git and open-science

"the practice of science in such a way that others can collaborate and contribute, where research data, lab notes, processes are freely available, under terms that enable reuse, redistribution and reproduction of the research and its underlying data and methods."

- FOSTER OpenScience

open-science: motivations

- **human right** to knowledge
- **accountability** → judged → *ethics consequences*
  - can reduce fraud, data manipulation, and selective reporting of results
- **reproducibility**: anyone, including you, can replicate the steps of an analysis
- **sustainability**
  - **value proposition**
  - **education**
  - **politics**

open-science: motivations

- a bit of pure "**healthy selfishness**"
  - open access citation advantage bias → greater visibility and exposure

obstacles

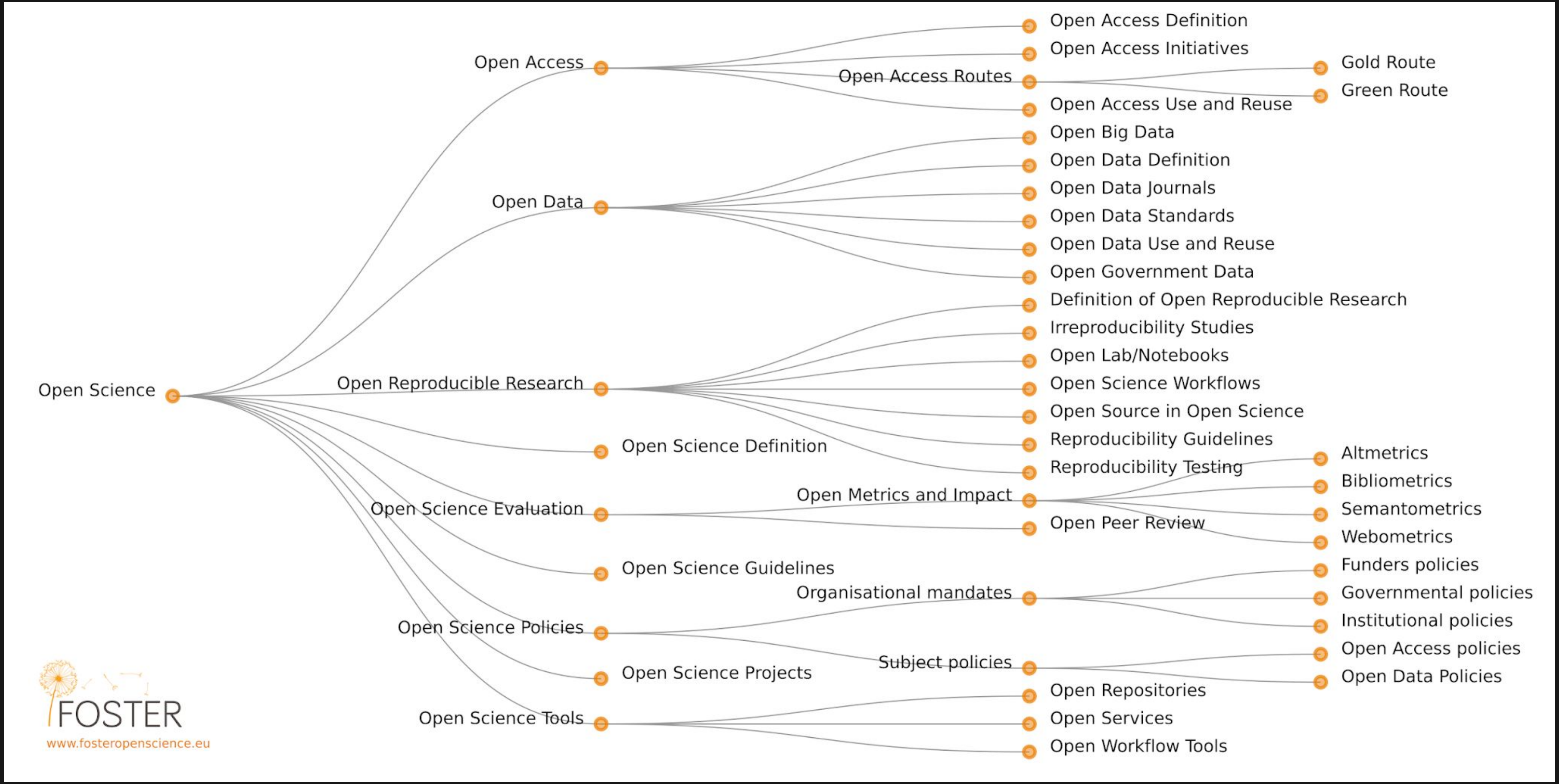
social

- ➢ prevailing trends and cultural inertia;
- ➢ academic culture can be hierarchical and conservative:
  - ○ resistance from peers, colleagues or coordinators;
- ➢ existing incentive mechanisms do not yet reflect the new culture of openness and collaboration;
- ➢ good understanding of ethical, social and academic benefits is required

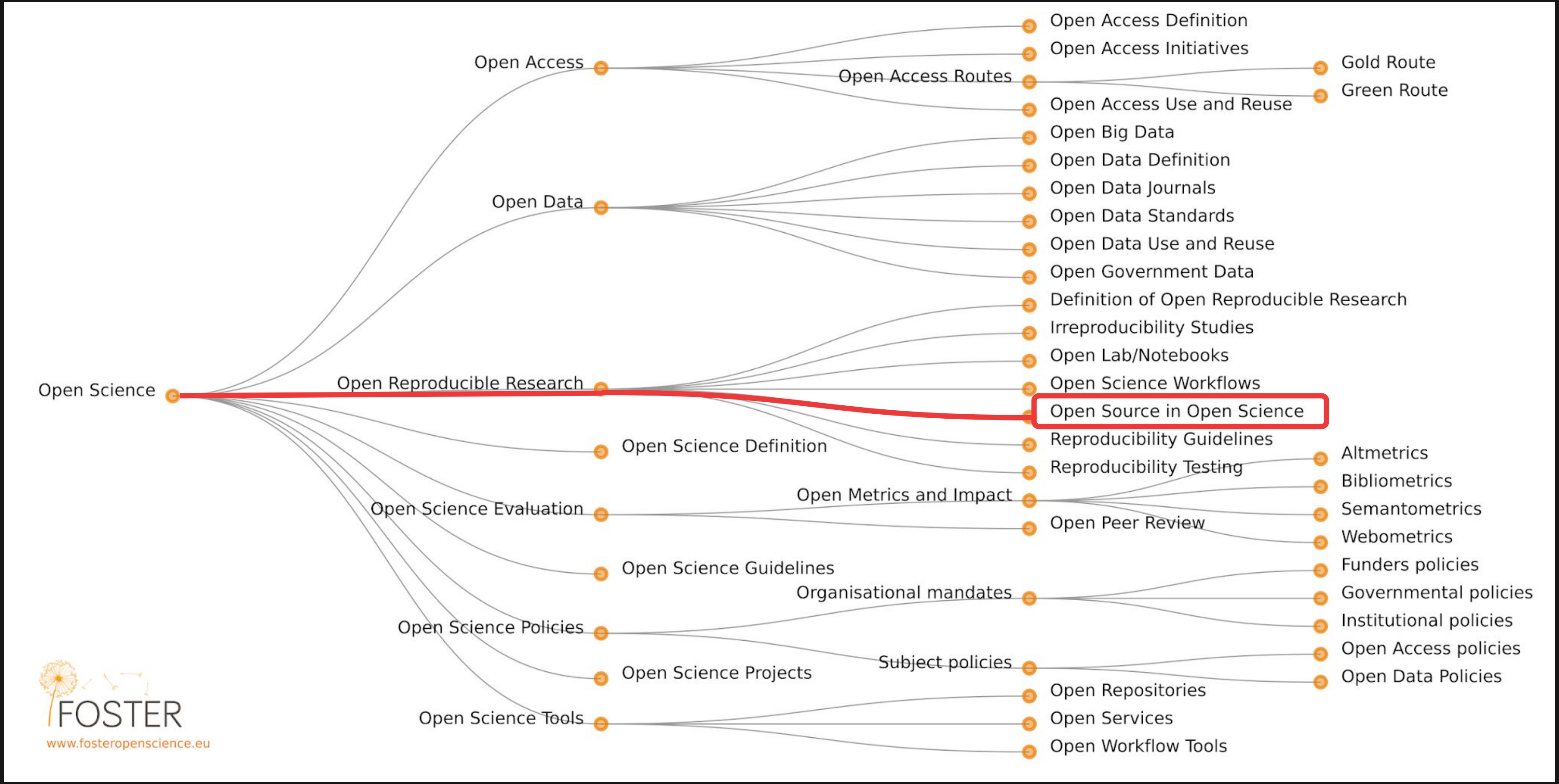
obstacles
economic

- high publication prices were made possible by monopoly-like mechanisms in academic publishing
  - unequal distribution of knowledge;

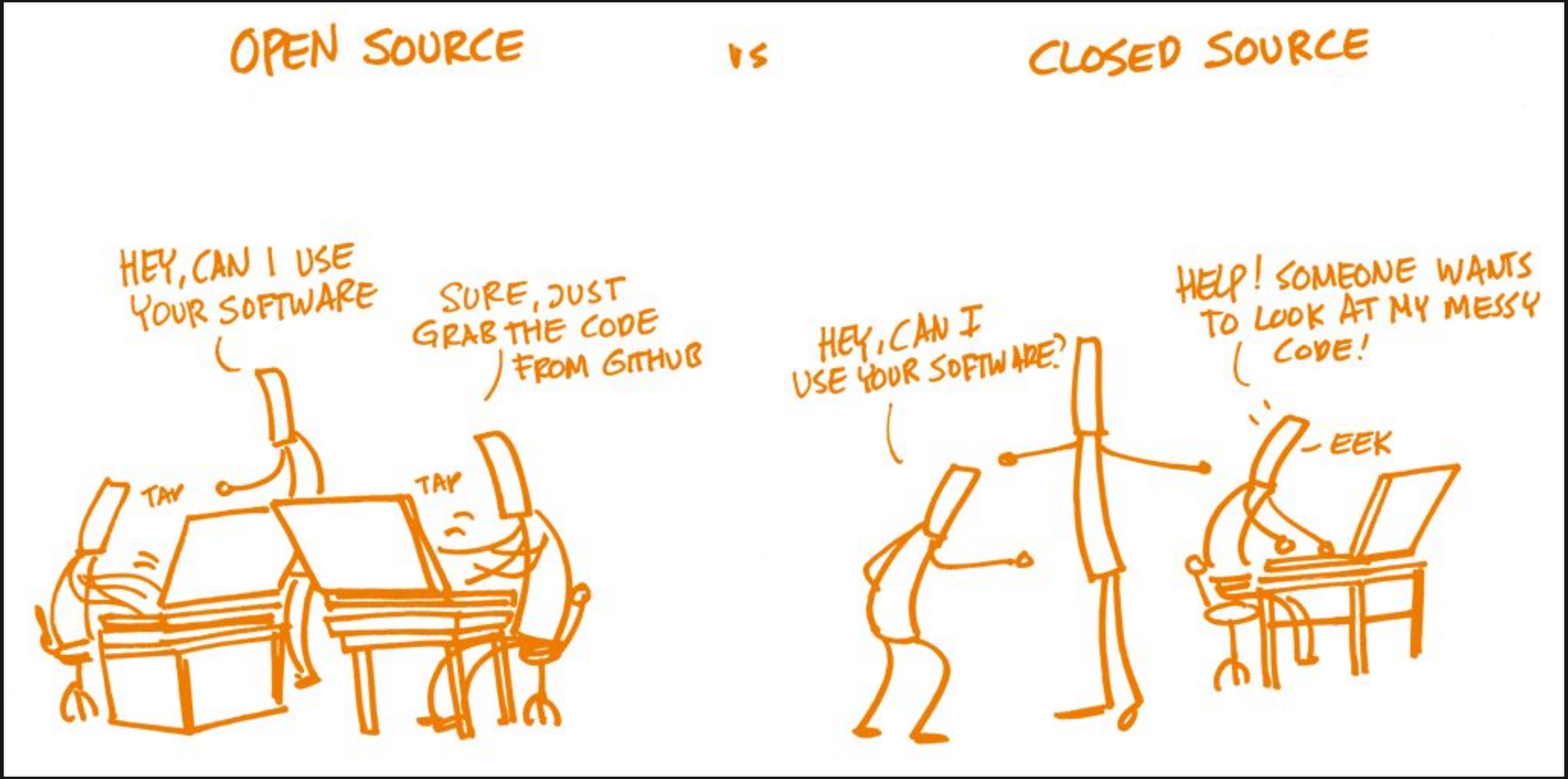
open-*everything*



open-*source*



Open Science
Open Access
Open Access Definition
Open Access Initiatives
Open Access Routes
Gold Route
Green Route
Open Access Use and Reuse
Open Data
Open Big Data
Open Data Definition
Open Data Journals
Open Data Standards
Open Data Use and Reuse
Open Government Data
Open Reproducible Research
Definition of Open Reproducible Research
Irreproducibility Studies
Open Lab/Notebooks
Open Science Workflows
Open Source in Open Science
Reproducibility Guidelines
Reproducibility Testing
Open Science Definition
Open Science Evaluation
Open Metrics and Impact
Altmetrics
Bibliometrics
Semantometrics
Webometrics
Open Peer Review
Open Science Guidelines
Open Science Policies
Organisational mandates
Funders policies
Governmental policies
Institutional policies
Subject policies
Open Access policies
Open Data Policies
Open Science Projects
Open Science Tools
Open Repositories
Open Services
Open Workflow Tools
FOSTER
www.fosteropenscience.eu


OPEN SOURCE
VS
CLOSED SOURCE
HEY, CAN I USE YOUR SOFTWARE
SURE, JUST GRAB THE CODE FROM GITHUB
TAP
TAP
HEY, CAN I USE YOUR SOFTWARE?
HELP! SOMEONE WANTS TO LOOK AT MY MESSY CODE!
EEK

philosophy equation

SCI-HUB* : CLOSE ACCESS

=

OPEN SOURCE : PROPRIETARY SW

(*) LibGen is fine as well...

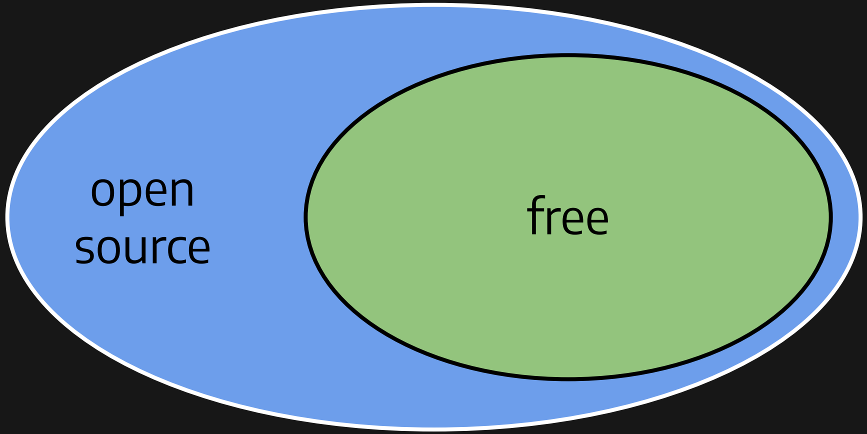
open-source

- principles
  - **transparency**: accessible materials → build upon each other's ideas and discoveries;
  - **collaboration**: solve problems that no one can solve alone;
  - **release often**: rapid prototypes → rapid discoveries;
  - **meritocracy**: good ideas can come from anywhere, and the best ideas should win;
  - **community**: shared goals >> individual interests

free software

- emphasis on **users' essential freedoms**: to run it, to study and change it, and to redistribute copies with or without changes;

- it is a matter of freedom, not price
  - think of "**free speech**" not "free beer"



what is (g)it?

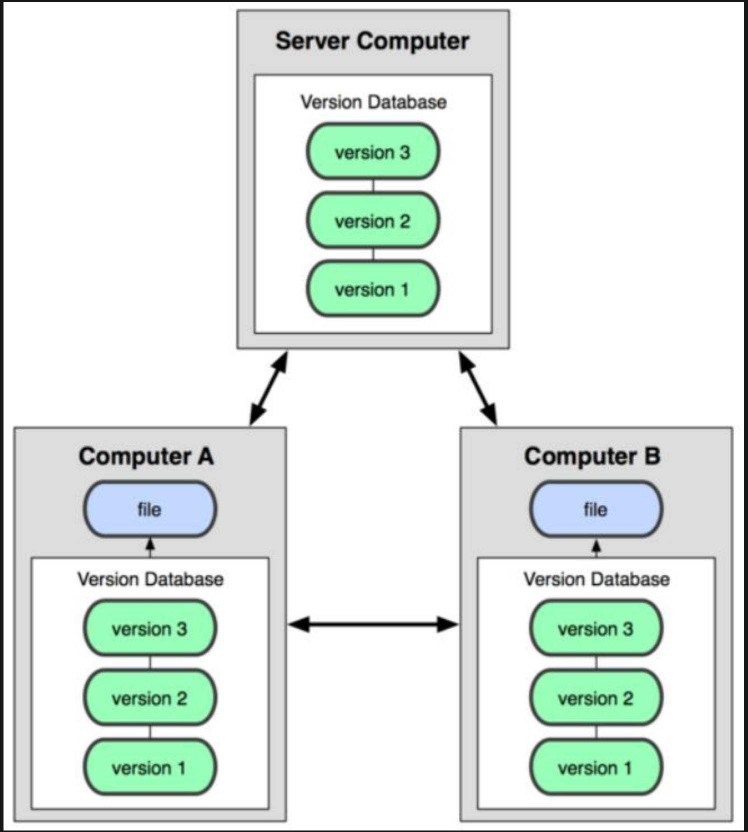
- developed by Linus Torvalds, the “inventor of linux”, in 2005
  - *de-facto* standard for software version control
  - free and open-source (GPLv2)
  - handles text files

- it is a **DVCS (*)**

- goals:
  - **fast** and capable of managing very large projects
  - fully **distributed**
  - supports non-linear development (**branches**)

git: distributed

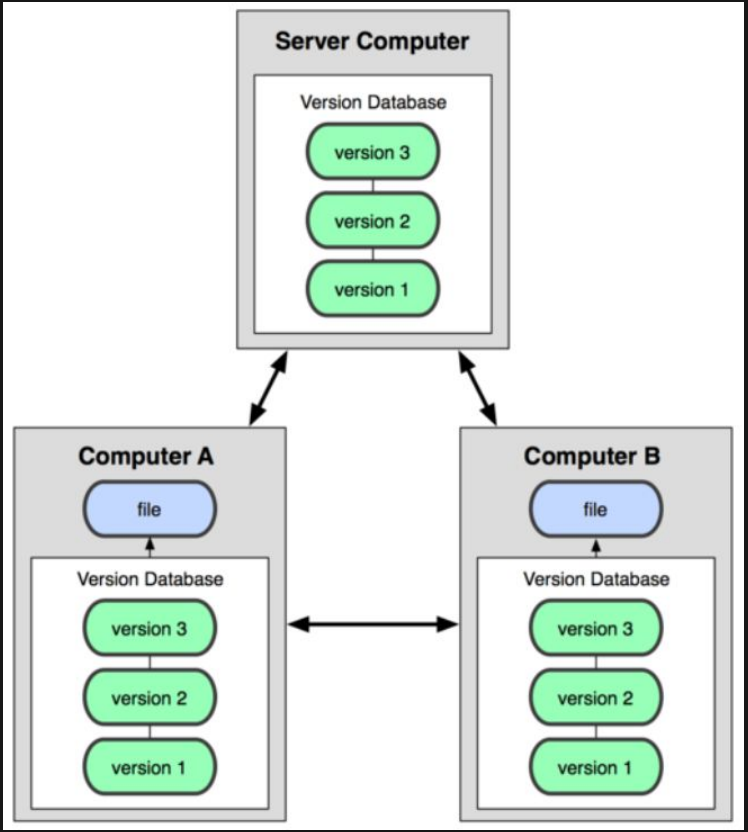
- you keep your files in a **repository** on your personal computer
- your local repo is a complete copy of everything on a **remote** server
- if you move to another machine, you just need to **clone** the repository again from a remote server
- everything is stored in a **.git directory**



git: distributed

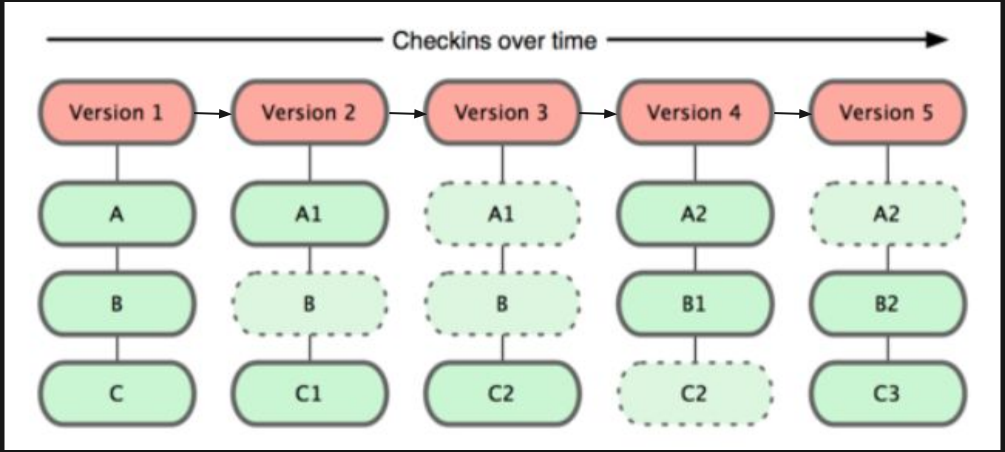
- you work on your local copy, when you are ready, you **push** your changes back to a server

- if your partners upload some changes to files, you can pick those up by synchronizing with (**pull**ing from) the server



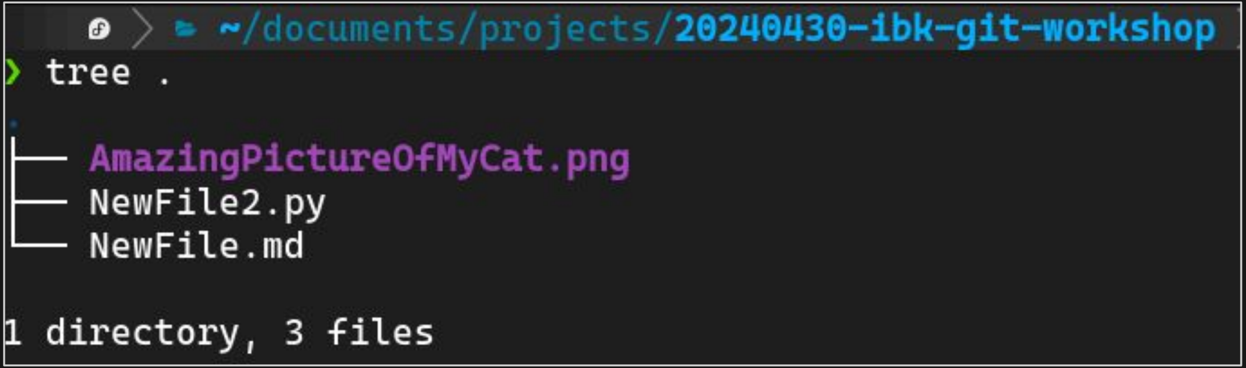
git: version-control

- a “version” is the state of the source code in a specific time in history
- version = **commit** (identified by an unique SHA1 hash, linked together)
- git keeps track of the history of your commits by using snapshots (Directed Acyclic Graph)
  - each version of the overall code has a copy of each file in it;
  - some files change on a given check-in, some do not;
  - redundant but fast



let’s start

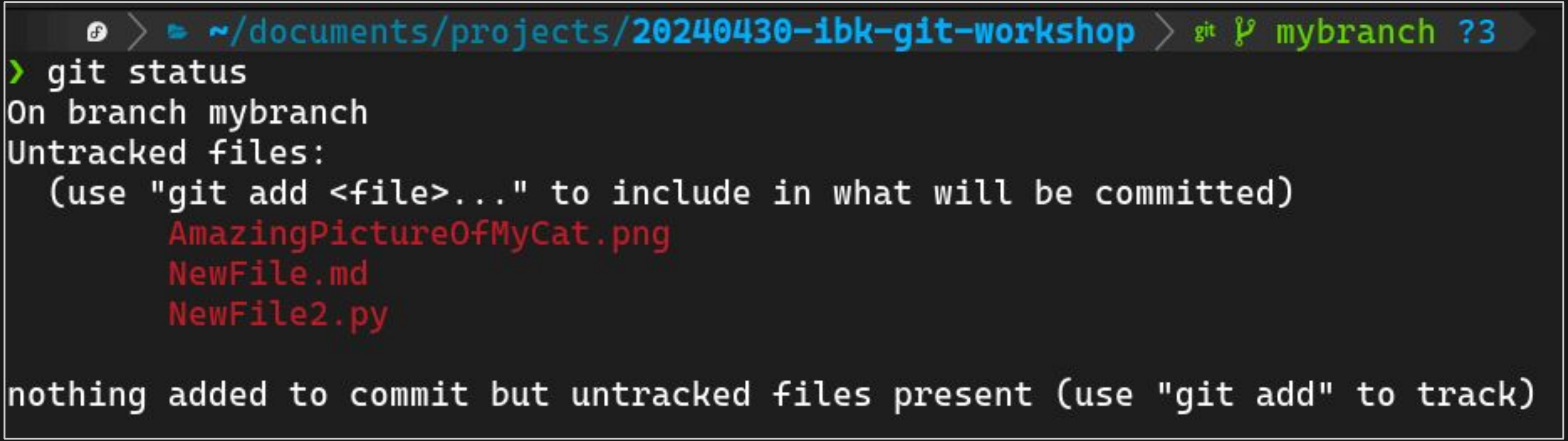
- simple directory with three files



```
 ~/documents/projects/20240430-ibk-git-workshop
❯ tree .
.
├── AmazingPictureOfMyCat.png
├── NewFile2.py
└── NewFile.md

1 directory, 3 files
```

working tree

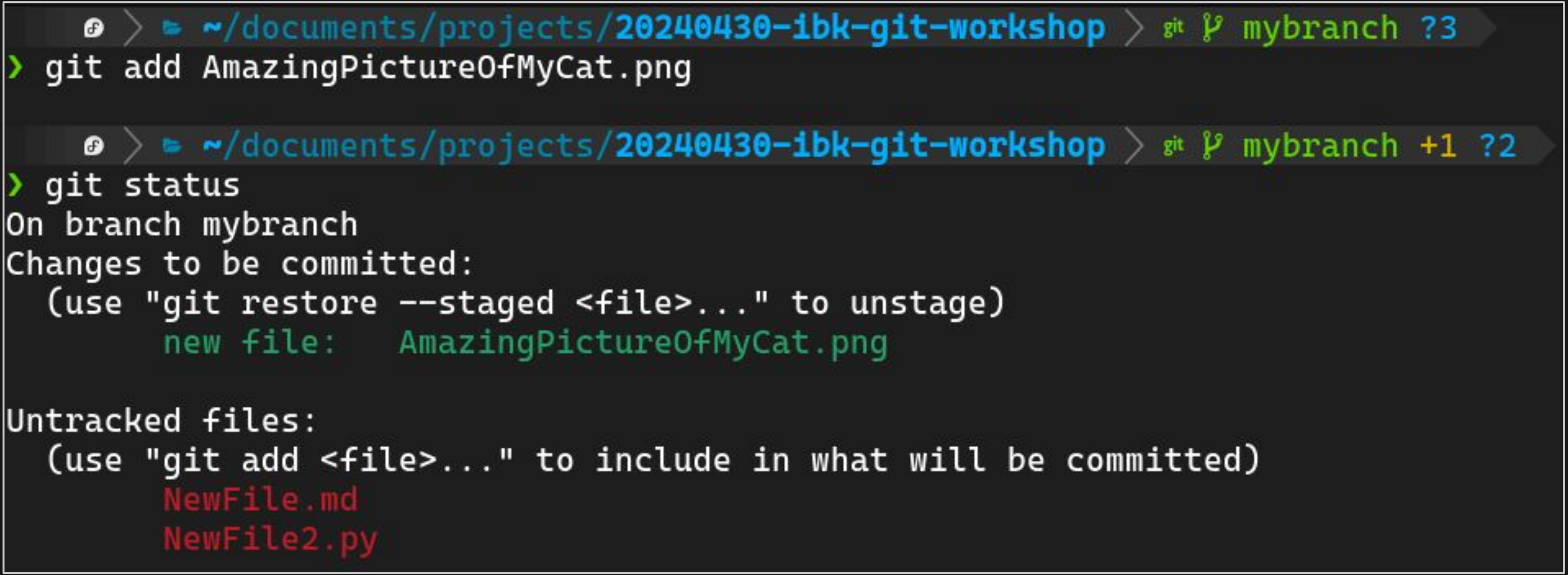
- is the area where you are currently working, where your files are;
- any change to files will be marked and seen by git;
- if you make changes and do not explicitly save them to git, you will lose the changes made to your file.



```
~/documents/projects/20240430-ibk-git-workshop  git  mybranch ?3
❯ git status
On branch mybranch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        AmazingPictureOfMyCat.png
        NewFile.md
        NewFile2.py

nothing added to commit but untracked files present (use "git add" to track)
```

staging area/index

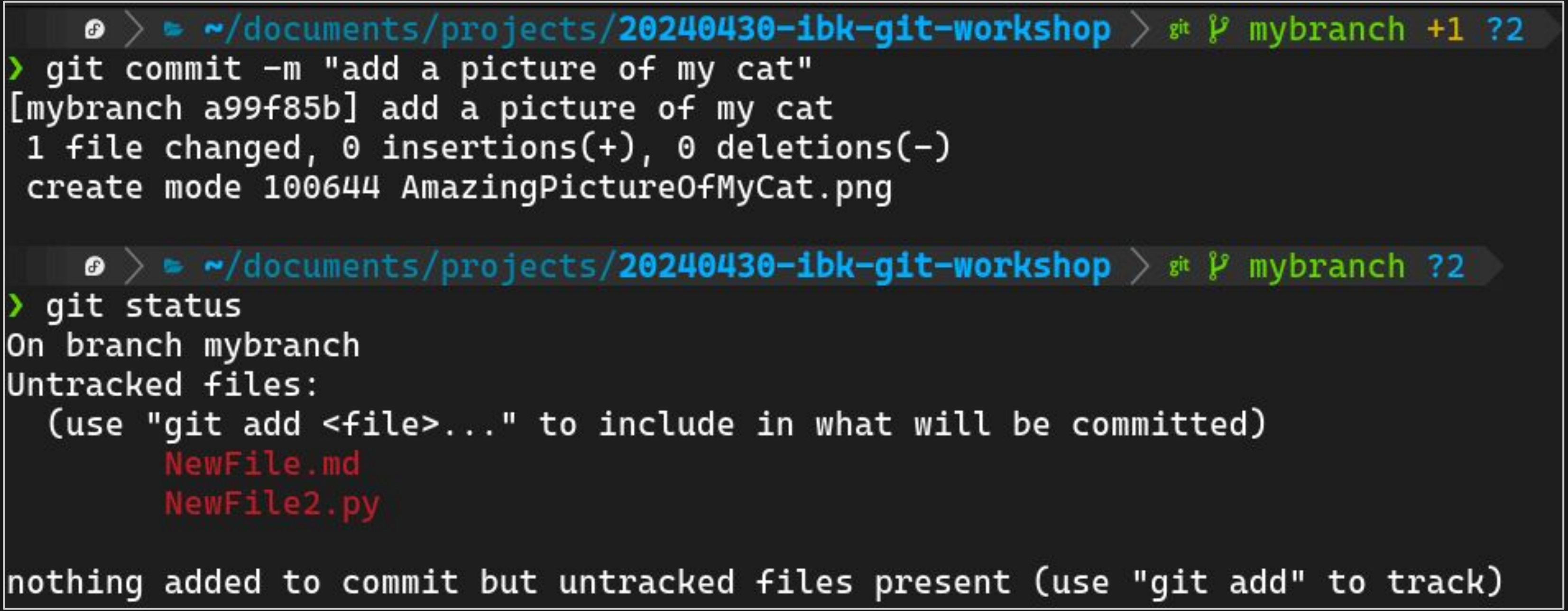
- is when git starts tracking and saving changes that occur in files
- if you make any more additional changes after adding a file to the staging area, you explicitly must inform git to take care of the edits in your files



```
 > ~/documents/projects/20240430-ibk-git-workshop > git mybranch ?3
> git add AmazingPictureOfMyCat.png

 > ~/documents/projects/20240430-ibk-git-workshop > git mybranch +1 ?2
> git status
On branch mybranch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   AmazingPictureOfMyCat.png

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        NewFile.md
        NewFile2.py
```

local repository

- ➢ is everything in your .git directory, mainly commits
- ➢ area that saves everything (don’t delete it)



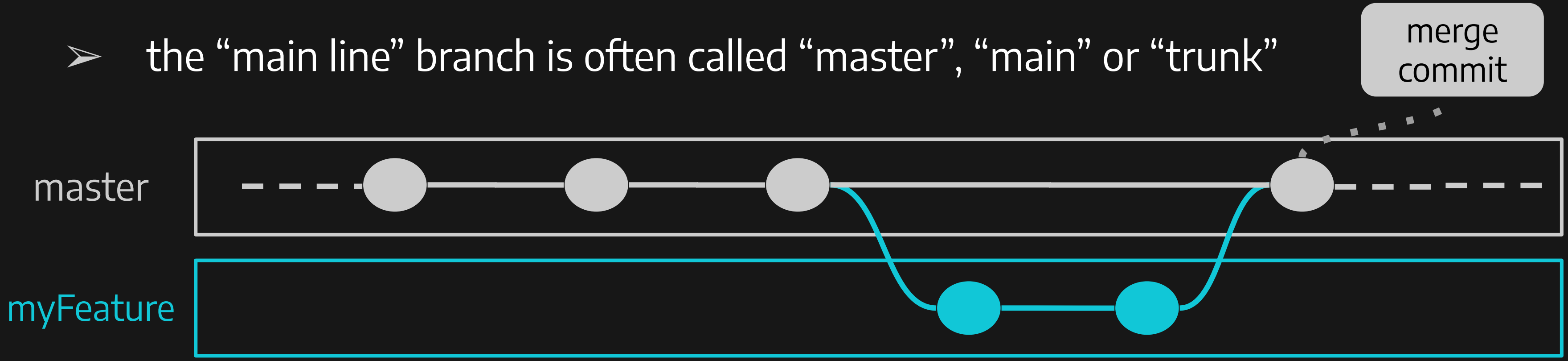
```
 ~/documents/projects/20240430-ibk-git-workshop  git  mybranch +1 ?2
❯ git commit -m "add a picture of my cat"
[mybranch a99f85b] add a picture of my cat
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 AmazingPictureOfMyCat.png

 ~/documents/projects/20240430-ibk-git-workshop  git  mybranch ?2
❯ git status
On branch mybranch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        NewFile.md
        NewFile2.py

nothing added to commit but untracked files present (use "git add" to track)
```

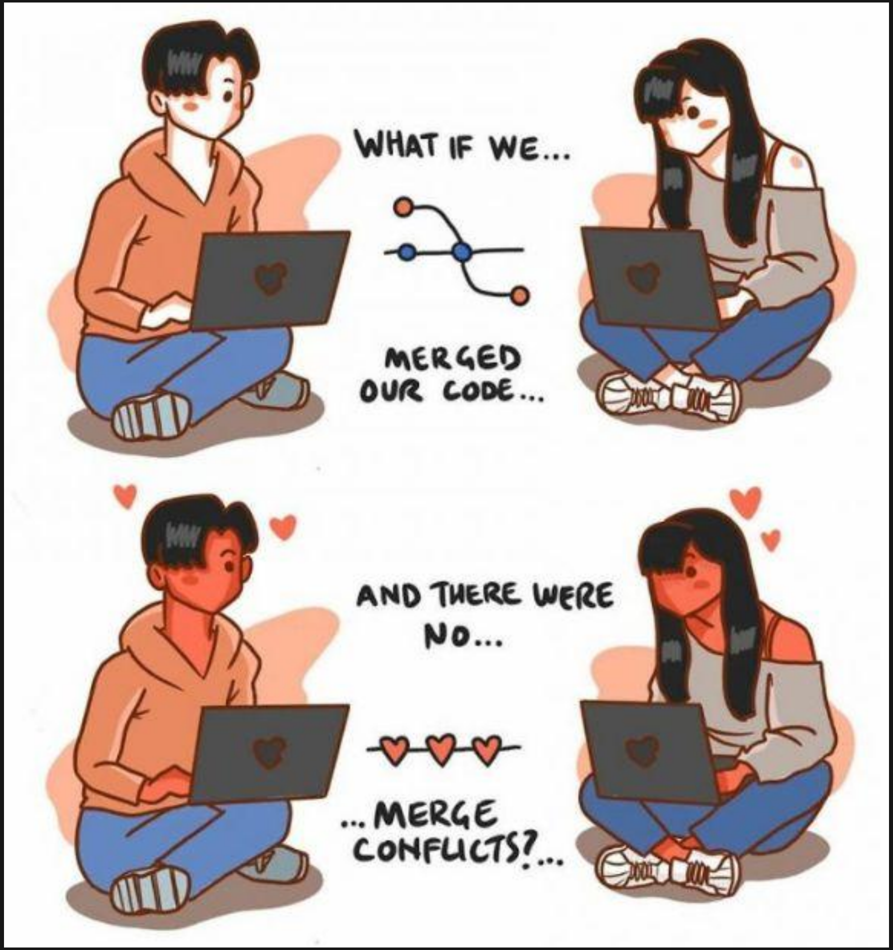
git features: branches

- “to branch” means that you diverge from the main line of development and continue to do work without messing with that main line

- you can think of branches as the train tracks in a train station: they split to handle different platforms (in git case, to encapsulate individual changes), and they will eventually converge (**merge**) into one when leaving the station

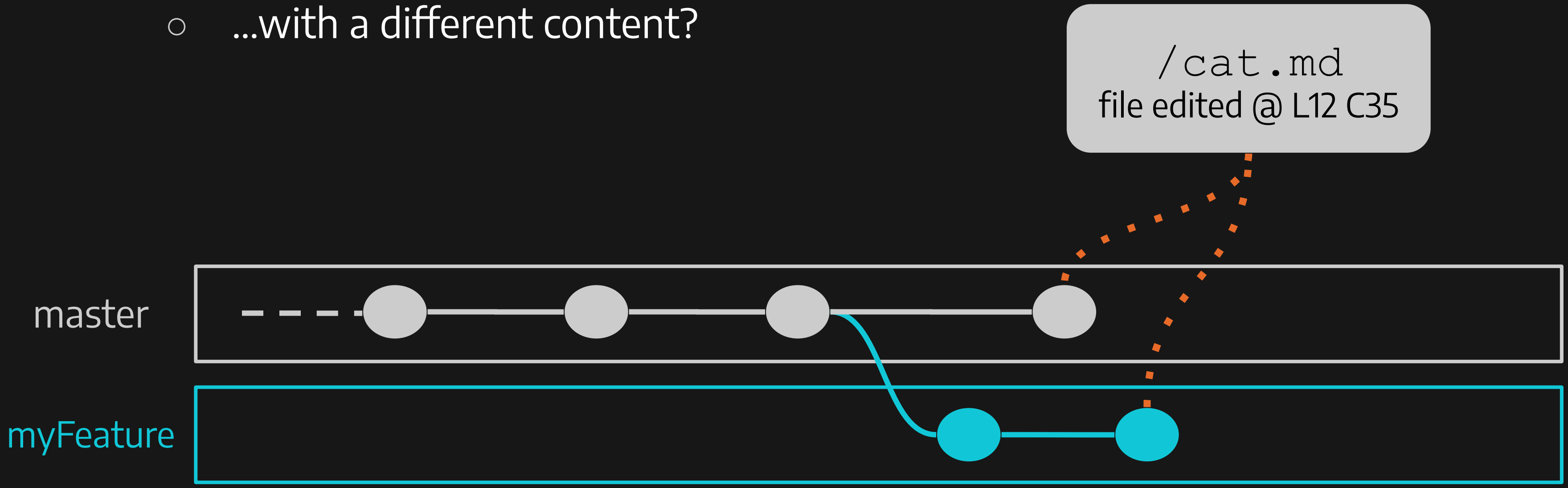
- the “main line” branch is often called “master”, “main” or “trunk”




WHAT IF WE...
MERGED OUR CODE...
AND THERE WERE NO...
...MERGE CONFLICTS?...

git features: merge

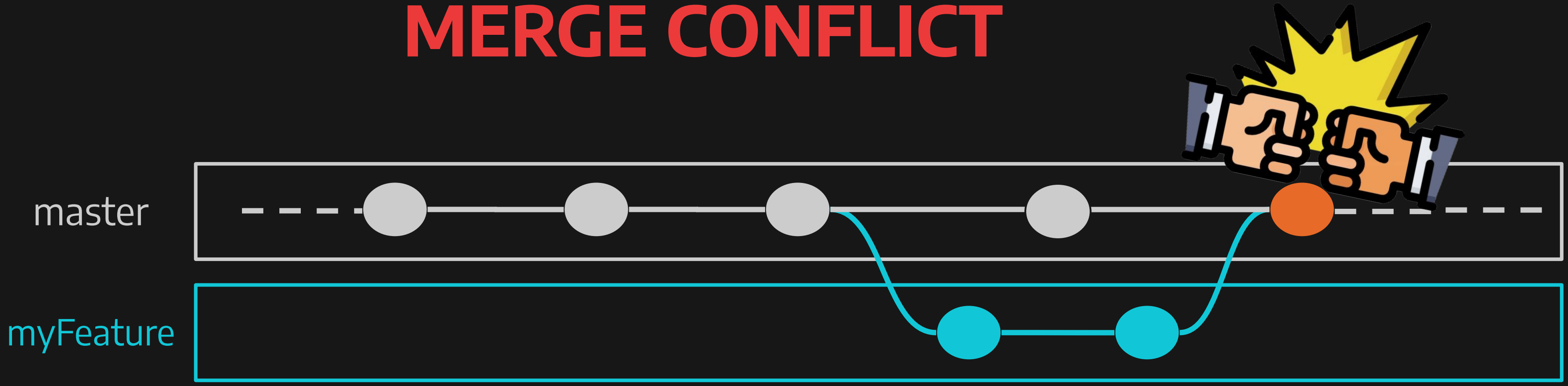
- what happens if we try to merge two branches and:
  - a file has been edited in the same line (L) and column (C)...
  - ...but in both branches...
  - ...with a different content?



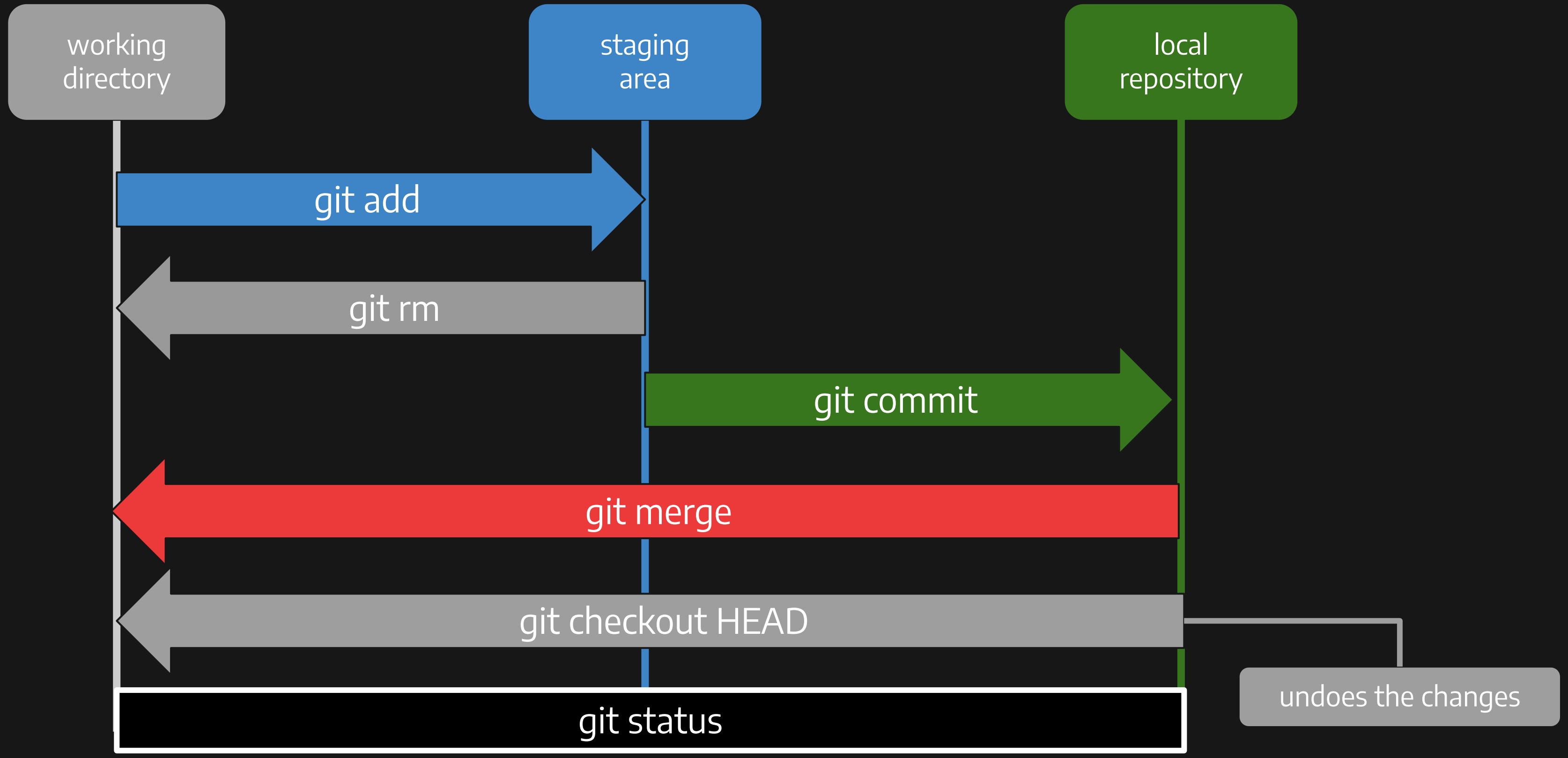
git features: merge

- what happens if we try to merge two branches and:
  - a file has been edited in the same line (L) and column (C)...
  - ...but in both branches...
  - ...with a different content?

MERGE CONFLICT



git: local operations recap

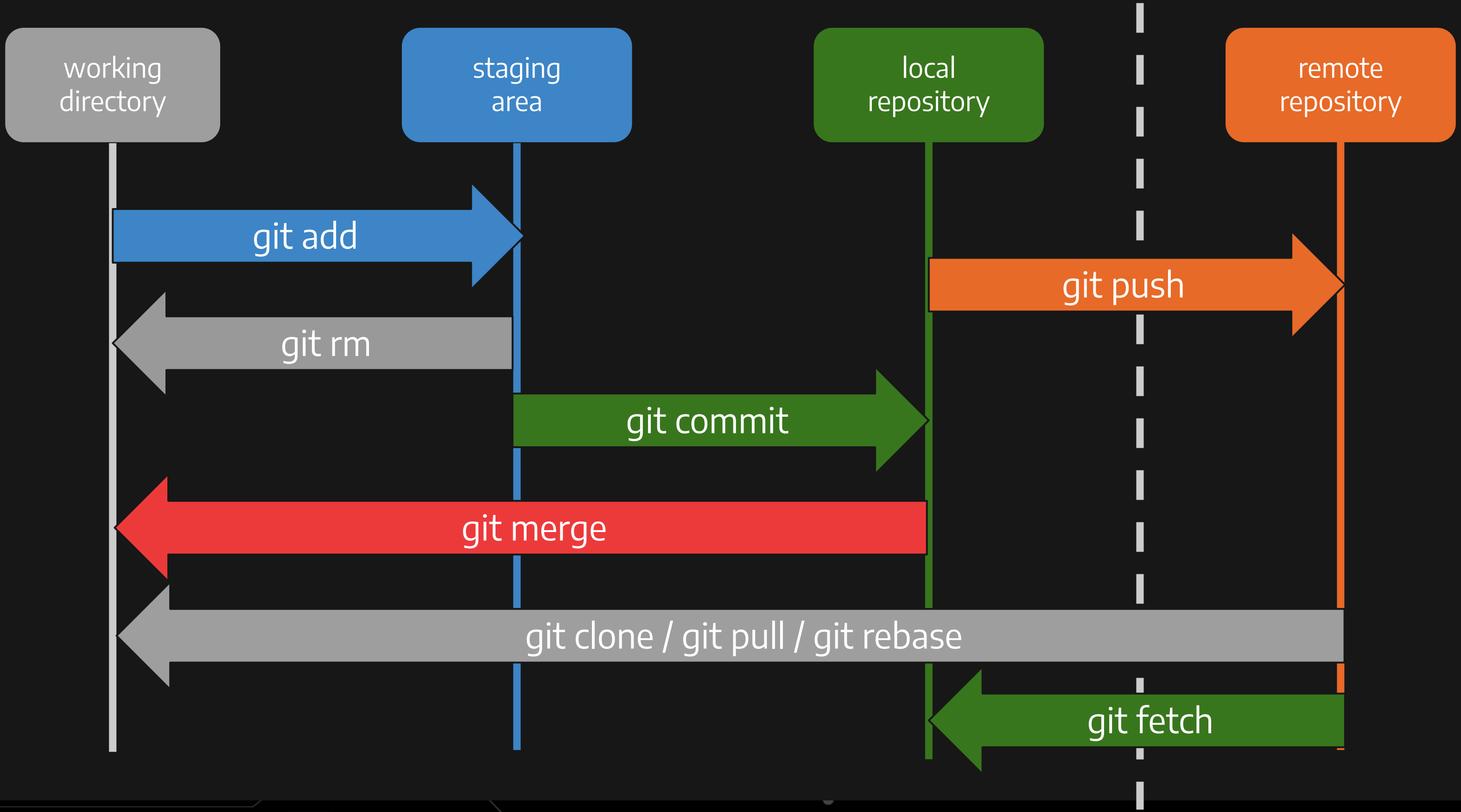


OK TEAM
LETS DO THIS!!!!

team work: git remotes

- **versions** of your project that are **hosted somewhere** on the Internet
- collaborating with others involves managing these remote repositories and **pushing and pulling data to and from them** when you need to share work
- references to remotes are held into the .git folder, and you can refer to them by using a label that points to that source (a URL)
  - a local branch linked to a remote branch is called an **upstream** branch
- you can create a local repository by **cloning** an existing one stored remotely
  - git automatically creates the “origin” remote pointing to that source

git: local and remote



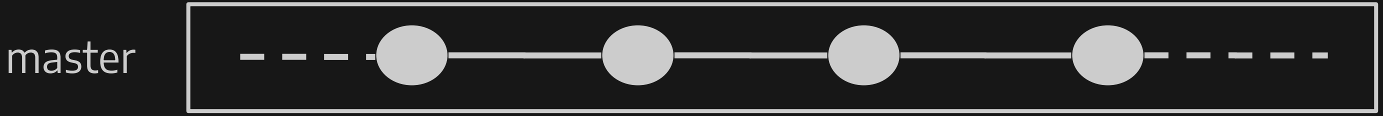
let’s test remotes

common git flows

- given git's focus on flexibility, there is no standardized process on how to interact with it
- when working with a team on a git-managed project, it’s important to **make sure the team is all in agreement** on how the flow of changes will be applied
- an agreed-upon git workflow should be developed or, even better, selected from a **vast plethora of alternatives** already available
- it is a difficult choice, let’s give a starting point…

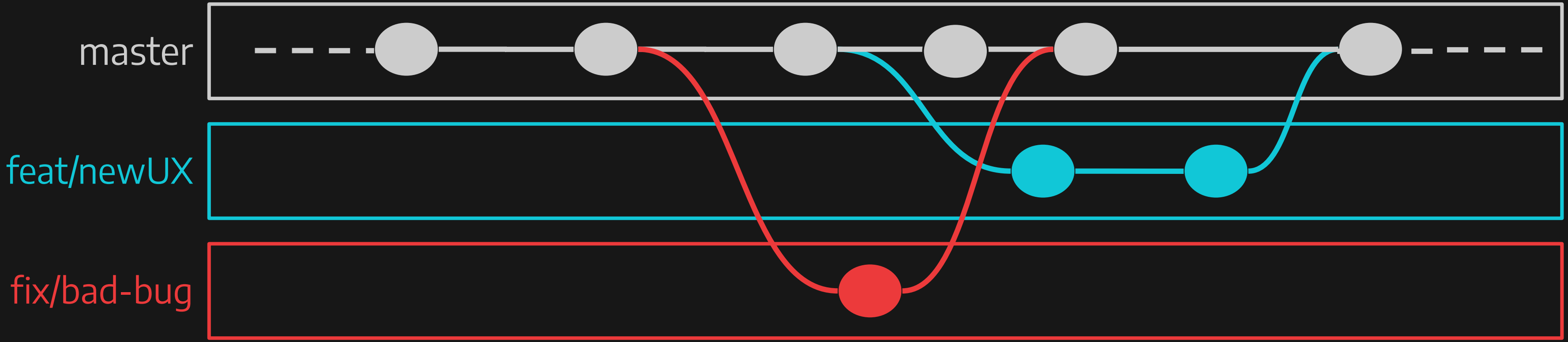
basic flow

- commits go directly on master
- very simple
- not really effective, can be adopted at most for small individual projects



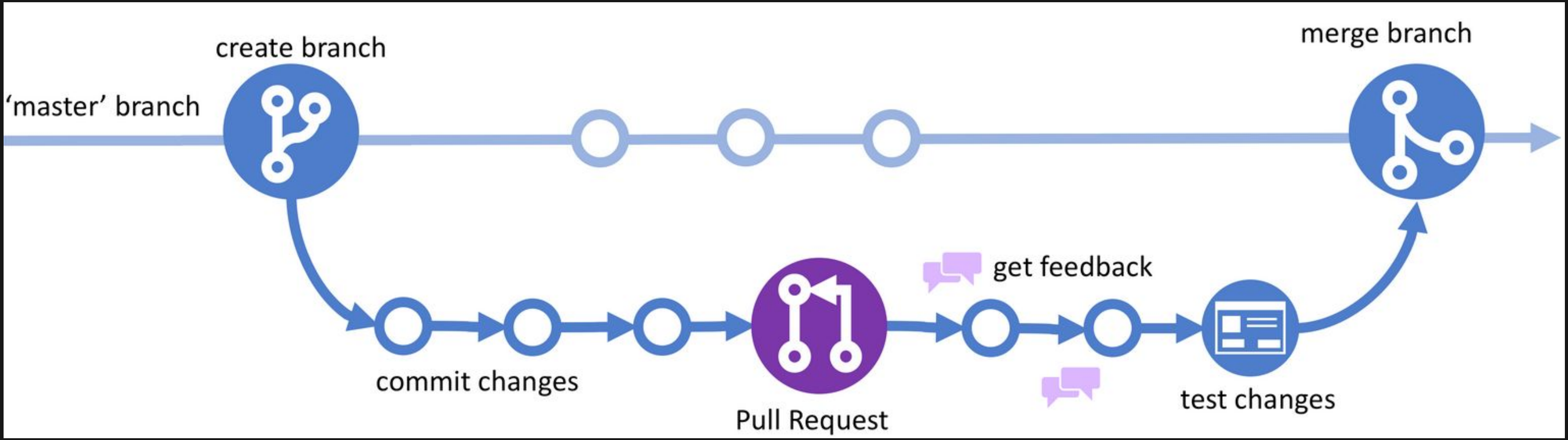
github flow

- pretty simple, branch based and useful for teams (not only developers)
- **create an isolated branch** with a descriptive name and make commits on it;
- make **small** changes* and commit them with clear messages to track progress;
- **open a PR** to propose changes → discussion and review** by team members;
- once approved, **merge the changes** into the master branch, ensuring seamless integration of new features.



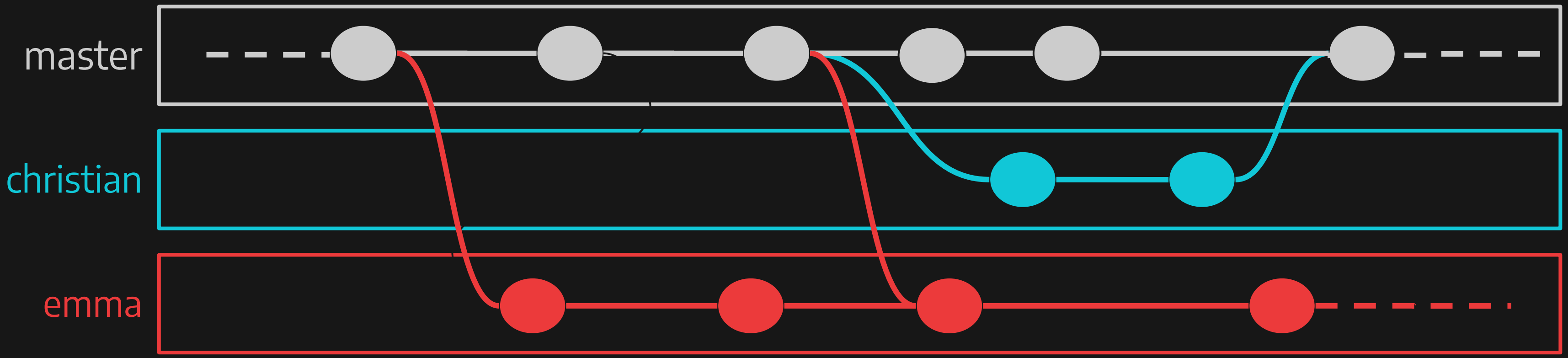
github PRs and reviews

- simple real life example
- complex real life example
- mandatory for **security** (roles) and **safety** (stability of the codebase)



centralized flow

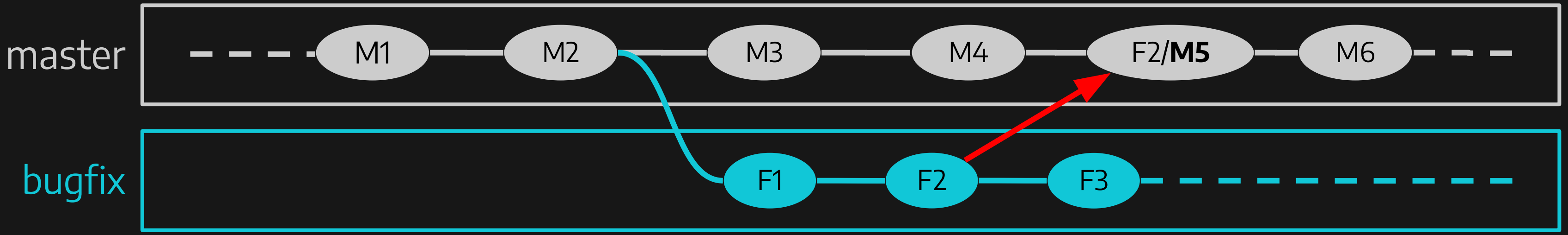
- ➢ well suited for isolated "environments";
- ➢ not recommended for software development;
- ➢ all team members clone a single central repository;
- ➢ each contributor has their branch for new work or fixes.



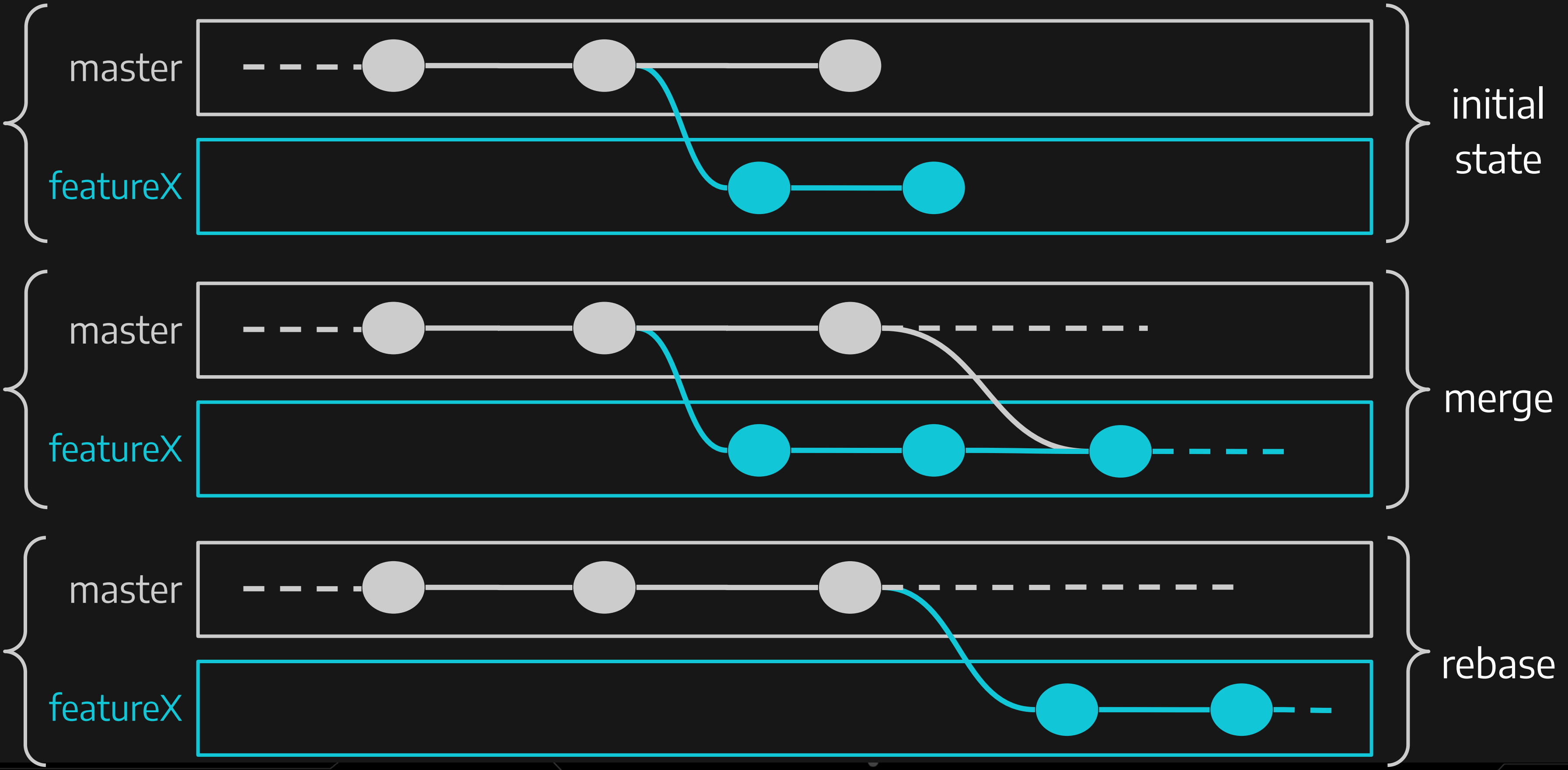
other features: cherry picking

- It is useful when you want to **selectively apply specific commits** from one branch to another, especially when those changes are relevant but merging the entire branch would include unwanted changes

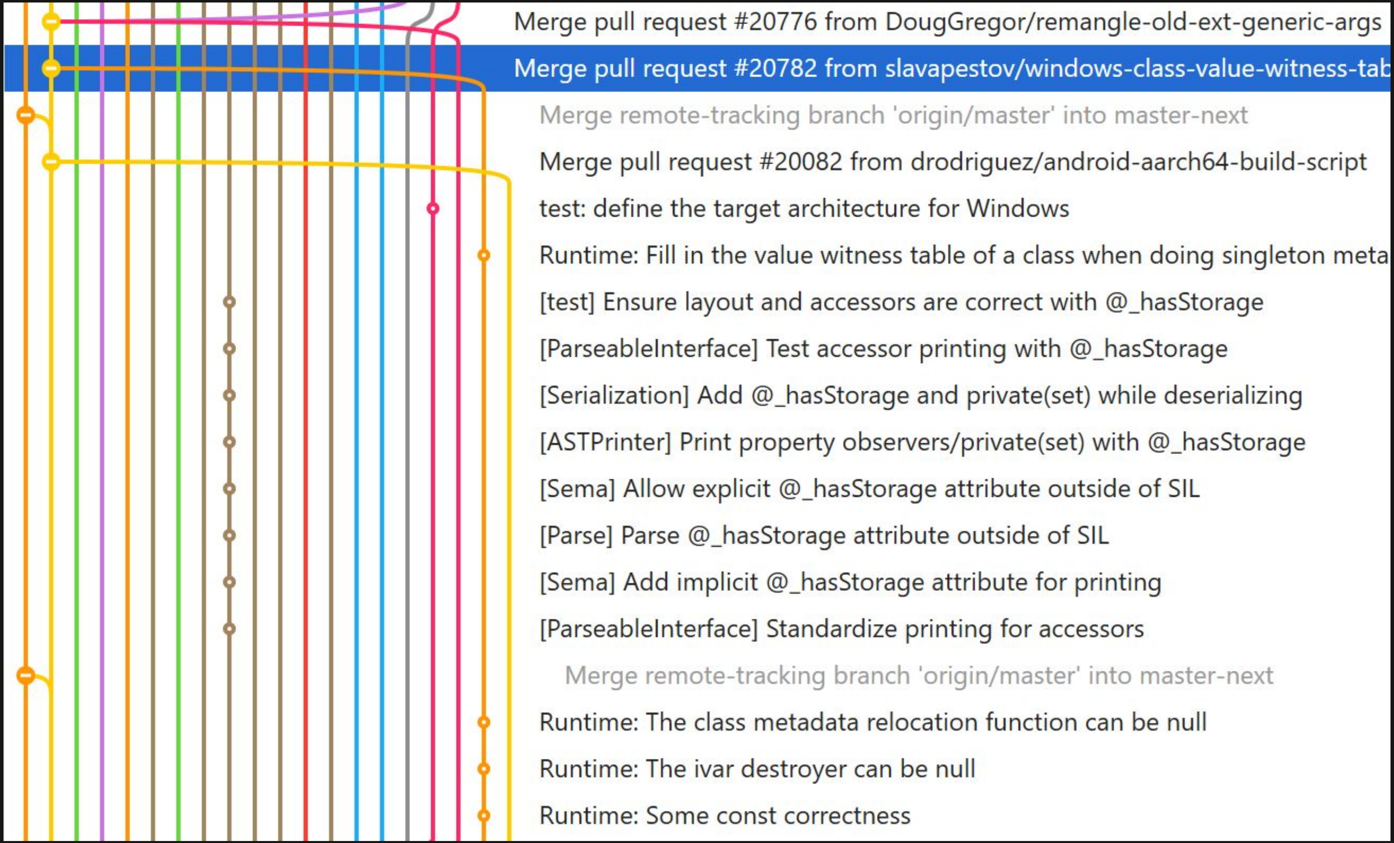
- it can lead to duplicated commits and potential conflicts, so it is best used for isolated and small changes rather than extensive updates
  - do not apply the same changes manually, ***never***



merge vs rebase



why rebasing?


Merge pull request #20776 from DougGregor/remangle-old-ext-generic-args
Merge pull request #20782 from slavapestov/windows-class-value-witness-tab
Merge remote-tracking branch 'origin/master' into master-next
Merge pull request #20082 from drodriguez/android-aarch64-build-script
test: define the target architecture for Windows
Runtime: Fill in the value witness table of a class when doing singleton meta
[test] Ensure layout and accessors are correct with @_hasStorage
[ParseableInterface] Test accessor printing with @_hasStorage
[Serialization] Add @_hasStorage and private(set) while deserializing
[ASTPrinter] Print property observers/private(set) with @_hasStorage
[Sema] Allow explicit @_hasStorage attribute outside of SIL
[Parse] Parse @_hasStorage attribute outside of SIL
[Sema] Add implicit @_hasStorage attribute for printing
[ParseableInterface] Standardize printing for accessors
Merge remote-tracking branch 'origin/master' into master-next
Runtime: The class metadata relocation function can be null
Runtime: The ivar destroyer can be null
Runtime: Some const correctness

LICENSE ≠ PATENT

Contracts that transfer intellectual property rights from the owner of the rights to a third party who wants to use them (**right to not be excluded**)

can also be applied to

20-year exclusive property right that entitles the inventor **to exclude others** from making, using, or selling the invention

Typically(*) defined by non-profits: **grants permission to use the invention**

Developed and enforced by a government: **gives monopoly over the invention**

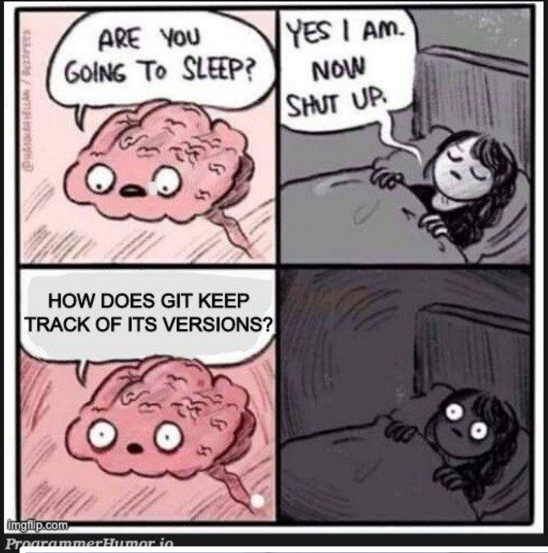
licenses and models

- **Creative Commons licenses**: for all kinds of creative works like websites, scholarship, music, film, photography, literature, courseware, etc.
- **GNU General Public License (GPL)**: primarily designed for software; conditions of licensing are standard and cannot be changed.
- **Open Source Initiative** (OSI): they run review processes to ensure that licenses comply to community standards; the goal is to minimize license duplication and proliferation.
- **Blue Oak Council Licenses**: publishes this list to identify permissive licenses, so that everyone can recognize, rely on, and use them without expensive legal help.

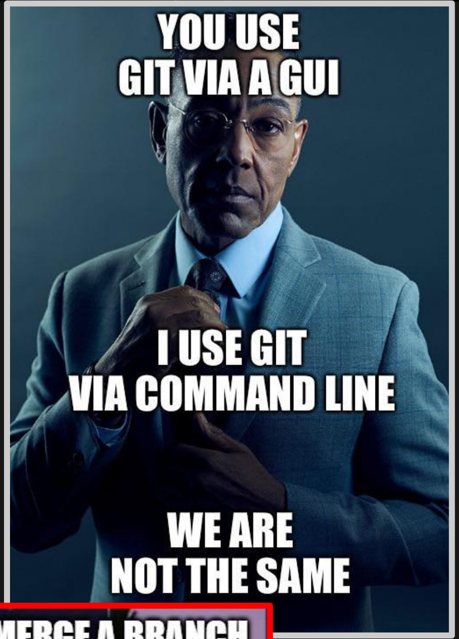
... and many more

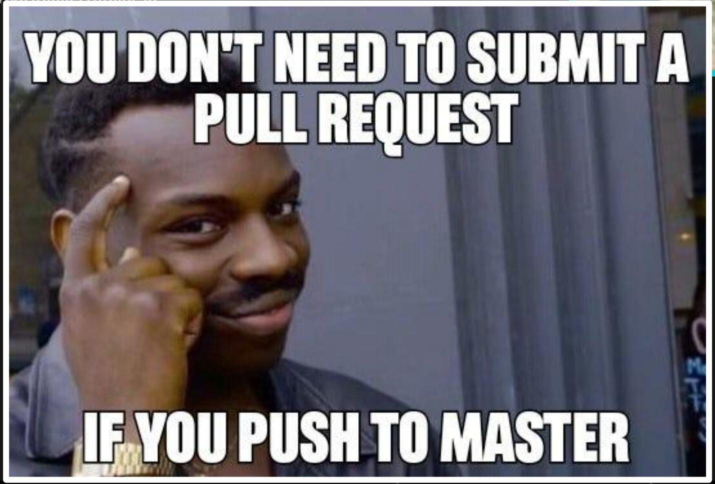
example: creative commons

License	Copy and publish	Attribution required	Commercial use	Modify & adapt	Change license
PUBLIC DOMAIN	👍	⛔	👍	👍	👍
CC BY	👍	👍	👍	👍	👍
CC BY SA	👍	👍	👍	👍	⛔
CC BY NC	👍	👍	⛔	👍	👍
CC BY NC SA	👍	👍	⛔	👍	⛔
CC BY ND	👍	👍	👍	⛔	⛔
CC BY NC ND	👍	👍	⛔	⛔	⛔


ARE YOU GOING TO SLEEP?
YES I AM. NOW SHUT UP.
HOW DOES GIT KEEP TRACK OF ITS VERSIONS?
imgflip.com
ProgrammerHumor.io

me
random GIT commands from internet
my branch, already messed up


YOU USE GIT VIA A GUI
I USE GIT VIA COMMAND LINE
WE ARE NOT THE SAME


YOU DON'T NEED TO SUBMIT A PULL REQUEST
IF YOU PUSH TO MASTER

WHEN THE TEAM LEAD IS ABOUT TO MERGE A BRANCH
AFTER MONTHS OF ISOLATED WORK
imgflip.com

let’s do it

https://tinyurl.com/20240430-ibk

references

- ★ Open Science Training Handbook
- ★ WhyOpenResearch
- ★ OpenScienceMooc: open principles
- ★ EarthLab: what is open reproducible science
- ★ SciHub about page
- ★ Recommendations for the Transition to Open Access in Austria
- ★ EGU 2018 SC1.13
- ★ The OpenSource way
- ★ Why open source misses the point of free software
- ★ UpCounsel: Copyright vs. Trademark vs. Patent vs. License
- ★ Hash functions
- ★ Git working tree, staging area and local repository deep dive
- ★ Git branches in a nutshell
- ★ Licenses: CreativeCommons, GNU, OSI, BlueOak, license chooser tool, TLDRLegal licenses in plain english
- ★ Images and icons: stolen somewhere over the internet (I'll put the refs here soon...)
- ★ How to waste time: "master vs main" controversy in git

download this presentation

https://github.com/LeoSpyke/slides