Git

Crash Course

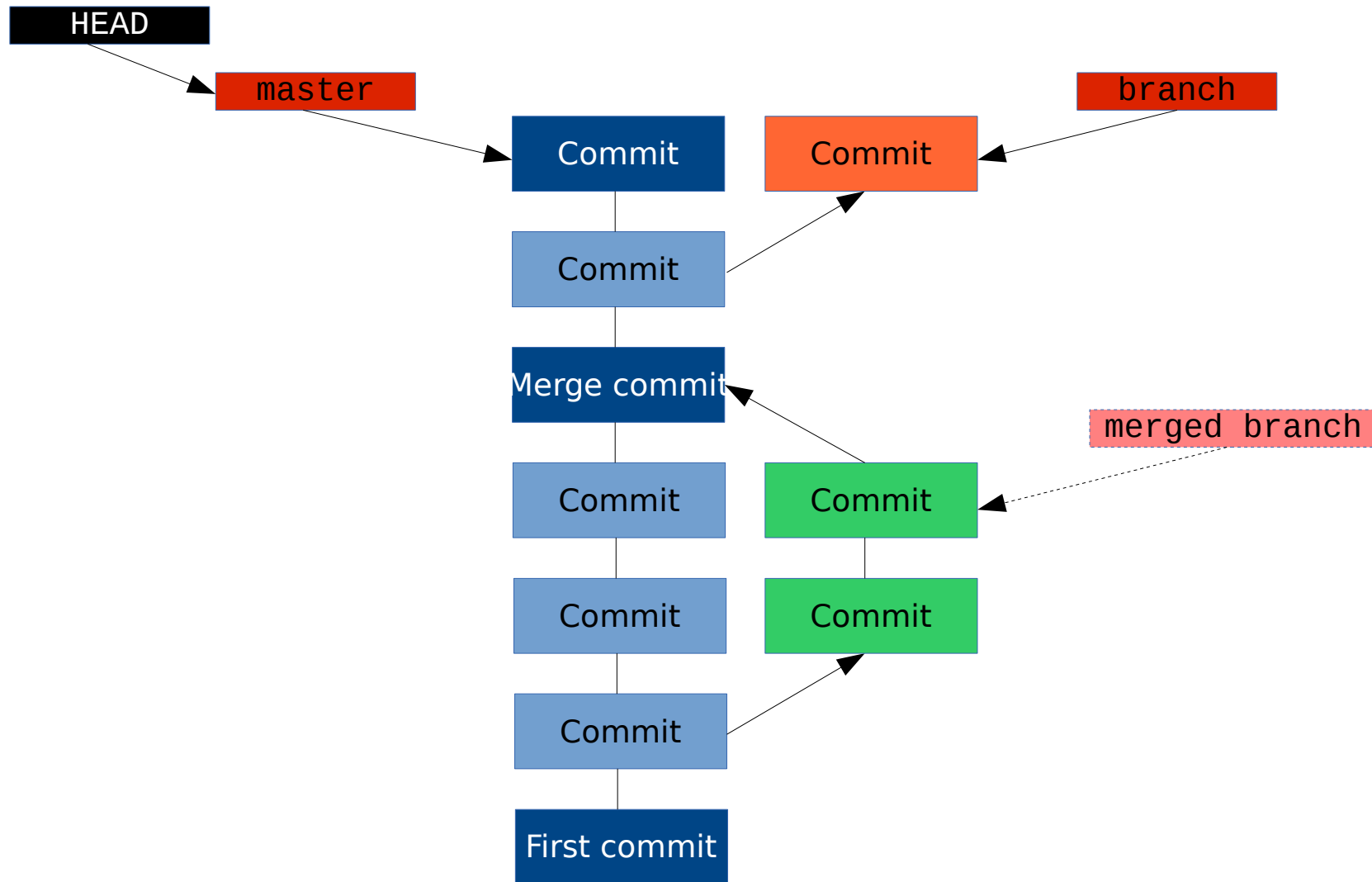## Ugo Pattacini

based on course by
D.Domenichelli

Version Control

- System that records changes to files.

- Allows you to restore a specific versions later.

- A better option than copying files with a different name or into another directory.

Git Design

- Fully distributed
- Fast
- Strong support for non-linear development
- Able to handle large projects
- Saves Snapshots, not patches
- Cryptographic Authentication of history
- Is not optimized to handle binary files (but there are extensions for this).

Git Repository History Example

HEAD

master

branch

Commit

Commit

Commit

Merge commit

merged branch

Commit

Commit

Commit

Commit

Commit

Commit

First commit

- "HEAD" is a reference to the commit that you checked out in your working tree
- A "merge commit" has two parents, the "first commit" has none.

LOCAL WORKFLOW
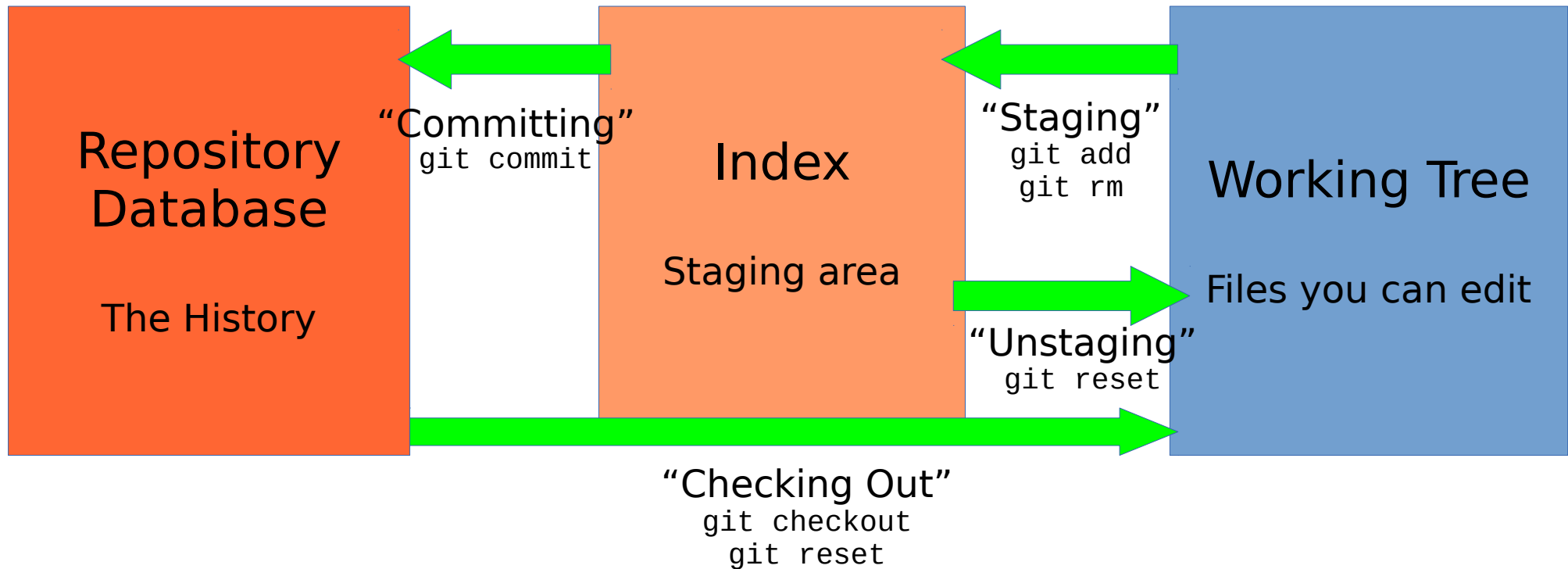
Git Repository Structure

Repository Database	Index	Working Tree
The History	Staging area	Files you can edit

Local Workflow

Repository Database

The History

"Committing"
`git commit`

Index

Staging area

"Staging"
`git add`
`git rm`

"Unstaging"
`git reset`

Working Tree

Files you can edit

"Checking Out"
`git checkout`
`git reset`

Find out more on this nice resource: https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified

Identity

- The first thing you should do is to set your user name and e-mail address.

  ```
  git config --global user.name "John Doe"

  git config --global user.email "john.doe@example.com"
  ```

Creating a git repository

- mkdir <folder>

- cd <folder>

- git init

- Now you have a fully working git repository.

- Don't forget that for now it's just on your computer.

Status

- `git status`

  - Shows staged, unstaged and untracked files
  - Do not be afraid to use it after every command
  - Better with colours enabled

- Now try to create a file and see what changes in the output of the command.

  - `touch README.md`
  - `git status`

Staging and Unstaging

- git add <file>

- git add .

- git reset <file>

- git reset

  - Nothing was committed yet!

  - The "index" is updated

  - Check what changes after each command using "git status"

- These commands work on the stage area, but also on the working tree:

  - git rm <file>

  - git mv <old> <new>
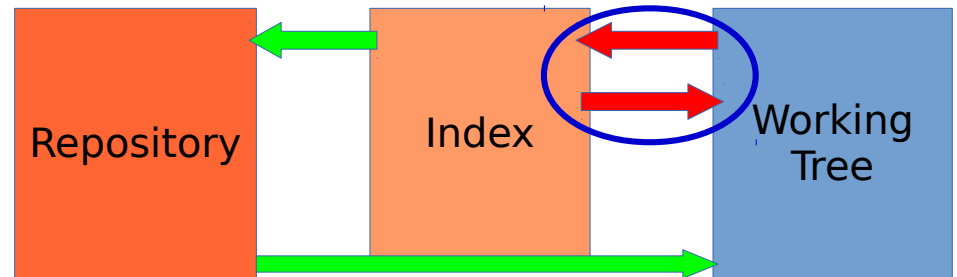
Committing

- Staged files only

  - `git commit -m "Log message"`

- Commit on a shared machine, where your name and email address are not configured

  - `git commit --author="John Doe <john.doe@example.com>"`
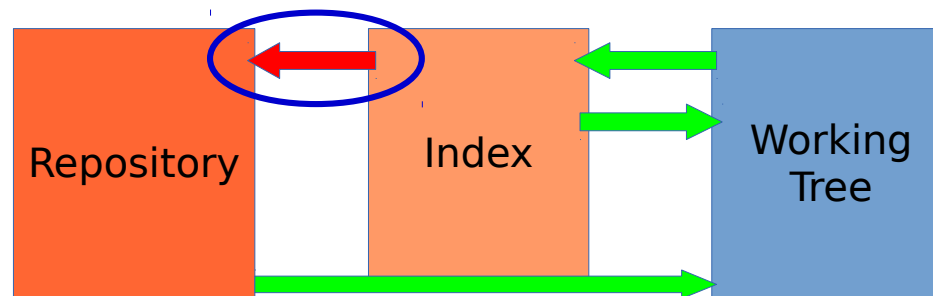
Log

- git log

  - Shows the log starting from HEAD

- git log <revision>

  - Shows the log starting from <revision>

- git log [--] path

  - Limits the log to the changes to a file or a directory

Diff

- `git diff`
  - Shows diff between index and working dir
- `git diff --staged`
  - Shows diff between HEAD and index
- `git diff <object>`
  - Shows diff between object and working dir
- `git diff <object> <object>`
  - Shows diff between two objects

Checking Out a Commit or a Branch

- git checkout <commit>

  - If you have some changes to files that would be overwritten when switching branch, the operation fails, and does not try to do anything
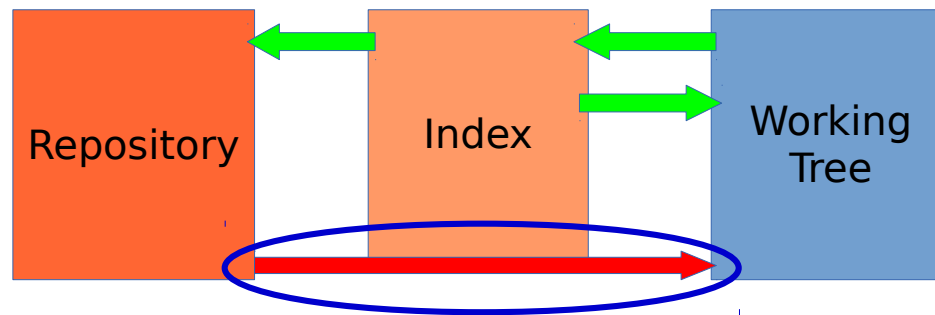
Branch

- `git branch (-l)`

    - Shows local branches only

- `git branch -r`

    - Shows remote branches only

- `git branch -a`

    - Shows all branches (local and remote)

Create local branches

- `git branch new-branch <commit>`

  - Create a branch "new-branch" on HEAD or on specified commit. Does not do a checkout of that branch

Merge

- `git merge <branch>`

  - Create a merge commit with multiple parents.
  - If it results in a conflict, user intervention is required

Rebase

- WARNING: Rewrites the history!

- `git rebase <commit>`
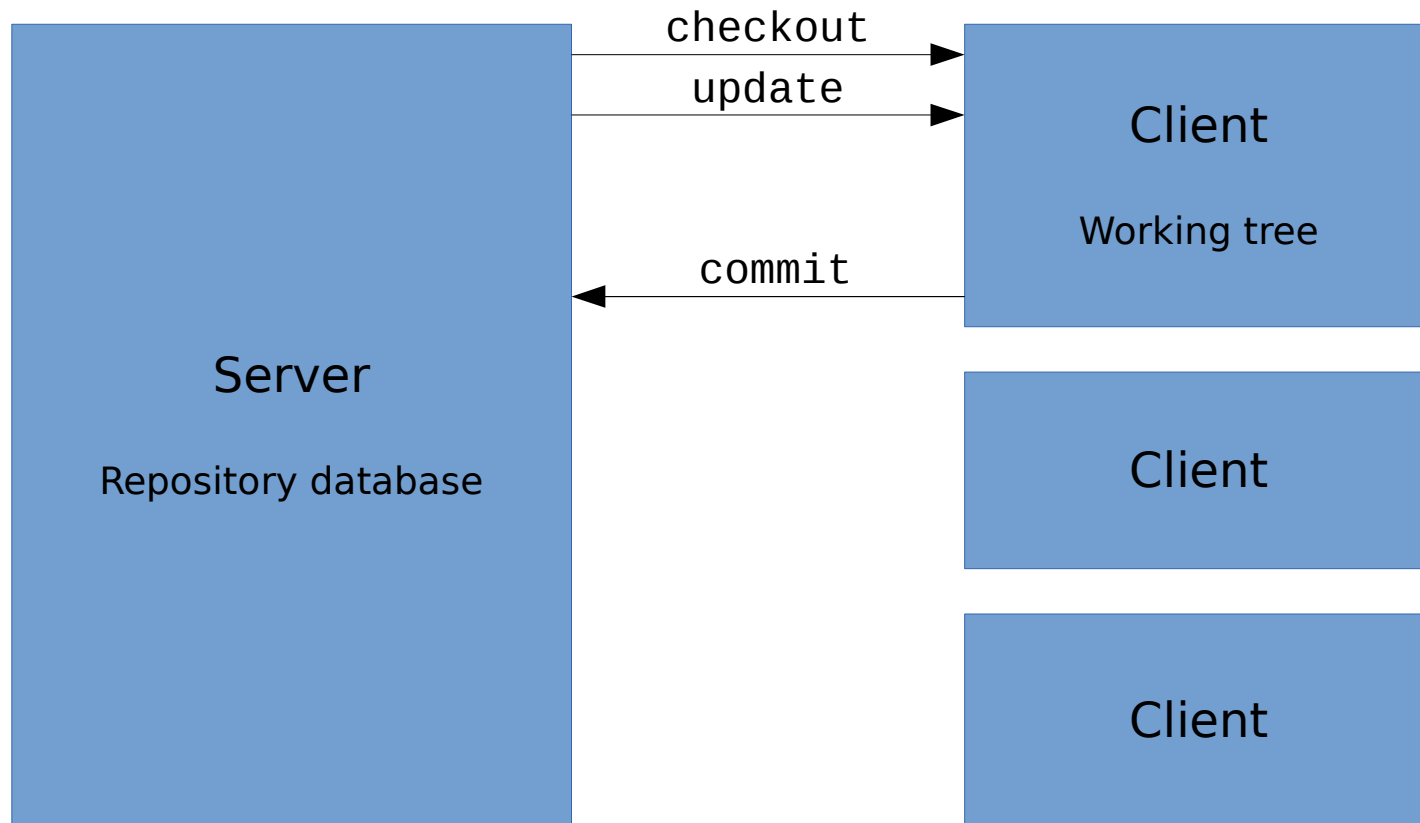
  - Takes all your commits and applies them onto <commit>

Revert

- `git revert <commit>`
  - Applies the reversed patch from <commit> to the repository and creates a new commit
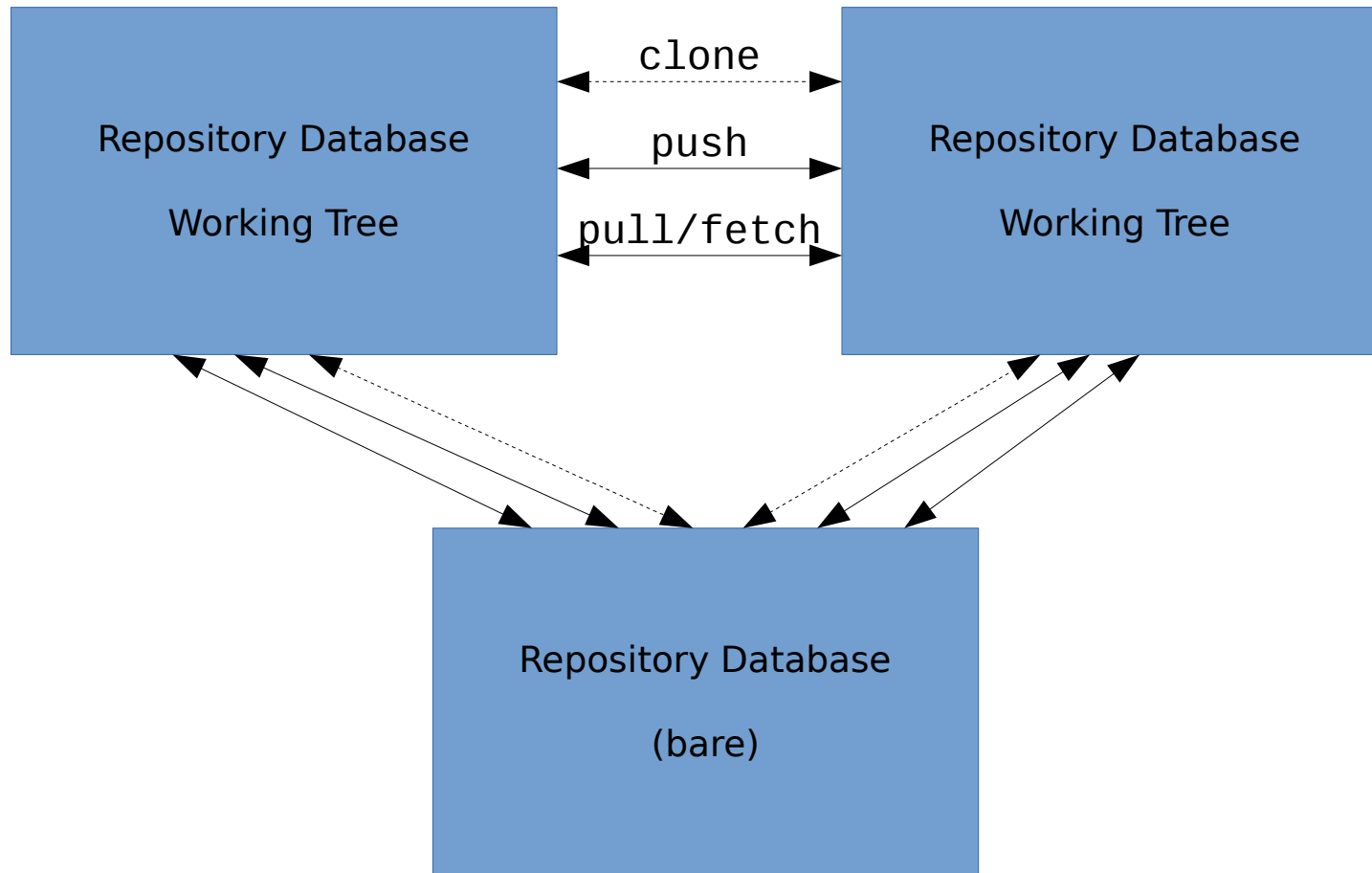
REMOTE WORKFLOW
(with GitHub)

Centralized VCS (Subversion, CVS)

```
                    checkout
                   ─────────────▶  ┌──────────────┐
                    update          │              │
                   ─────────────▶   │    Client    │
┌────────────────┐                  │              │
│                │                  │ Working tree │
│                │     commit       │              │
│                │  ◀─────────────  └──────────────┘
│     Server     │
│                │                  ┌──────────────┐
│                │                  │              │
│   Repository   │                  │    Client    │
│    database    │                  │              │
│                │                  └──────────────┘
│                │
│                │                  ┌──────────────┐
│                │                  │              │
│                │                  │    Client    │
│                │                  │              │
└────────────────┘                  └──────────────┘
```

- Most commands require a server, including "informational" commands (log, diff, blame)

Distributed VCS (Git)

Repository Database	clone	**Repository Database**
	push	
Working Tree	pull/fetch	**Working Tree**

Repository Database

(bare)

- Any "clone" can be a remote, i.e. a server. for another clone.
- Most of the operations are performed locally.
- All the "clones" contain all the history of the repository.

Let's start...

- For now we have just our local repository on our computer.

- Now we'll share this repository with the rest of the world using GitHub as a "central repository"

Working with remotes

- All the commands mentioned until now can be used locally, you don't need network or a server

- A remote can be

  - Another local repository

    - `file:///home/user/repo.git`

  - A remote repository (on a server, but also on your friend's computer)

    - `https://github.com/robotology/yarp.git`
    - `ssh://git@github.com:robotology/yarp.git`
    - `git://github.com/robotology/yarp.git`

- Remotes can be read/write or read only

Creating a Git Repository on GitHub
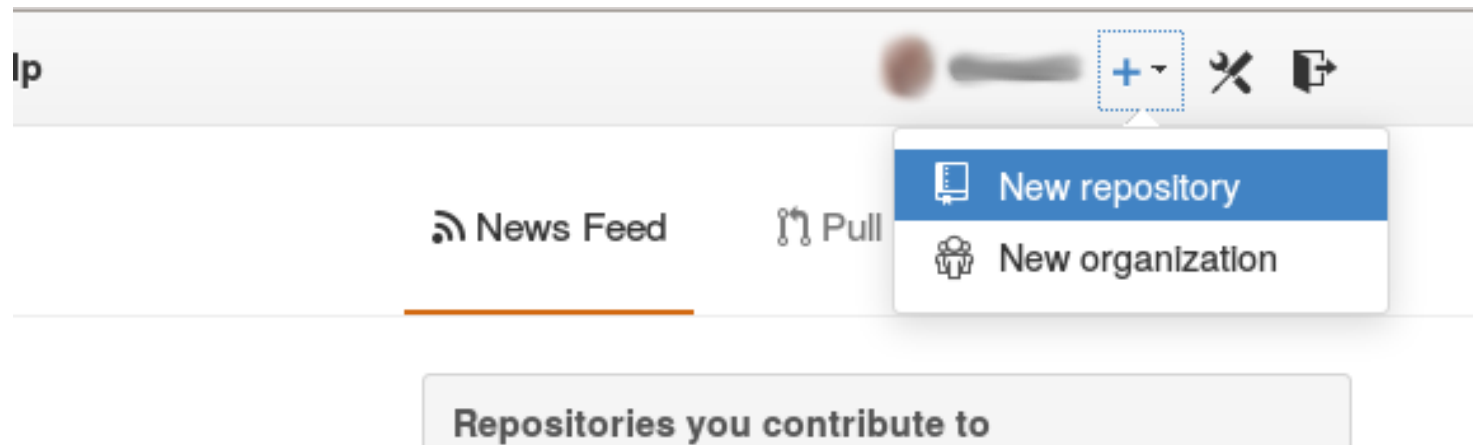
- Get an account if you don't have one
- Log in
- Click on "New repository"
- Follow the instructions
- Congratulations, you just created an empty repository on GitHub

Local repository
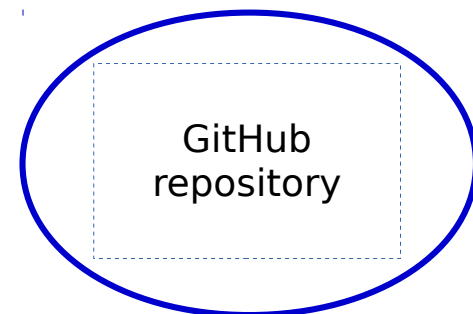
GitHub repository

# Configure Your Local Repository

- Our local git repository needs to know that the new "remote" exists

  - `git remote add origin https://github.com/vvv-school/git-training.git`

- Our local git repository now knows "origin".

- This is the name that we assigned to the remote on GitHub, and we can refer to that name from now on.

- The "`origin`" remote is still empty!
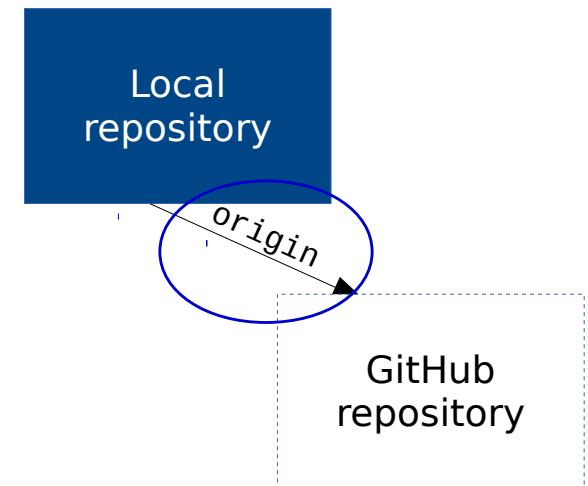
Now "Push" our Commits

- `git push origin master`

- `origin` is the name of the remote where we want to push our changes.

- `master` is the name of the branch that we want to push.

- The `master` branch on GitHub repository now contains all the commits that we had locally in our local `master` branch.

- Now refresh the page on GitHub and see what happened.

- "`master`" is your local master branch.

- "`origin/master`" is the master branch on the origin remote (it can be different from the local one if you have some commits or if someone else made some commits on the remote)

- The server can reject the push for different reasons.

Local repository

origin
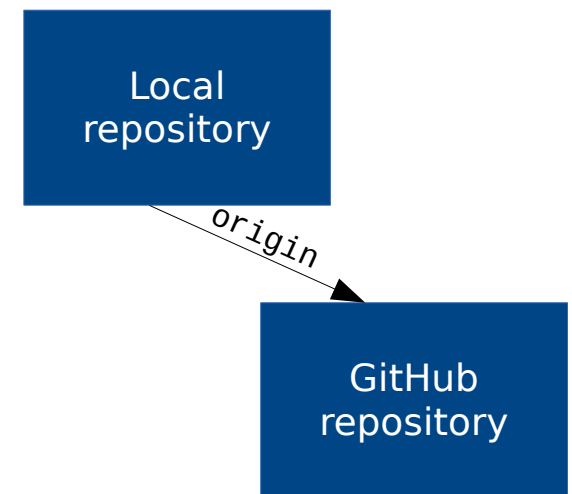
GitHub repository

Let's Do Some Team Work

- We are ready to start working in team. Our co-worker wants to grab your files and do some changes

- First of all, he will make a "clone" of the repository, i.e. will download it locally.
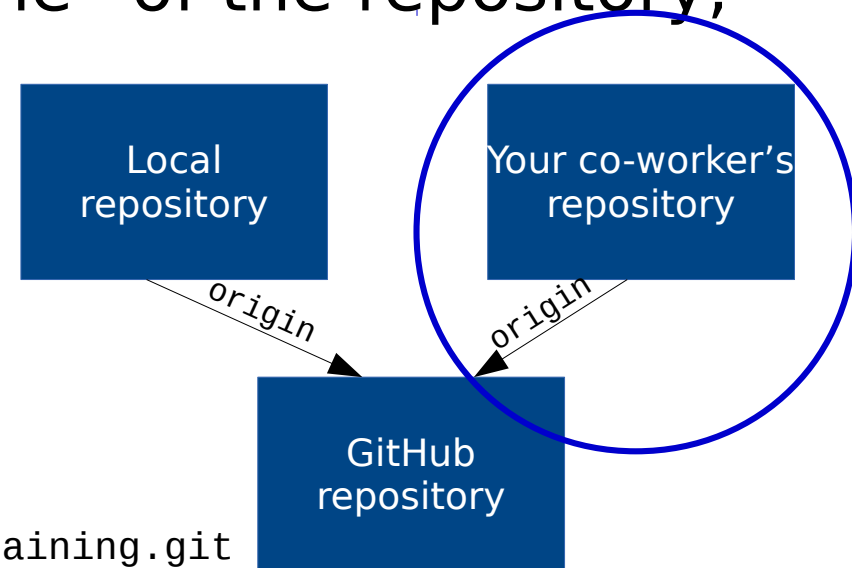
- `git clone https://github.com/vvv-school/git-training.git`

- Your co-worker now has a full copy of the repository configured with an "`origin`" remote pointing to the GitHub repository.

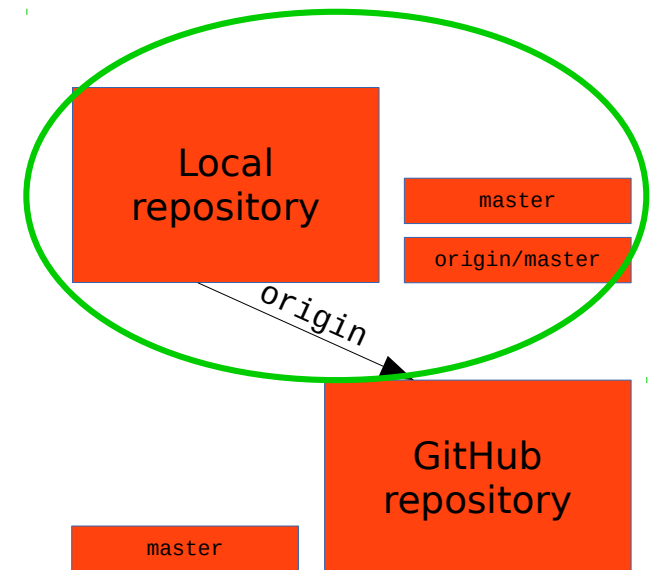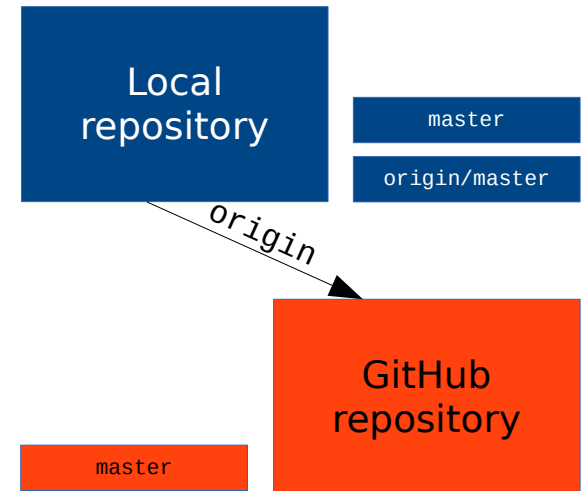- He can now push new commits on this repository.

Check Changes on a Remote

- Someone pushed some new commit on the GitHub repository.

- `git fetch origin` (or just `git fetch`)

  - Retrieves all the new objects (commits, branches, tags) in the `origin` remote and saves them in your local repository, but does not apply anything to your working directory.
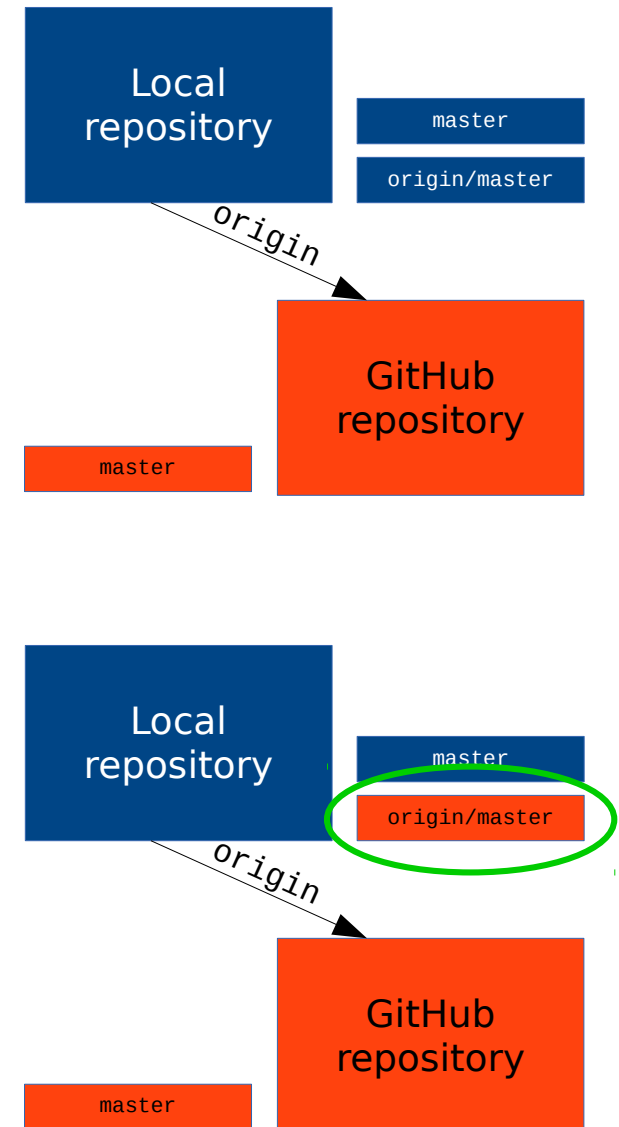
Pull Other People's Changes

- `git pull origin master`
  - Retrieves all the new commits in the `master` branch from the `origin` remote and merge them into your current branch (if you have local commits you get a new local merge commit)
  - Might result in a conflict
- `git pull --rebase origin master`
  - Same as previous command, but does not create an extra merge commit. Instead it rebases your local changes on top of the remote branch. (i.e. applies your local commits on top of `origin/master`)
  - Usually this is the recommended way to do, because the extra merge commits will make the history confused.

Local repository

master

origin/master

origin

GitHub repository

master

Local repository

master

origin/master

origin

GitHub repository

master

Fork & Pull Request on GitHub

- GitHub flow: *https://guides.github.com/introduction/flow*

- **Fork**: copy of a repository that lets you experiment with changes without affecting the original project.

- **Pull Request**: a way to propose to integrate your changes upstream, i.e. in the original project, enabling *code review*.

Local
repository

fork

origin

GitHub
personal
fork

Pull Request
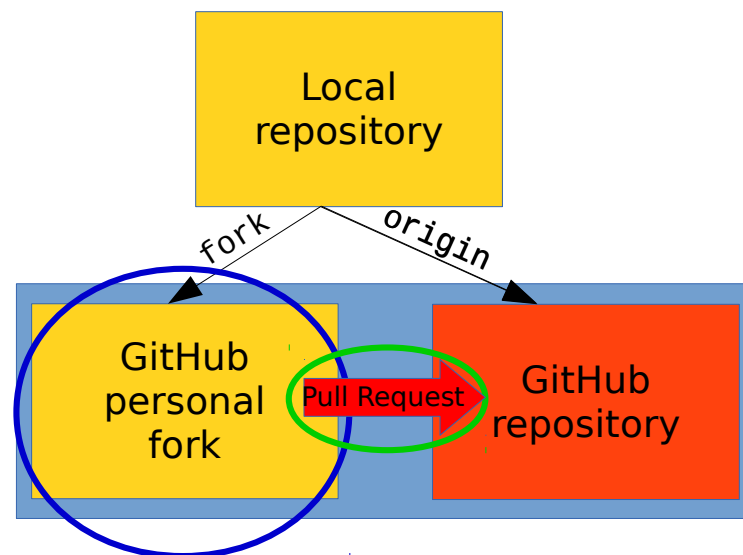
GitHub
repository

Interactive Tutorials
& Cheat Sheets

- You should try them, even if you already know how to use git, they can teach you some new tricks.

- `try.github.io`

- `learnGitBranching.js.org`

- `education.github.com/git-cheat-sheet-education.pdf`

- `ndpsoftware.com/git-cheatsheet.html`