# 00 Lab
# PPS Lab Introduction and Setup

Mirko Viroli, Roberto Casadei, Gianluca Aguzzi
{mirko.viroli,roby.casadei,gianluca.aguzzi}@unibo.it

C.D.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2024/2025

# On how to follow lab lessons

- Students will receive in advance the full material related to each lab lesson (slide and code)
- The teacher and tutor will be in lab for its entire duration
  - exercise assignation: students will try to solve them
- Students can require dedicated interactions with the teacher and the tutor
  - to discuss and receive feedback on proposed solutions, ...
- This first lesson is dedicated to the setup of the development environment,
  - So please configure both git and IntelliJ Idea with Scala Plugin to be ready for the next lessons

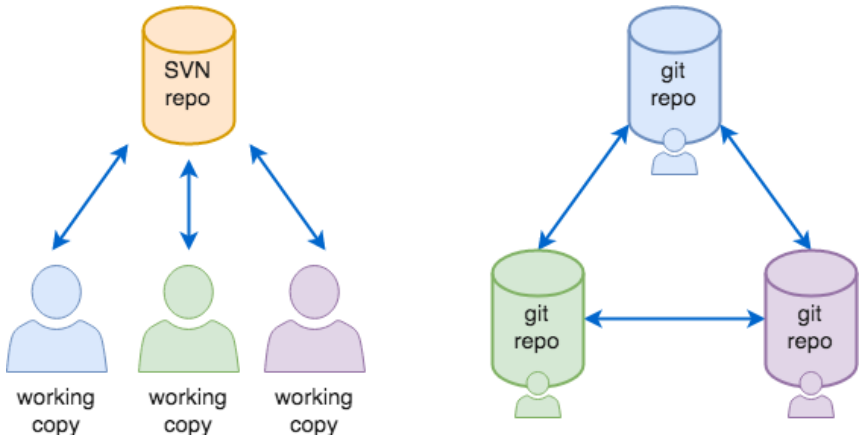
# Outline

- A brief recall on DVCS and Git
- Setting up IntelliJ Idea (and Scala Plugin)

# git: brief intro (i)

## Basics on version control

- **Version Control System (VCS)**: system for the management of "versions" of data
- **Repository**: stores a set of files and the history of their versions (or the history changes made to them)
- **Centralized VCSs**: single remote repo + working copies
  - A **working copy** is a local copy of the repo
- **Distributed VCSs**: each developer has a copy of the entire repository



# git: brief intro (ii)

## Basics on git

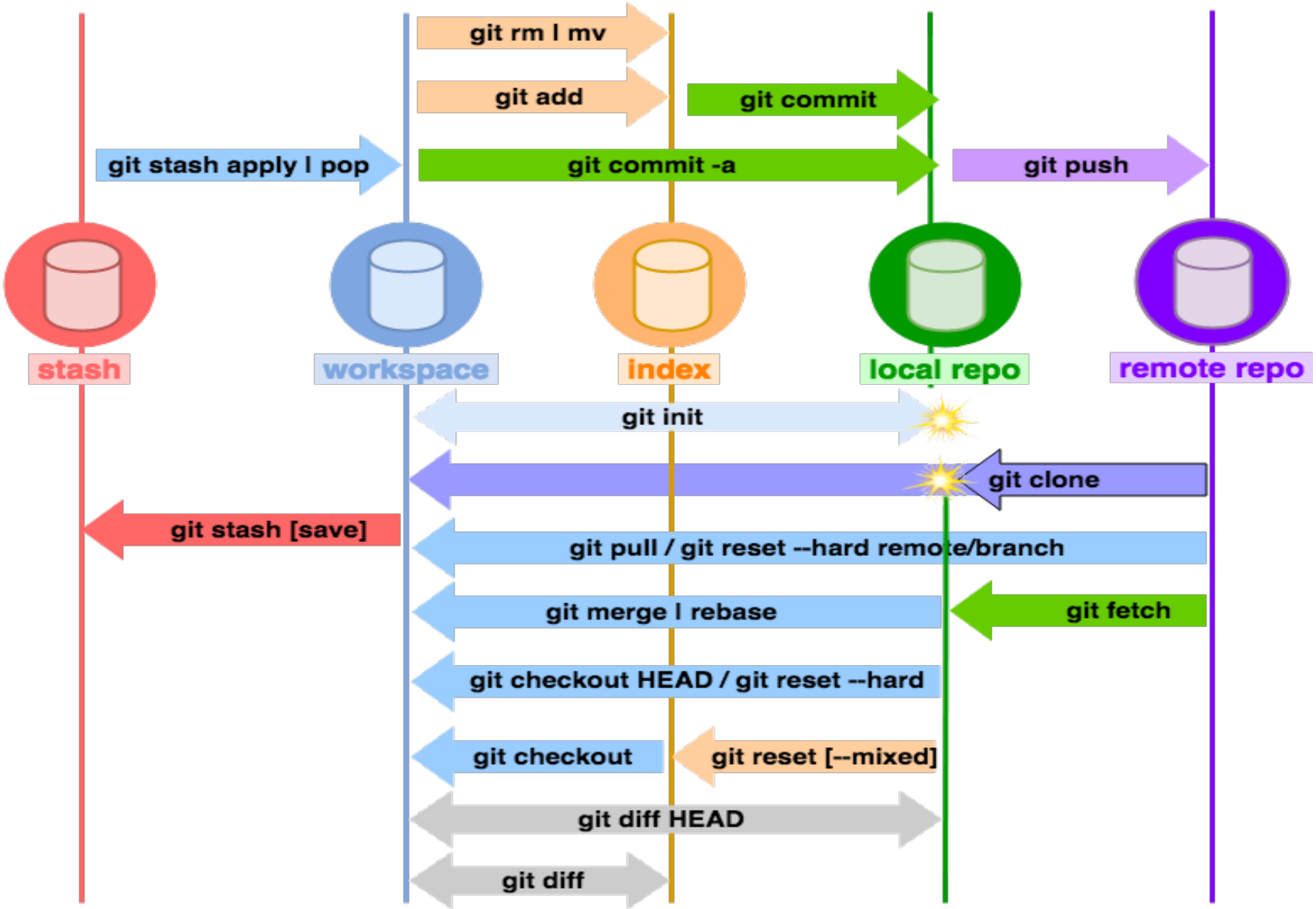
- git thinks of data as a stream of **snapshots** (rather than a list of file-based changes)
  - But does use **deltas** for storage!
- **Git directory**: stores metadata and object DB (commits, branches..)
- **Working tree**: a filesystem tree containing a single **checkout** of one version of the project (+ local changes)
- **Staging area** (aka **index**): keeps track of the snapshot (actually, delta w.r.t. HEAD) that will go into the next commit
- Files in working tree can be in two states: **tracked** or **untracked**
- Tracked files can be: **unmodified** or **modified** and/or **staged**
- **Branch**: independent line of development; new commits are recorded in the history of current branch
- **HEAD**: pointer to the current snapshot (branch or commit)

# git: brief intro (iii)

## Key commands

- **Init**: initialize a new git repository
- **Clone**: downloads a project and its entire version history
- **Status**: shows untracked files, modified files, staged files
- **Add**: propose changes by adding them as a snapshot to the **index** (the staging area); also resolves merge-conflicted files
- **Diff**: shows difference between a commit and the working tree, or between the working tree and the index
- **Commit (check in)**: the changes are committed to the HEAD
- **Push**: send local changes to another (possibly remote) repository
- **Checkout**: updates the working tree by switching to a branch/commit
- **Branch**: manages branches (list, create, delete, ..)
- **Fetch**: download objects and references from another repository
- **Pull**: incorporates changes from a remote repository into the current branch
- **Push**: update remote references along with associated objects
- **Merge**: incorporates changes from a branch into the current branch

# git: a pictorial view



# git: workflows (i)

## Simple workflow

- Create and/or clone a remote repository
- Create or select a branch of choice
- Create and/or modify files in the working tree
- Stage your changes
- Commit your changes
- When you are done, merge your branch into the main branch (master)
  - If any conflict, resolve them, add (mark) them as resolved and commit
- Push your changes to the remote
  - May fail if someone pushed new stuff; in this case, pull and incorporate changes before retrying the push.

# git: workflows (ii)

# Workflow for practice in lab (i)

## Introduce yourself to git

- `$ git config --global user.name "Name"`
- `$ git config --global user.email your@email.com`

## Lab workflow

- We give you a remote repository with code for lab
  - e.g., `https://github.com/unibo-pps/pps-lab00`
- If it is the first time, you clone the repo
  - `$ git clone git@github.com:unibo-pps/pps-24-25-lab00.git`

  otherwise you pull from the remote
  - `$ git pull origin` (origin: default remote name for a cloned repo)
- Create or modify files in the working tree, stage your changes, and commit them
  - `$ git add .`
  - `$ git commit -m "Commit msg"`

# On forks (i)

You may want to keep your own remote repo and push changes to it

## Forking

a) Create a new, empty remote repo

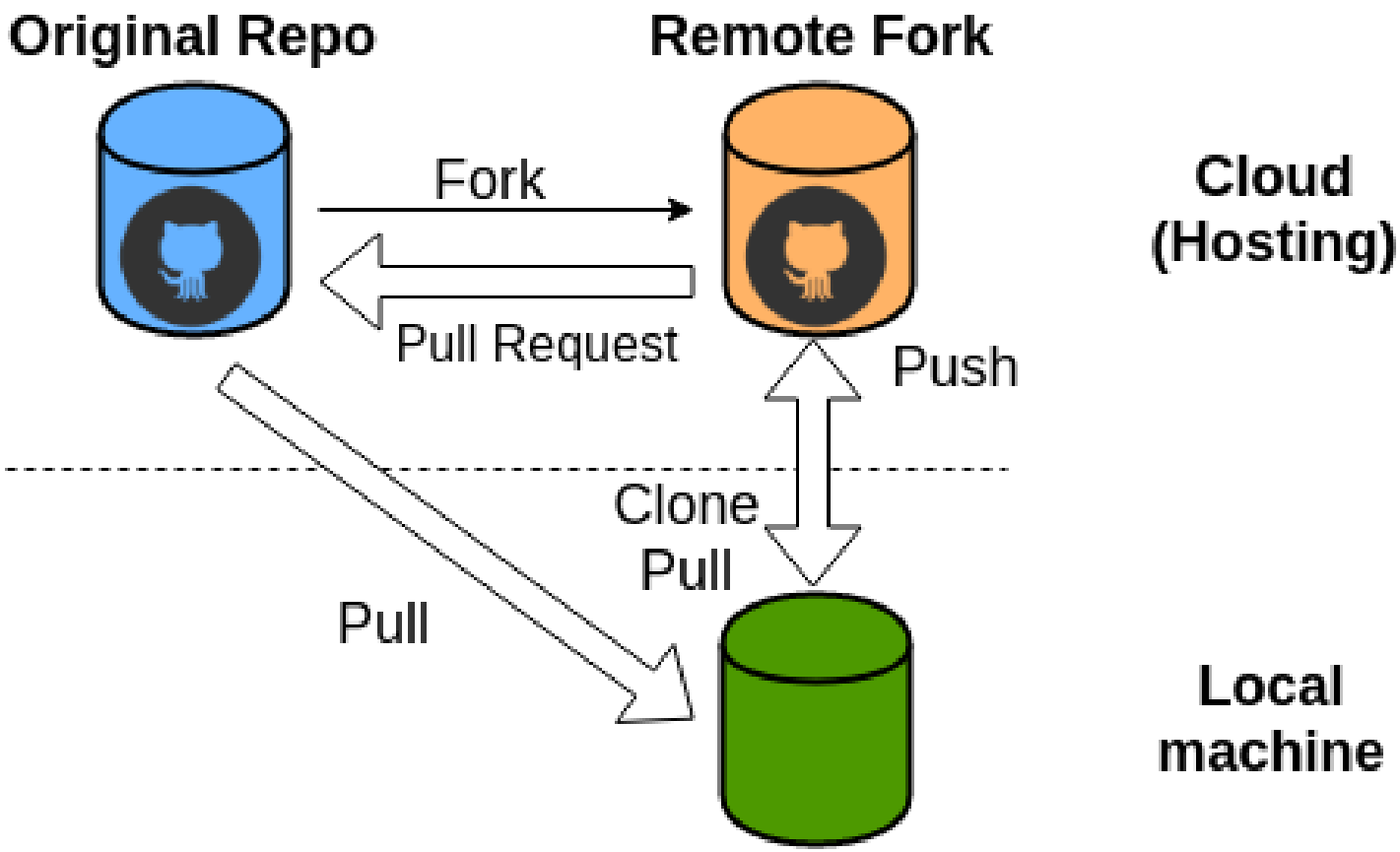
b) Explicitly create a fork (e.g., on GitHub or Bitbucket)
   - Preferred, as this supports *pull requests*

- Clone locally your own remote fork
- Add the remote referring to the original repo:

  `$ git remote add sync <originalRepo>`
- Sync with the original repo `$ git pull sync master`
- Set up a local branch to track the original one (for pulls)

  `$ git branch originalMaster --track sync/master`
- Push into the forked repo and track it `$ git push -u origin master`
- Use `sync` to fetch from the original repo
- Use `origin` to pull and push to your forked repo

# On forks (ii)


Original Repo
Remote Fork
Fork
Pull Request
Push
Cloud
(Hosting)
Clone
Pull
Pull
Local
machine

# Setting up the development environment

## To install

- git
  - `https://git-scm.com/downloads`
  - more on how to use git: `https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup`
- JDK 21 (suggested)
  - installation using adoptium `https://adoptium.net/`
- IntelliJ IDEA Community Edition 2024.3
  - `https://www.jetbrains.com/idea/download/?section=linux`
- Scala Plugin for IntelliJ IDEA
  - `https://www.jetbrains.com/help/idea/discover-intellij-idea-for-scala.html`
- Scala 3.3
  - `https://www.scala-lang.org/download/scala3.html`

# First steps in IntelliJ IDEA (i)

## Key notions in IntelliJ

- Project settings: `https://www.jetbrains.com/help/idea/`
- Core concepts:
  - **Project**: Complete software solution with settings in `.idea` folder
  - **Module**: Independent unit within a project that can be built/run separately
  - **SDK**: Required development kit (e.g., Java SDK) configured via `File | Project Structure`
- Version control: Do not include any IDE-specific files in the repo
  - **.idea** folder: IDE settings
  - **.iml** files: module settings

# Lab Setup

- Clone (or fork and clone) the repository at `https://github.com/unibo-pps/pps-lab00`
- Open the project in IntelliJ Idea
  - File — Open — select the folder of the cloned project
- It should correctly import the project and set up the SDK
  - We mainly leverage external building tools (sbt, maven, gradle) for building and running the project, reducing the need for IDE-specific settings

# Setup Exercise

1. Analyse the proposed code to understand the application logic
2. Run the application by clicking Run near the main method in `src/main/java/Main.java`
3. Try to modify/refactor some parts of the proposed code to become familiar with the IDE
4. Run the proposed test suite (see `test/model/PersonTest.java`) by the green arrow near the class name
   - Run all tests using the green arrow near in the class name
5. Try to add other simple tests for the application