Version control with git

or how to write code without going crazy

Agenda

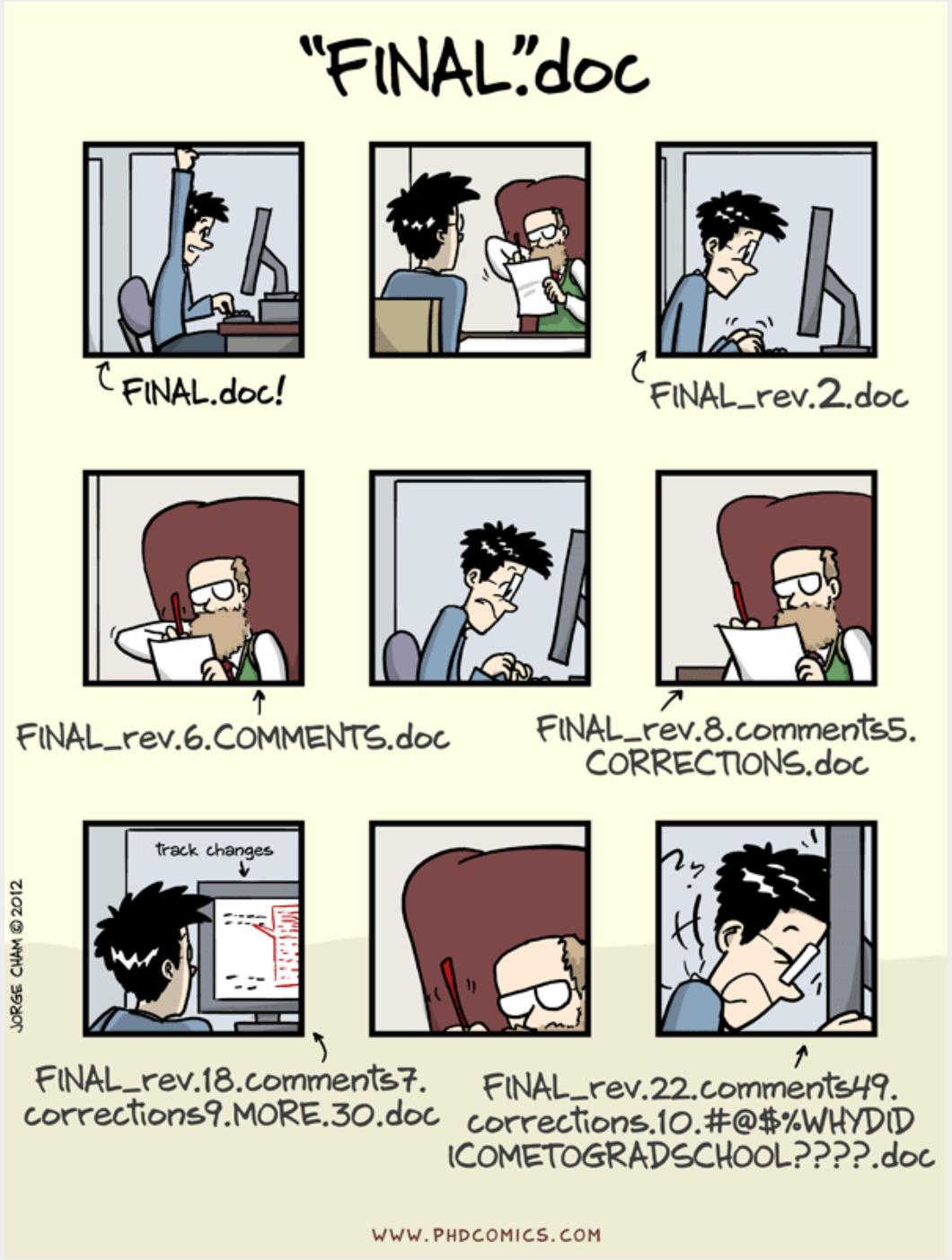
- Breaks?
- Who am I?
- Who are you?
- Short history of version control
- Working with git
  - Basic commands
  - Github
  - Lots of practice!

Who am I?

- Programming since young age
- Medical Biotechnology@TUB
- 10 years NGS and Bioinformatics Core Facility at the Robert Koch Institute
- Short stop at ministry of finance
- Since 10/2019: Professor@HTW: Applied Computer Science, Bioinformatics

Who are you?

- More in-depth in NGS course
- Now short intro:
  - Name
  - Background
    - Education
    - Previous work
    - Data/code management
  - Expectations



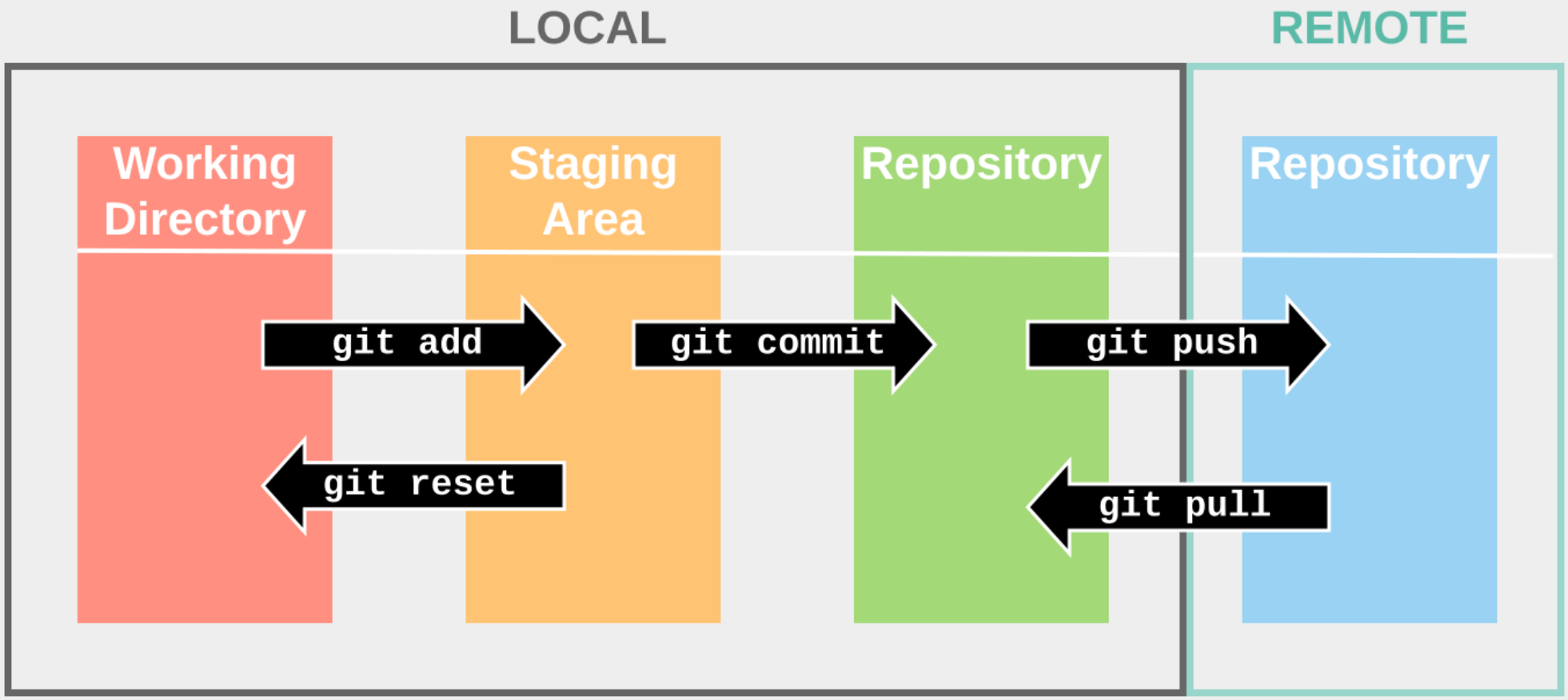
Version control

- Aim: Not losing control!
  - 1972: SCCS, single user
- Bonus: Coordination of work
  - 1986: CVS (central)
  - 2005: git (distributed)
  - In the meantime: SVN, TFVC, BitKeeper, Mercurial etc.
- For all text based content (code, LaTeX, Markdown etc.)

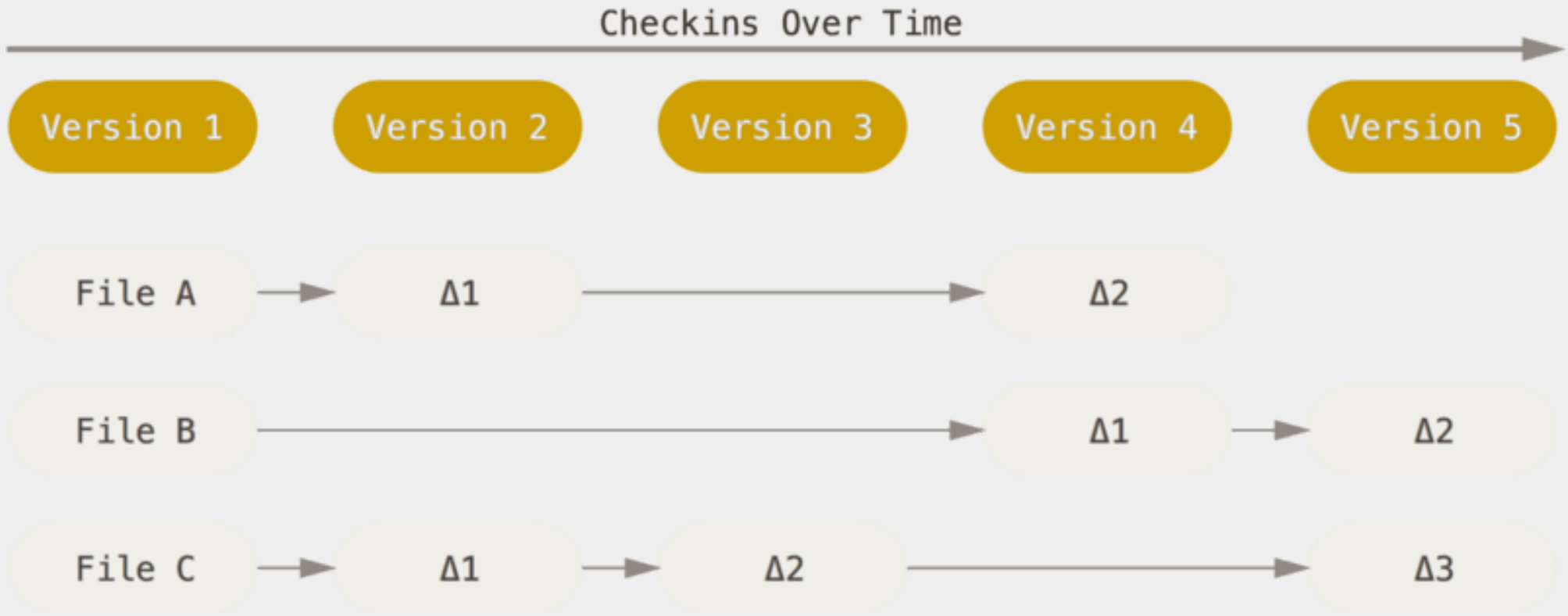
Our story

- Course diary
  - General notes
  - Cheatsheets
  - Link collection
  - Code for the exercises
- No chaos!
- Requirements may change over time

Basic idea of git



Files over time in the repository

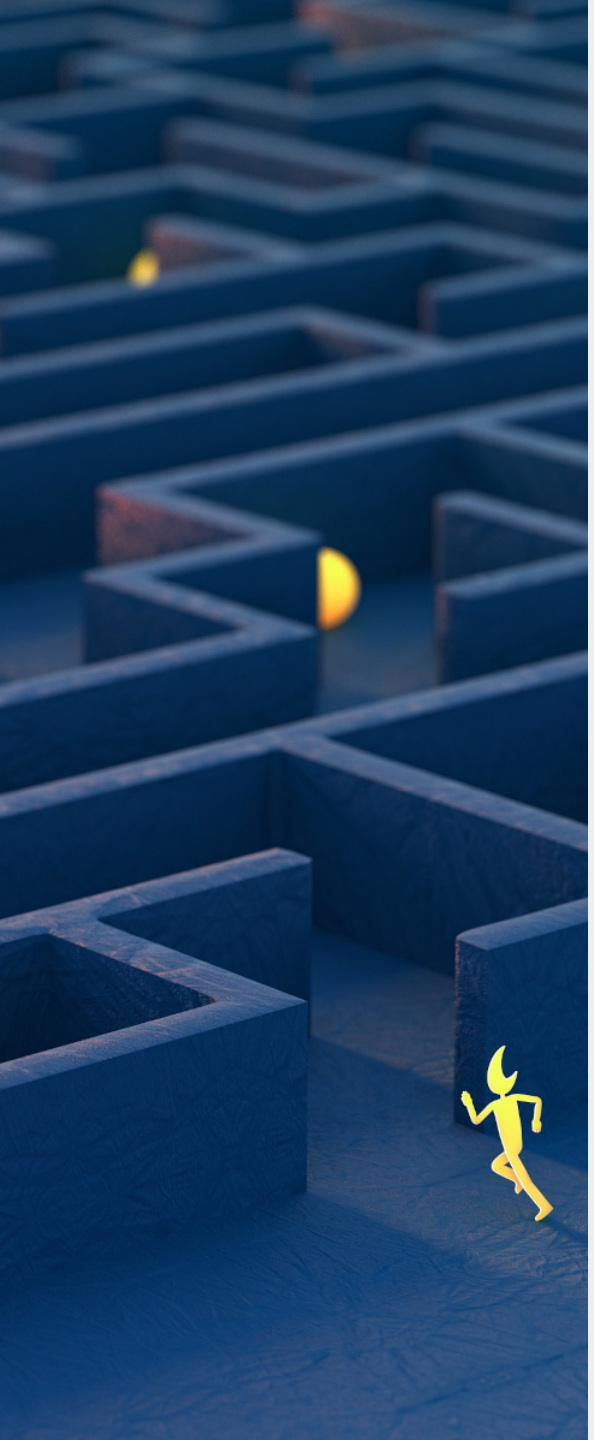


Providers

- github
  - Many extensions (Zenodo integration, CITATION.cff)
  - Limited number of private repositories
  - Owned by Microsoft
- gitlab
  - Unlimited number of private repositories
  - Can be self-hosted
  - Not owned by Microsoft

Repositories

- Public vs. private repositories
  - When should it be public vs. private?
  - Wann should a private repo be made public?
- Together:
  - Creating a repository
  - Permission management
- Short discussion: PW reuse, PW manager

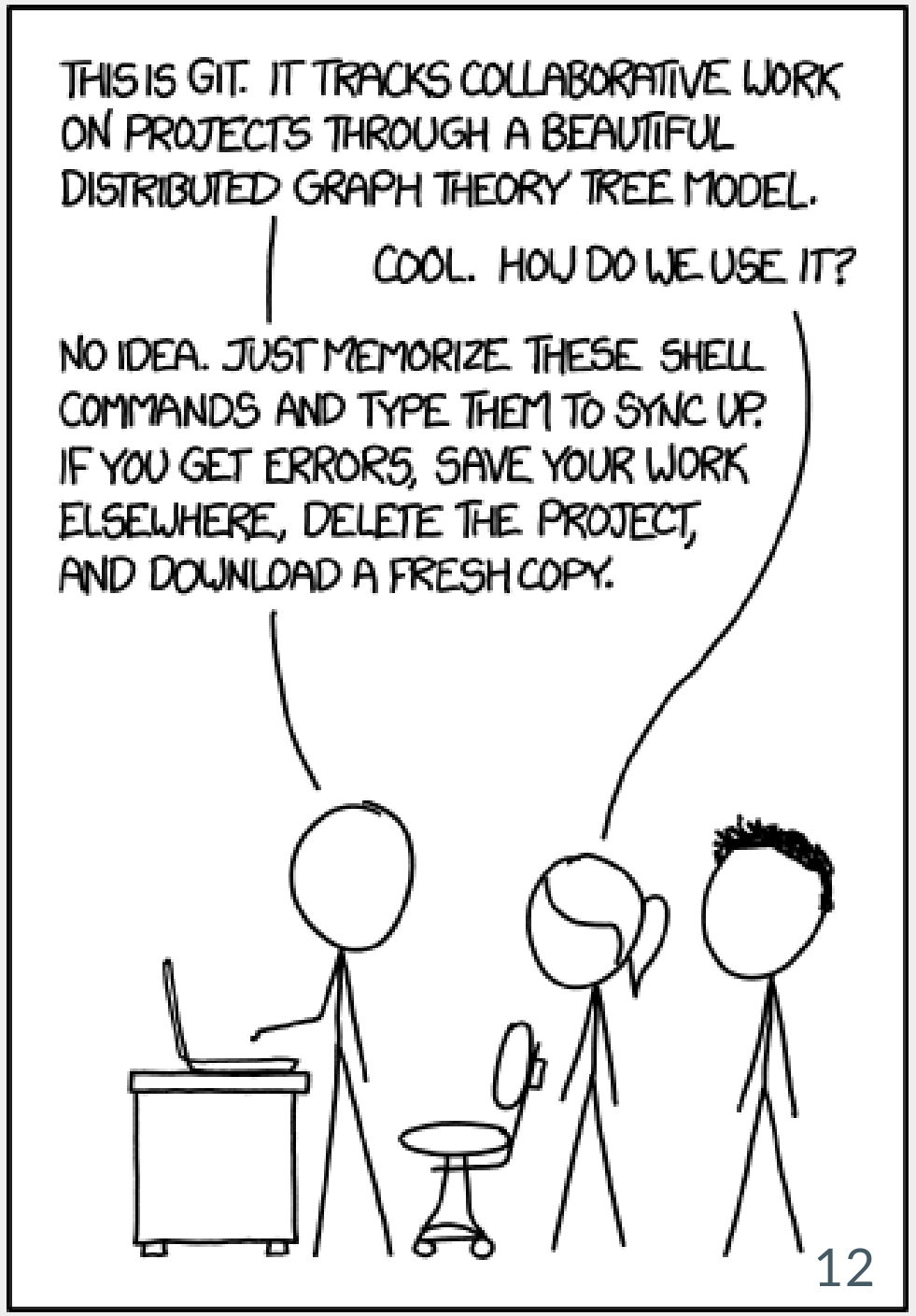


Exercise: Creating a repo

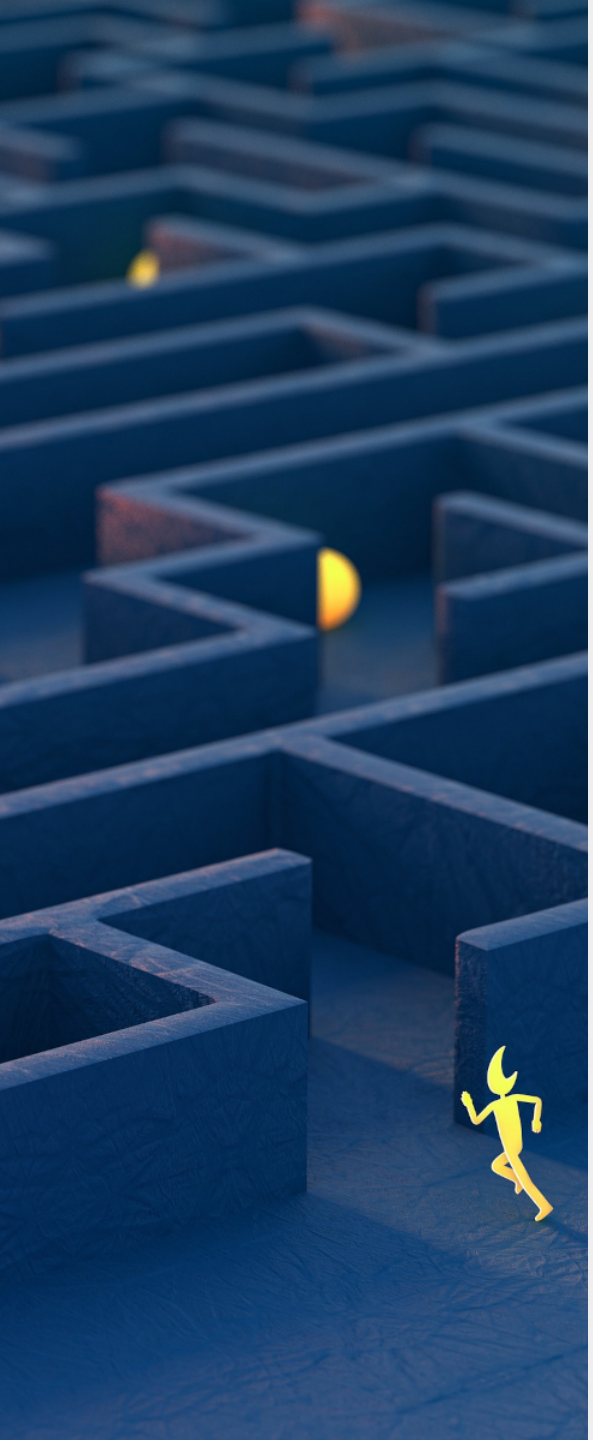
- Create account at github (since we need CITATION.cff and zenodo)
- Create a public (to use insights) repository
- Invite me

Important commands

- Overview: NeSI reference sheet
- First `git clone https://repo.git`
- Typical workflow afterwards:
  - `git add file1 file2 ...`
  - `git status`
  - `git commit -m "Habe fertig!"`
  - `git push`
- In case of fire: Oh Shit, Git!?!



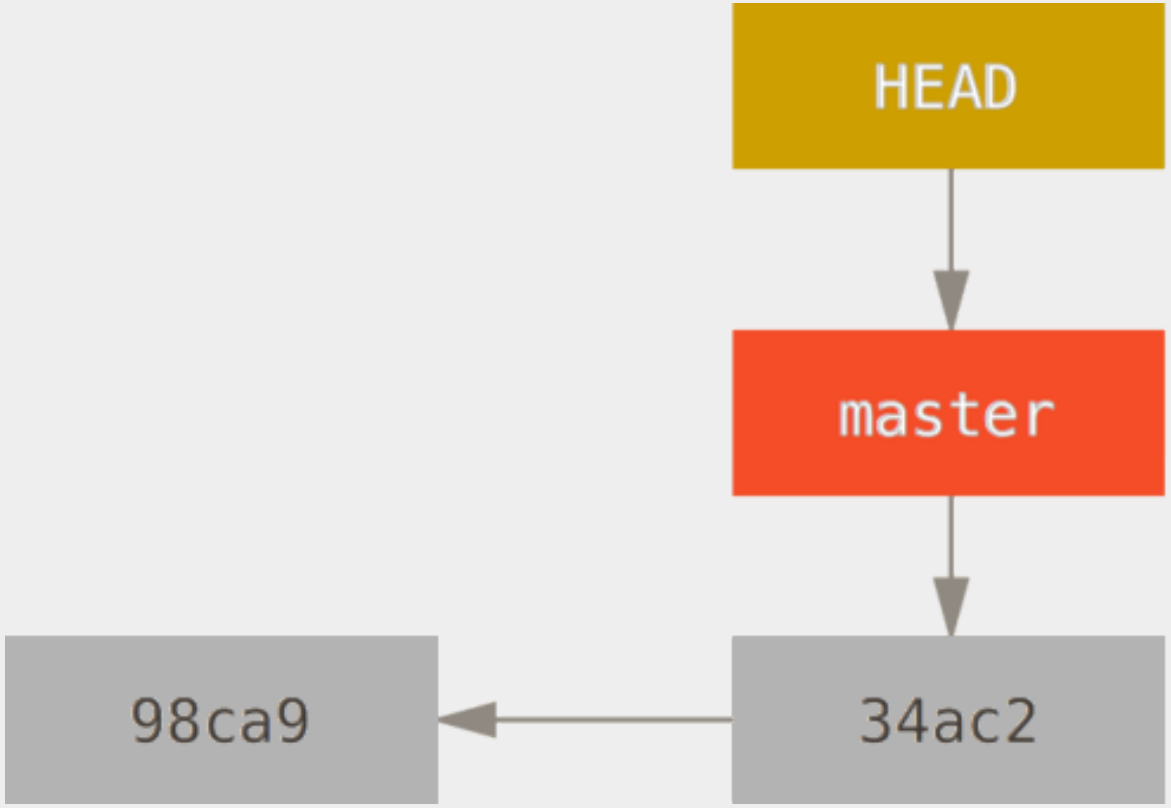
Fresh repository ("Network graph")



Exercise: Commit + push

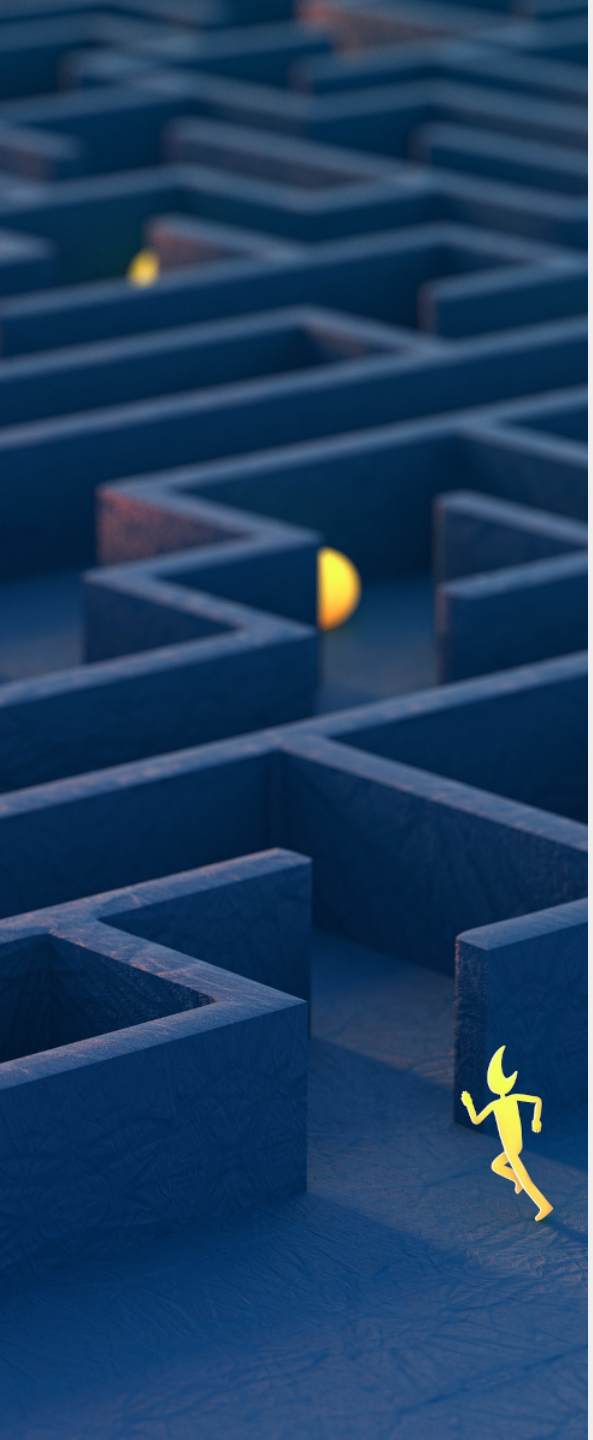
- Create a new text file: git cheat sheet
- Write down the 4 basic commands (slide 11), 2 lines per command: `command`, `short description`
- add, status, commit
- Look at repo & graph online: What happened?
- push
- Look at repo & graph online: Difference?

Repository after one commit



Markdown, marp

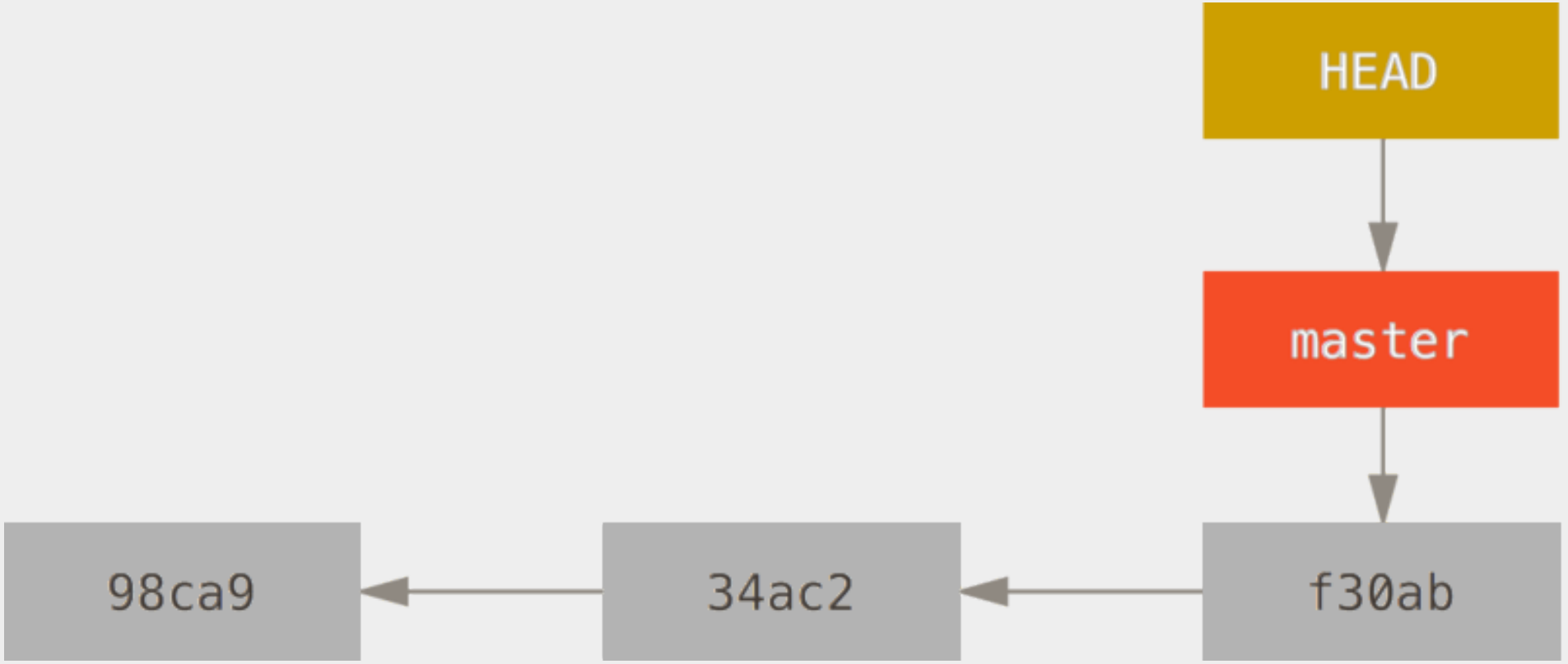
- Plaintext for cheatsheet etc. is not pretty
- Markdown: populat text-based format with many features
  - Headings, bold, italic...
  - Lists, tables, images
  - Syntax highlighting for many languages, blockquotes
- Useful for notes in Jupyter Notebooks
- Extension for slides: marp (e.g. these slides)
- Together: dillinger.io, Joplin, VS code, slides



Exercise: Second commit

- Move git cheat sheet to markdown:
  - Rename to .md (with `git mv`!)
  - Title: "Basic git commands"
  - Subtitle for each command, comment underneath
  - Remove ("forget") line for `git status`
  - Add `git mv`
- add, status, commit, push

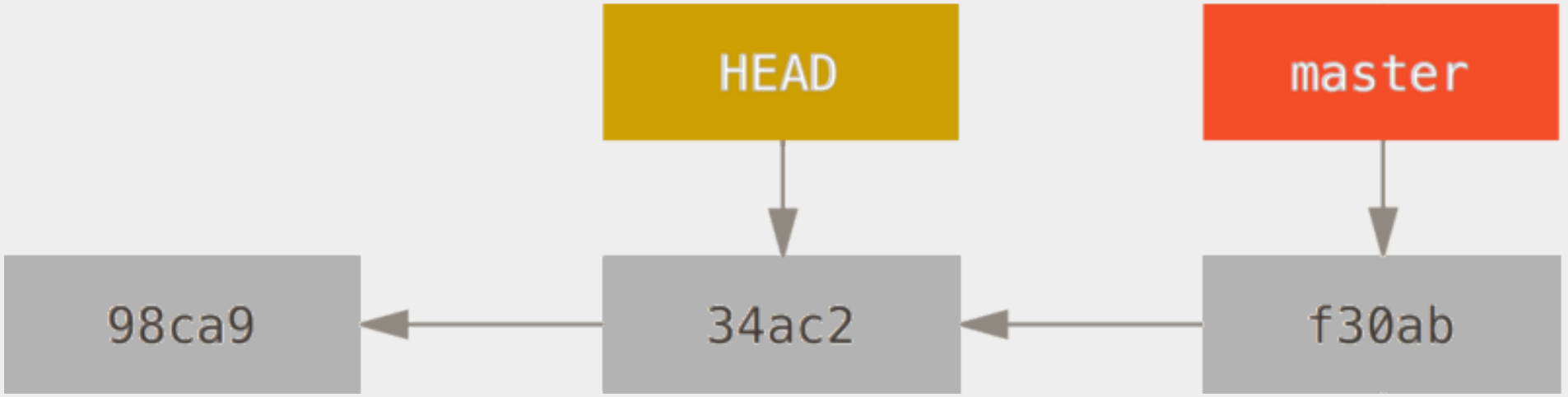
Repository after two commits

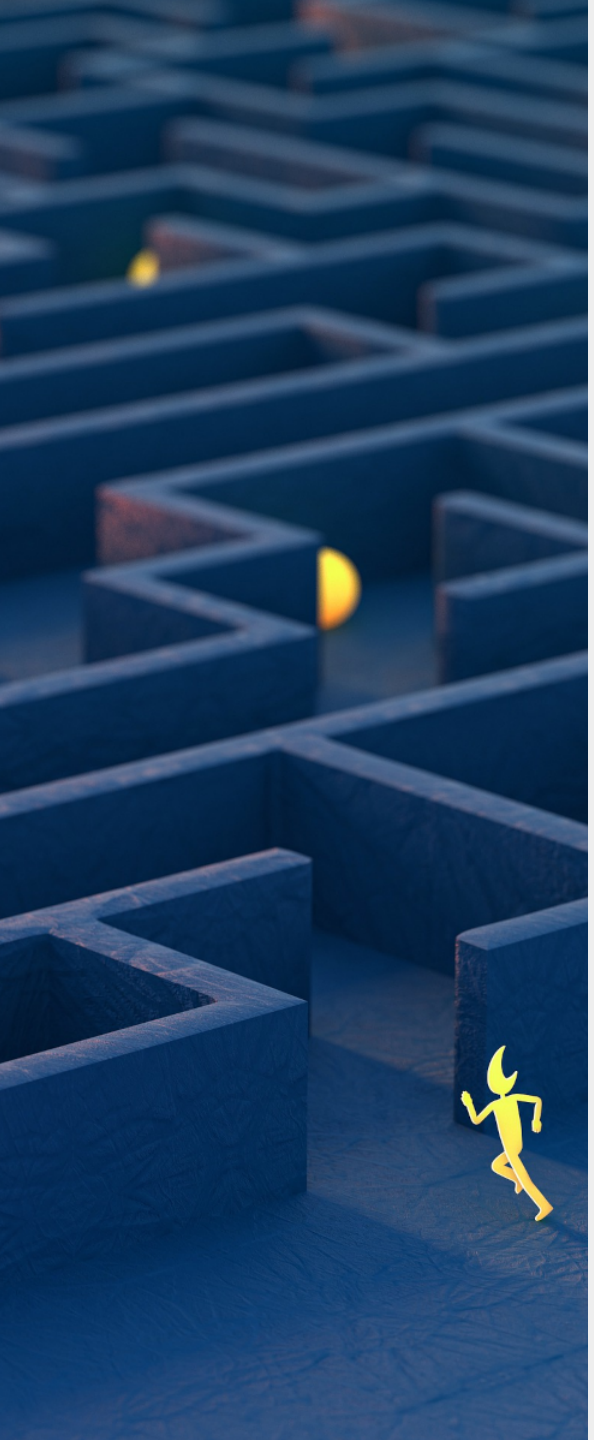


Webinterface and history

- History view online:
  - List of commits
  - Diff between two versions of a file
- Recover previous versions:
  - `git log`
  - Optionally `git diff --name-only <hash>`
  - `git diff <hash> -- <filename>`
  - `git checkout <hash>`

Detached head?





Exercise: Handling mistakes

- Checkout commit with text file
- Copy the text file somewhere else
- Checkout master
- Add `git status` line from text file to markdown file
- add, commit, push

Pandoc

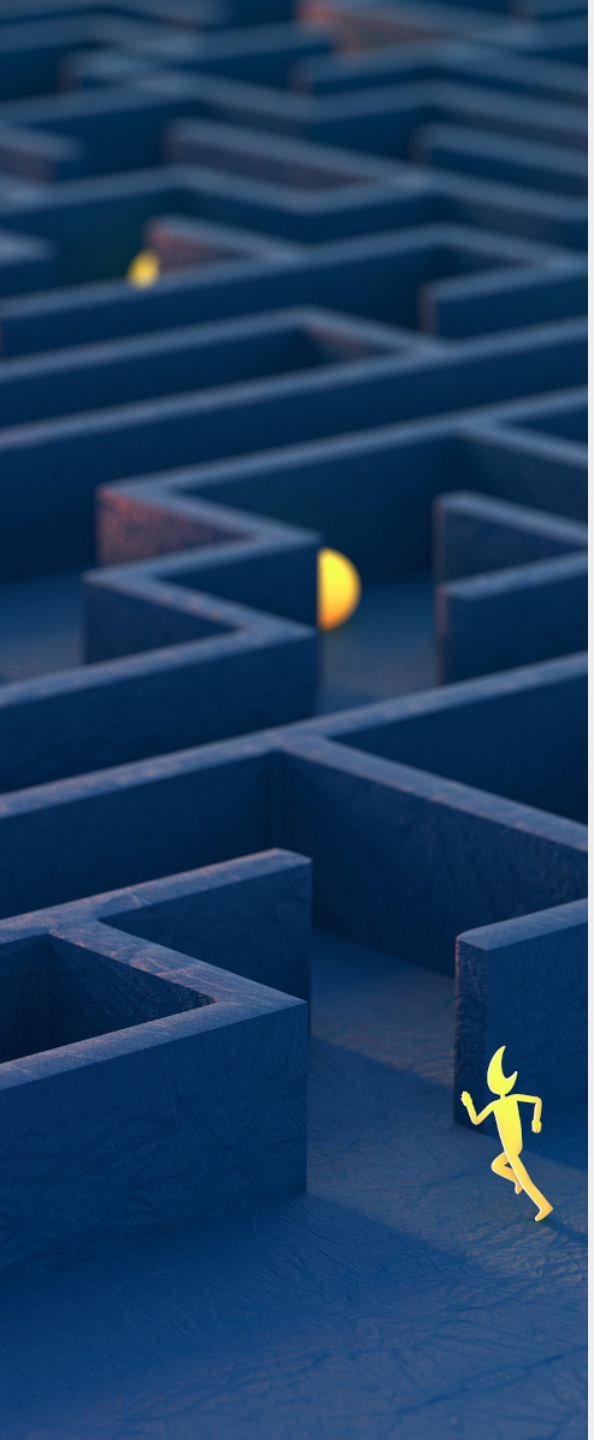
- One advantage of markdown: Conversion to other formats
- Typically: `pandoc -o file.pdf file.md`
- Formattig hints can be put in header of markdown file:

```
---
geometry: margin=25mm
colorlinks: true
---
```

Keeping the repository clean

- `git status`: Does the pdf belong in the repository?
- Files to ignore can be listed in `.gitignore`
- Documentation in git-scm
- Generally: Patterns with `*`, comments with `#`, negation with `!`

```
# Ignore all temporary editor files
.*
# Ignore all files in the tmp subdirectory...
/tmp/*
# ...except temp.config
!/tmp/temp.config
```



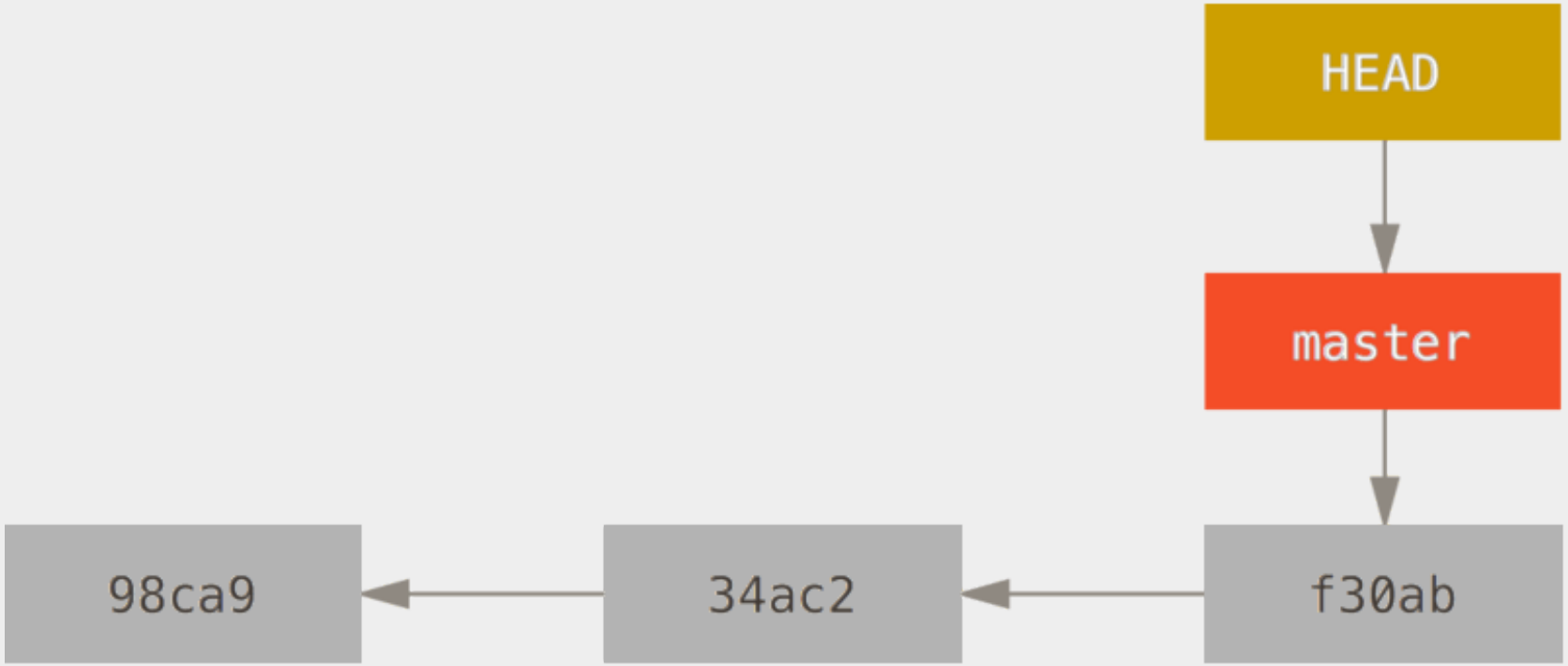
Exercise: .gitignore

- build the pdf
- Create `.gitignore`
- `git status` - what files are listed?
- Add pattern for pdf files to `.gitignore`
- `git status`: Compare to before
- add, commit, push

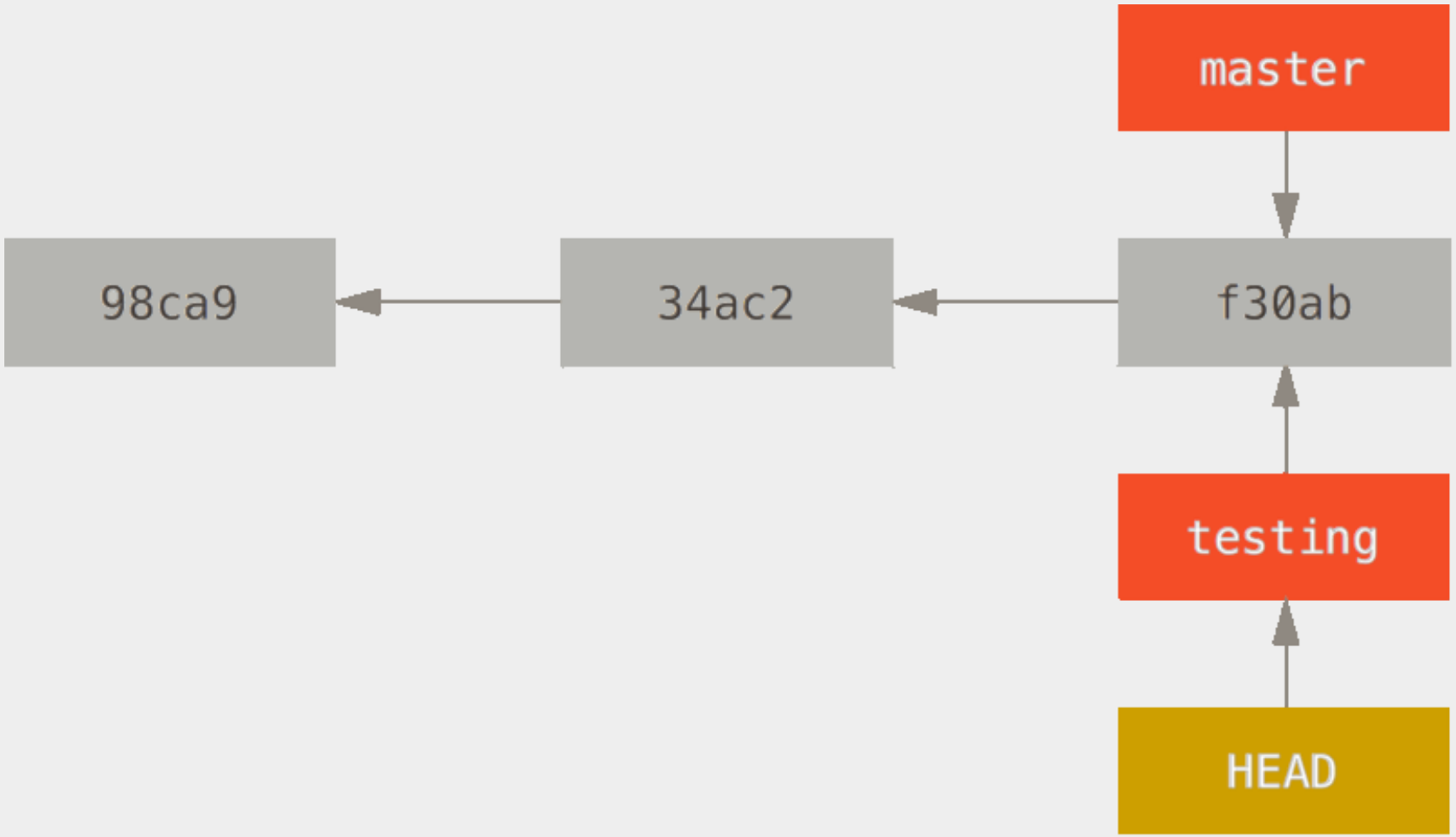
Branching

- Problem: Broken intermediary states
- Solution: Feature branches (nice tutorial at git-scm)
- Create branch: `git checkout -b branchname` (if it doesn't exist)
- Commit the branch: `git commit -m "message"`
- Push the branch: `git push -u origin branchname`
- Merge two branches (here `branchname` with `master`):
  - `git checkout master`
  - `git merge branchname` (In case of conflict: Edit, commit)

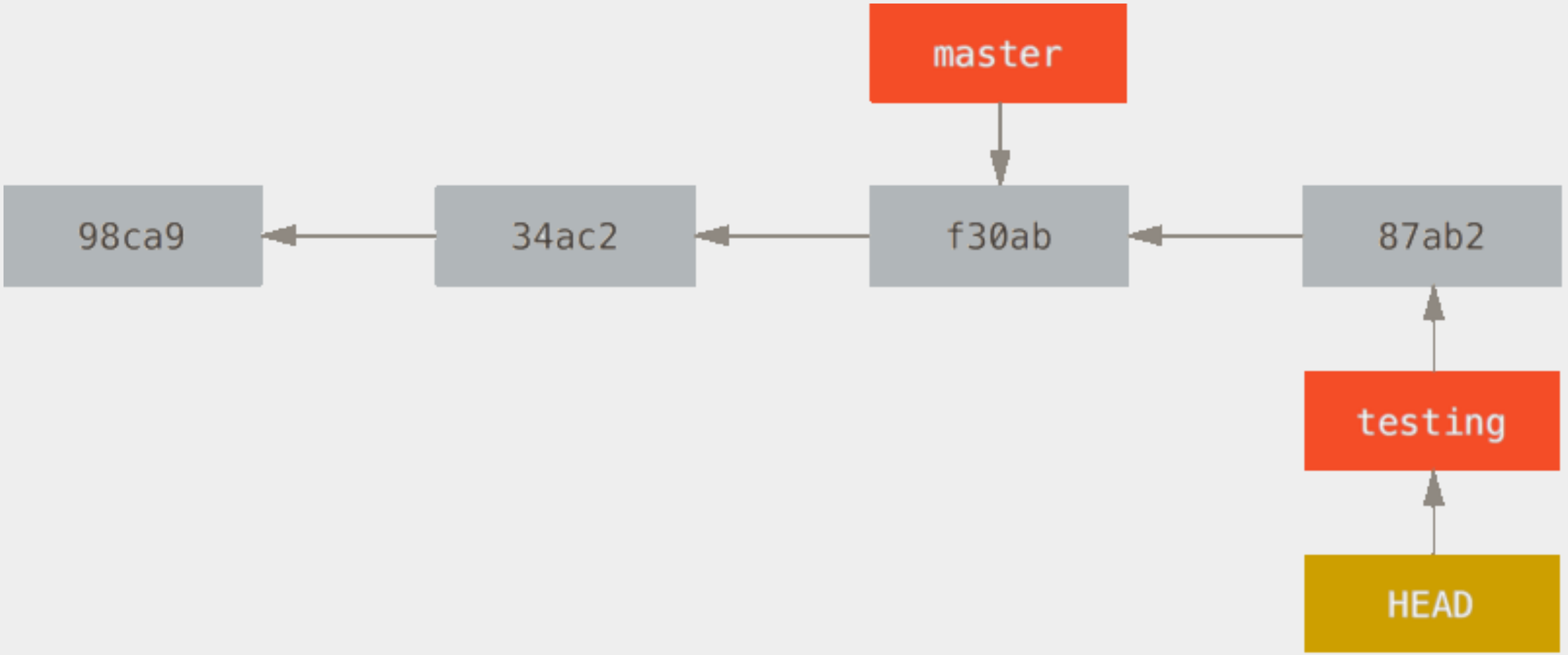
Repository before branching



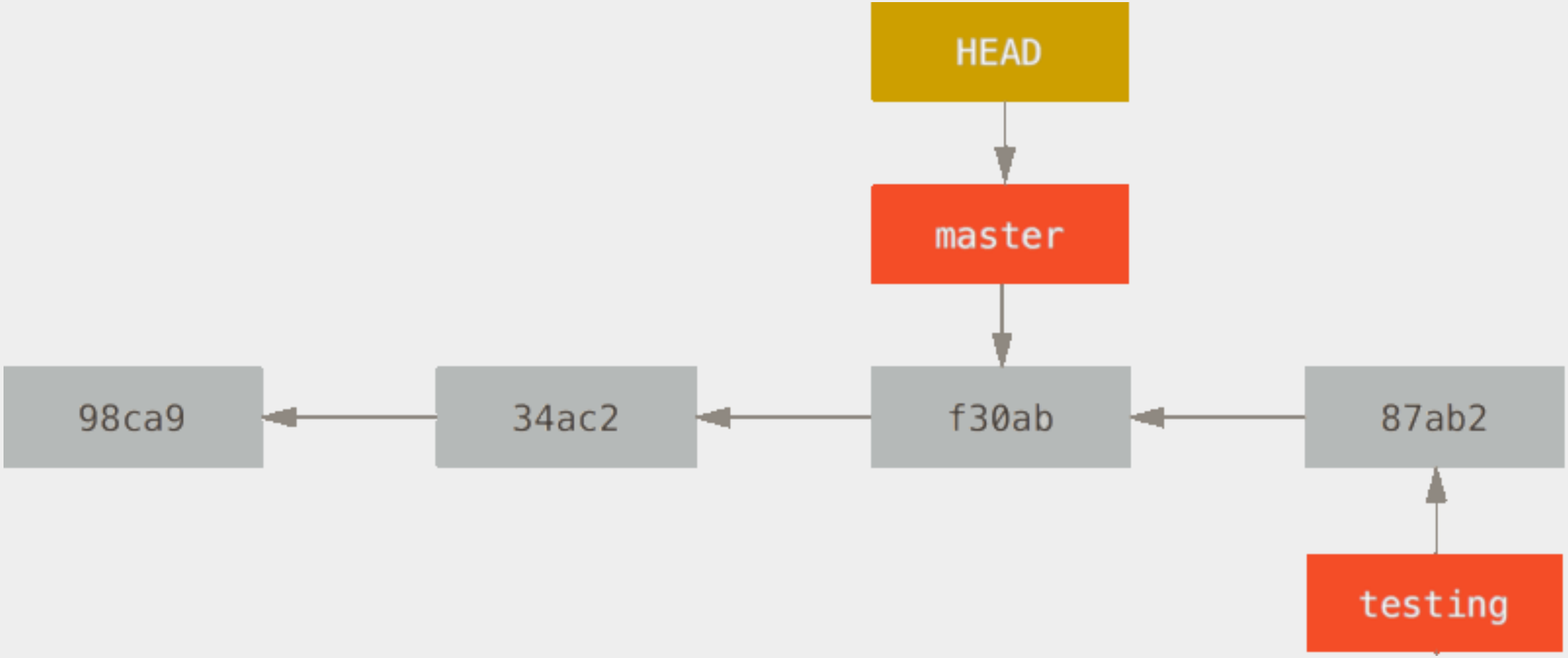
After `git checkout -b testing`



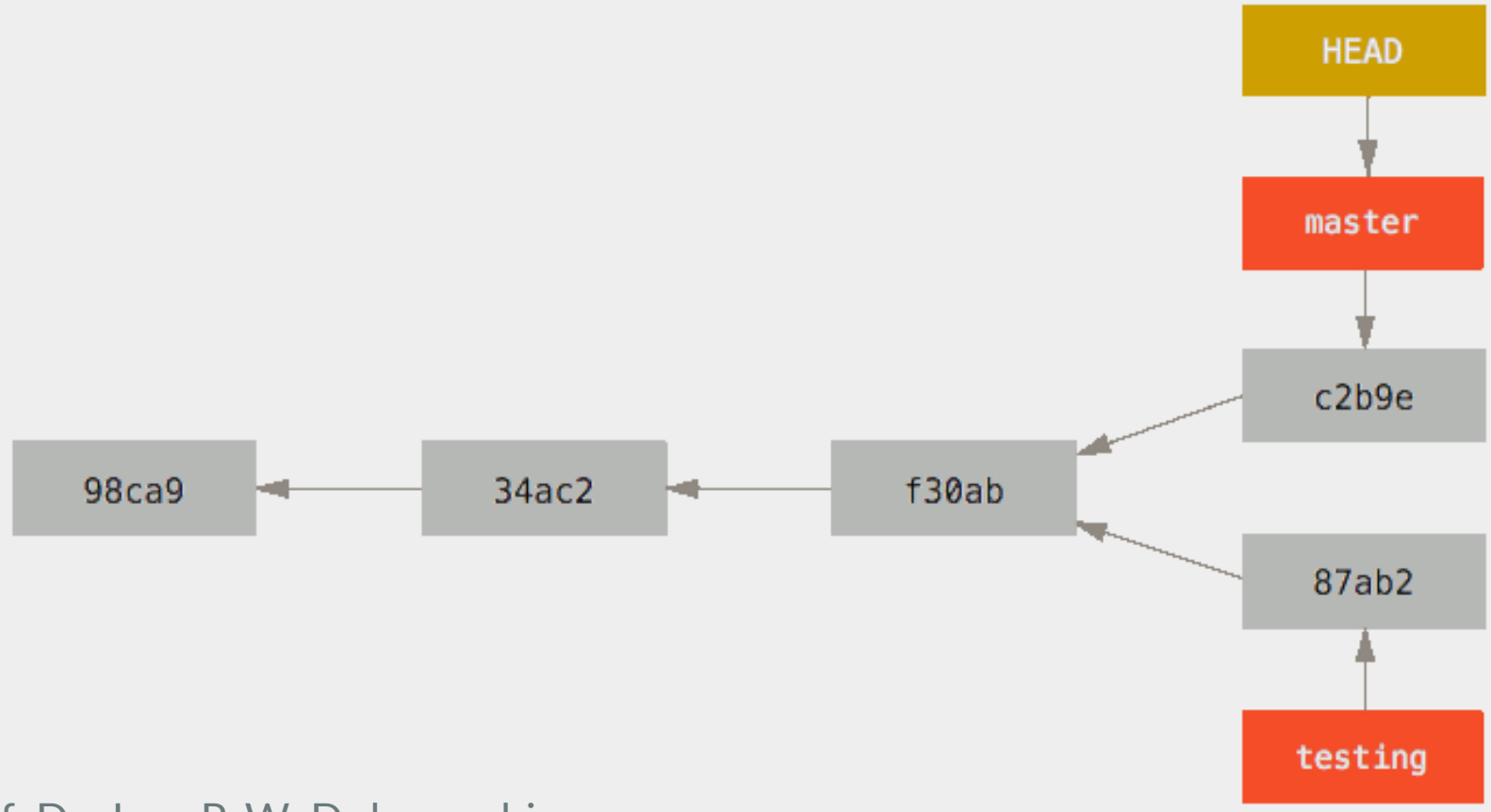
After `git commit -m "testing commit"`



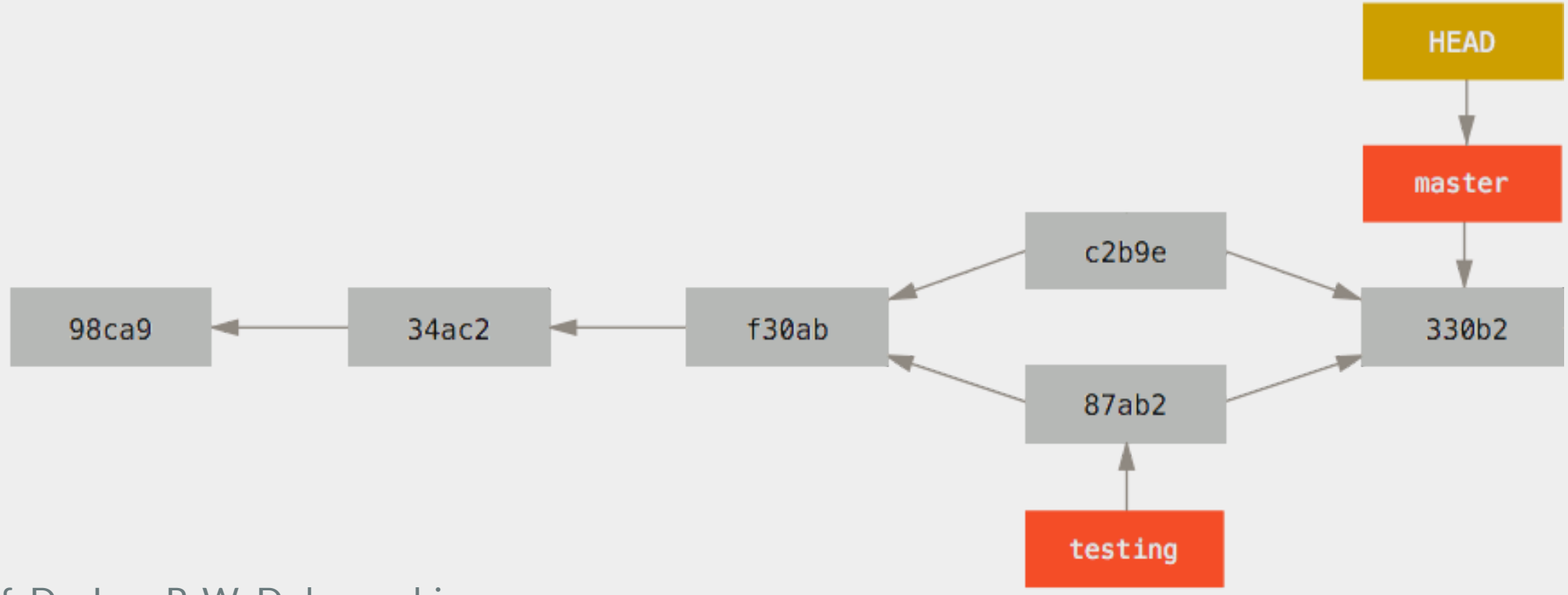
After `git checkout master`

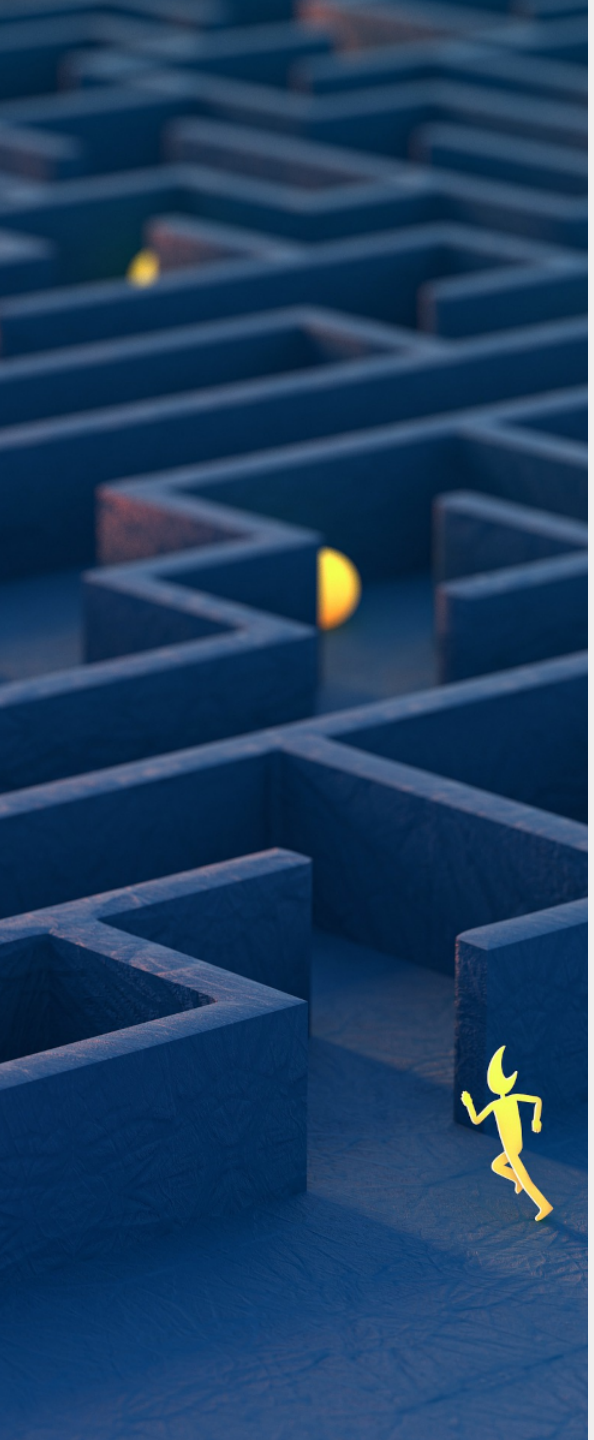


After `git commit -m "master commit"`



After `git merge testing`





Exercise: Feature branch

- New feature branch for Markdown-Cheatsheet
- Write a few notes about Markdown syntax in a new markdown file
- Have at least two commits on the branch (small changes, commit intermediate states)
- Push each commit, then look at the network graph
- Merge to master, push, look at the network graph

README

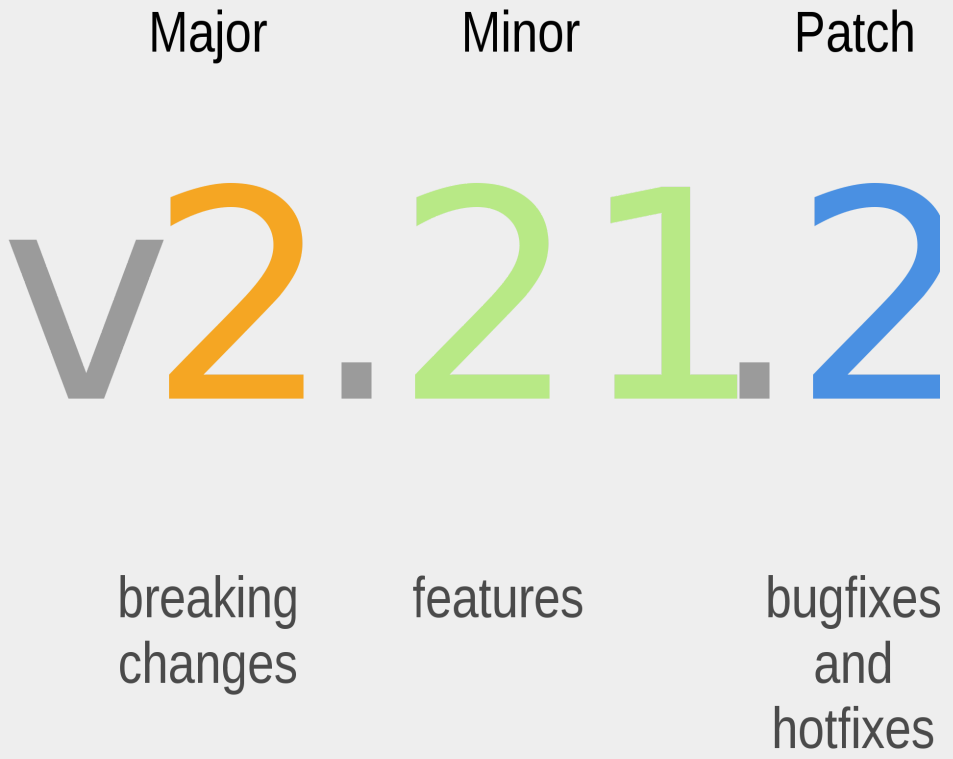
- Default file shown on repo landing page
- Main information source for users
  - What is the tool/repo for?
  - How is it supposed to be used?
    - Installation (easy! e.g. release)
    - Example call with example data
    - Expected output with explanation
    - What can go wrong?

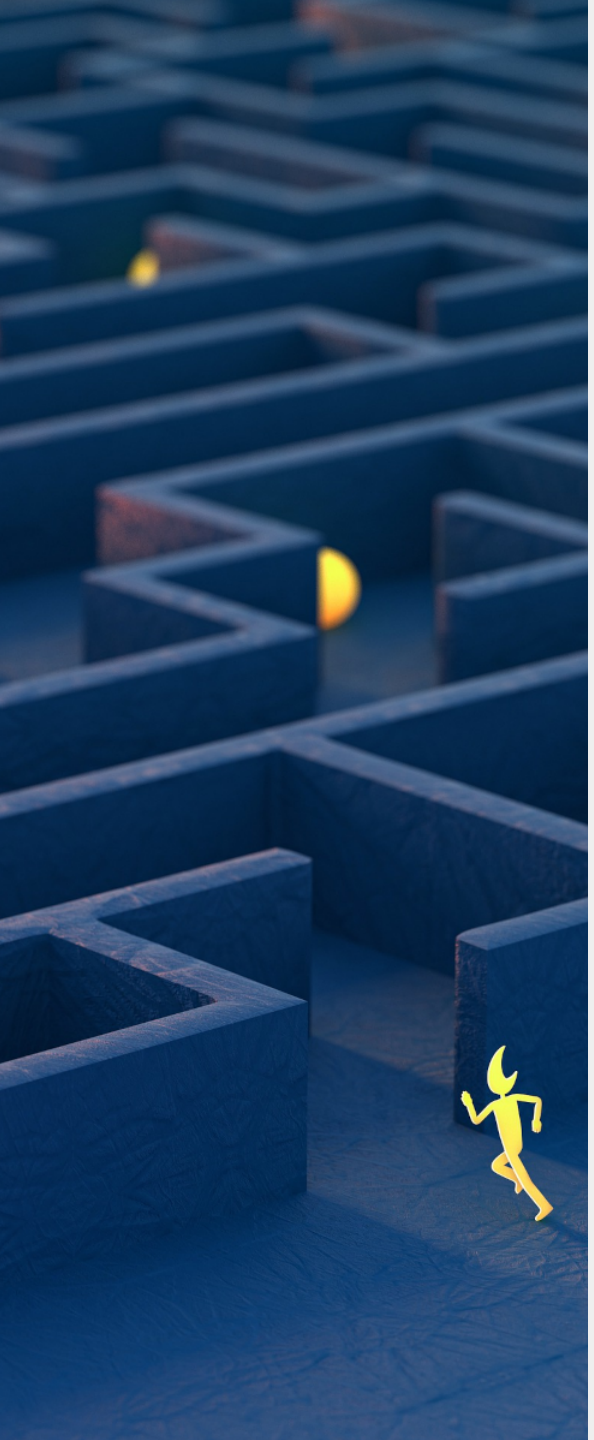
Tags

- Generally: Link to a specific commit (see git-scm)
- Contain metadata like date and comment (except lightweight tags)
- Creation: `git tag -a v0.1 -m "The first version"`
- Show info: `git show v0.1`
- push: `git push origin v0.1` or `git push origin --tags`
- Checkout:
  - Only look, no touch: `git checkout v0.1` (detached HEAD)
  - Change (new branch): `git checkout -b branch_v0.1 v0.1`

Semantic versioning

- Convension for versioning of software
- Clear information on effects of changes and compatibility
- No defined standard, just a convention



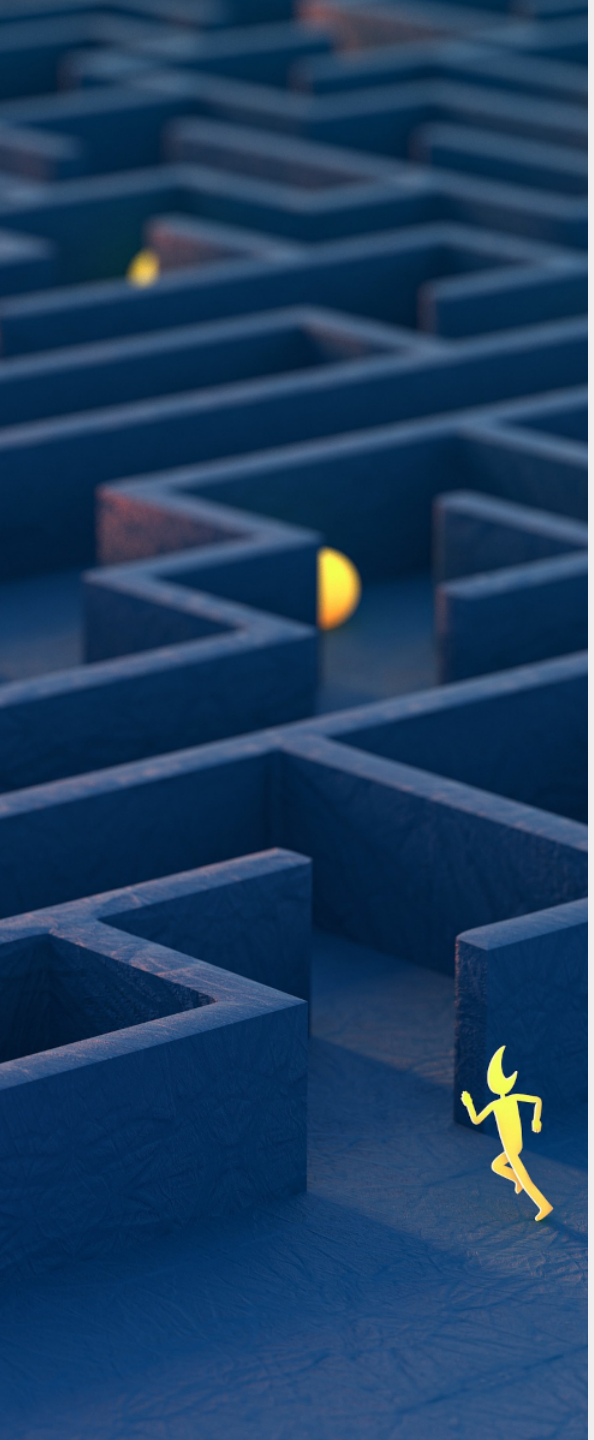


Exercise: Tagging

- Two changes (in two separate feature branches):
  - Add README.md
  - Update git cheatsheet
- merge with master
- tag as "v0.1"

Releases

- Stable "snapshots" of the repositorie's state
- Can be enrichted with further data/files (e.g. binaries)
- Simple mechanism for distributing software
- Based on tags
- `.zip` and `.tar.gz` of repository are automatically generated
- Described in depth in the github documentation



Exercise: Release

- Build PDFs from your cheatsheets
- Create a release from the v1.0 tag
- Include the PDFs in the release

Licenses - basics

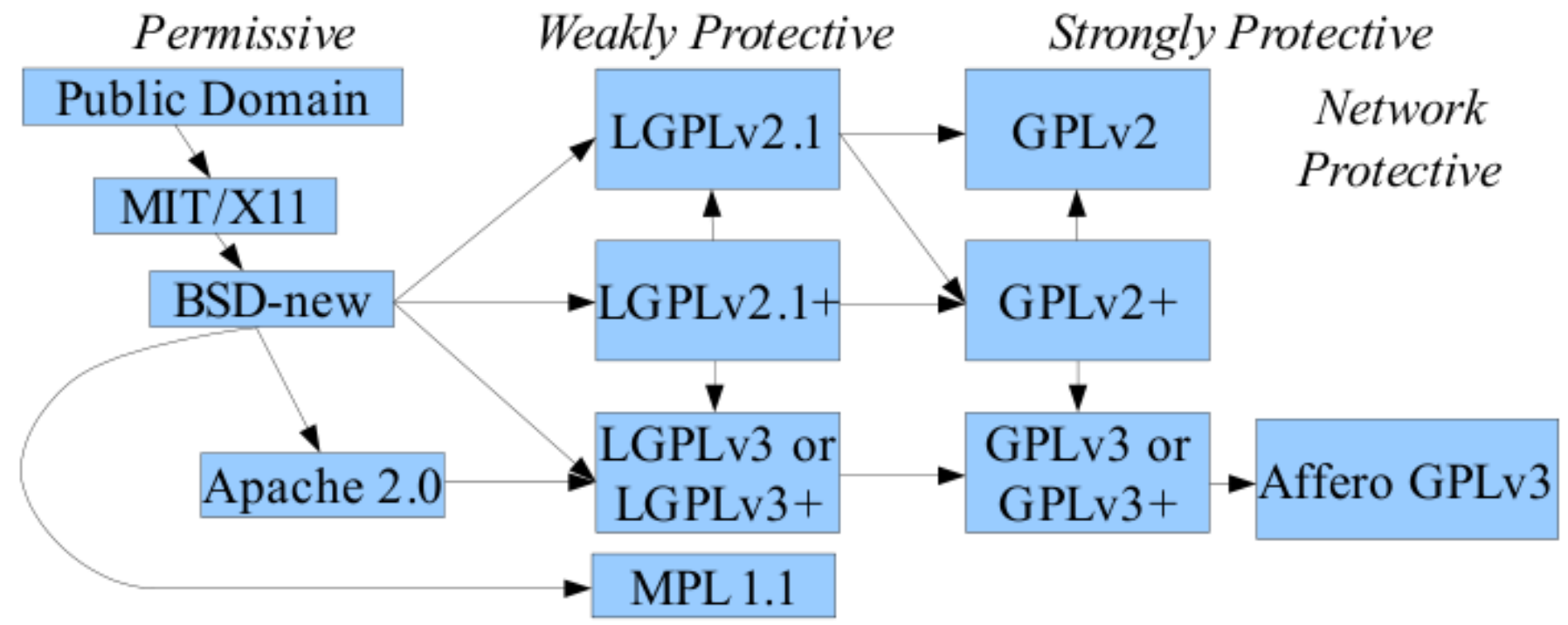
- Disclaimer: IANAL
- Important: Do set a license! It ensures:
  - Re-usability
  - Own rights
  - Self-protection, e.g. liability (don't feed the lawyer)
- Basic differences:
  - Open Source vs. Closed Source
  - Copyright vs. Copyleft

Licenses - short how-to

- LICENSE in repository root
  - gitlab, github: Templates available for typical licenses
  - Look at it together - LICENSE in example repository
- Attention: Licenses can be incompatible!
  - Caution generally required when:
    - Modifying code (compatible with own license?)
    - Redistribution (requirements, e.g. making source available)
  - Simple use without modification usually unproblematic

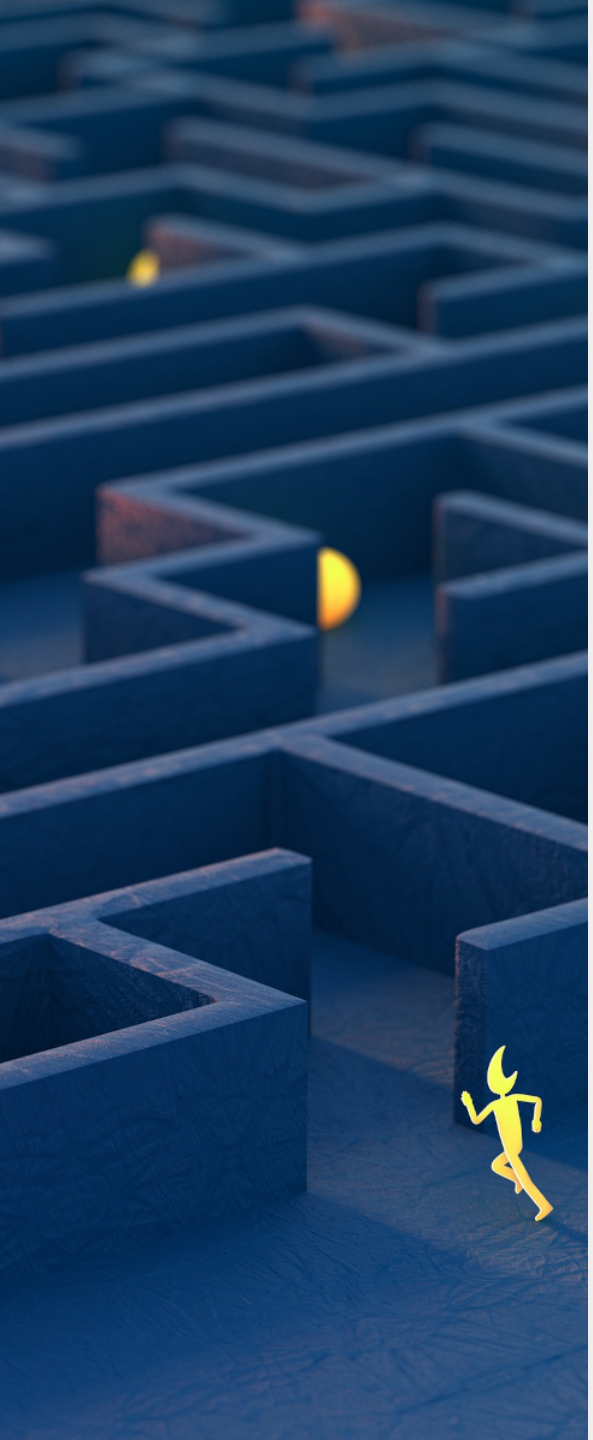
Licenses - cheat sheet

- Overview: https://tldrlegal.com/, https://choosealicense.com/
- Compatibility chart by David A. Wheeler



Zitierhinweise

- Erleichtert anderen das Zitieren der Software
- Eintrag in citation file format in `CITATION.cff`
  - In github: Automatische Anzeige als "Cite this repository"
  - Standard: Software citation
  - Wenn Paper publiziert wurde: `preferred-citation` auf `article` setzen und freuen!
- Beispiel: Github-Dokumentationsseite



Exercise: Citability

- Finalise the repository:
  - Set a license
  - Add last notes
  - Add new tag
  - Register the repository with zenodo
  - Create another release on github
  - Add a `CITATION.cff` with the DOI from zenodo

More practice: Oh my git!

- Game for practicing and visualizing git concepts
  - Real git, interaction via commandline or playing cards
  - Visualization of the repository structure (like "network" in github)
  - Separate "missions" for different concepts

Image sources

Folie 1: "Staub", tommyvideo, Pixabay
Folie 2: "Road", Navi, Unsplash
Folie 3: "Nerd", cottonbro, Pexels
Folie 4: "Group of friends", StockSnap, Pixabay
Folie 5: "notFinal.doc", Jorge Cham, "Piled Higher and Deeper", PHD comics
Folie 6: "Diary", Pexels, Pixabay
Folie 7: "Git: Reference Sheet", New Zealand eScience Infrastructure
Folie 8: "Deltas", Scott Chacon und Ben Straub, Pro Git book
Folie 12: "Git", Randall Munroe, XKCD
Folien 13, 15, 18, 20, 26-31: "Git Branching", Scott Chacon und Ben Straub, Pro Git book
Folie 33: "Will it work", Randall Munroe, XKCD
Folie 35: "Versionsnummern gemäß Semantic Versioning", Surjit Bains, Wikipedia
Folie 41: "FLOSS License chart", David A. Wheeler
Folie 44: "Oh my git! logo", bleeptrack & blinry
Exercise slides: "Mondmärchen", CDD20, Pixabay