PROGRAMMING IN PYTHON II

Version Control and Collaboration – git

Andreas Schörgenhumer
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Contact

Andreas Schörgenhumer

Institute for Machine Learning
Johannes Kepler University
Altenberger Str. 69
A-4040 Linz

E-Mail: `schoergenhumer@ml.jku.at`

Write mails only for personal questions

Institute ML Homepage

Copyright Statement

Motivation

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

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

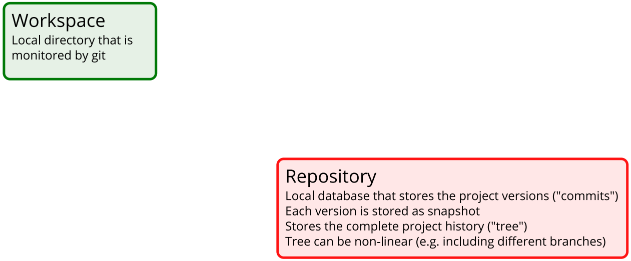
- **Hash value**
  - Computed via **hash function** from file content (e.g., bytes)
  - Fixed-sized vector (independent of input length)
  - Fast to compute (in the average case)
  - Minimal number of **collisions** (=multiple inputs resulting in the same hash value)
- Compare the (shorter) fixed-sized hash values instead of complete file contents
- **Salt**: For sensitive applications (e.g., comparing passwords), salt (=random data) is added to the input before hashing

Git

- git
  - Distributed version control system
  - Efficient tracking of files and their changes
  - Independent of types of files/data
  - Large datasets usually not in git or version control
  - Very common and useful for collaboration
- We will not go into details but look at the basics of how to use git
- Download: `https://git-scm.com/downloads`

Workspace

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

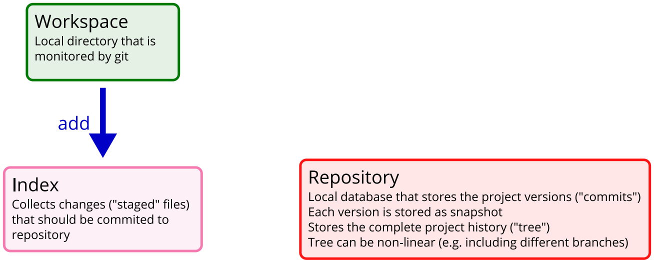


Workspace
Local directory that is monitored by git

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)

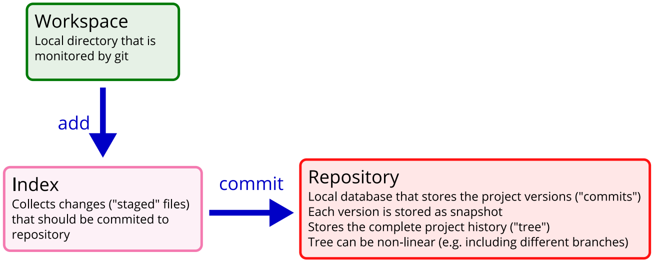
Index

- 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** or **cached**
  - File changes are determined via hash values


Workspace
Local directory that is monitored by git
add
Index
Collects changes ("staged" files) that should be commited to repository
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

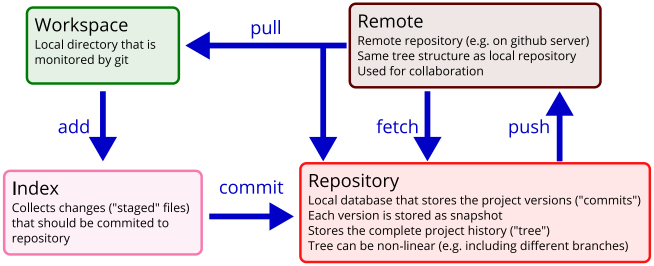

Workspace
Local directory that is monitored by git
add
Index
Collects changes ("staged" files) that should be commited to repository
commit
Repository
Local database that stores the project versions ("commits")
Each version is stored as snapshot
Stores the complete project history ("tree")
Tree can be non-linear (e.g. including different branches)

Repository (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**)
  - **fetch**: update/download remote branches
  - **merge**: combine branches in your repository version (will create a new commit)
  - **pull**: get commit history (tree) from remote (fetch + merge)
  - **push**: push commits in your repository to remote


Workspace
Local directory that is monitored by git
pull
Remote
Remote repository (e.g. on github server)
Same tree structure as local repository
Used for collaboration
add
fetch
push
Index
Collects changes ("staged" files) that should be commited to repository
commit
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)

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 has a command line interface but many GUIs exist
  - PyCharm git integration
  - GitHub Desktop
  - ...
- Literature and command-line git guide:
  - https://git-scm.com/book/en/v2
  - https://rogerdudler.github.io/git-guide/

GitHub

- In computer science and artificial intelligence, you will most likely have to use GitHub (https://github.com) (or some other platform such as GitLab) at some point
  - Platform that hosts git projects
  - Many features for project/task management
  - Free accounts (for public and private projects)
  - Many open-source projects
  - Over 384 million repositories[1]
  - Example: https://github.com/git/git

[1] https://github.com/search

GitHub

- You can create your **own** repositories (with a free account) that are hosted on GitHub
- On GitHub, 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 provides management for **issues**
  - Can be tagged and assigned to users
  - If you create an issue, always make sure to do it properly
    - Check if there are similar issues already
    - Provide input/output/exact descriptions of the issue
    - Read the rules if specified

Tasks (1)

1. Install and configure git
2. Activate git integration for a PyCharm project
3. Perform the following operations with the command line interface, PyCharm or any GUI client:
   - ☐ 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
   - ☐ Take a look at the **Log**
   - ☐ Modify a file and commit it to your local clone of the GitHub repository

Tasks (2)

4. If you have a GitHub account:
   - ☐ **Fork** a GitHub project to your GitHub account (any project will do)
   - ☐ 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
5. If you have a GitHub account:
   - ☐ Create your **own** repository
   - ☐ Experiment with the different functionality provided by GitHub