Introduction to Git

FH Sustainable Computing Workshop
September 8, 2023

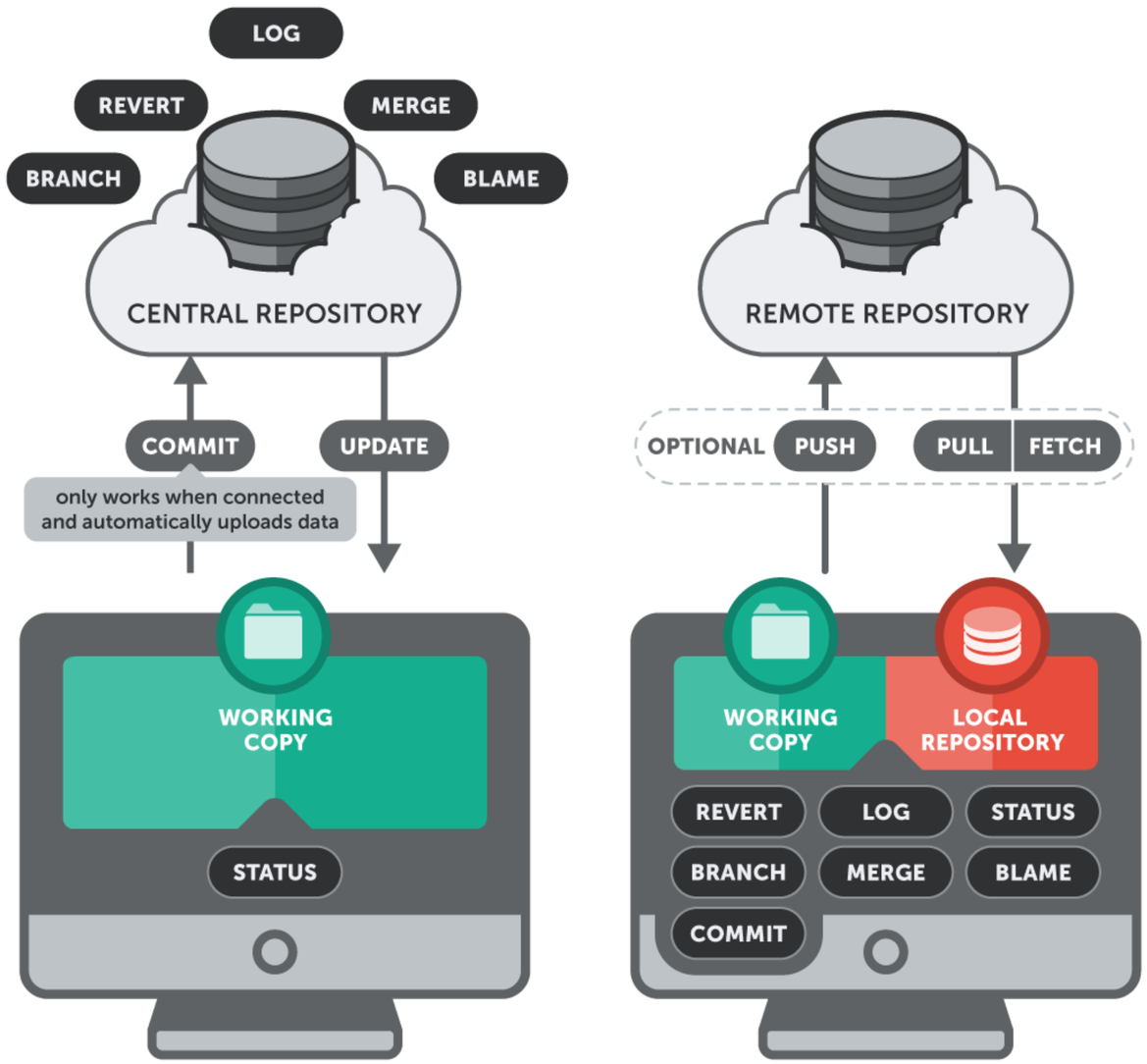
Tadej Novak

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Why Version Control?

- Keeping history of the code, documents, configuration, general text files.
- Collaborating with other people and easily merge the work.
- Save checkpoints in time:
  - revert to any point in time if something goes wrong
  - be able to look at specific revisions of documents/code
- Automatically build or deploy on every change.

Git — a Distributed Version Controls System (VCS)



- git is distributed
- each developer has (at least) one copy of the repo — the local repository, and interacts with one/multiple remote repositories — remotes
- all version control actions can be performed offline
- online access needed only to share your work with others and obtain the changes introduced by others
- mostly for textual files, binary files discouraged

Git Basics

- initialise a git repository: `git init`
- add a file/change to the staging area:
  - `git add <filename>` (add a file with all changes)
  - `git add -p` (cycle through changes and add one by one)
  - never use `git commit -a` or `git add *`
- commit the staging area to the local repository:
  `git commit -m "My commit message"`
- display the current status of your working copy (and its relationship with remotes): `git status`
- check the history: `git log`

Some Technicalities

- The local repo is located under a folder named `.git` in your working copy.
- Git uses 160-bit (SHA-1) hashes to point to a given state of your repository.
- This hash is always unique (collision unlikely).
- It is hard to memorise and not very practical — we use references.
  - HEAD is a reference to the hash describing the current status
  - `main` or `master` is a branch (usually the main branch)
  - `release/21.0.21` is a tag
- A given hash points to the same content, even if from different repositories, branches, tags, ...
- For small repositories, 7 leading digits of the hash are (usually) enough.

Examples:

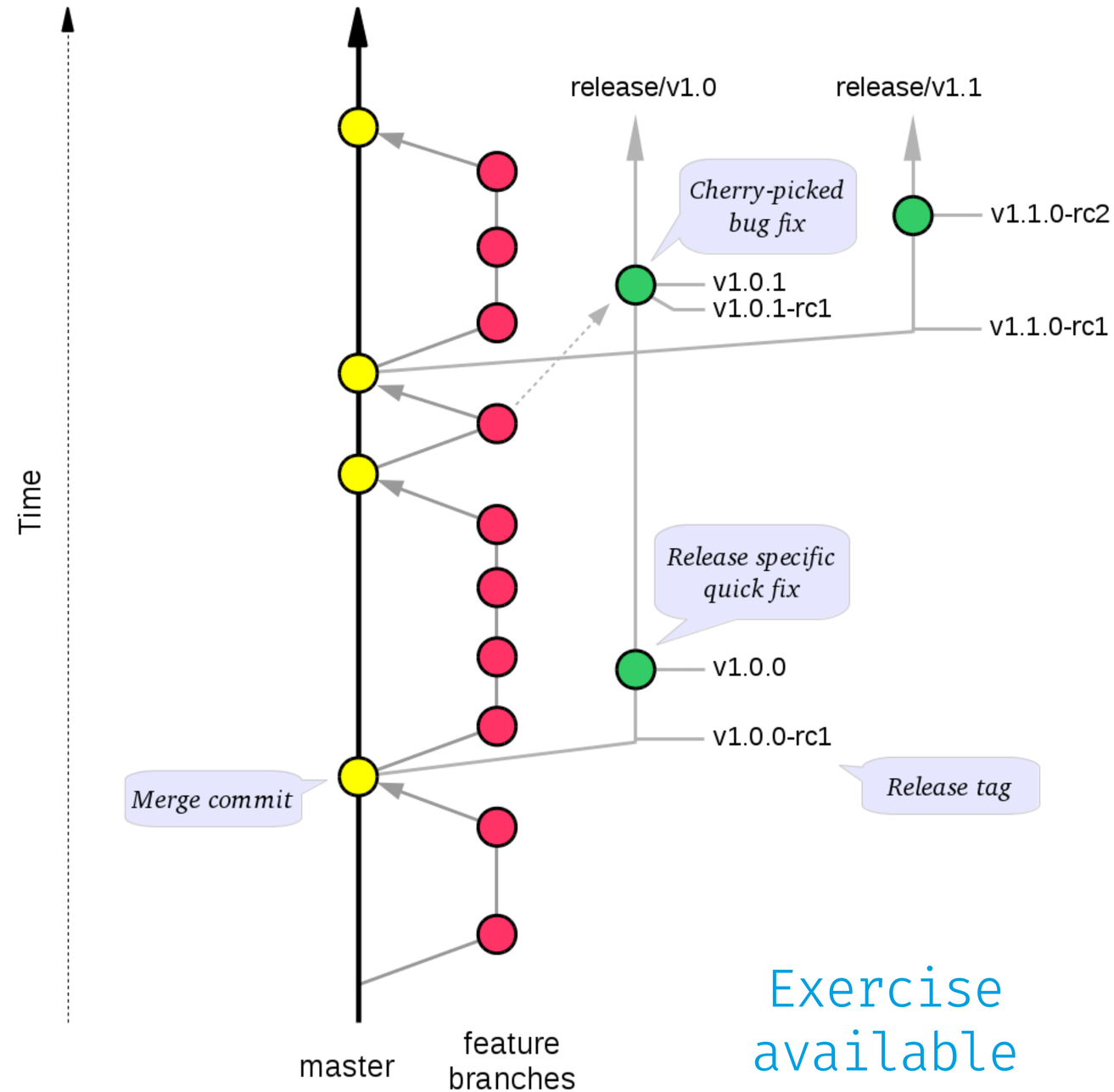
- show the contents of one commit: `git show e3153d7`

Organising Your Work

- `main` (formerly `master`) is the main branch of your repository (can be changed)
- you work on larger chunks in branches
- to reference a specific state of your repository you use tags

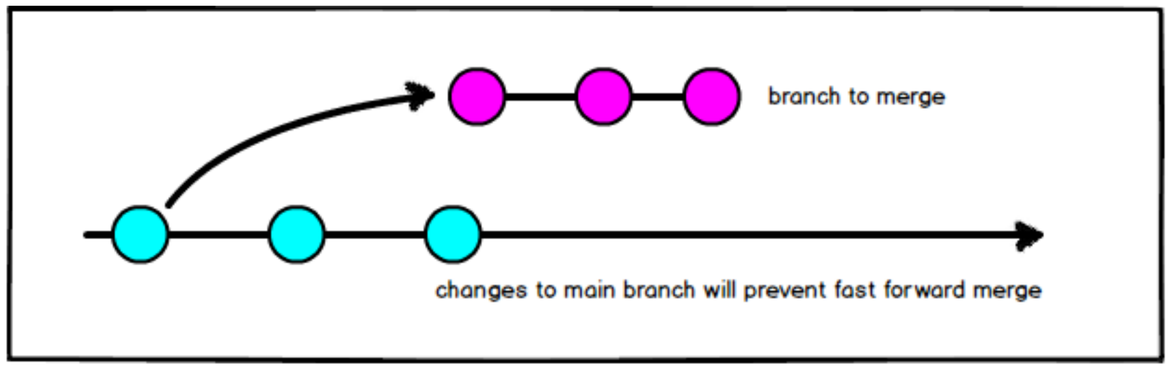
Examples:

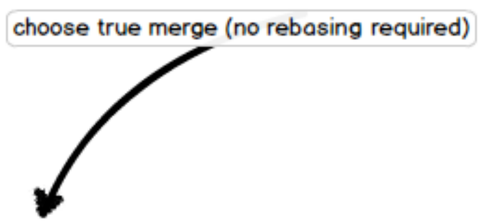
- Move to a branch:
  `git checkout <branch>`
- Create a branch out of another branch/tag/hash:
  `git checkout -b <reference>`
- Create a tag out of current state:
  `git tag v1.0.1`

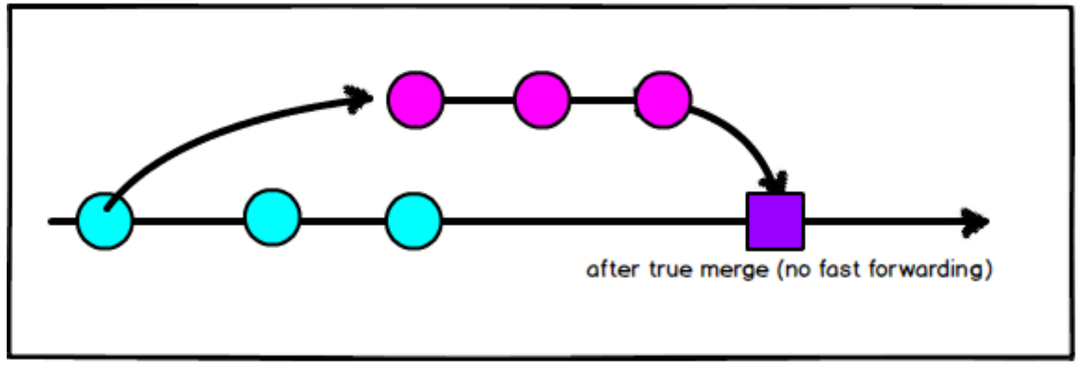


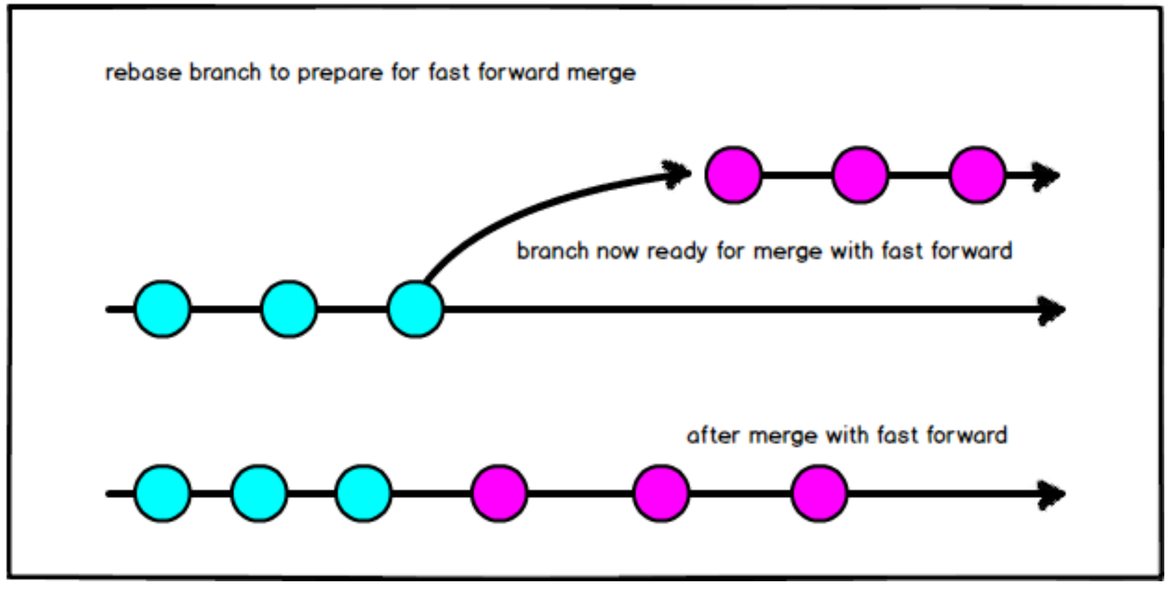
Exercise available

Merging the Changes — Merge vs Rebase









Exercise available

Conflicts:

- merge: you edit someone else's code
- rebase: you edit your own code

Merging the Changes — Commands

Merge

- Always go to the branch you want to merge your changes to:
  `git checkout master`
- Merge the changes:
  `git merge super-new-feature`

Rebase

- Go to the branch you are working on:
  `git checkout super-new-feature`
- Rebase onto a branch (e.g. master):
  `git rebase master`
- Merge like usual.

Interacting With Remote Git Repositories

- Initialise a copy of a remote repo: `git clone <url> <local-folder>`
- Display which remote repositories are known: `git remote`
  - add a remote: `git remote add <name> <url>`
  - update a remote: `git fetch <remote>` (or `git fetch --all`)

What happens when you clone a repository?

- By default, the main branch is checked-out (add `-b <branch>` for another).
- We're set to track the master branch of the remote — upstream branch, "Your branch is up-to-date with origin/master"
- The remote is by default mapped to the name origin and each remote branch is referenced under origin/branch.

Making Your Changes Available To Others

Exercise available

- Upload/copy your changes to the remote repository: `git push <remote> <branch/tag>`
  - shorthand for the current branch: `git push`
  - first time you push use the `-u` flag to enable tracking
  - force push — one needs to be very careful!

What if I am behind the remote?

- `git pull --rebase` — always recommended to rebase
- use fetch if not trivial

Some Useful Commands

Stash — temporarily save and remove your local changes

- add to stash: `git stash save <name>`
- list stashed: `git stash list`
- retrieve from stash
  - but do not remove it: `git apply <name>`
  - and remove it: `git pop <name>`
- name is always optional

Cherry-pick

- Take one hash and apply it to your current staging area:
  `git cherry-pick e3153d7`

Collaborating With Others

- Using one of the shared or private git hosting services (e.g. DESY GitLab, GitHub, ...).
- You make a fork of the repository — your personal online copy.
- Always clone your own copy.
- Add a remote for the original — upstream repository.
- When ready, push to your remote and submit a Pull/Merge Request (PR/MR)

Examples

- `git clone git@gitlab.desy.de:fh-sustainability-forum/sustainable-coding-tutorial/git-exercises.git`
- `git remote add origin git@gitlab.desy.de:fh-sustainability-forum/sustainable-coding-tutorial/git-exercises.git`

Services to Share Code: GitLab

Exercise available

- DESY provides an instance of GitLab: https://gitlab.desy.de
  - A powerful software suite to not only host git repositories, but also track progress/issues, test the code, ...
  - Many other institutes have their own instances (e.g. CERN).
- GitHub is one of the other popular hosting providers.
  - Will not go into detail today.

- Three exercises provided on DESY GitLab: https://gitlab.desy.de/fh-sustainability-forum/sustainable-coding-tutorial/git-exercises

What If?

I've made a lot of changes but now I want to submit just some of them?

- Look at the log and decide what you want (`git log`).
- Make a new branch from upstream/master:
  `git checkout -b feature-a upstream/master`
- Cherry-pick commit(s) that you found in the log:
  `git cherry-pick e3153d7`
  `git cherry-pick f249a34`
- Push to the origin:
  `git push -u origin feature-a`

What If?

I've made a commit but I forgot a file?

- Add the file that is missing:
  `git add missing.txt`
- Update — amend the commit:
  `git commit --amend`

What If?

I've added too much to the staging area/commit?

- If not committed yet, reset the file:
  `git reset my-file.txt`
- I have already committed — reset the state to the previous HEAD:
  `git reset --soft HEAD~1`

Conclusions

- Git helps you keep track of the history of your text files
  - code
  - configuration
  - thesis
  - recipes
- A lot of material covered today — you should understand the philosophy.

Useful links:

- https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf
- https://ohshitgit.com


DESY.