Key concepts

- Each developer uses a private local repository
  - *clone*: full mirror of some existing repository

- Operations performed on the local repository
  - very fast, off-line

- Synchronization
  - Operations *push* and *pull*
  - Exchanging code patches
Comparing distributed and centralized VCS

- Centralized
  - Everything visible in the central repository
  - Private branches (work) not possible

- Distributed
  - Private repositories (and branches) useful for experimental development
Tools

- Git
- Mercurial
- Bazaar
Git

Software Development Tools

Distributed Version Control
Main features

• Versions: snapshots of the project (working dir)

• Committed revisions form a direct acyclic graph
  ▪ Multiple “latest” versions (leaf nodes)

• Each commit has an author and committer
  ▪ Distributing changesets via patches (email)

• Whole repository stored in `.git` (files, metadata)

• Confusing for most people (good for advanced users)
• Commands have names similar to SVN
Usage scenario

Local Operations

- working directory
- staging area
- git directory (repository)

- checkout the project
- stage files
- commit

Task 1

- Configure your identity
  
  - `git config --global user.name "<your full name>"
  
  - `git config --global user.email "<your email address>"

- Stored in `$HOME/.gitconfig`
Basic commands

- Create repository in the current directory: `git init`
- Print status of the working tree: `git status`
- Start tracking new files: `git add <work dir path>`
- Add files to the staging area: `git add <path>`
- Commit staged modifications: `git commit -m "..."`
- Print uncommitted unstaged changes: `git diff`
- Print staged uncommitted changes:
  ```
  git diff --staged
  ```
- Automatically stage every tracked file and commit
  ```
  git commit -a -m "..."
  ```
- Revert modifications: `git checkout -- <path>`
File status lifecycle

- untracked
- unmodified
- modified
- staged

- add the file
- remove the file
- edit the file
- stage the file
- commit

Task 2

- Create repository in a specific directory
- Create some new files (e.g., hello world)
- Print current status of your repository and the working directory
- Stage all the new files
- Print current status
- Modify one of the files
- Print current status
  - Inspect differences from the previous invocation
- Commit all staged modifications
- Print current status
Managing files

- Make the given file untracked
  
  `git rm <work dir path>`

- Renaming file (directory)
  
  `git mv <old path> <new path>`
Pick your changes

- Full interactive mode: `git add -i`

- Select patch hunks: `git add -p`

- Additional information with examples
Project history

- List all the commits
  
  `git log [-p] [--<N>] [--stat]
  
- More options
  
  `[--pretty=oneline|short|full|fuller]
  
  `[--graph]
  
  `[--since=YYYY-MM-DD]
  
  `[--until=YYYY-MM-DD]
  
  `[--author=<name>]`
Task 3

- Try out file management commands (rm, mv)

- Play with the “git log” command
  - Explore different parameters (-p, -<N>, --stat, --pretty, --graph)

- Run the program “gitk” and try it

- Make some changes to a particular file and use interactive staging
Using remote repositories

- Clone a remote repository in the current local directory: `git clone <repo url>
- Get recent changes in all branches from the remote repository: `git fetch origin`
- Get recent changes in the “master” branch and merge into your working copy: `git pull`
  - Announcements via pull requests
- Publish local changes in the remote repository: `git push origin master`
Branches in Git
Branches in Git

- Branch: pointer to a node in the revision DAG
- Default branch: **master**
- Commit: branch pointer moves forward

What happens after concurrent modification

Branches in Git: commands

- **Create new branch:** `git branch <name>`
- **Switch to given branch:** `git checkout <name>`
- **Shortcut:** `git checkout -b <name>`
- **Merge branch into current working directory**
  `git merge <branch name>`
- **Deleting unnecessary branch**
  `git branch -d <branch name>`
- **List all branches:** `git branch [-a]`
  - Current branch marked with *
Comparing branches

- `git diff <branch 1>..<branch 2>`
  - Compare heads of the two branches
  - Note the characters ‘..’

- `git diff <branch 1>...<branch 2>`
  - Print changes on the branch 2 (e.g., master) since the branch 1 (feature) was created from it
  - Note the characters ‘...’
Three-way merge

- Common ancestor
- Target branch
- Source branch

- Conflicts happen also with Git
  - Standard markers `<<<<<< ====== >>>>>>>`
  - Marking resolved files: `git add`

- Graphical merging tool: `git mergetool`
Task 4

- Create new branch B and switch to it
- Modify some files and commit them
- Switch back to the master branch
- Modify some files and then commit
- Merge your branch B into the master
- Delete the now unnecessary branch

- Try switching branches with uncommitted changes in the working copy
- Try graphical merging tool on some conflicts
Advanced features I.

• Using stack of unfinished changes (stashing)
  - `git stash [push]`
  - `git stash pop`
  - `git stash apply [<stash name>]`
  - `git stash list`

• How to undo some changes
  - `git reset <commit>`
    - Moves the branch HEAD to a given commit
  - Several variants
    - `--soft`: undo commit (just in history of revisions)
    - `--mixed` (default): undo commit and changes in staging area
    - `--hard`: undo everything (commit, staging area, working dir)
Advanced features II.

- Symbolic names of versions
  - HEAD, HEAD~1, HEAD^2

- `git rebase`
  - Replaying changes done in a branch onto another branch
  - Very powerful command but also tricky (be careful !!)

- Modifying committed history
  - e.g., commit messages (`git commit --amend`)

- Ignoring certain files
  - List patterns in the file `.gitignore`

- Tagging: `git tag`

- Bare repository
  - No working copy
Mercurial

- Basic principles: like Git
- Simpler learning curve
- Commands very similar
  - `init, clone, add, commit, merge, push, pull`
Work-flow models (cooperation)
Work-flow models (cooperation)

- Anything possible technically with DVCS
- “Network of trust” between developers

Examples

- Single “central” repository
- Multiple release repositories
- Many public repositories
- Total anarchy
Single “central” repository

Privileged Developer Repository

Central Repository

Privileged Developer Repository

Normal Developer Repository

Normal Developer Repository

Software Development Tools

Distributed Version Control
Multiple release repositories

Main Repository development

- Developer Repository GUI branch
- Developer Repository DB branch
- Release 1 Repository
- Release 2 Repository
- Release 3 Repository
Many public repositories

- Linux kernel

Diagram:
- Official Release
  - Vendor Release
  - Main Development integration
    - Module Development experiments
      - Module Development experiments
      - Module Development experiments
      - Module Development experiments

Software Development Tools
Distributed Version Control
Total anarchy

Repository no. 1

Repository no. 2

Repository no. 3

Repository no. 4

Repository no. 5

Software Development Tools

Distributed Version Control
Contributing to [open-source] projects

• Typical scenario
  ▪ Project hosted on some public repository server
  ▪ Write access to official repository is not possible

• Important concepts
  ▪ Forking of the official repository
  ▪ Publishing via pull requests
Contributing to [open-source] projects

- Official central repository (upstream)
  - https://github.com/projectname

- Fork on the same server
  - https://github.com/user/projectname

- Clone to local repository
  - From https://github.com/user/projectname to $HOME/projectname

- Synchronizing fork with official repository
  - git fetch upstream
  - git merge upstream/master

- Publishing changes to the upstream repository
  - Creating pull requests (processed later by maintainer)
Links

- Git documentation
  - http://git-scm.com/doc

- Mercurial

- Repository servers
  - https://github.com/
  - https://bitbucket.org/
  - https://gitlab.com/

- Tools
  - Git for Windows (http://msysgit.github.io/), TortoiseGit (Win), SmartGit (http://www.syntevo.com/smartgit/)
  - TortoiseHg (Mercurial GUI, Windows)
  - SourceTree (https://www.sourcetreeapp.com/, Git and Mercurial)
Homework

- Assignment
  - [https://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/](https://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/)

- Deadline
  - 13.10.2021 / 18.10.2021