Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Big Data
National Research Programme

Introduction to Version Control with Git

Patrick Schwab and **Walter Karlen**

Mobile Health Systems Lab
Institute of Robotics and Intelligent Systems
ETH Zurich

10th April 2019

Outline

- Theory
  - What is version control?
  - What is git?
  - Git terminology and use cases
- Practice
  - Several examples

Goal

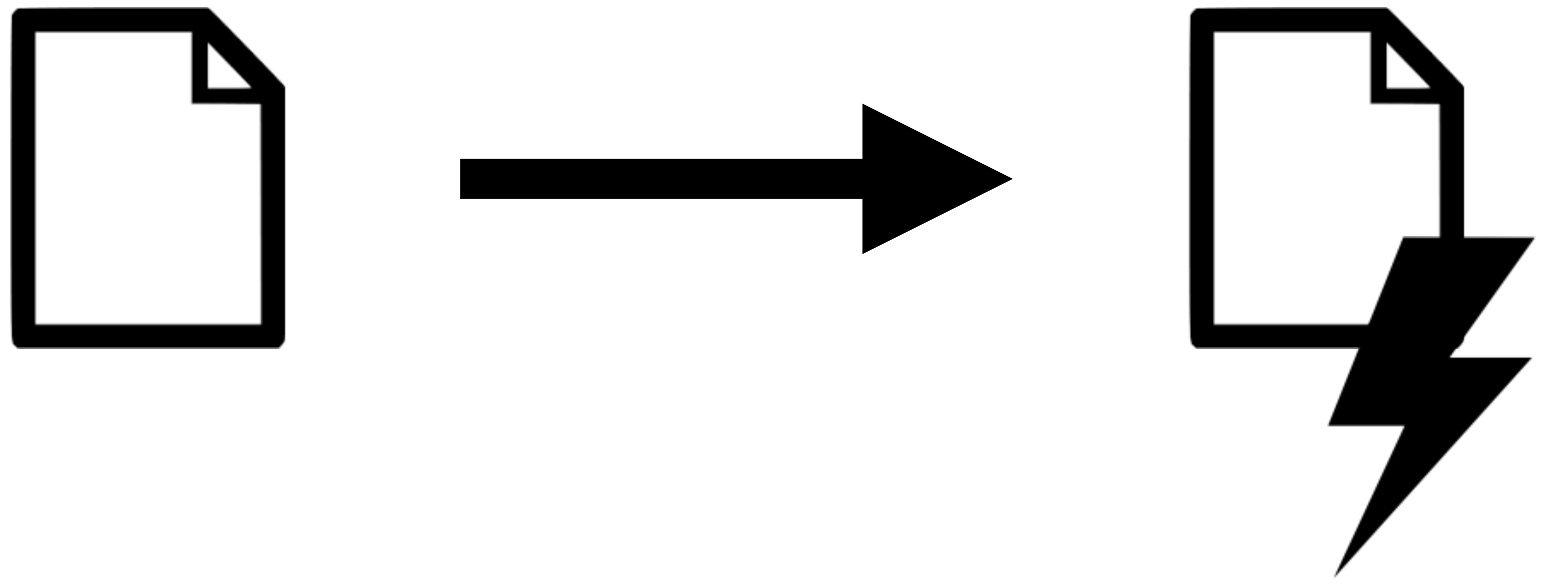
Learn how to use git for the most common day-to-day use cases.

Theory

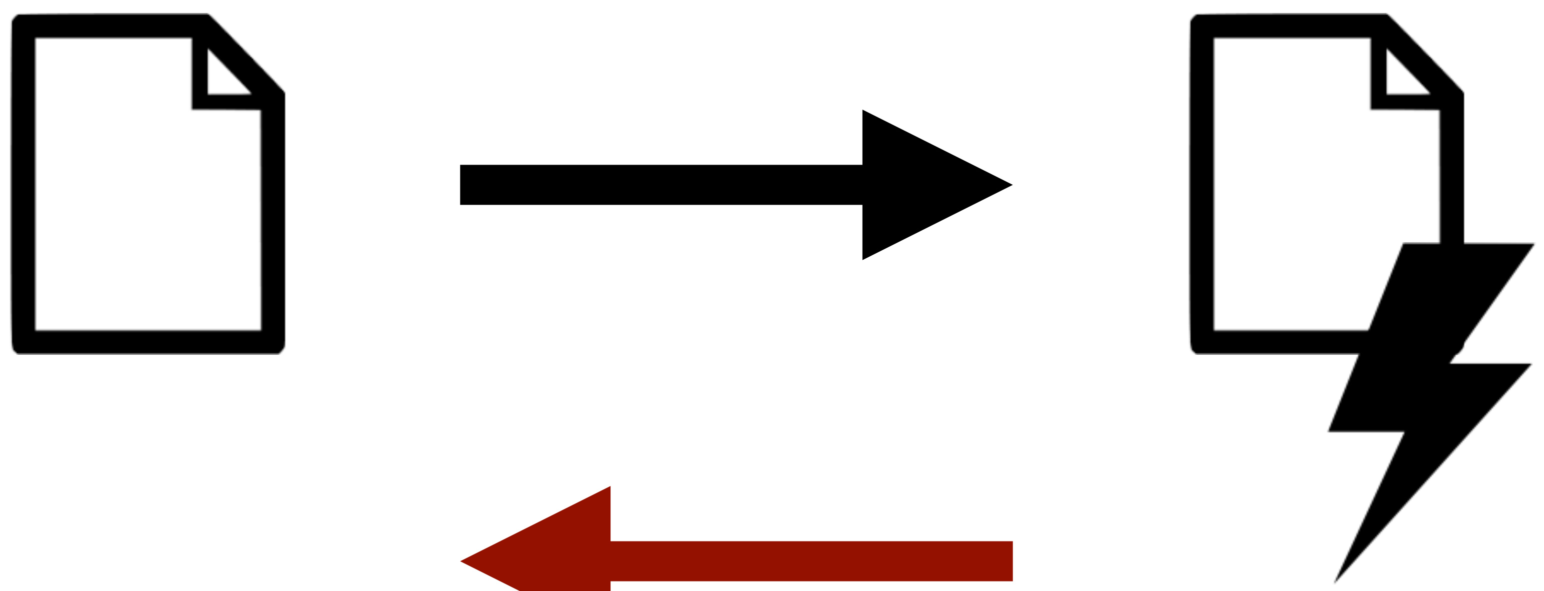
What is Version Control?

- **Software tools to …**
  - **manage changes**
  - **collaborate**
  - **preserve knowledge**
- **It is not just a file storage system**
  - **e.g. Dropbox, Google Drive**

Managing Changes

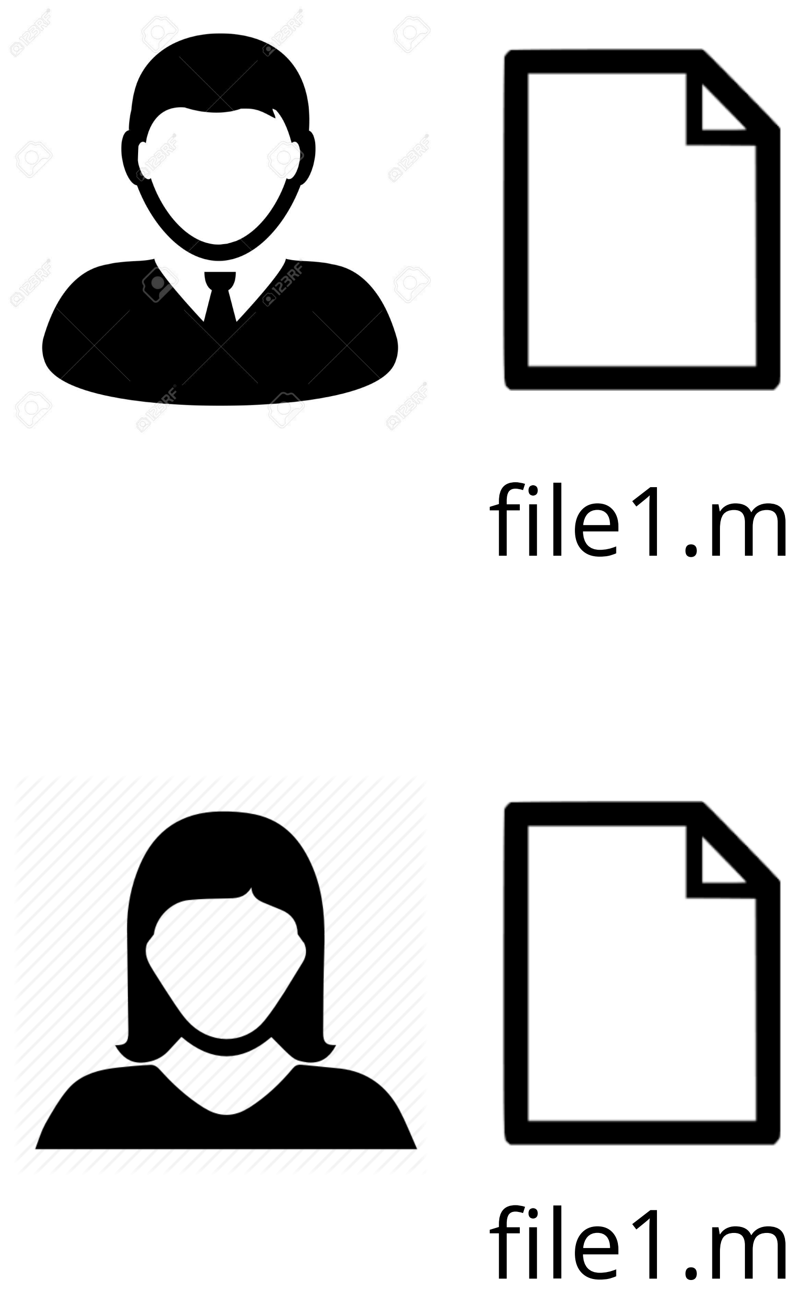


Managing Changes

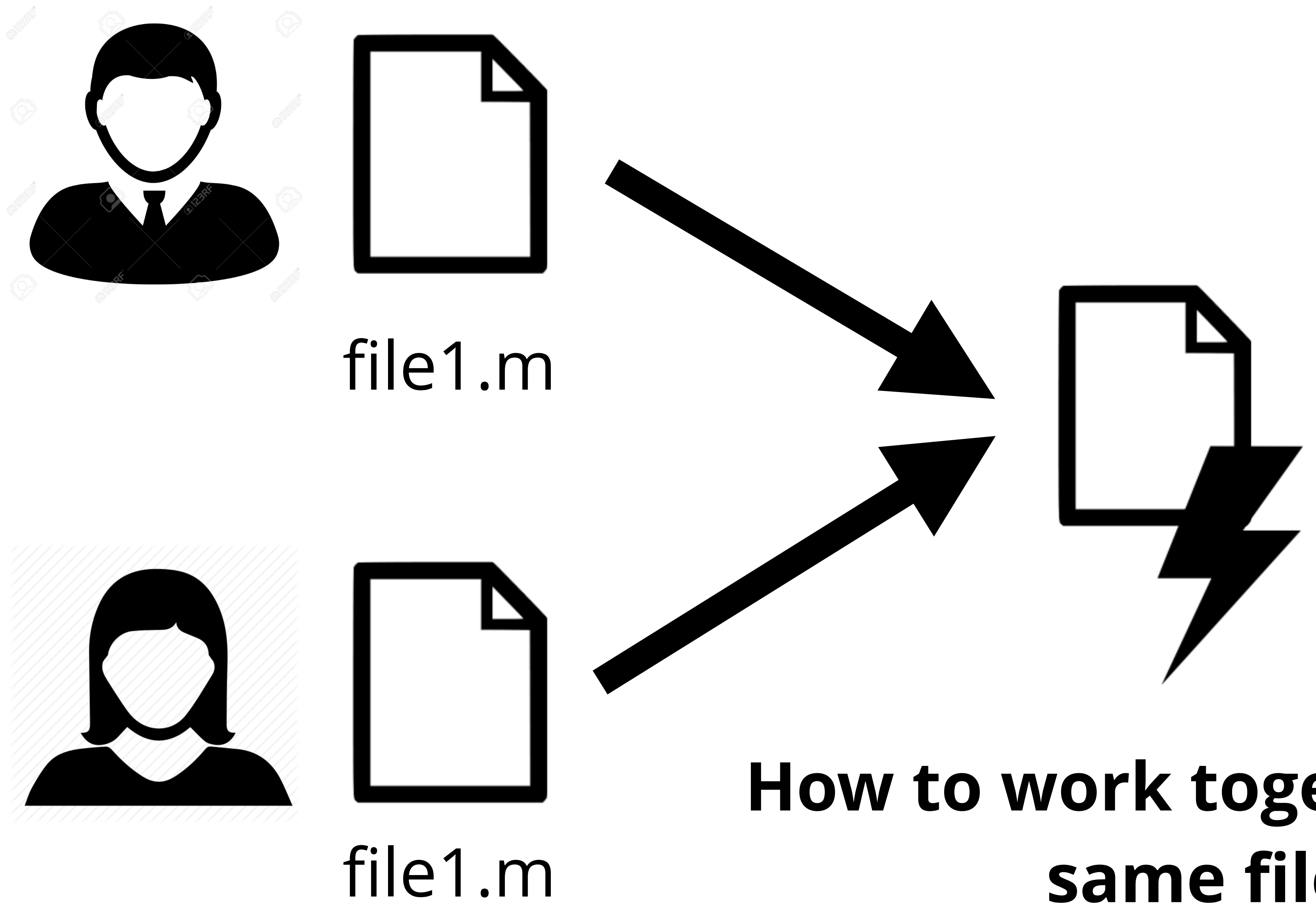


How to return to a previous version?

Collaboration

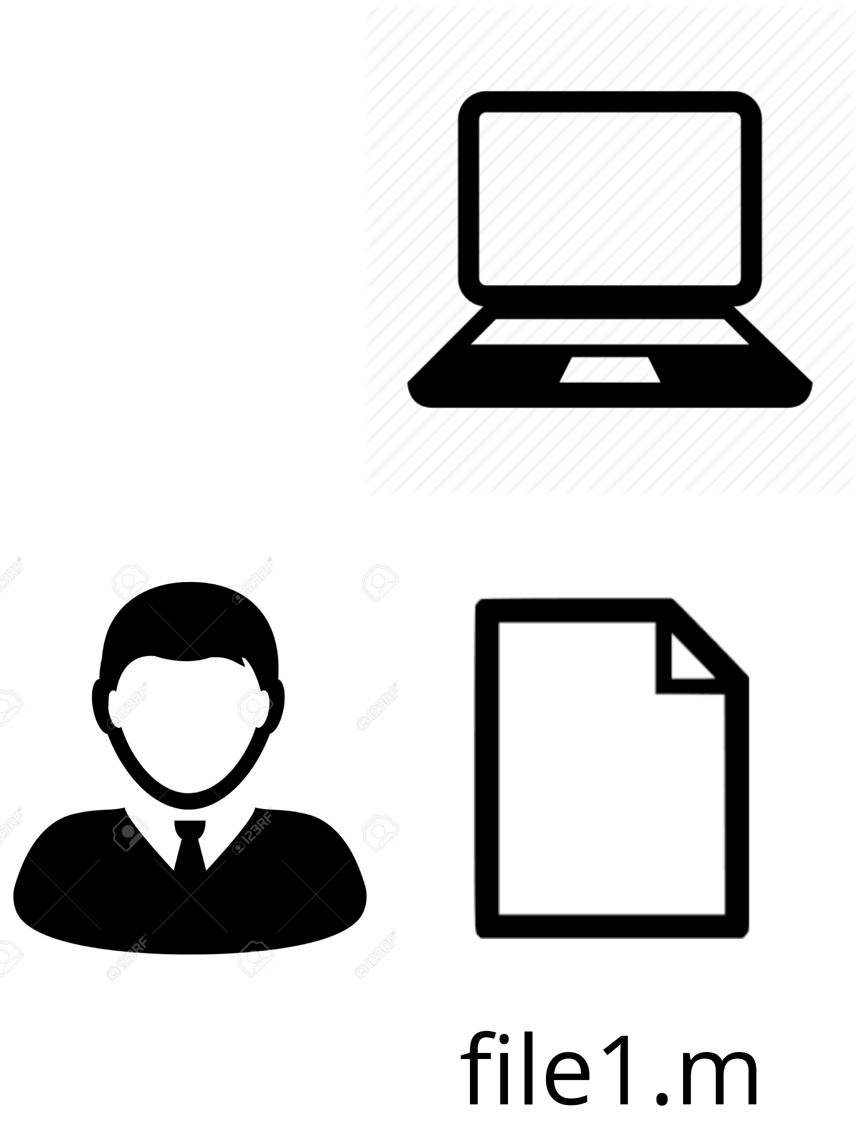


Collaboration

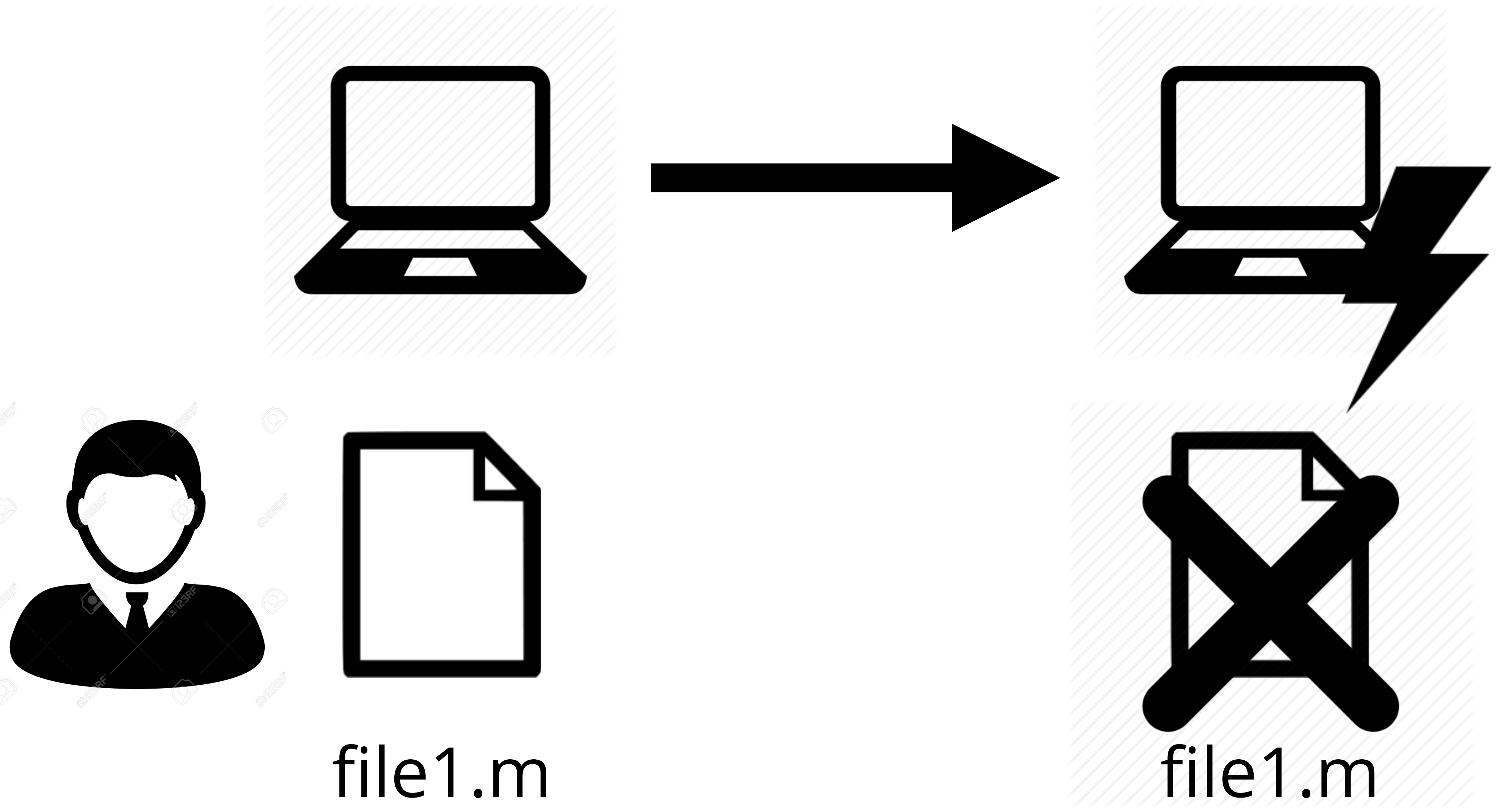


How to work together on the same files?

Preserve Knowledge



Preserve Knowledge



How to ensure all versions of code are preserved for the future?

Summary

- Version control software is widely accepted as a standard tool for version management
- It is required both in academia and industry
- You should never write code without version control in place

Git

- De-facto standard tool for version control
- Popularised through open source movement
    - e.g. github.com, bitbucket.org
- Decentralised version control system

Git Overview

local repository

file1.m

Git Overview

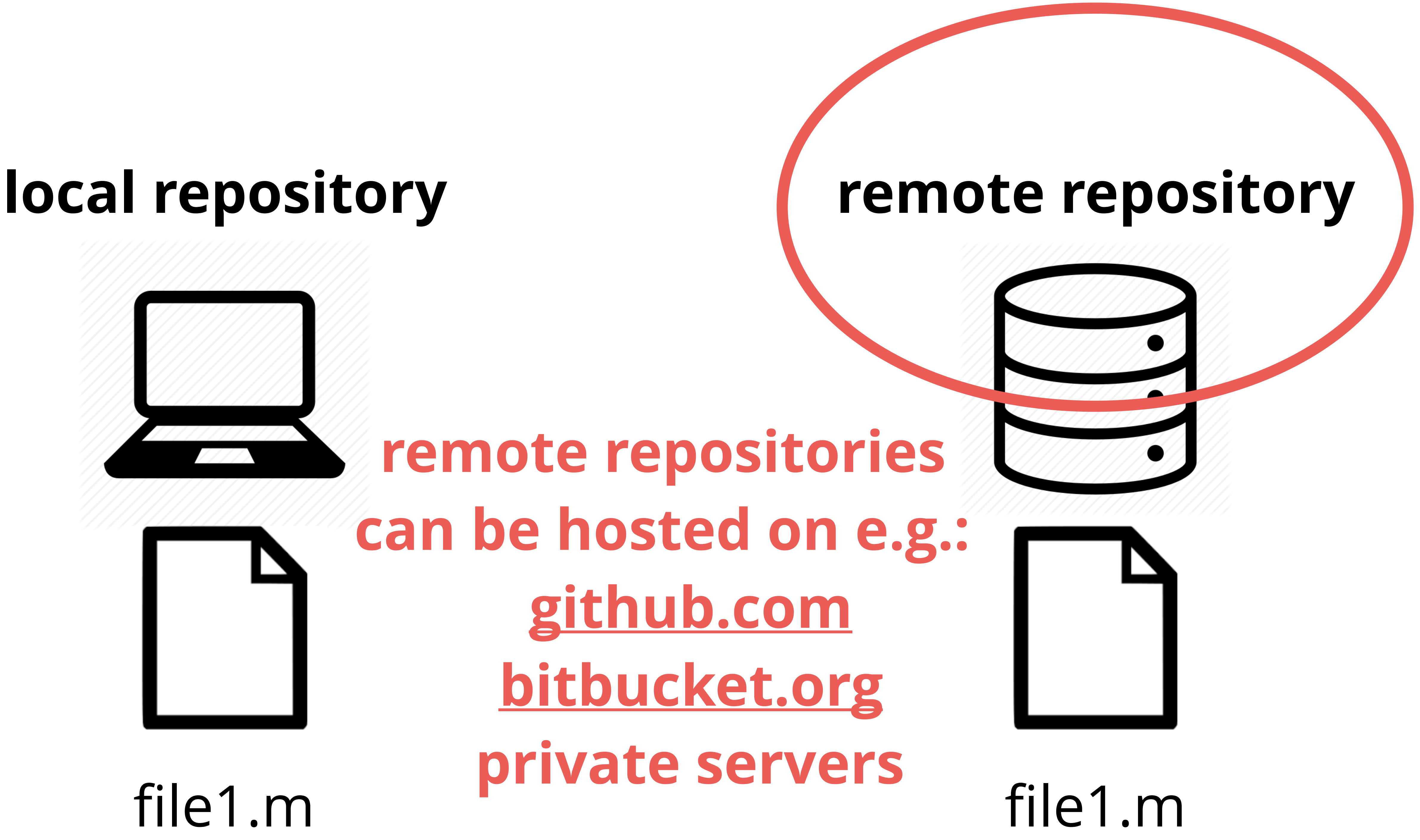
local repository

file1.m

remote repository

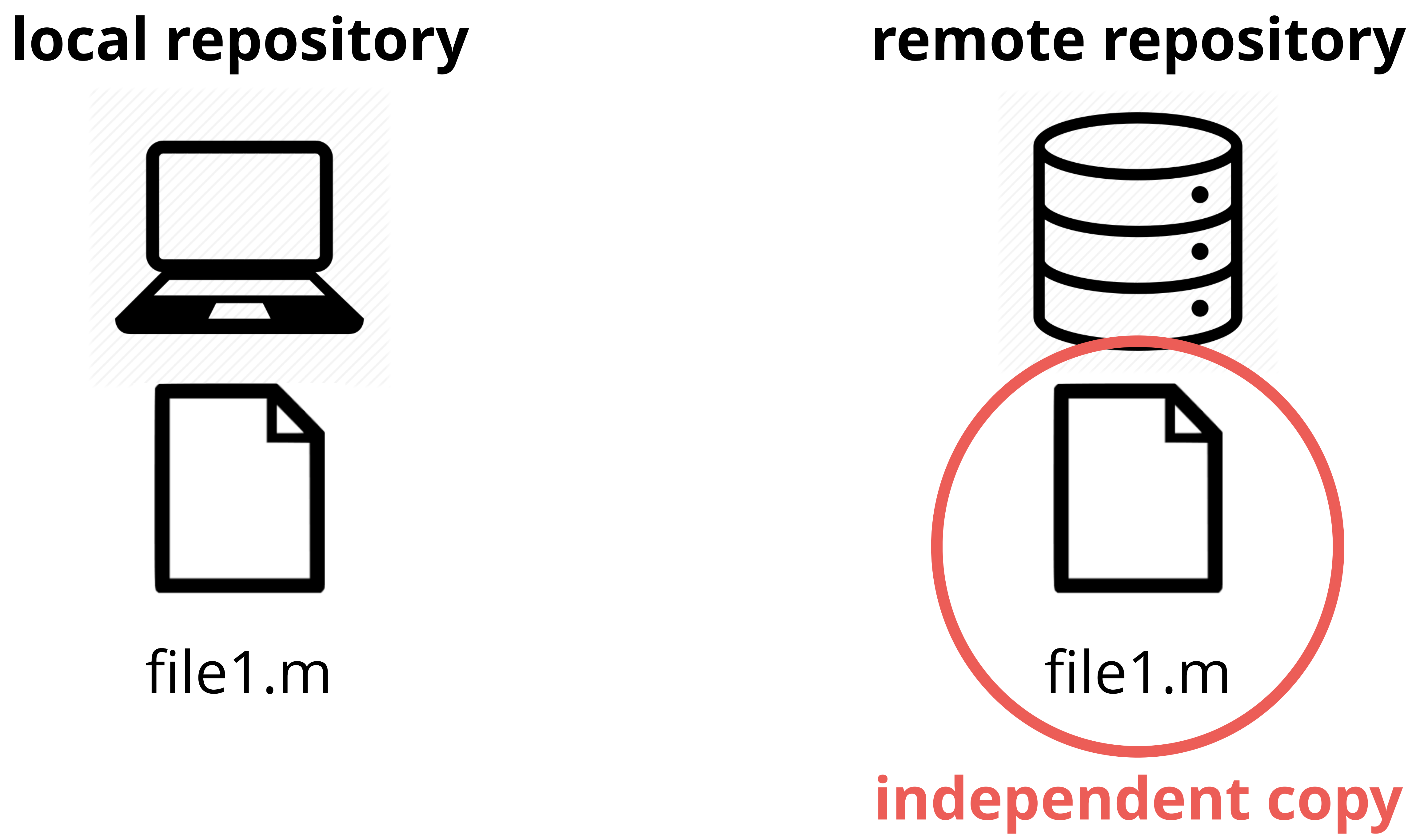
file1.m

Git Overview

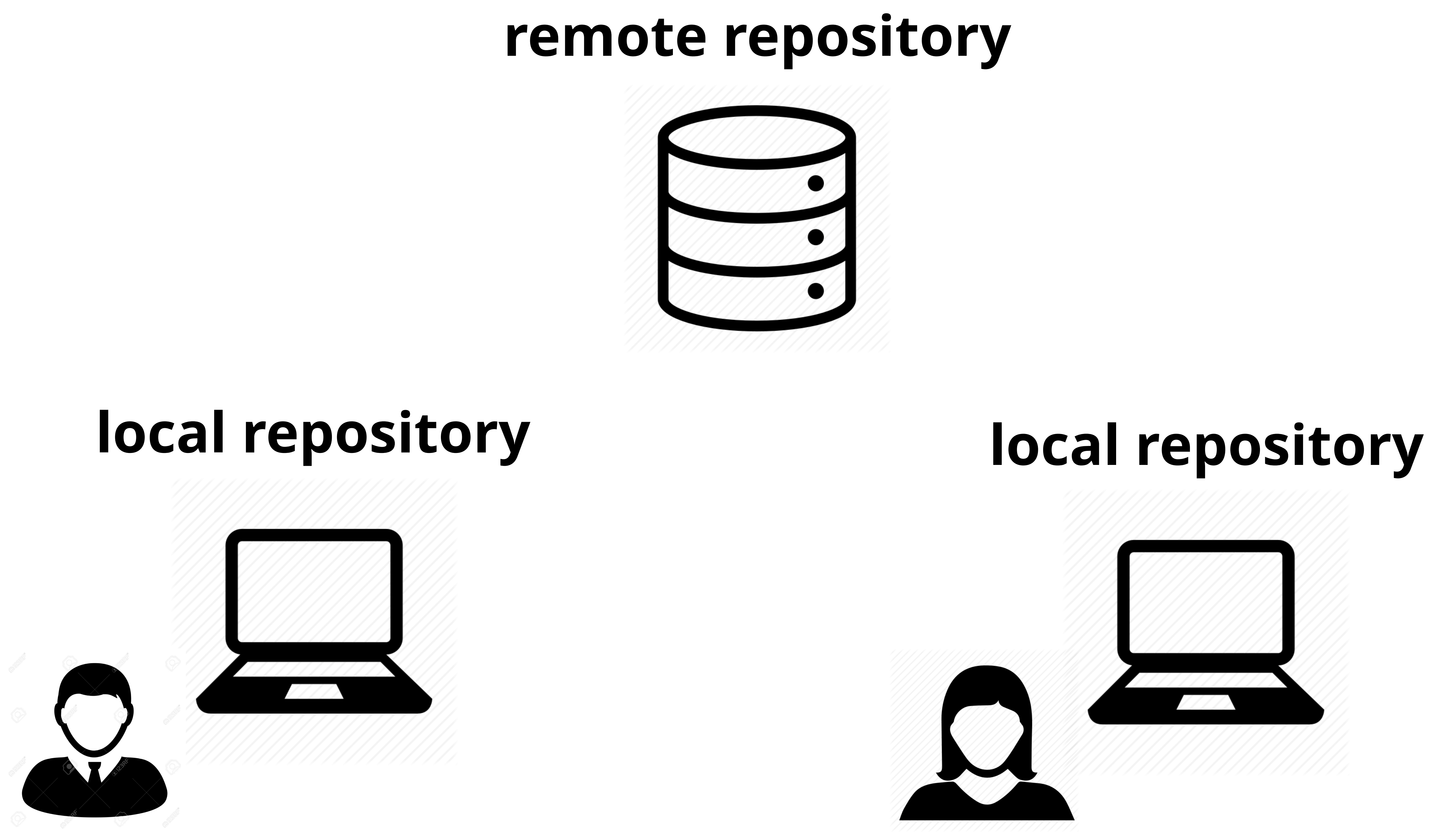


remote repositories
can be hosted on e.g.:
github.com
bitbucket.org
private servers

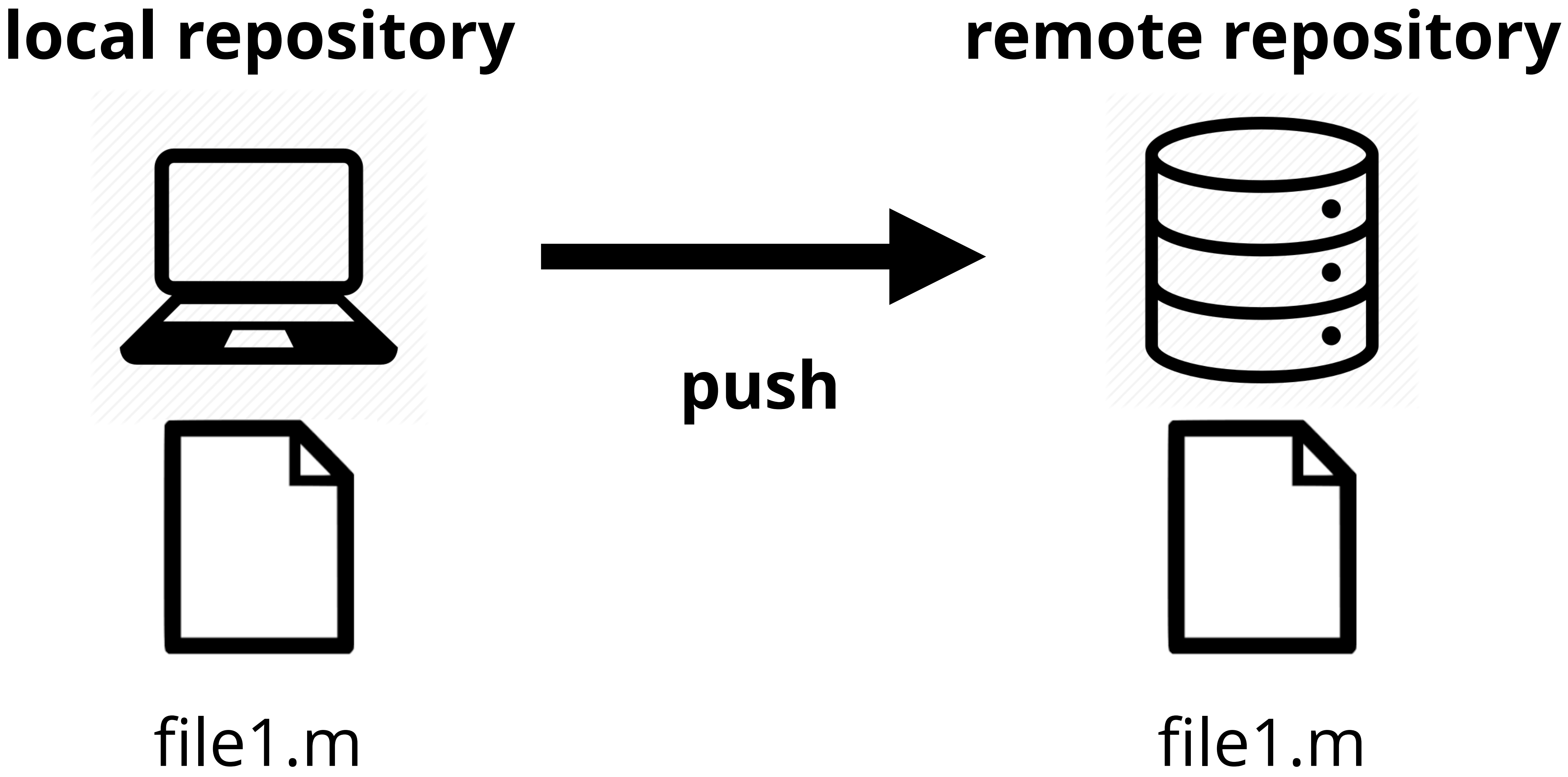
Git Overview



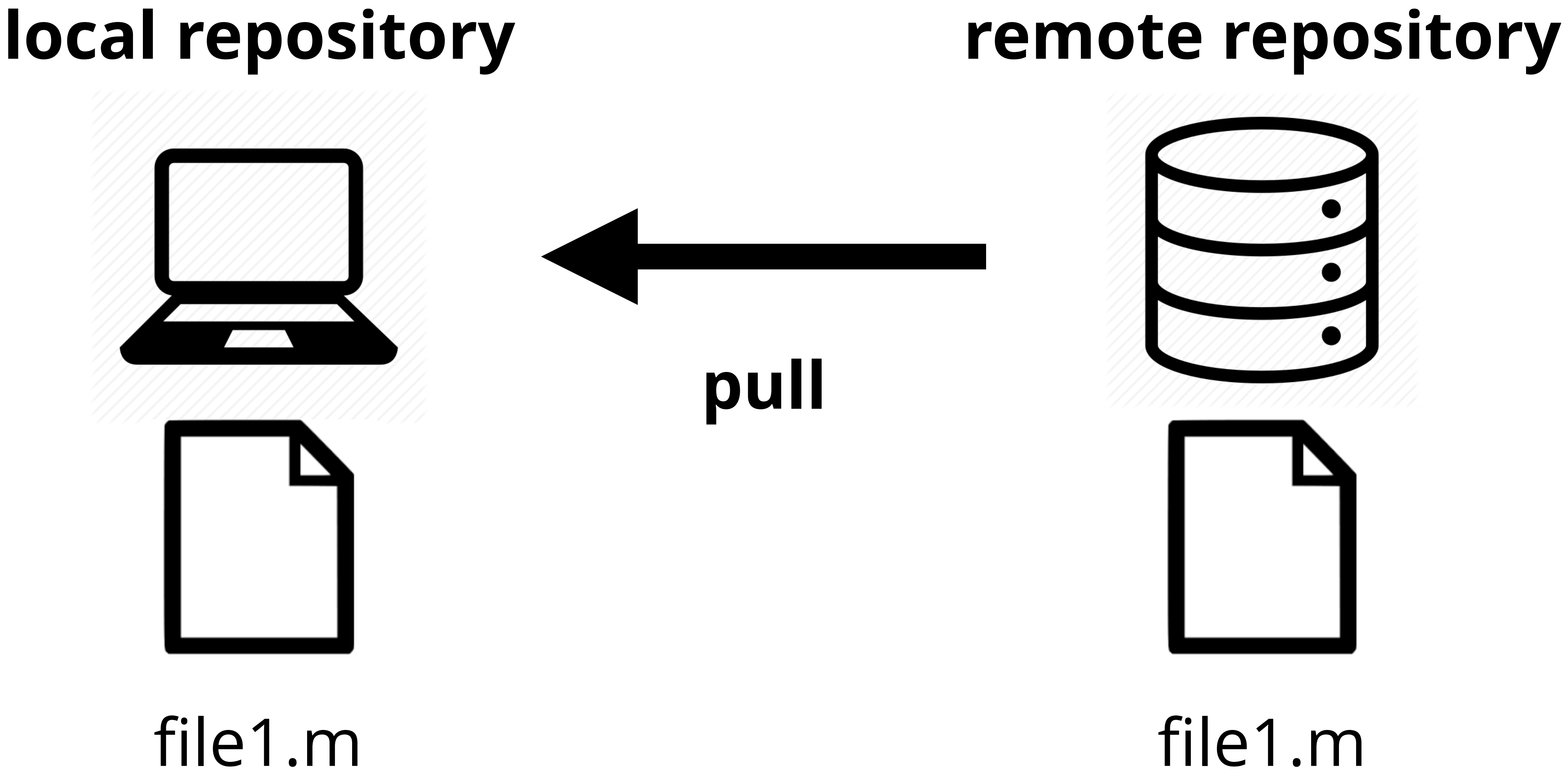
Decentralisation



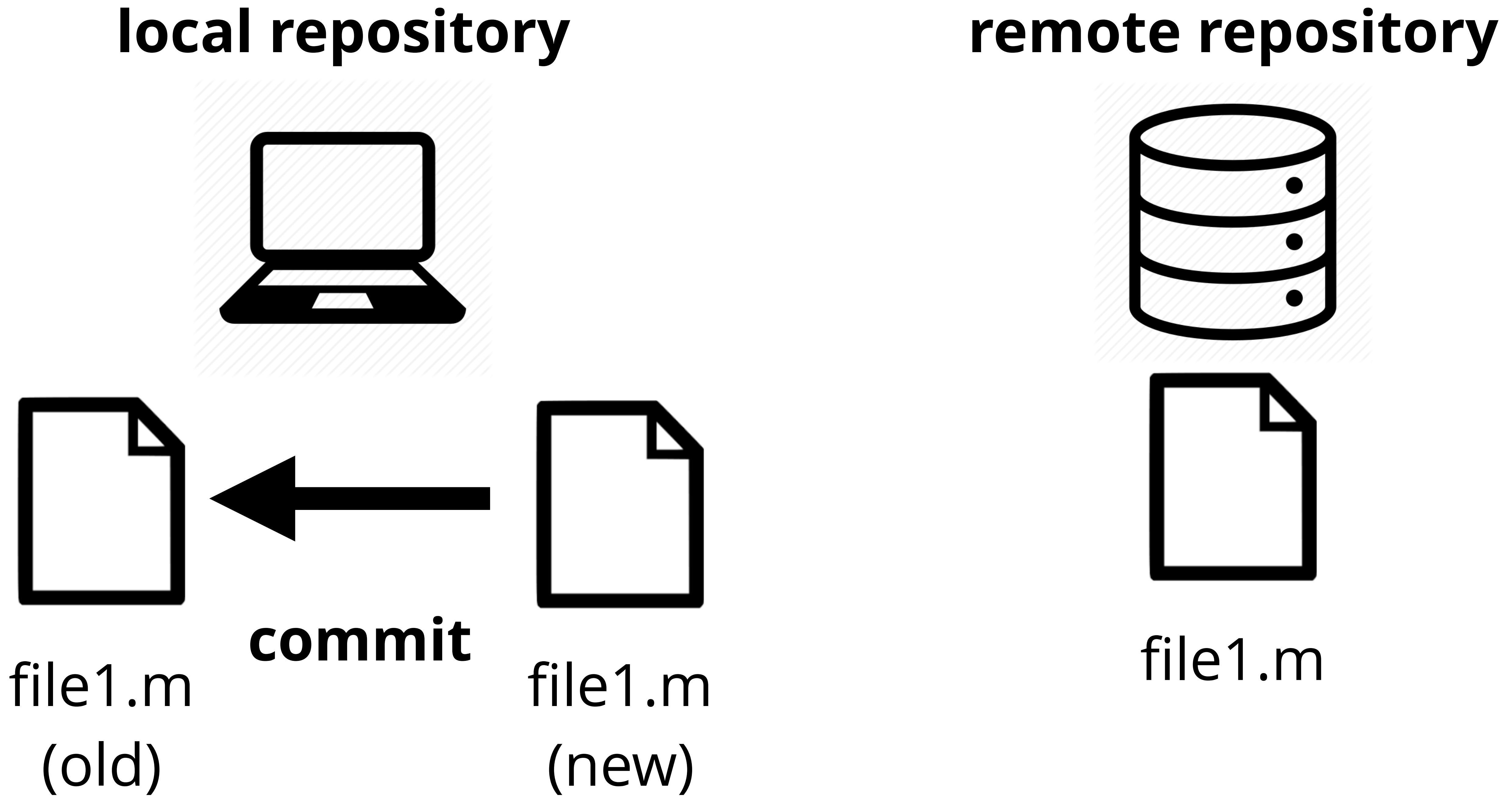
Git Overview



Git Overview

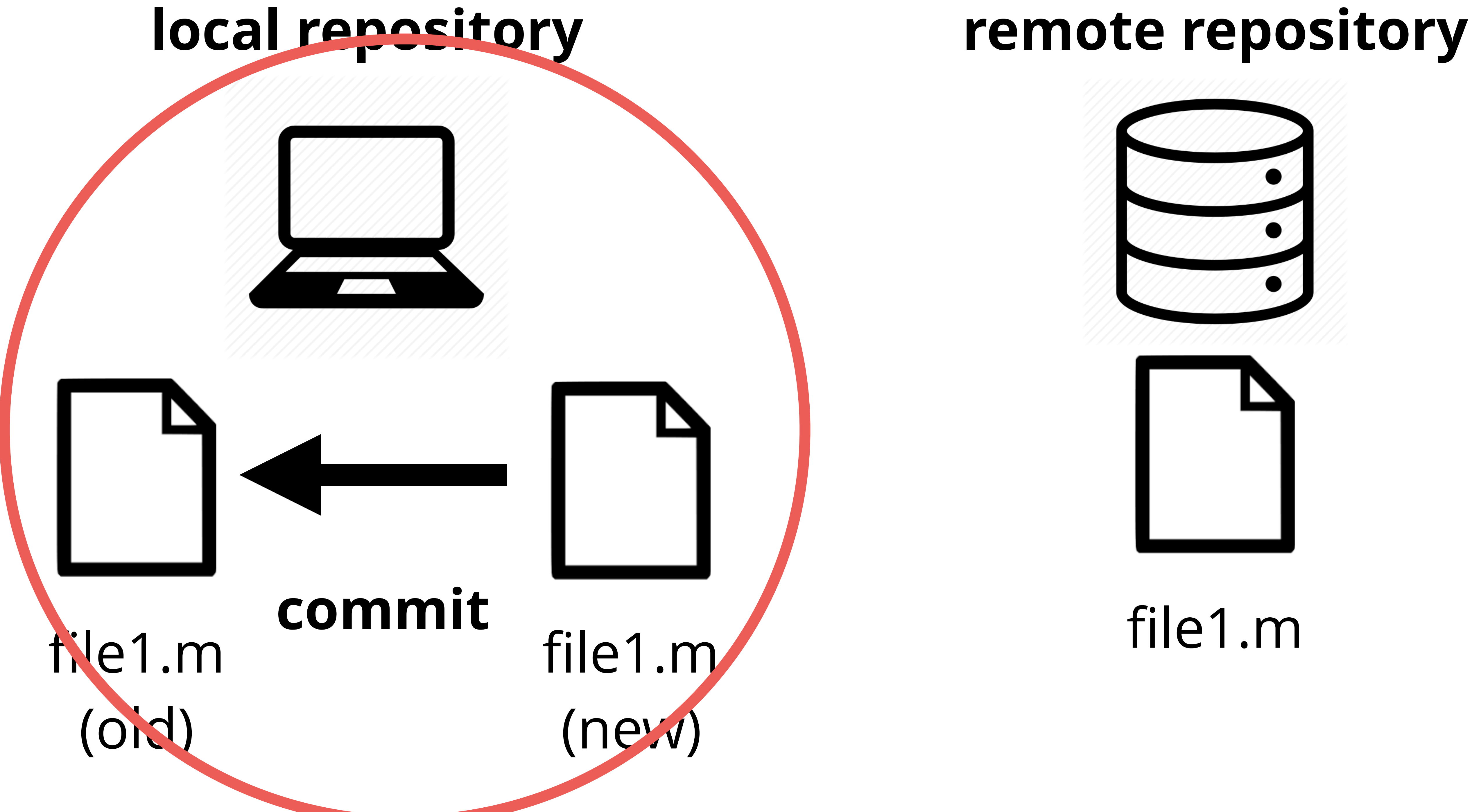


Git Overview



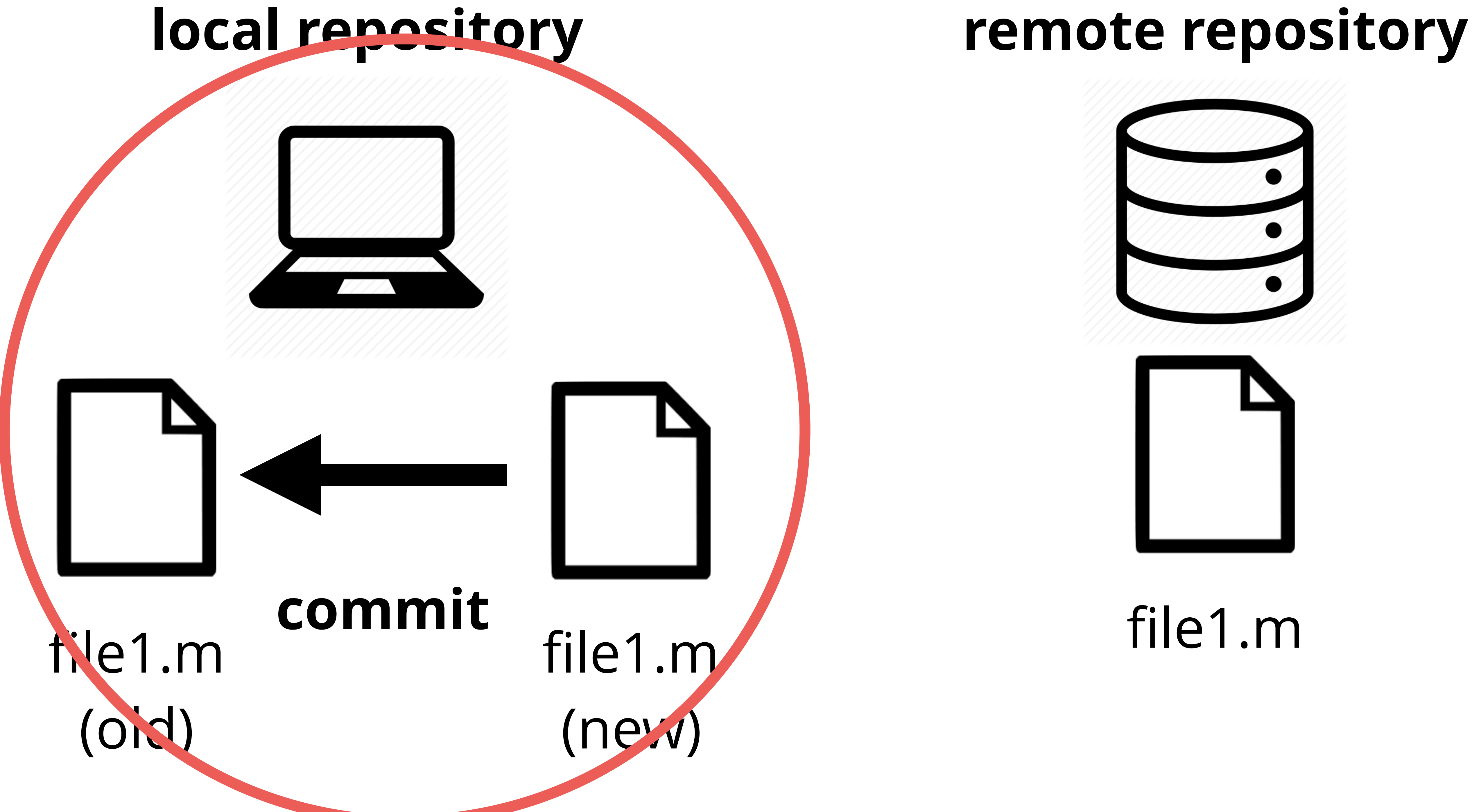
Git Overview

A commit alone does not affect the remote repository - must be coupled with a push.

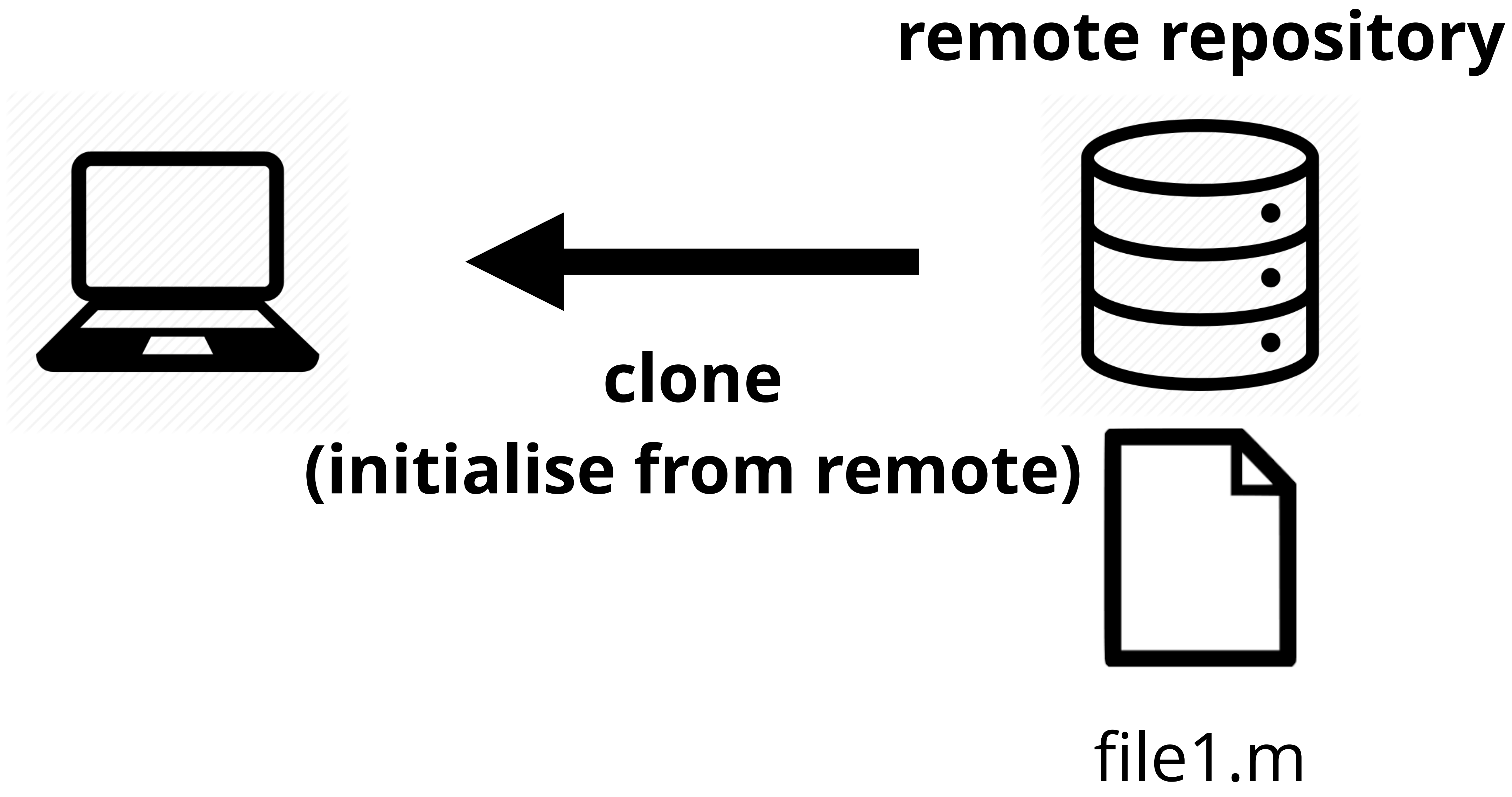


Git Overview

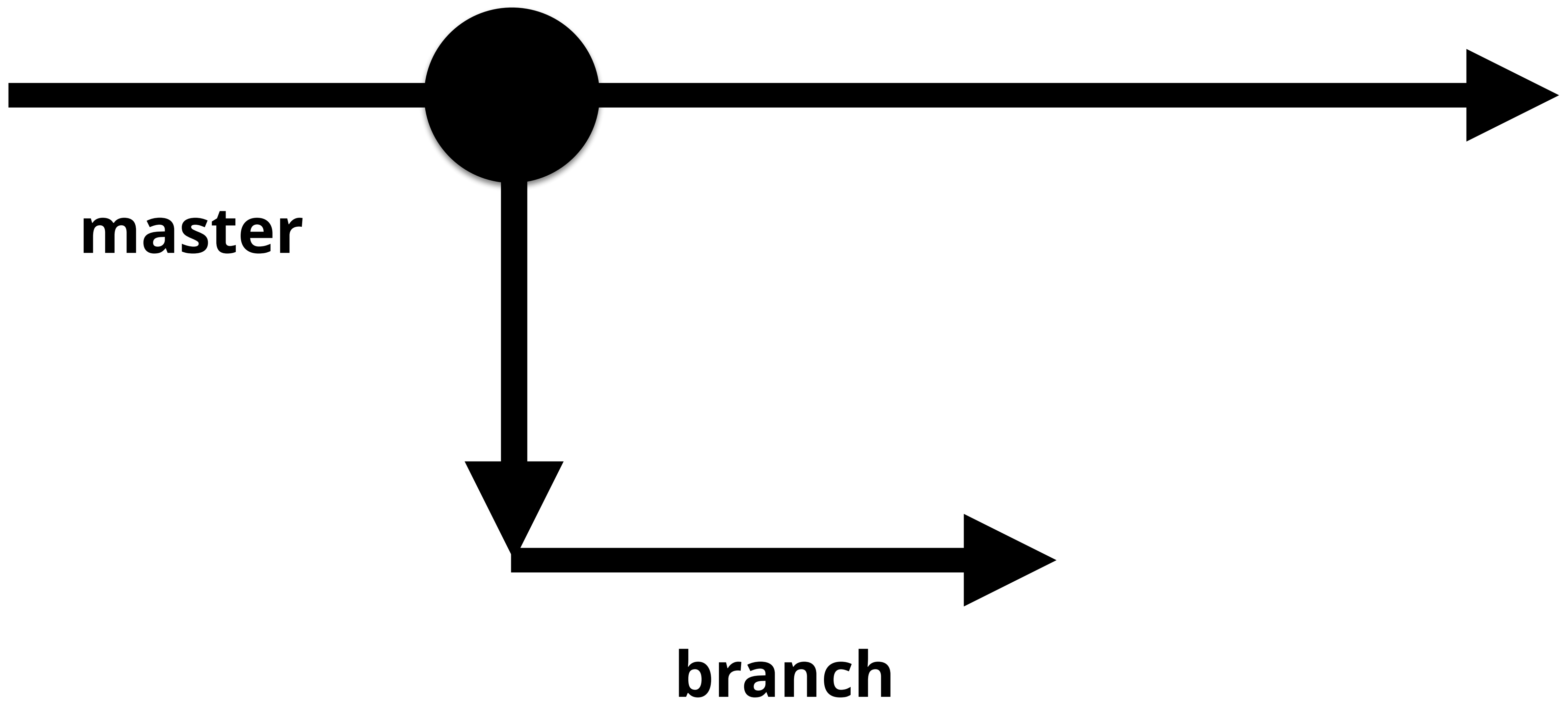
The file contents for every commit are saved in the history.



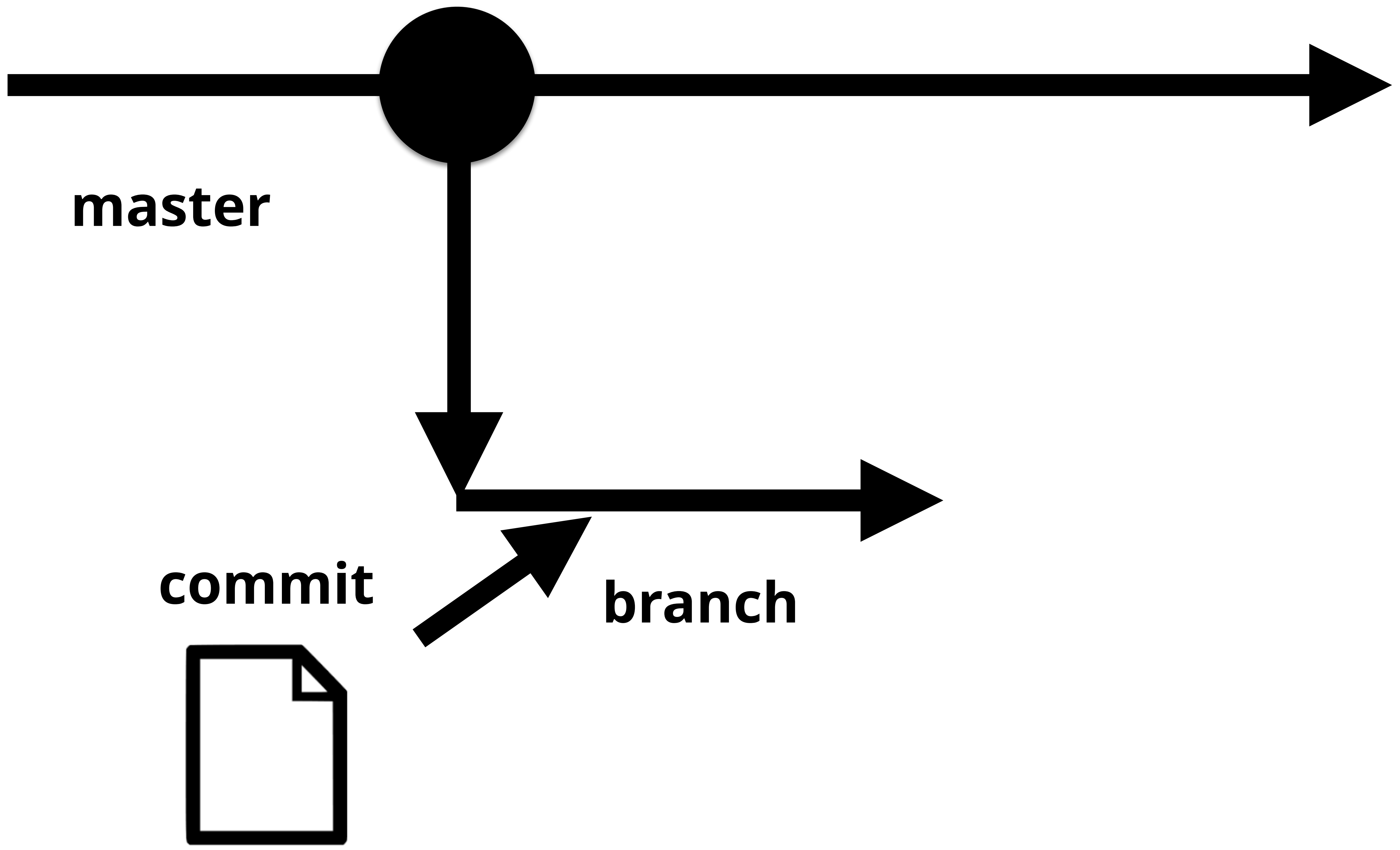
Git Overview



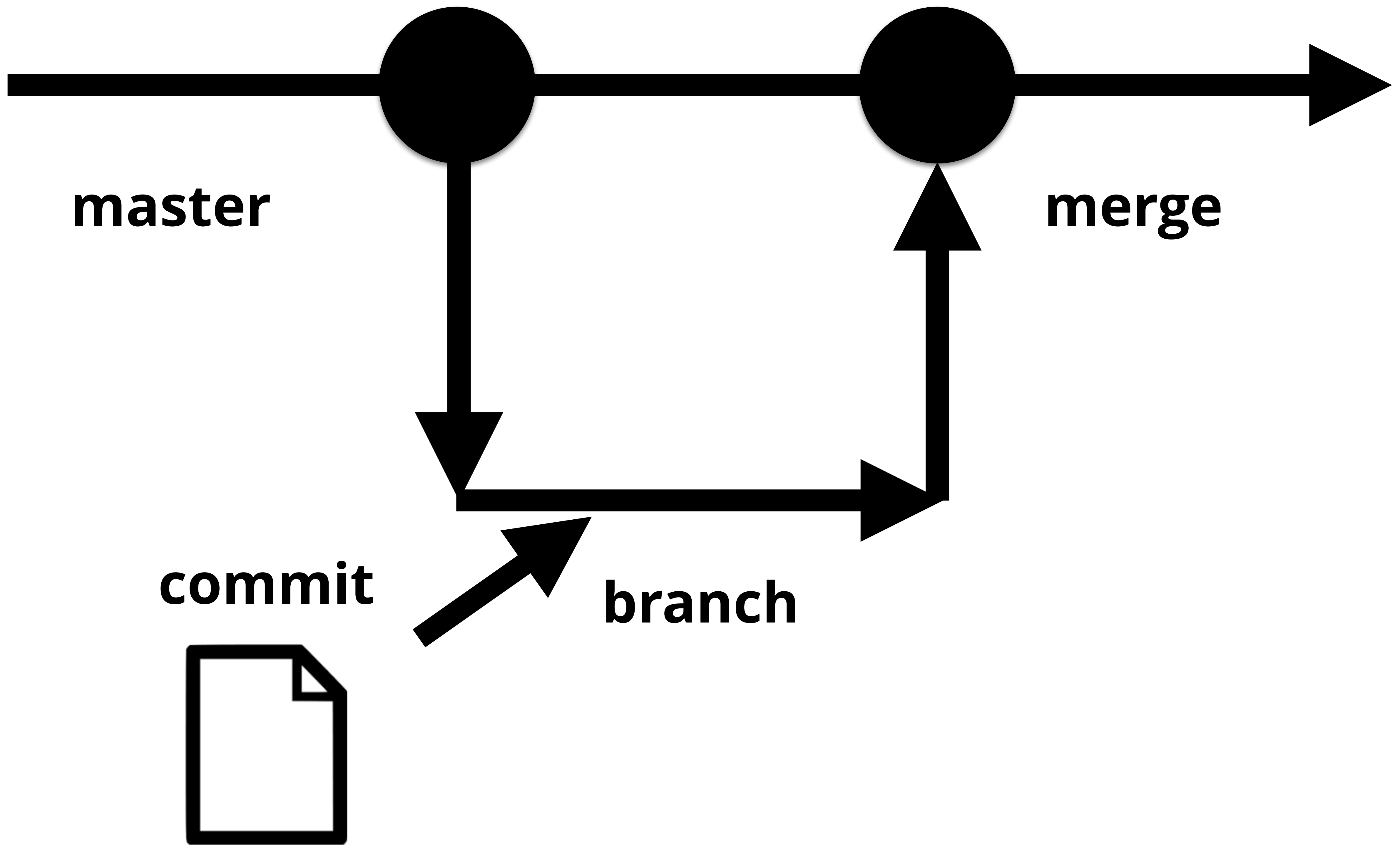
Branches



Branches



Branches



Questions?

Practice

1 - Creating a Git Repository

- **Create a new repository on bitbucket.org**
- **Initialise a new local repository on your machine**
- **Create a README.txt file with any content**
- **Add and commit the README.txt file with a meaningful message to the master branch**
- **Push your commit to the bitbucket.org repository**

FAQ

- **How often should I commit?**
  - **Every time you finish a significant feature**
  - **Should be multiple times (~4-6) a day**
- **What should be my commit message?**
  - **A brief and complete description of what you have changed.**
- **How often should I push?**
  - **Ideally, after every commit; otherwise: risk of loss**

FAQ

- **Which files should I commit?**
  - All source and configuration files necessary to reproduce your work
  - You should **not** include:
    - Dependencies and libraries (instead provide an installation command)
    - Binaries, intermediary results, or executables (instead provide a reproducible build command)
    - Source data (host on file storage instead and provide instructions on how to obtain)
  - Use a **.gitignore** file to ignore all of the above files in the local repository

2 - Branching

- **Create and switch to a new branch called test_branch in your repository**
- **Add and commit a new file file1.txt with content “Hello world” to the new branch in your local repository**
- **Merge the newly created branch into the master branch**

FAQ

- **Why should I have branches?**
  - **Code isolation**
    - **Minimise the number of conflicts by isolating code branches until they are fully implemented**
  - **Maintaining working code**
    - **Ability to return to switch between your work in progress and the latest working version without interruption or breaking existing code**

FAQ

- **When should I create a new branch?**
  - **A good practice is to have:**
    - **One branch that contains the latest working release of your code (typically the master branch)**
    - **One branch that contains development code that will later be merged into a release (typically called “develop”)**
    - **A number of other branches for individual features (called feature branches) to be merged into the development branch**

3 - Conflicts and Conflict Resolution

- Switch back to the test_branch
  - Change the content of file1.txt to “Hello git”
  - Commit file1.txt
- Switch back to the master branch
  - Create a new branch mhsl_branch
  - Change the contents of file1.txt to “Hello mhsl”
- Switch back to the master branch and then merge first the test_branch and then the mhsl_branch into master
- View file1.txt, resolve the conflict, and commit the result to the master branch.

FAQ

- **How can I avoid regular conflicts?**
  - **Use branching to isolate your new code features**
  - **Write modular code instead of a single function / class with all features**
    - **Most changes can be merged automatically by git - only changes in the same line and file lead to conflicts**
  - **Coordinate with your team to not rewrite the same modules concurrently**

4 - Restoring Old Versions

- **Suppose you preferred the first version of file1.txt**
- **View the commit history for file1.txt using git log**
- **Revert file1.txt to its first version**
- **Commit your changes to the master branch**

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Big Data
National Research Programme

Questions?

Patrick Schwab

@schwabpa

patrick.schwab@hest.ethz.ch
Institute for Robotics and Intelligent Systems
ETH Zurich