Introduction to Git and GitLab

Pictures from www.atlassian.com/git/tutorials
Slides at https://indico.him.uni-mainz.de/event/37/

What is a Version Control System (VCS)?

- Track changes among (any type of) files
- Long term history of changes
- Collaborate on files (e.g. source code)

- Answer: Who ? Why ? When ?

Advantages of Git

- Distributed VCS
  - Every user keeps the full set of changes
  - Independent of network connection or central server
- Easy collaboration on code via pull/merge requests
- Forking of projects
- Became very popular is combination with web interfaces GitLab, GitHub, Bitbucket

Pictures from www.atlassian.com/git/tutorials
Slides at https://indico.him.uni-mainz.de/event/37/

Getting started - `git init/add/commit`

- Getting help

```
man git
man git-<subcommand>, e.g. man git-add
```

- Initialize a new repository

```
git init
```

- Create file or add existing ones

```
git add testfile
git add *.cxx
```

- First commit

```
git commit –a –m “Initial commit”
git commit --amend
```

- Further changes – show differences

```
git status
git diff
```

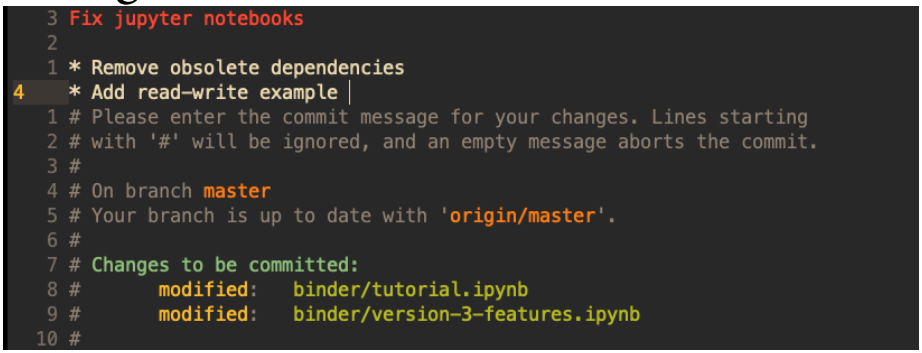
- Exclude files using `.gitignore`

Inspecting a repository - `git log/diff/blame`

- Inspect commit history

```
git log
git log --oneline
```

- Write proper log messages!



- Show changes since some commit

```
git diff HEAD~3
git diff f44596f
git diff f44596f827263d90cb52fe10c3104bd8cce742bc
git diff HEAD~3 -- README.rst
```

- Show who's responsible for a certain line of code

`git blame README.rst` (Use the IDE of your choice)

Fix mistakes - `git checkout/reset/revert`

- Fix your mistakes – copy file from previous commit

```
git checkout HEAD -- README.rst
```

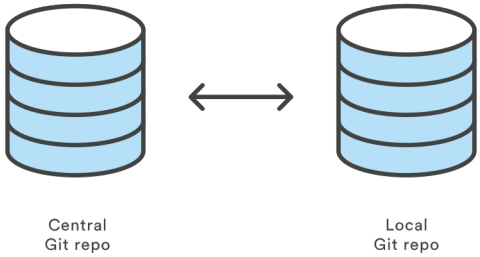
- Reset all changes to HEAD

```
git reset --hard HEAD
```

- Revert the changes of a single commit

```
git revert HEAD~3
```

Branches - `git branch/clone`



- List branches

```
git branch (-a)
```

- Create a new branch

```
git branch new
git checkout new
```

} `git checkout -b new`

- Clone remote repository

```
git clone https://git....
git clone git@git....
git clone /local/path/to/repo
```

- Merge branches (`new` into `master`)

```
git checkout new
git merge master
```

- Remote repositories

```
git remote –v
git remote add/remove
```

Exercise I

I. Create a repository, add files, commit

II. Get repository

```
git@gitlab.rlp.net:
    git-introduction/example-multibranch.git
```

OR:

```
https://gitlab.rlp.net/
    git-introduction/example-multibranch.git
```

1. Show all (remote) branches
2. Pull remote branch `new`:

   ```
   git checkout origin/new
   ```

3. Merge branch `new` into `master`
4. Open the conflicting file and solve the merge conflict
5. Commit and push your changes
6. Check the history using

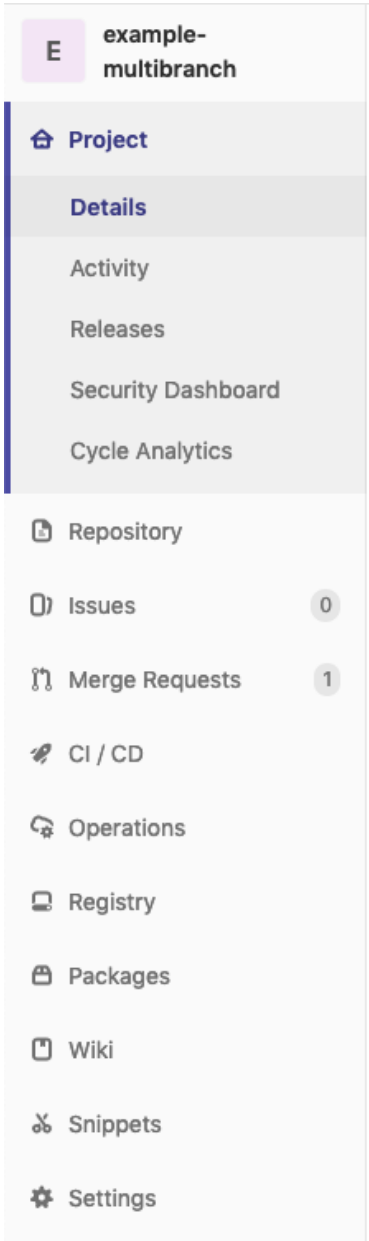
   ```
   git log --oneline --graph --decorate
   ```

   (or use some graphical tool like `gitk`)

Collaborating on git repositories - GitLab

- Public services `www.gitlab.com`
  `www.github.com`
- University hosted service `gitlab.rlp.com`

- Web interface to many git functions
- Access management to the repository
- Management of bug reports (Issues)
- Code analysis
- Provides releases
- Wiki pages
- Continuous integration (CI)
- Backup and long time storage



Exercise II

I. Create a repository on `gitlab.rlp.net`
   1. Use an existing project and initialize a git repository
   2. Add a new remote to your existing repository

```
git remote add origin git@gitlab.rlp.net:
      git-introduction/example-multibranch.git
```

   3. Push your repository to GitLab

II. Fork an existing repository in GitLab
https://gitlab.rlp.net/git-introduction/example-multibranch.git
   1. Clone the fork
   2. Perform a change and push it to your fork
   3. Create a merge request (pull request) on GitLab

Advanced topics

Submodules

Large file support

Continuous integration

Submodules – `git submodule`

- Modular organization of my own code
- Dependency of (fast) changing code
- Avoid another system dependency
  - Software typically not available on all Unix systems

- Add submodule

```
git submodule add git@gitlab.rlp.net:...
```

- Clone respository with submodules

```
git clone [URL]
git submodule init
git submodule update
```

`git clone --recursive [URL]`

- Update submodules

```
git submodule update --remote
```

Large file support – `git lfs`

- Size of the repository matters
  - Each clone copies every version of every files ever existed within the repository
  - Problem with large files that change frequently
- Full histroy is not necessary for e.g. pictures, ROOT files
- `lfs` only downloads versions referenced by current commit

- Mark such files as `lfs`:

```
git lfs install
git lfs
```

- Download a repository with lfs

```
git clone [URL]
```

- Downloaded during `git checkout` (not `git clone`)

Continuous integration (CI)

- Verify each commit
  - Build code (on several systems)
  - Run tests
  - ThirdParty tools on code quality
- Configure on GitLab via `.gitlab-ci.yml`
- Images available on `https://hub.docker.com/`
- Templates are provided on GitLab

```
image: gcc

build:
  stage: build
  before_script:
      - apt update && \
        apt -y install make
  script:
    - g++ helloworld.cpp \
          -o mybinary
  artifacts:
    paths:
      - mybinary

test:
  stage: test
  script:
    - ./runmytests.sh
```

Exercise II

I. Submodules
   1. Clone including submodules
      `https://github.com/ComPWA/ComPWA.git`
   2. Update submodules

II. LFS
   1. Use a custom project and add some files to be tracked via lfs
   2. Push the repository to GitLab

III. CI
   1. Create a repository and add a LaTeX document
   2. Upload it to gitlab
   3. Setup CI to recreate the document every time the repository is updated

Summary

Git and GitLab

- Git tracks changes in your code (or other files)
- GitLab organizes collaboration on Code
  - Issues
  - Merge requests
  - Access control

There is more....

- Advanced git commands
  `git rebase/bisect`
- Filetypes different from pure text
  Word, TeX, jupyter notebooks
- GitLab Docker registry to store images generated by CI