Git basics

Matthieu Haefele

Pau, January 2026

Agenda

Session 1

- What is version control ?
- Git principles
- Hands-on: serious game on git branching
- Hands-on: git in real life

Session 2

- Hands-on
- Gitlab interface

Agenda

Session 3

- Git as a first step towards open science
- Hands-on
- Setting up projects on UPPA gitlab

When you do not know how git works

Without version control

Situation

- Oh, I have a working version, but I need to change a lot of things to implement the next step.
- What shall I do ? 🤔

Without version control

What I do

- I copy the folder of my application into `myapp_backup`
- Later, I copy the folder of my application into `myapp_backup1`, `myapp_backup2`, ..., `myapp_backup125`

Without version control

Situation

- Oh, I sent a tarball of the sources to a colleague
- She sent me back a tarball with code modifications
- I have implemented new things into the code meanwhile
- What shall I do ? 🤔

Without version control

What I do

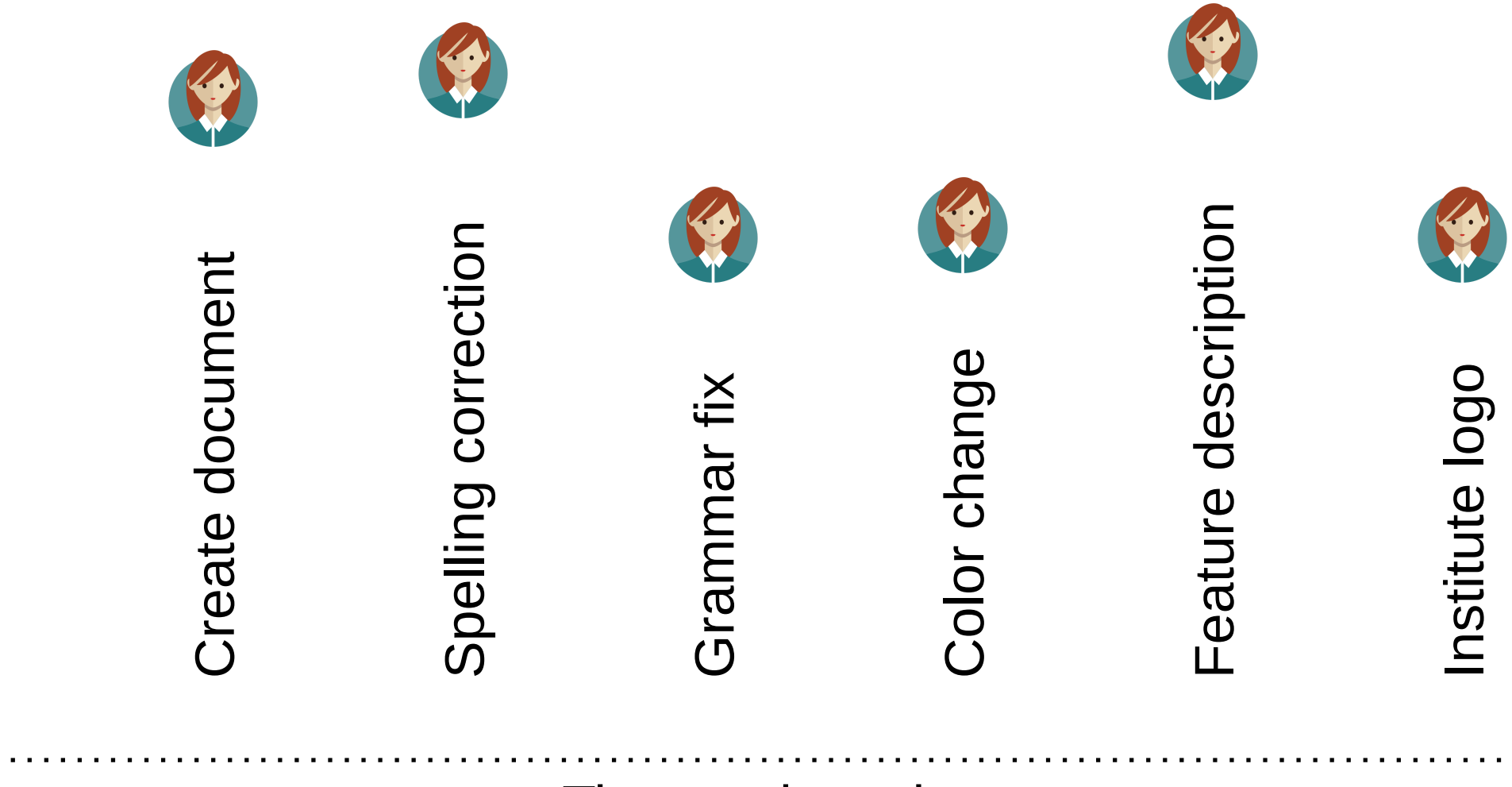
- I look at all differences in my version
- I look at all differences in my colleague's version
- I merge her differences into my version the best I can
- I run tests to check everything has been merged correctly ! If there are any...

What is version control ?

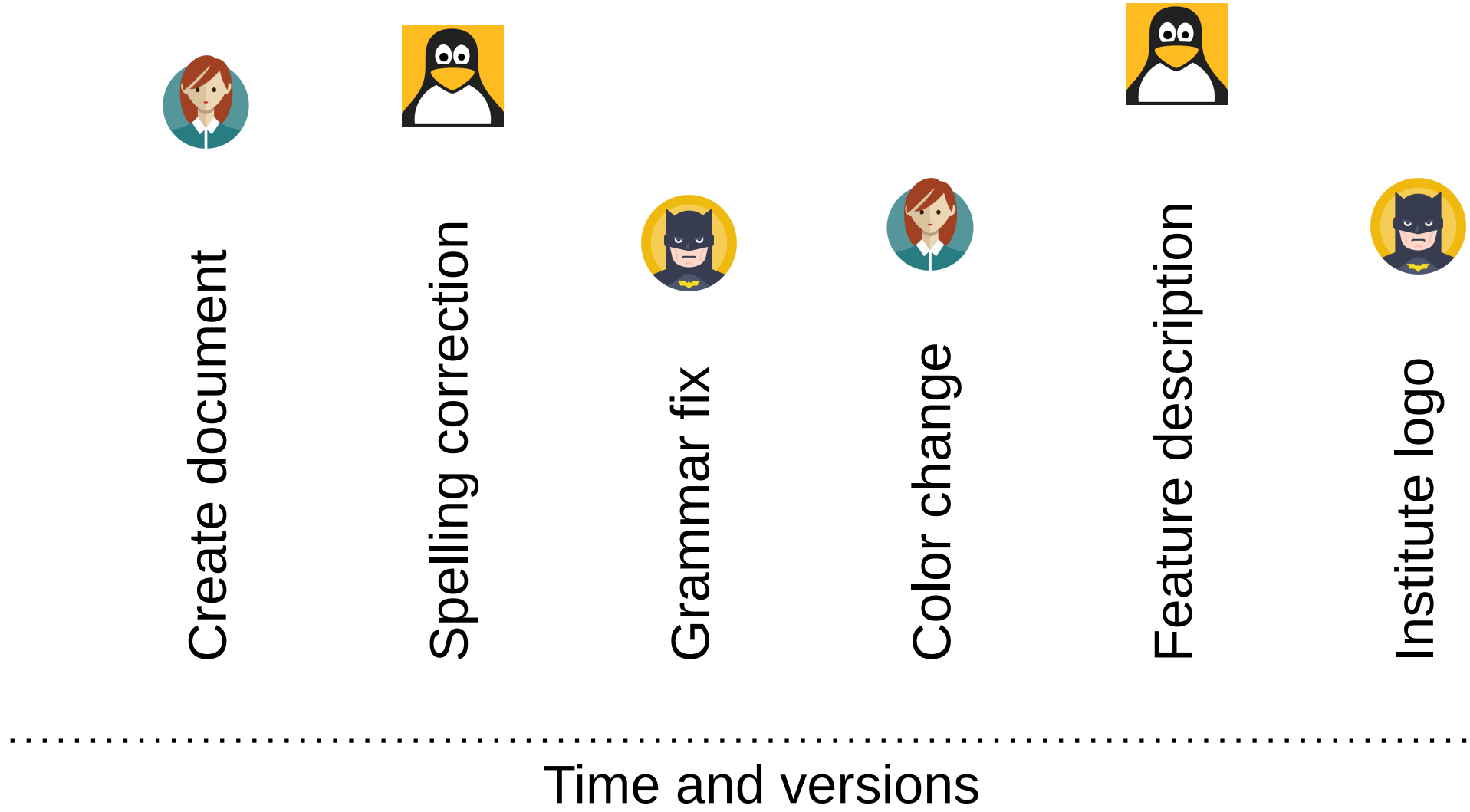
Knowledge worker main steps:

- **Create** things
- **Save** things
- **Edit** things
- Save the thing **again**

What is version control ?



What is version control ?



Acknowledgement to Matthew McCullough

Version control

- Git implements a *fast* version control system
- Git gives access to the **history** of content change
- Git facilitates **collaborative changes** to files
- Git is easy to use

SCM vs. VCS

- SCM: Source Control Manager
  - git
  - Mercurial (hg)
- VCS: Version Control System
  - Subversion (svn)
  - Concurrent Versions System (cvs)
- Main differences
  - Incremental storage of versions in VCS vs complete project at each commit in SCM
  - Branching / merging mechanism much more powerful in SCM
  - Distributed system

Git short history

Linux Kernel development

- 1991-2002: File archive + patches
- 2002-2005: Usage of BitKeeper proprietary software
- 2005: BitKeeper not anymore free of charge for the Linux project
- Development of git
- Linux kernel using git since

git-scm.com

Video: Git history by Linus Torvalds @ Google Talk

Git project spirit

- Speed
- Simple design
- **Strong support for non-linear development (thousands of parallel branches)**
- Fully distributed
- Able to handle efficiently large projects like the Linux kernel (speed and data size)
- Distributed under GNU GPL software license

Services on top of git

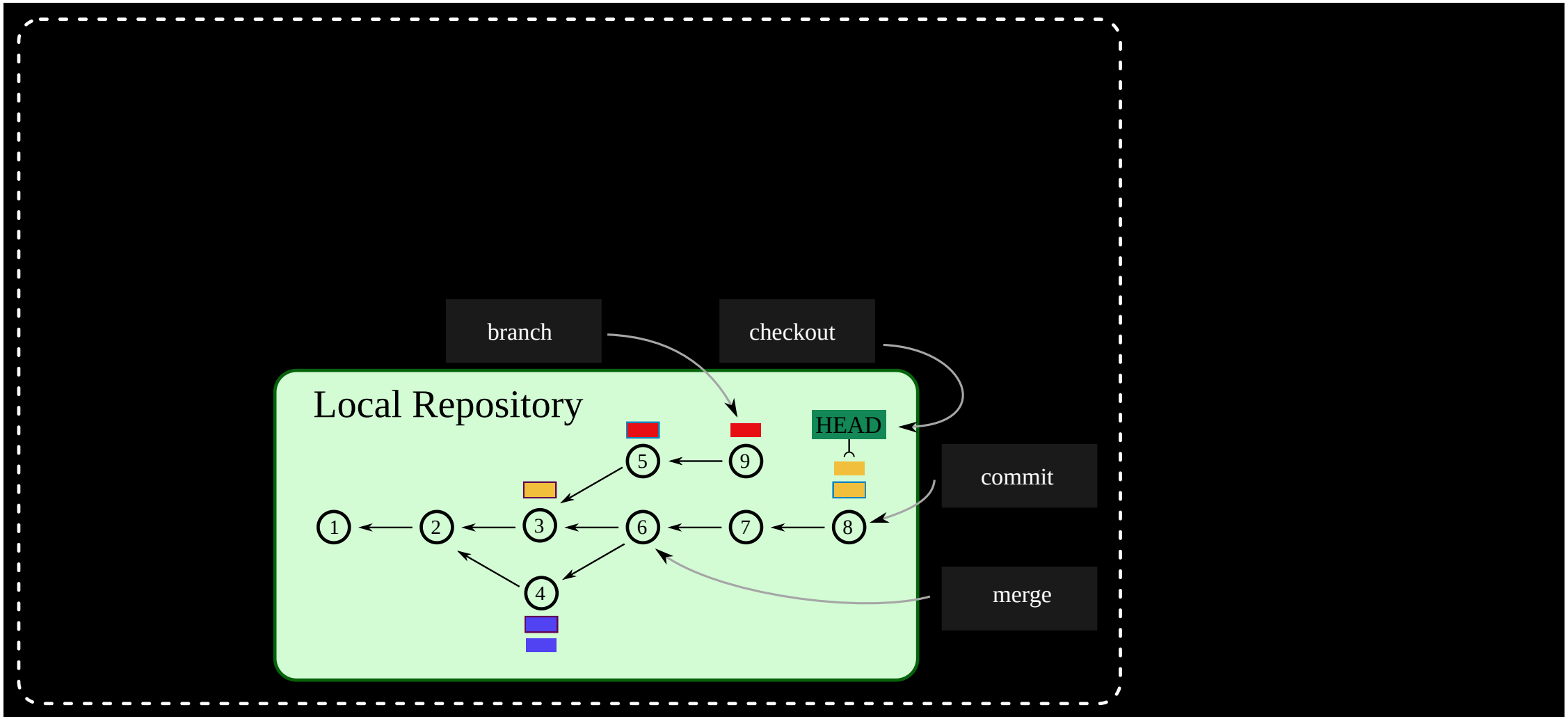
git (the GPL project)

- Keep the different versions of your software
- Merge the contributions from many collaborators

Web interfaces

- Git repository server
- Collaborative work
- Continuous integration
- github.com commercial
- Gitlab: GPL web interface (gitlab.com)

Commands overview



4 commands (to start with…)

- `commit` creates a new revision of the project
- `checkout` moves HEAD
- `branch` creates a branch
- `merge` creates one commit with the merge of two commits

What is HEAD ?

HEAD is particular internal marker that can point at

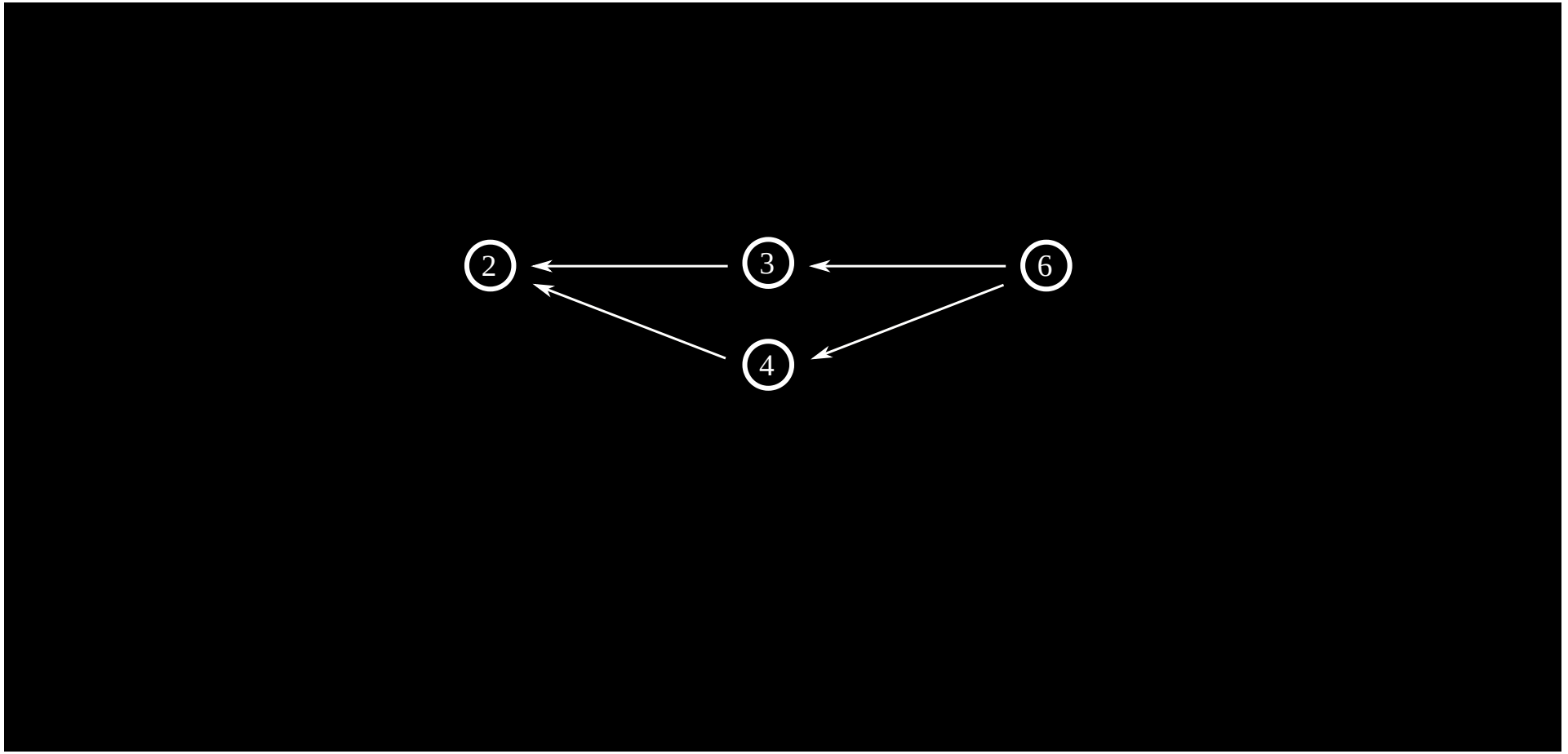
- commits
- branches

HEAD moves when doing

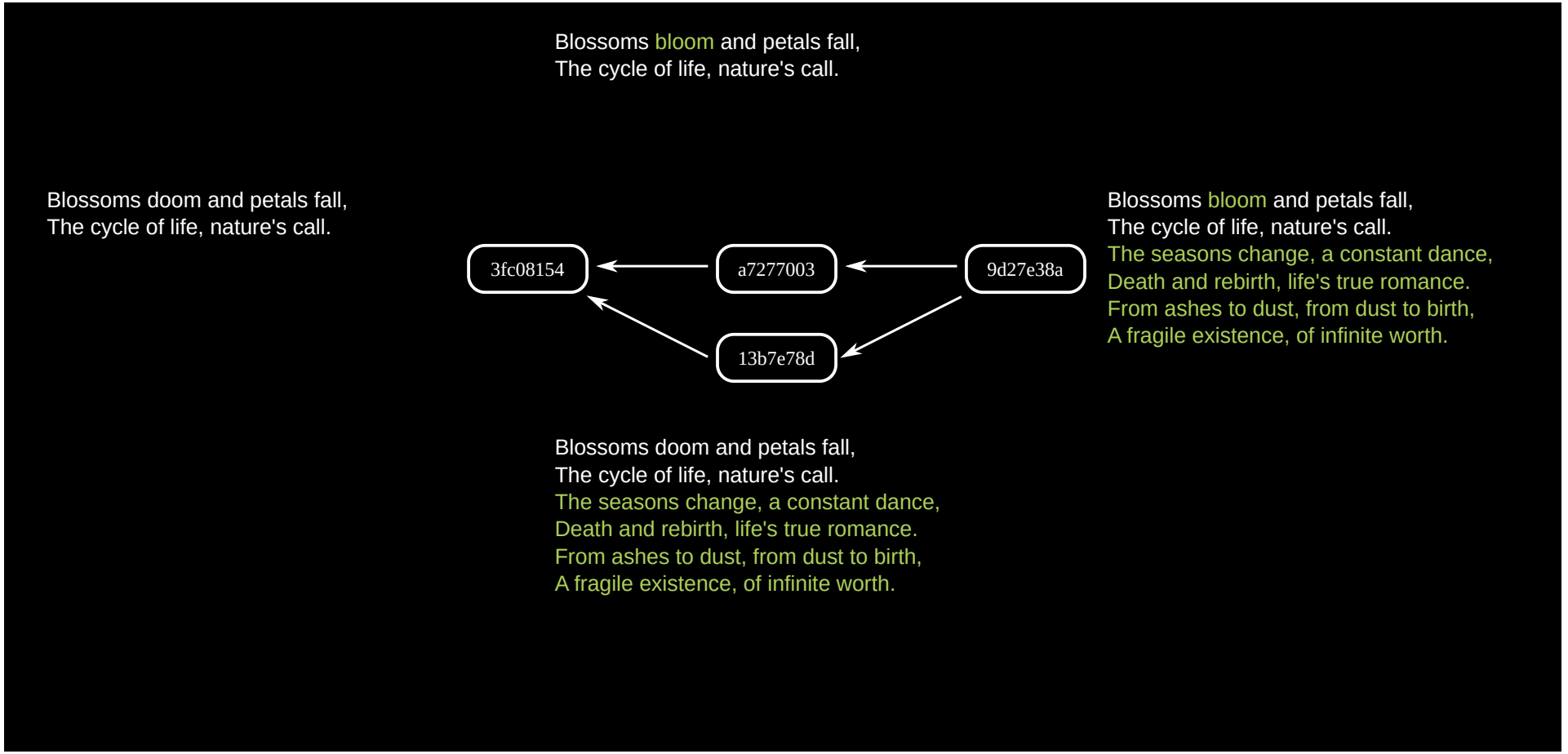
- checkout
- commit
- reset

If a git project is a tape, HEAD is the reading head of the tape reader, so your position on the tape.

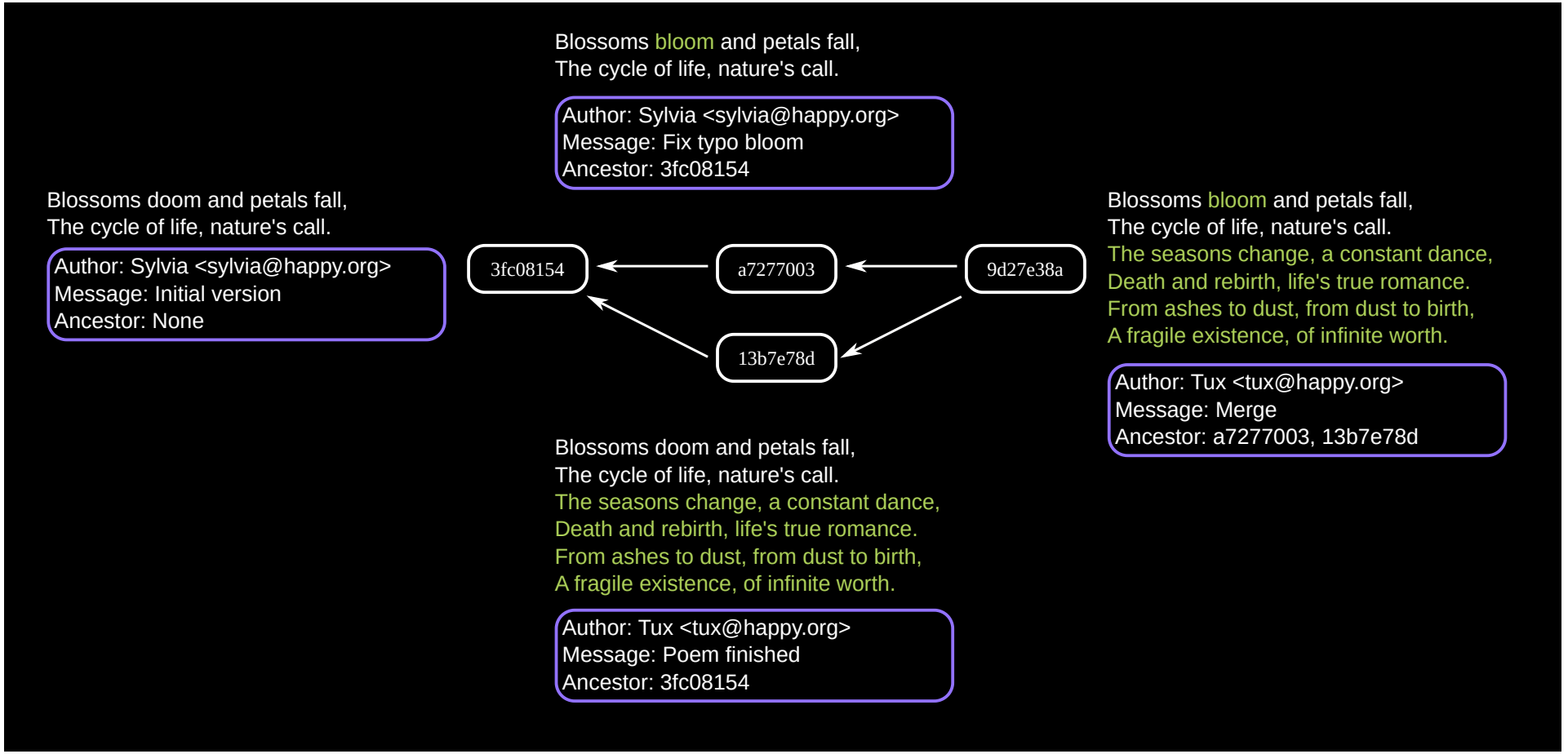
What are commits ?



What are commits ?



What are commits ?



What are commits ?

A commit is:

- Content of the project at a particular point in time
- Author
- Date
- Message
- Ancestor(s)

Let's play !

Lab1: start to be familiar with commits, branches and merges with the `main`

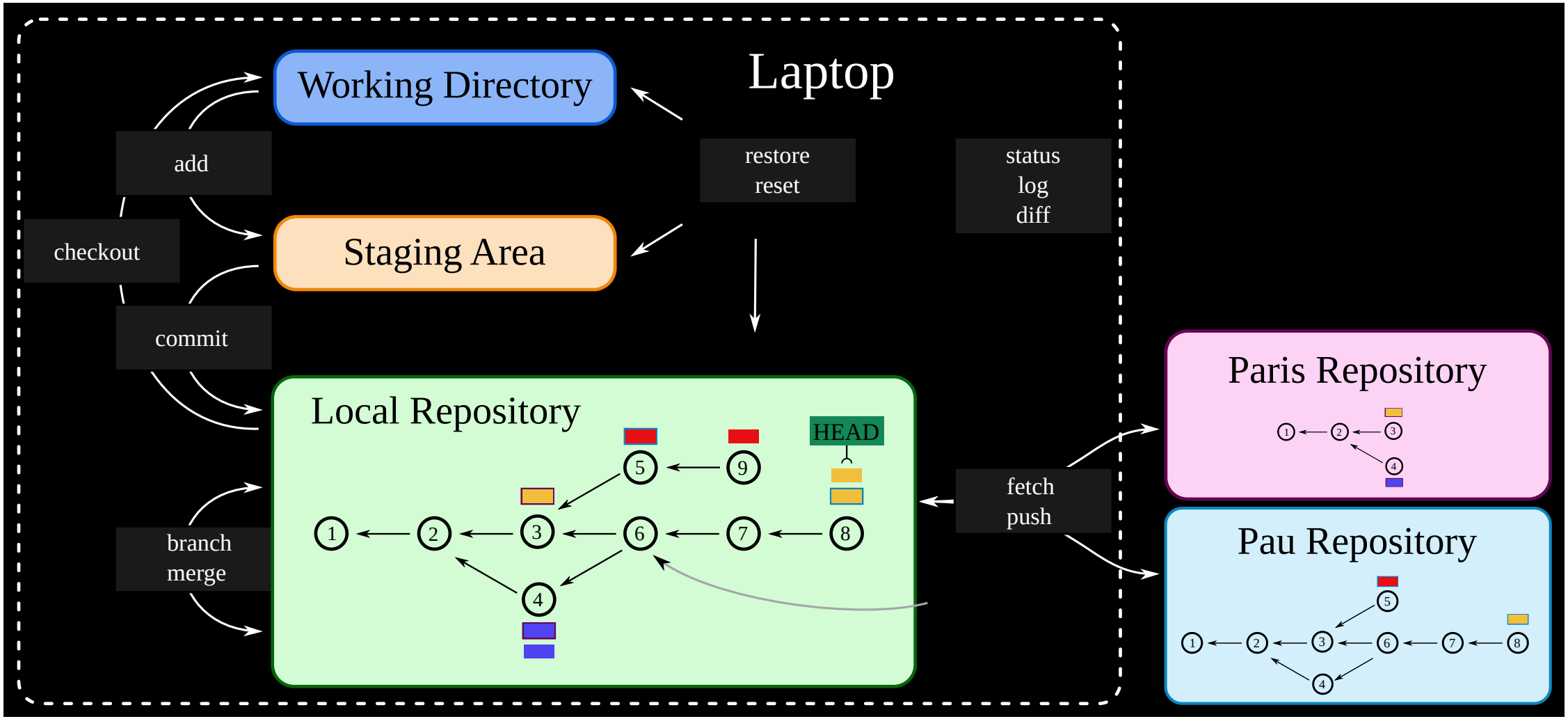
exercices of Learn git branching

Project history

Project history = commits with their relations

- `log` shows the history (commit id, authors and message)
- `diff` shows project difference between two commits

Commands overview



Three different "spaces"

Working directory

- contains the project at the current revision (pointed by HEAD)
- it is where you modify the project to create new version

Staging area

- contains the modifications "tagged" to be part of the next revision

Local repository

- contains the whole history of the project
- contains a copy of remote repositories

6 other commands (to start with...)

- `add` adds project modifications to the staging area
- `checkout` moves HEAD and updates the working directory with content coming from the local repository
- `reset / restore` do fancy things on all three spaces !
- `status` shows the status of working dir and staging area according to the local repository
- `log` shows the history of the project
- `fetch` brings content from remote repositories
- `push` sends content to remote repositories

Typical work flow

- Implement feature / bug fix
- Test if the feature / bug fix is well implemented
- Add the modified files to the staging area
- Commit
- Push/pull sometimes to share your work with collaborators

Let's play !

Lab2: do the same as in the first game but in the real world with GameShellGit