PROGRAMMING IN PYTHON II

Version Control and Collaboration – git

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Outline

1. Motivation
2. Hashing
3. git
4. github and gitlab
5. Tasks

MOTIVATION

Motivation

- Typically your ML project will contain code, data, and reports

Motivation

- Typically your ML project will contain code, data, and reports
- We want to keep track of our project history (Version Control)
  1. Reproducibility
  2. Reusability
  3. Bug-fixing
  4. Version conflicts, different published versions

Motivation

- Typically your ML project will contain code, data, and reports
- We want to keep track of our project history (Version Control)
  1. Reproducibility
  2. Reusability
  3. Bug-fixing
  4. Version conflicts, different published versions
- We want to be able to collaborate (or let others use our code)
  1. Common interface and tools for communication
  2. Modifications
  3. Different versions

HASHING

Hashing: Motivation

- For version control but also for ML data analysis, we sometimes have to check if two files have the same content
- We could compare the complete file contents byte by byte
  - Problem: Correct but slow for larger files

Hashing: Motivation

- For version control but also for ML data analysis, we sometimes have to check if two files have the same content
- We could compare the complete file contents byte by byte
  - Problem: Correct but slow for larger files

Solution: Hash values!

Hashing: Idea

- Hash value
  - Computed via hash function from file content (e.g. bytes)
  - Fixed-sized vector (independent of input length)
  - Fast to compute (in the average case)
  - Minimal number of collisions (=multiple inputs resulting in the same hash value)

→ Compare the (shorter) fixed-sized hash values instead of complete file contents

Hashing: Idea

- Hash value
  - Computed via hash function from file content (e.g. bytes)
  - Fixed-sized vector (independent of input length)
  - Fast to compute (in the average case)
  - Minimal number of collisions (=multiple inputs resulting in the same hash value)

→ Compare the (shorter) fixed-sized hash values instead of complete file contents

- Salt: For sensitive applications (e.g. comparing passwords), salt (=secret random byte offset) is added to the input before hashing

GIT

git

- 
  - Efficient tracking of directory contents
  - Independent of types of files/data
  - Suitable for version control and collaboration
  - Supports distributed, non-linear workflows
  - Commonly used in ML
  - Large datasets usually not in git or version control
- We will not go into details but look at the basics of how to use git
- Tech Talk: Linus Torvalds on git

  `https://www.youtube.com/watch?v=4XpnKHJAok8`

Workspace

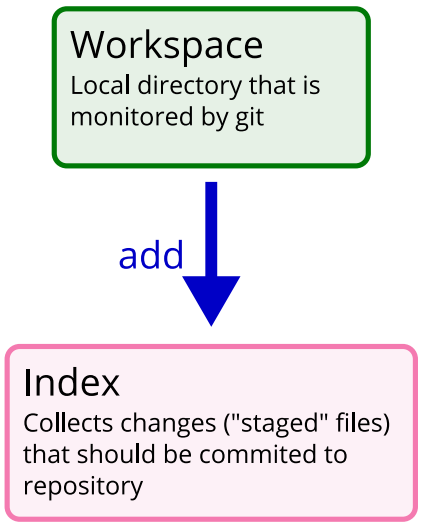
- Create a git repository for a workspace on your machine
  - Create a new repository or clone an existing repository

Workspace
Local directory that is monitored by git

Repository
Local database that stores the project versions ("commits")
Each version is stored as snapshot
Stores the complete project history ("tree")
Tree can be non-linear (e.g. including different branches)

Index

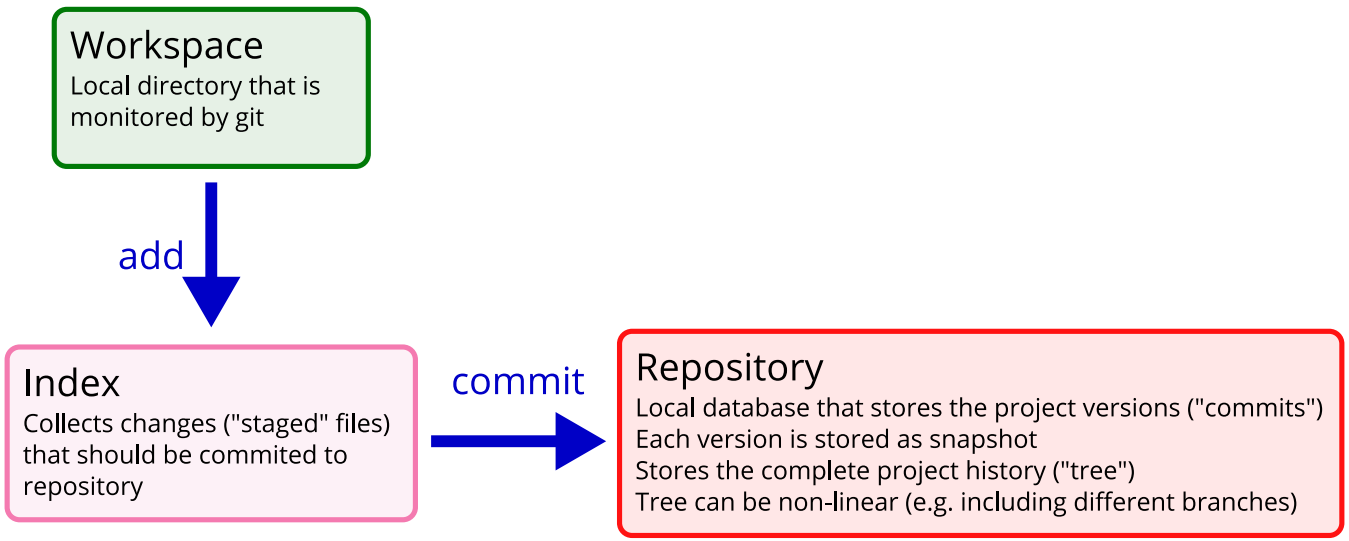
- Create a file locally and add it to the index
- The index collects changes that we want to have in our new project version
  - Files in the index are referred to as staged
  - File changes are determined via hash values



Repository
Local database that stores the project versions ("commits")
Each version is stored as snapshot
Stores the complete project history ("tree")
Tree can be non-linear (e.g. including different branches)

Repository (1)

- Commit the staged files to the git repository
  - The committed file states are now saved in the git repository
  - Each commit includes a commit message

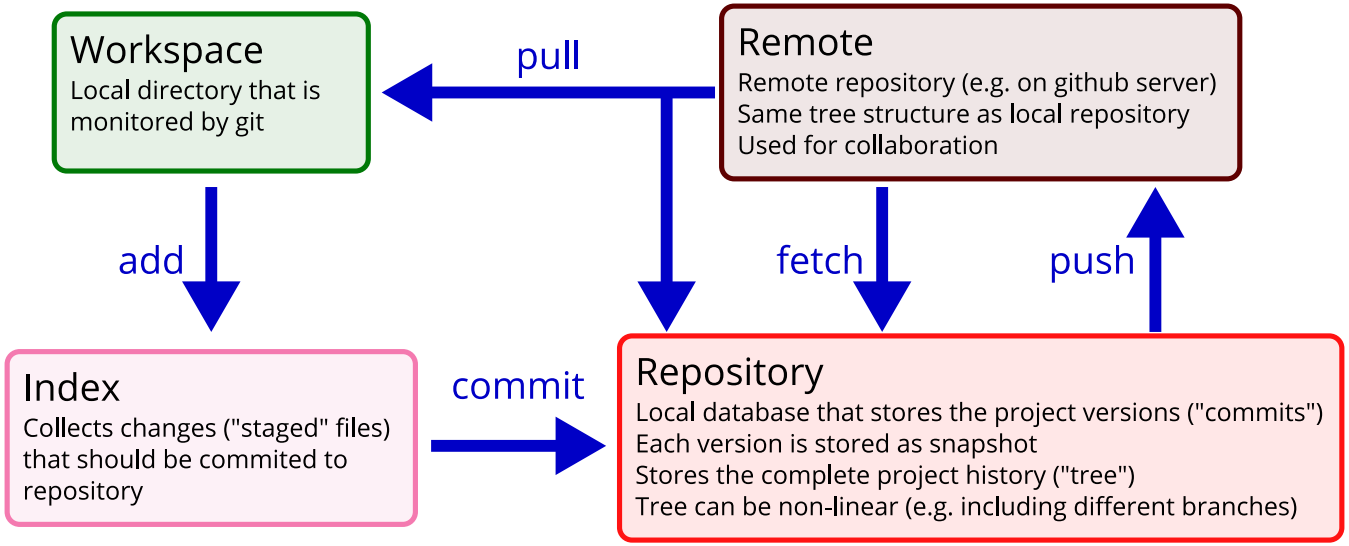


Repository (2)

- git stores snapshots of a directory
  - Each project version is a snapshot, also referred to as commit
  - The snapshot includes the file contents and the directory structure
  - For unchanged files between versions, only references to old snapshots are stored
- The commits make up the project history (tree)
  - Can be non-linear (there can be different branches)
  - Includes complete development history
- Commits are stored in a local database (repository)

Remotes

- You can interact with remote repositories (remotes)
  - pull: get commit history (tree) from remote
  - push: push commits in your repository to remote
  - merge: get commits of remote and combine with your repository version (will create a new commit)



How to use git? – Practice

- Install git
  - `https://git-scm.com/book/en/v2/Getting-Started-Installing-git`
- Setup git using `git config`
  - `https://git-scm.com/book/en/v2/Getting-Started-First-Time-git-Setup`
- git is command-line but many GUIs exist
  - E.g. PyCharm git integration
- Literature and command-line git guide:
  - `https://git-scm.com/book/en/v2`
  - `https://rogerdudler.github.io/git-guide/`

How to use git? – PyCharm

- Video guide is available in Moodle

GITHUB AND GITLAB

github and gitlab (1)

- In ML you will most likely have to use github or gitlab at some point
- `https://github.com`
  - Platform that hosts git projects
  - Many features for project/task management
  - Free accounts (for public and private projects)
  - Many open-source projects
  - Over 100 million repositories[1]
  - Example: `https://github.com/git/git`

[1] `https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/`

github and gitlab (2)

- On github/gitlab you may create a fork of an existing repository
  - Creates a copy of the repository on your account
  - Can be modified by you
  - Can be pushed to the parent repository using a pull request
- github/gitlab provide management for issues
  - Can be tagged and assigned to users
  - If you create an issue, always make sure to do it properly or you will get ignored/banned
    - Check if there are similar issues already!
    - Do not hijack, stay on-topic!
    - Provide input/output/exact descriptions of the issue!
    - Read the rules if specified!
    - You want something from someone, not the other way around!

TASKS

Tasks (1)

1. Look up git readme to find what "git" stands for
2. Install and configure git
3. Activate git intergration for a PyCharm project
   - ☐ Perform a commit
   - ☐ Perform a second commit
   - ☐ Create a new branch and commit it
   - ☐ Take a look at the git Log
   - ☐ Perform a checkout of an old commit version
4. Clone a github project using PyCharm
   - ☐ Take a look at the Log
   - ☐ Modify a file and commit it to your local clone of the github repository

Tasks (2)

5. If you have a github account:
   - ☐ Fork a github project to your github account (any project will do, e.g. the course github https://github.com/widmi/programming-in-python)
   - ☐ Clone the forked repository from your github account to your local PC
   - ☐ Modify a file, commit it, and push the changes back to your github repository