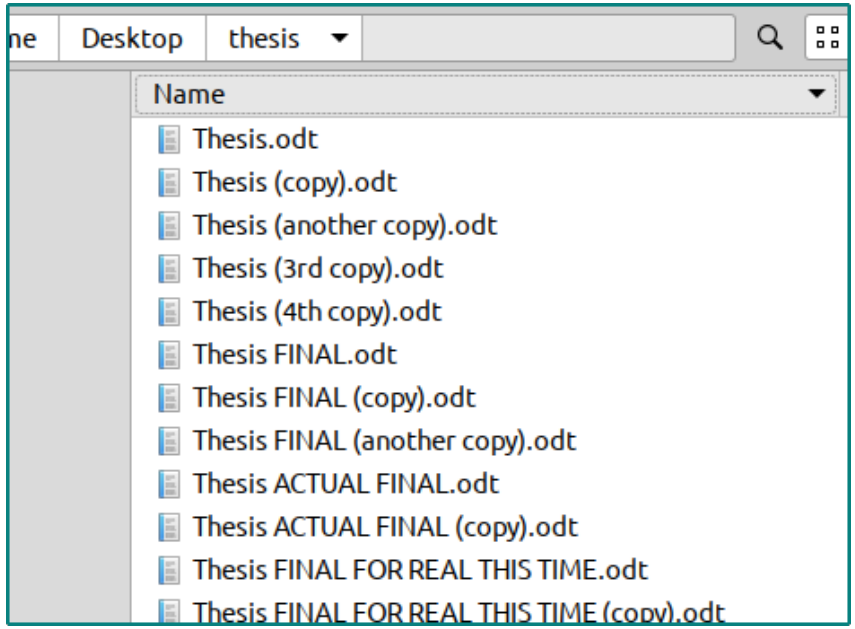
27 Aug 2021

Doorway to Human History

What is Git?

Why?

- I don't like to lose work.
- I like it when people work together without getting in each other's way.
- I like to be able to make mistakes.



Short fact sheet

- Git is a Distributed Version Control System.
- Created in 2005 by Linus Torvalds (the creator of Linux).
- Free and Open-Source Software.
- Probably the most popular VCS out there.

Photo: Krd (CC BY-SA 3.0), via Wikimedia Commons

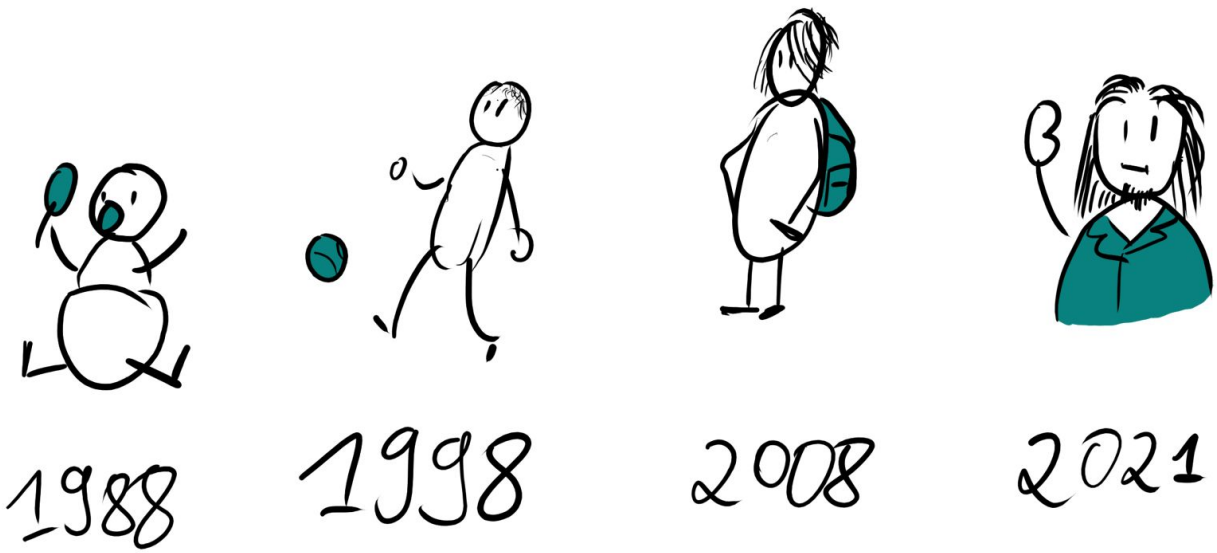
Version control

A Version Control System (VCS) keeps track of different versions of a project.

Two meanings of ‘version’

Meaning 1

‘Version’ as in the state of a project at a given point in time.
$\rightarrow$ i. e. ‘old version’ vs. ‘new version’



Two meanings of ‘version’ (ctd.)

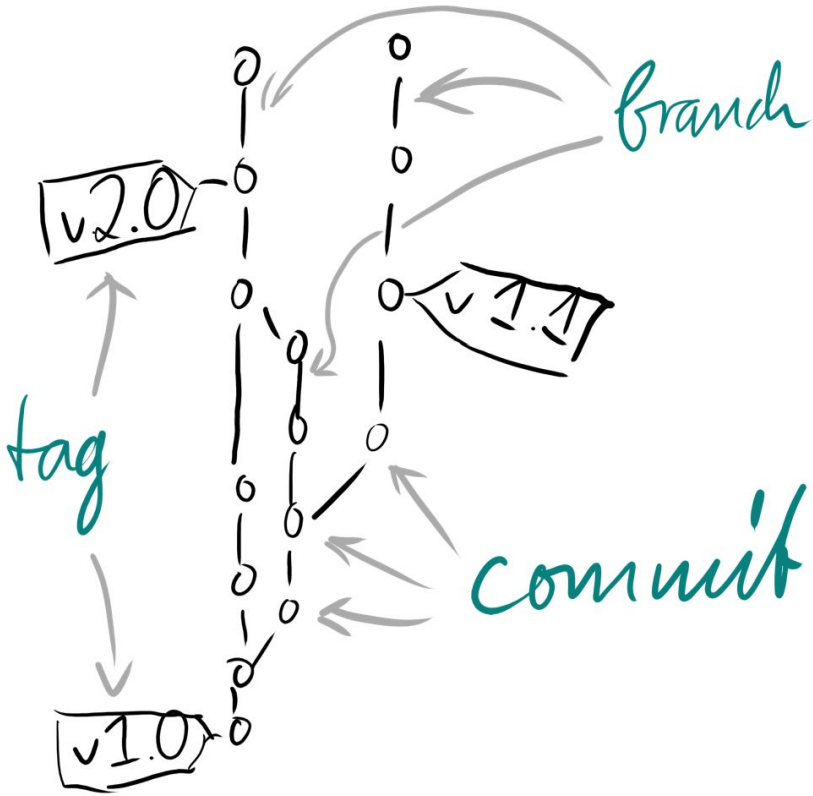
Meaning 2

‘Version’ as in different alternative editions of the same project, existing next to each other.

$\rightarrow$ ‘regular version’ vs. ‘evil alternate universe version’

Two meanings of ‘version’ (ctd.)

Version Control Systems do both!



What can VCS's be used for?

- Programming
- Configuration files
- Writing an article together in LaTeX, Markdown, etc.
- Reproducible and citable research data
  (we like that around here)
- Personal to-do lists, notes, etc.
- These slides ;)

Know your limits!

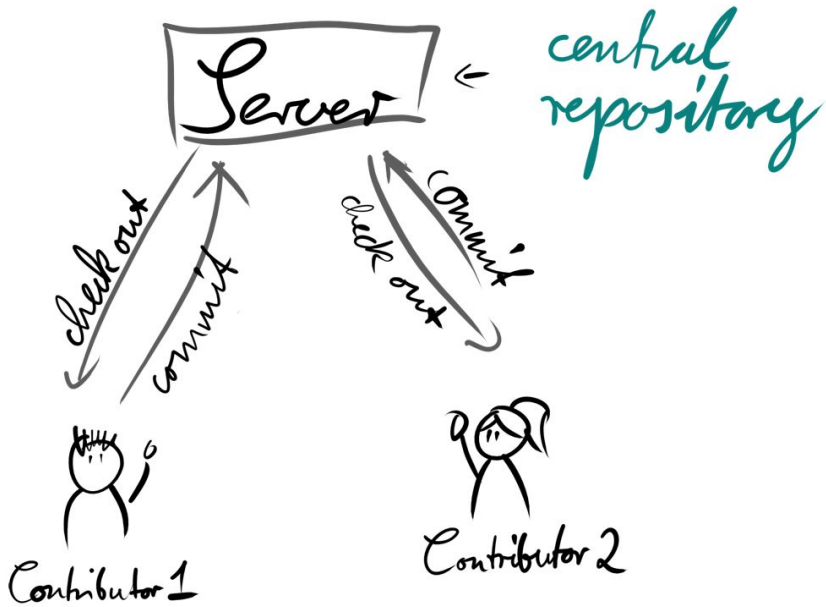
What are VCS's good at?

Text-based data (plain-text files, program code, XML/HTML, LaTeX, CSV tables, etc.)

What are (most) VCS's bad at?

Binary data (images, audio files, pdfs, zip archives, etc.)

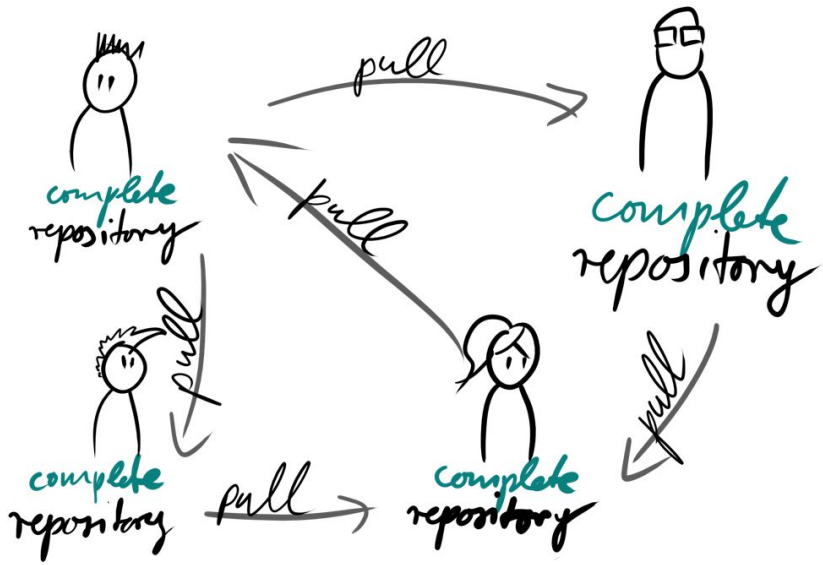
Centralised vs. Distributed VCS



Centralised VCS (e. g. CVS, subversion)

- Central repository on some server somewhere.
- People download the current state of the project to their computers.
- People upload any changes they made back onto the server.

Centralised vs. Distributed VCS (ctd.)



Distributed Version Control (e. g. git, mercurial)

- Everybody has a complete copy of the project (and all its history) on their hard drives.
- Changes are pushed and pulled between the different repositories.

Centralised vs. Distributed VCS (ctd.)

Note

In reality, people put a separate copy of the project on a server somewhere and pull/push their changes through that.

- Some project spin up their own git servers, e. g.:
  - Linux kernel$^{\hookrightarrow}$
  - GNU operating system$^{\hookrightarrow}$
- Others use third-party hosting sites:
  - Microsoft's Github$^{\hookrightarrow}$ (most popular)
  - Atlassian's Bitbucket$^{\hookrightarrow}$
  - Gitlab$^{\hookrightarrow}$

Know your limits! (ctd.)

Advantages of Distributed Version Control

- No need for a constant internet connection.
- Quick and comfy workflow (local branches are the best!).
- The massive redundancy reduces the chance of data loss.

Drawbacks

- Workflow of pulling/merging can be a bit finicky at times.
- Copies get out of sync with each other more easily, increasing the number of merge conflicts.
- Copying around the entire history wastes disk space and bandwidth, and also just takes longer.

Quick terminology note

What is a ‘repository’?

(coll.: repo or repos)

- General tech speak: online storage space
- Git speak: a copy of a version-controlled project folder

Getting git set up

Download/Installation

- GNU/Linux, *BSD: it's in the repos
- Windows: https://git-scm.com/downloads
- macOS:
  - Option 1: install the XCode command-line tools:
    ```
    $ xcode-select --install
    ```
  - Option 2: https://git-scm.com/downloads
  - Option 3: it's on homebrew

User interface

- Git uses a command-line interface
- There are graphical interfaces and editor plugins for common tasks, but at the end of the day the command-line is the preferred way to use git

```
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--ir
           [-p | --paginate | -P | --no-pager] [--no-replace-objec
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<n
           <command> [<args>]

These are common Git commands used in various situations:
[...]
```


First setup

Tell git your name and email address

```
$ git config --global user.name "Johannes Englisch"
$ git config --global user.email "englisch@shh.mpg.de"
```

- The name and address will be attached to every commit you make.
- This is completely unrelated to any Github/Bitbucket/ etc. accounts you might have!

First setup (ctd.)

Tell git about your text editor

```
$ git config --global core.editor \
    "C:\Program Files\Notepad++\Notepad++.exe"
$ git config --global core.editor "/usr/bin/emacs"
$ ...
```

→ The official installer also allows to set the editor (at least on Windows).

First setup (ctd.)

Avoid potential headache regarding line endings

- Windows:

```
$ git config --global core.autocrlf true
```

- GNU/Linux, *BSD, macOS, etc.:

```
$ git config --global core.autocrlf input
```


First setup (ctd.)

Review your settings

```
$ git config -l
```


Exploring a git repo

Cloning an existing repo

```
$ git clone https://github.com/dictionaria/kalamang kalamang
Cloning into 'kalamang'...
remote: Enumerating objects: 155, done.
remote: Counting objects: 100% (155/155), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 155 (delta 81), reused 139 (delta 68), pack-reused 0
Receiving objects: 100% (155/155), 927.47 KiB | 7.73 MiB/s, done.
Resolving deltas: 100% (81/81), done.
```


Status report!

```
$ cd kalamang
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.

nothing to commit, working tree clean
```


History lesson

```
$ git log
commit 5f28ae268e88d70e2f5f9ca2497a7e8fed257611 (HEAD -> main, tag
Author: Robert Forkel <...>
Date:   Fri Apr 9 13:27:29 2021 +0200

    re-compiled against Glottolog 4.3

commit 3d98441885adb1037afcbde9478d8ae4c0330567
Author: Johannes Englisch <englisch@shh.mpg.de>
Date:   Wed Apr 7 14:41:57 2021 +0200

    regen with current pydictionaria version
[...]
```


Branches, Tags

```
$ git branch
* main
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/link-to-concepticon
  remotes/origin/main
$ git tag
v1.0
v1.1
```


Time to time travel

```
$ git checkout v1.0
[...]
HEAD is now at 64ba30e release 1.0
$ git checkout link-to-concepticon
[...]
Switched to a new branch 'link-to-concepticon'
$ git checkout d1e45b9
[...]
HEAD is now at d1e45b9 metadata regen
```

$\rightarrow$ re. commits: you don't have to type all 40 digits – only enough to make it unique (usually like 7 or so digits)

Your very own git repo

Creating a new git repo

```
$ cd <folder>
$ git init
Initialised empty Git repository in <folder>/.git
```


Reviewing your changes

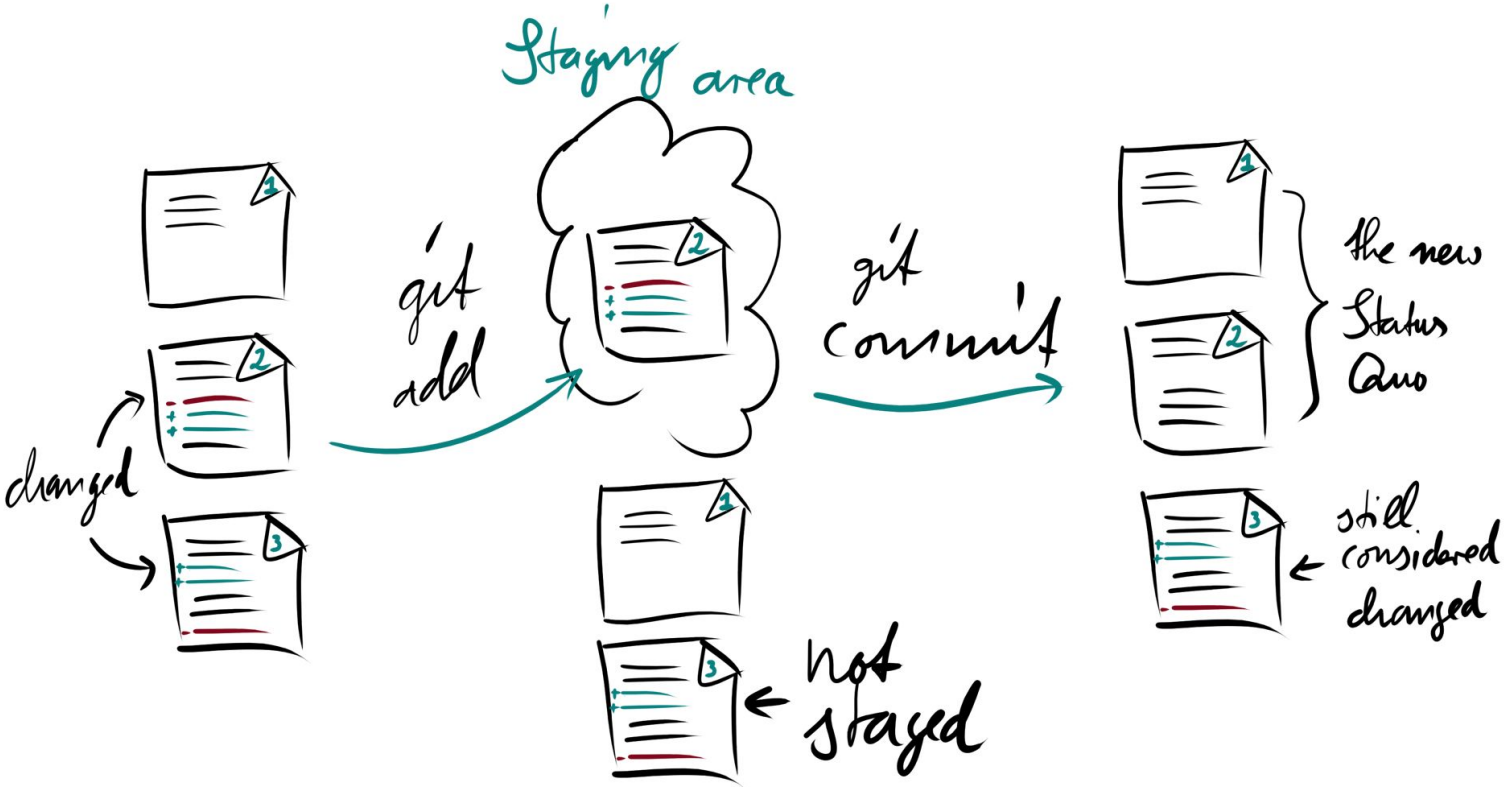
What files changed since the last commit?

```
$ git status
```

What lines changed since the last commit?

```
$ git diff
```


Making a commit is a two-step process



1. Add changes to the Staging Area.
2. Commit to these changes (and leave a commit message).

Adding changes to the staging area

```
$ git add file1 file2 [...]
```


Time for some commitment

Short version

```
$ git commit -m "short, yet helpful commit message :D"
```

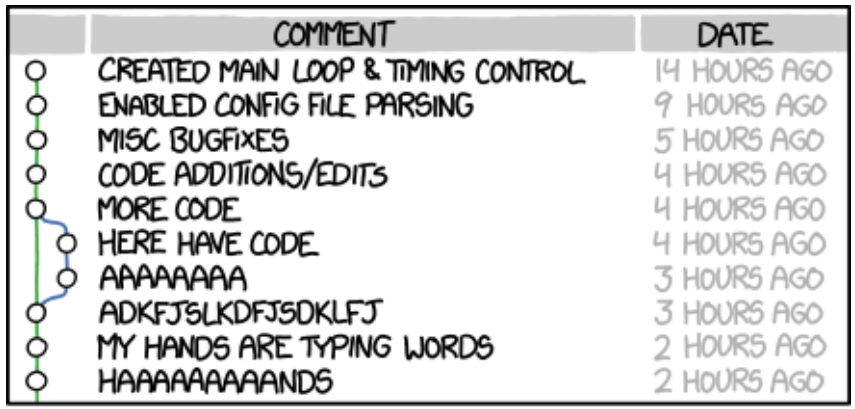
‘Proper’ version

```
$ git commit
```

The latter will open your text editor for your commit message:

- Save+quit to create the commit.
- Leave the message blank (or quit without saving) to abort the commit.

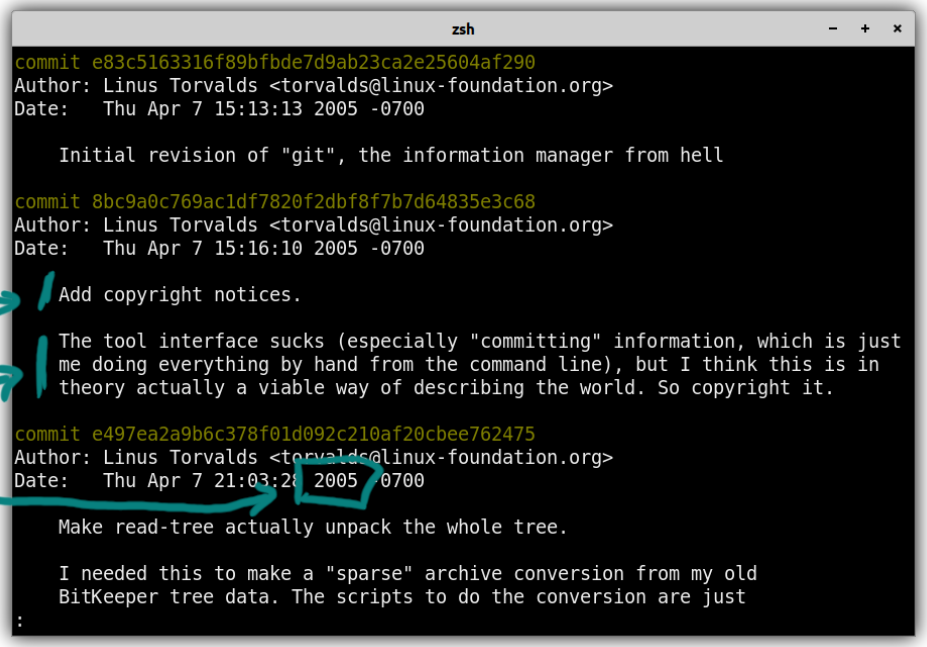
Good and helpful commit messages



by Randall Monroe (CC BY-NC 2.5), https://xkcd.com/1296/

Good and helpful commit messages (ctd.)

- One-line summary (max. 50 chars.)
- More detailed description of what changed and why
- Keep in mind that commit messages stick around for a long time



Good and helpful commit messages (ctd.)

Good one-liners

A good technique for a one-line commit message is to focus more on why a commit was made.

Example

don't: ~~`updated readme`~~

do: `forgot Steve in the author listing`

Changing your mind

I'm not ready to commit to this yet!

```
$ git reset file.txt
```

Note: This does not delete or change any files – it just removes them from the Staging Area.

Changing your mind (ctd.)

Crap, I messed around with the project and now everything is terrible! Let's just throw this away and start over...

```
$ git checkout file.txt
```

- This will undo any changes to `file.txt` that haven't been committed or staged, yet.
- This process is irreversible – tread with caution!
- Yes, `checkout` serves double-duty...

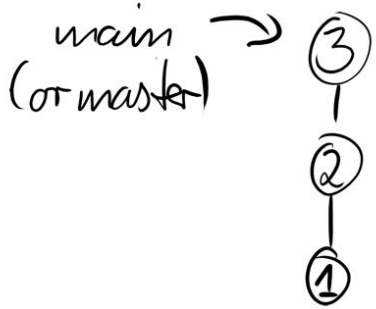
Ignore me

There are a bunch of (temp) files in my folder that I don't want to add to my git repo

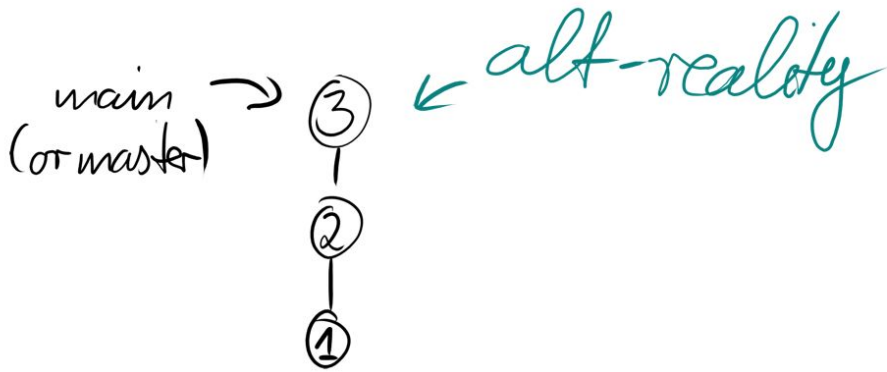
- Create a text file called `.gitignore` (nothing before the dot; no extension at the end).
- Add a new line for every file you want to ignore.
- The `.gitignore` file is just another file you can add/commit/etc. to your git repo.

Casually making parallel universes

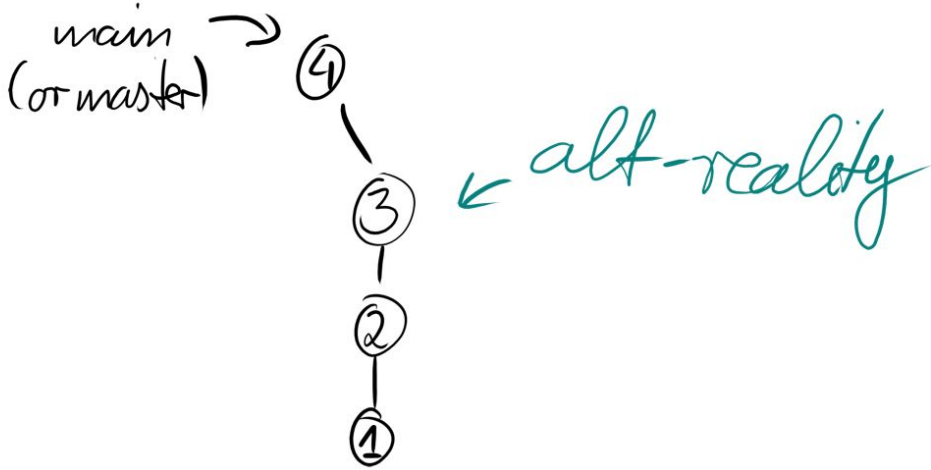
How branches work



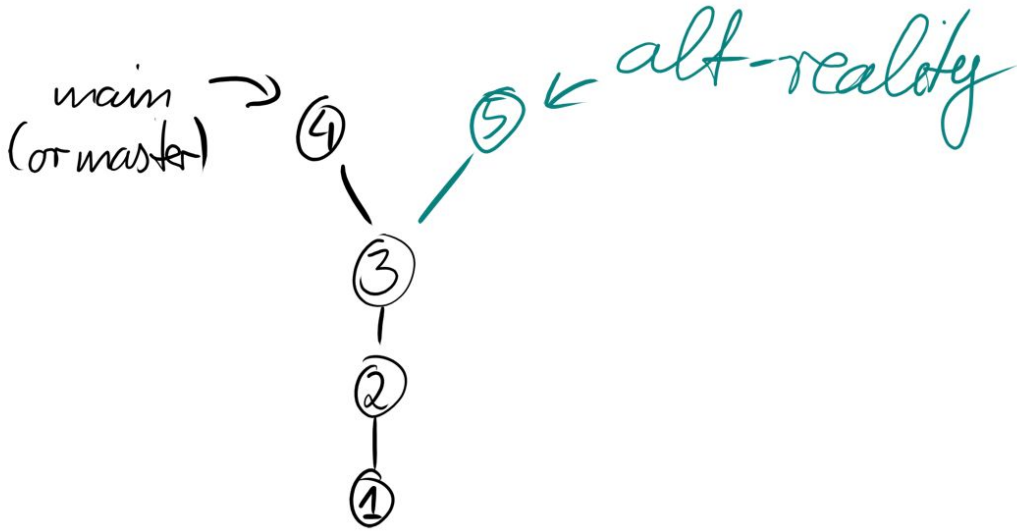
How branches work (ctd.)



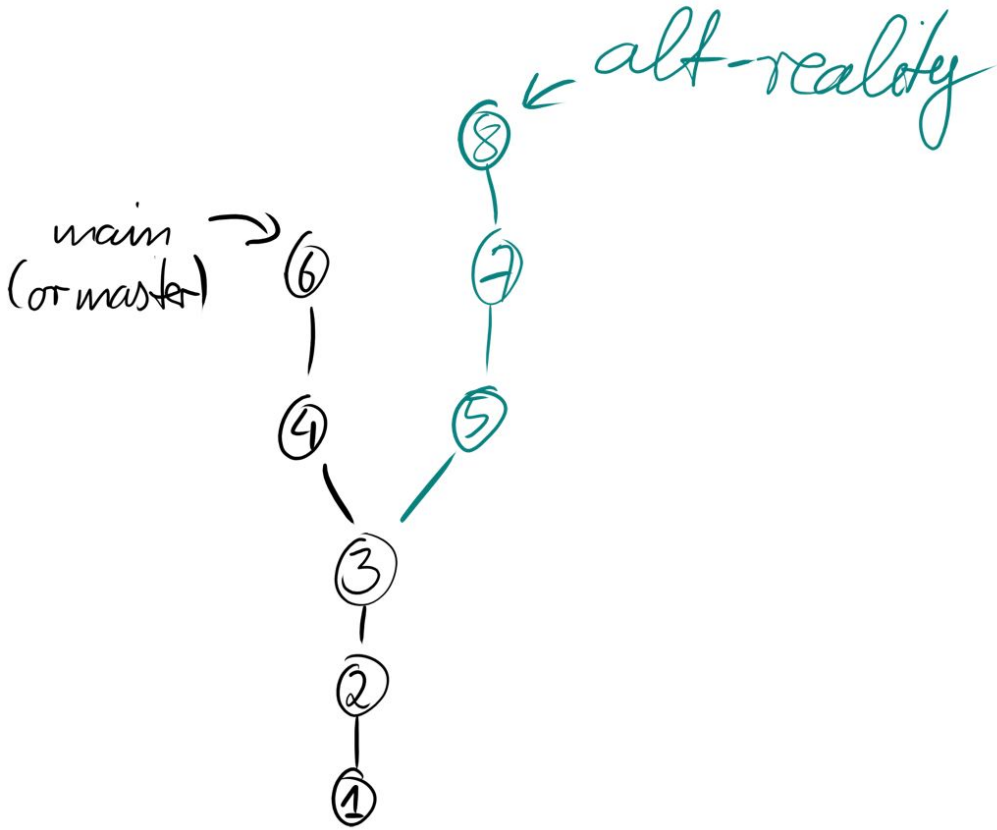
How branches work (ctd.)



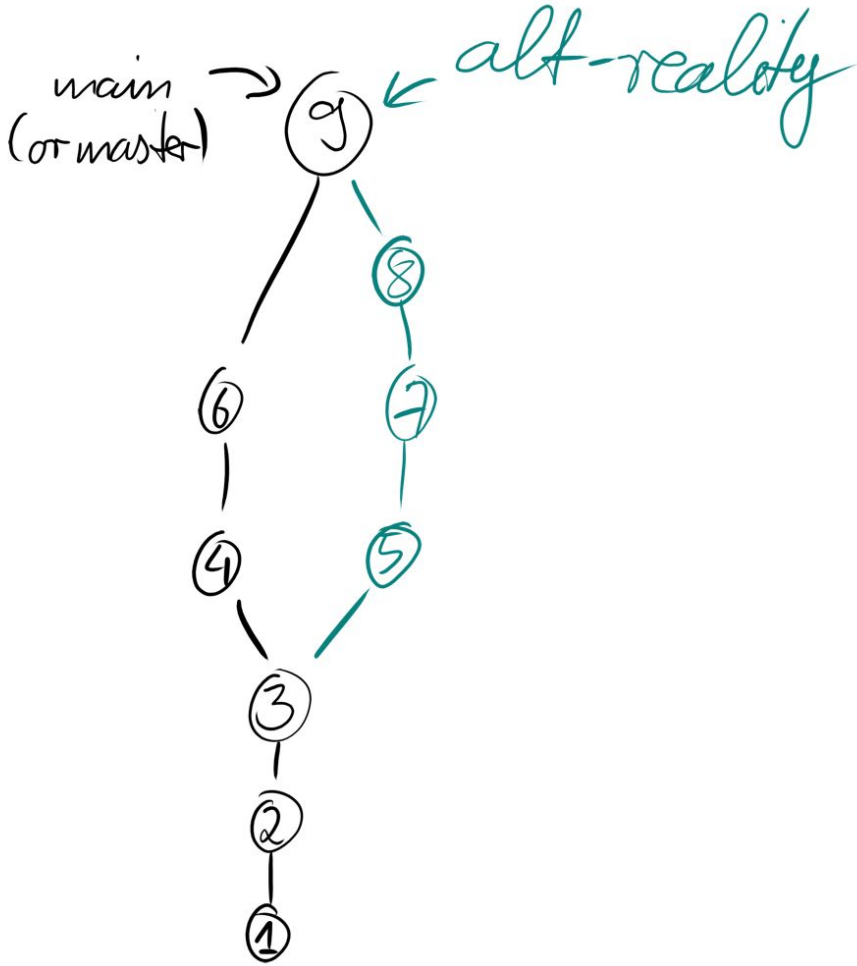
How branches work (ctd.)



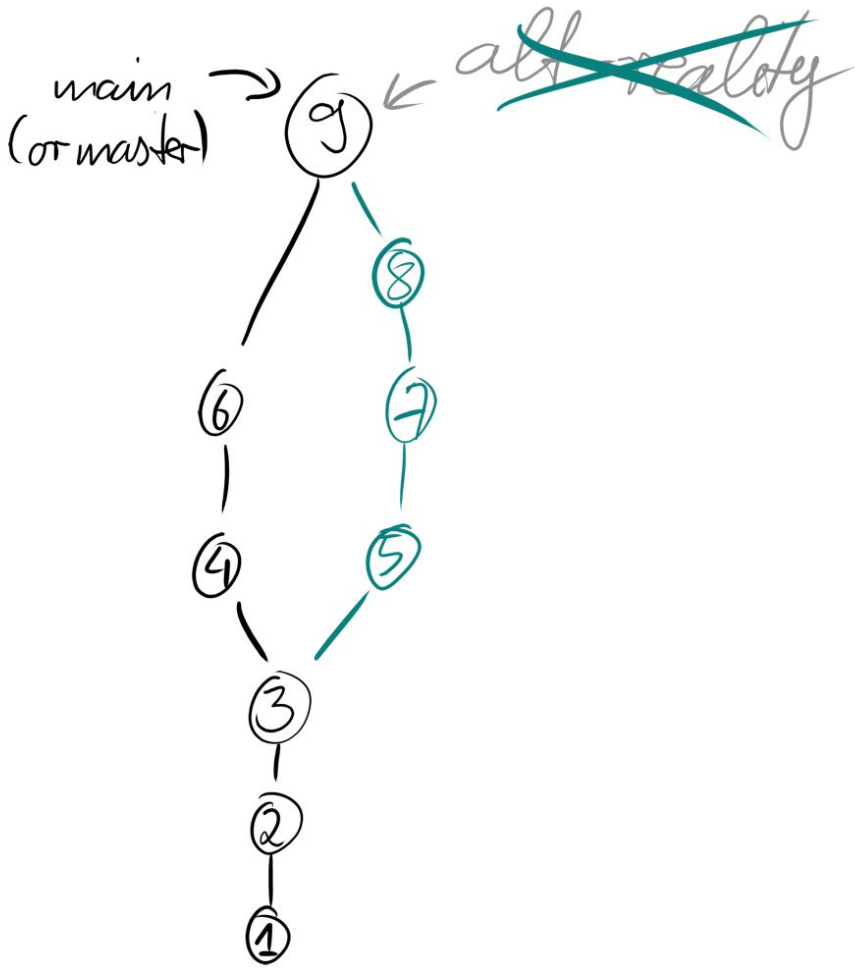
How branches work (ctd.)



How branches work (ctd.)



How branches work (ctd.)



Relevant commands

Make new branch calles `alt-reality`

```
$ git branch alt-reality
```

Switch to branch

```
$ git checkout alt-reality
```


Relevant commands (ctd.)

Merge branches

```
$ git branch main
$ git merge alt-reality
```

Remove the merged branch

```
$ git branch -d alt-reality
```


How does merging work?

Best case scenario

Git applies all the changes made in both branches to the project and everybody is happy. `\(^^)/`

How does merging work? (ctd.)

More annoying scenario

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


How does merging work? (ctd.)

What to do?

- Look at `git status` to see which files are affected.
- Manually edit the relevant files.
- Especially look out for this pattern:

```
<<<<<<< HEAD
stuff in the base branch
=======
stuff in the merged branch
>>>>>>> alt-reality
```

- `add` and `commit`

Publish your work on Github

Basic idea

- Create an empty git repo on Github (or anywhere else).
- Tell your local git repo about it.
- Push your local history to the remote repo.

Alternative workflow for new projects

- Create an empty git repo on Github (or anywhere else).
- Clone the empty onto your computer.

Github does not know who you are

- You can only clone/pull from repos you have read access to.
- You can only push to repos you have write access to.
- Git needs to send some sort of ID or else Github will refuse to cooperate.
- Git cannot look into your browser, so being logged on to the Github website doesn't do anything...

Two methods

- Personal access token
  (not gonna cover that one here – check Github's documentation$^{\hookrightarrow}$ for details)
- Secure Shell (SSH)

Secure Shell (SSH)

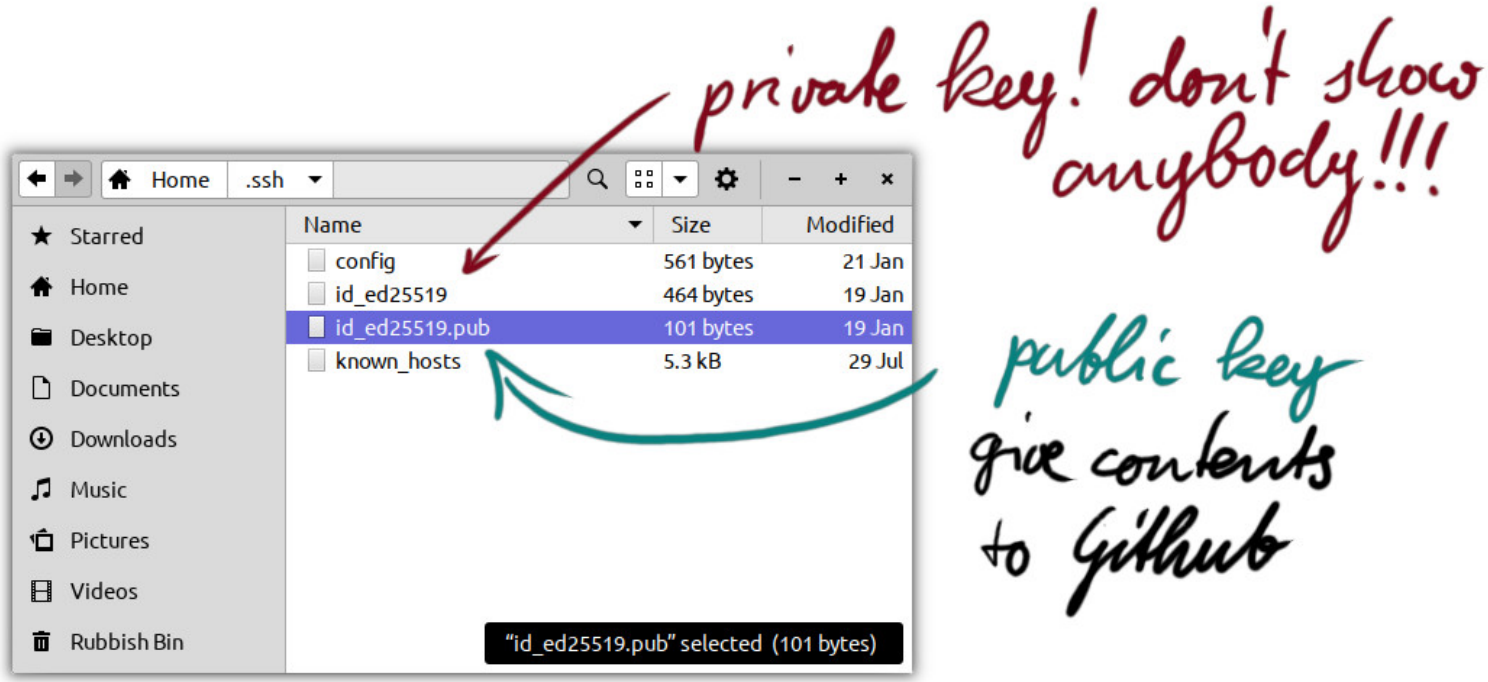
Create a key pair (if you don't have one yet)

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

→ see Github's documentation on creating an SSH key↪

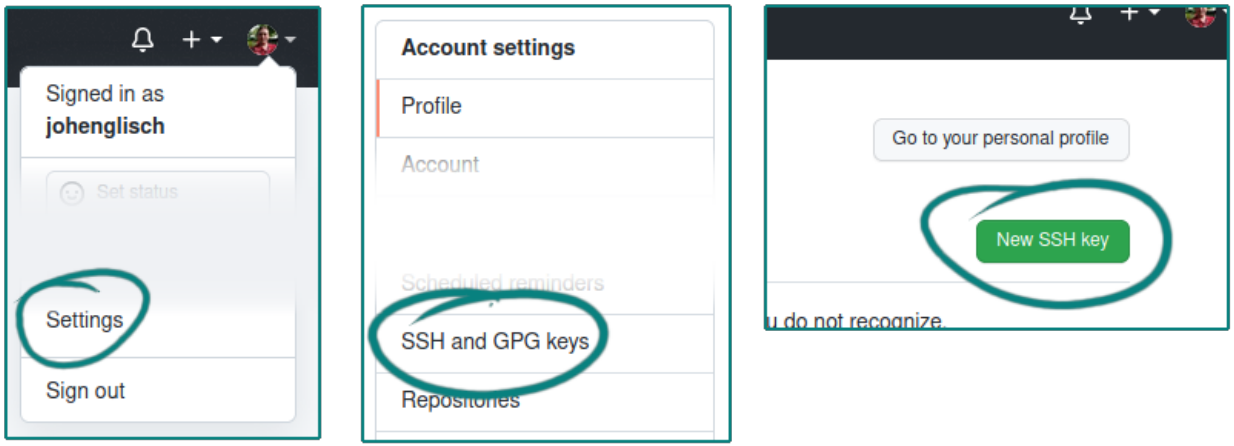
Secure Shell (ctd.)

Find your public key



Secure Shell (ctd.)

Give your public key to Github



- Copy the contents of your public key file into the text field.
- See Github's documentation on adding an SSH key

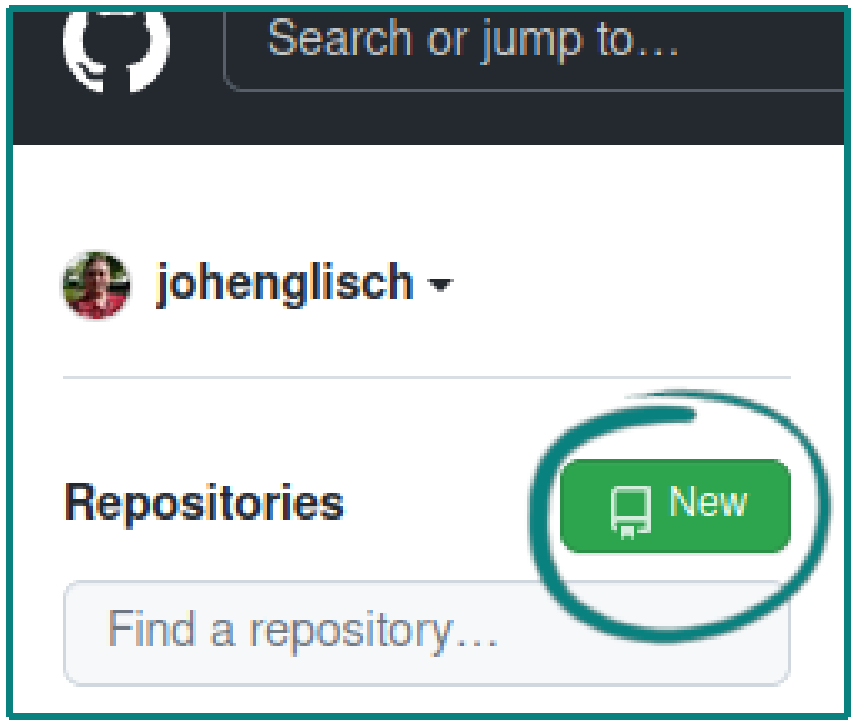
Cloning a git repo over SSH

- Cloning over SSH requires a different URL than regular cloning:

  `git@github.com:<user name>/<repo name>`

```
$ git clone git@github.com:johenglisch/git-intro-2021
```


Creating a new repo on Github



Hooking your local git repo up with Github

```
$ git remote add origin \
    git@github.com:johenglisch/git-intro-2021
```

- This connects your local git repo to a remote repo.
- Note that we are adding the SSH URL rather than the regular HTTPS one.

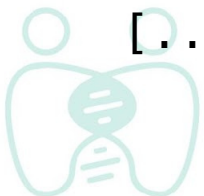
Keeping the repos in sync

Uploading for the first time

```
$ git push -u origin HEAD
```


Keeping the repos in sync (ctd.)

```
$ git pull
remote: Enumerating objects: 364, done.
remote: Counting objects: 100% (364/364), done.
remote: Compressing objects: 100% (197/197), done.
remote: Total 364 (delta 232), reused 256 (delta 127), pack-reused
Receiving objects: 100% (364/364), 85.43 KiB | 4.07 MiB/s, done.
Resolving deltas: 100% (232/232), completed with 19 local objects.
From https://github.com/cldf/pycldf
   61bfadd..275bb9f  master     -> origin/master
 * [new tag]         v1.20.0    -> v1.20.0
[...]
 * [new tag]         v1.23.0    -> v1.23.0
Updating 61bfadd..275bb9f
Fast-forward
 CHANGELOG.md                                       |  39 +
 CONTRIBUTING.md                                    |  10 +-
[...]
```



I am a dirty liar (<_<)"

Actually, `git pull` does two things:

1. `git fetch`: Sync all branches from the remote repository.
2. `git merge`: Merge the remote version of the current branch (e. g. `origin/main`) with the local version of the current branch (e. g. `main`)

Keeping the repos in sync (ctd.)

Upload new changes

```
$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 701 bytes | 350.00 KiB/s, done.
Total 7 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), completed with 4 local objec
To github.com:johenglisch/git-intro-2021
   e96bea1..a27e261  master -> master
```


Finishing notes

Links

- Tutorial on the Git website:
  https://git-scm.com/docs/gittutorial
- Software carpentry lesson on git:
  https://swcarpentry.github.io/git-novice/