

14 – Further Git features

Bálint Aradi

Scientific Programming in Python (2025)

<https://atticlectures.net/scipro/python-2025/>

Rename files

- **Rename a file** under version control:

```
git mv README README.txt
```

```
git status
```

```
On branch main
```

```
Changes to be committed:
```

```
renamed: README -> README.txt
```

```
git commit -m "Add .txt extension to readme file"
```

- Corresponding **file in working directory will be renamed** immediately
- The name **change must be committed** like any other change

Delete files

- **Delete (remove)** a file under version control

```
git rm unnecessary_file
```

```
git status
```

On branch main

Changes to be committed:

```
deleted:    unnecessary_file
```

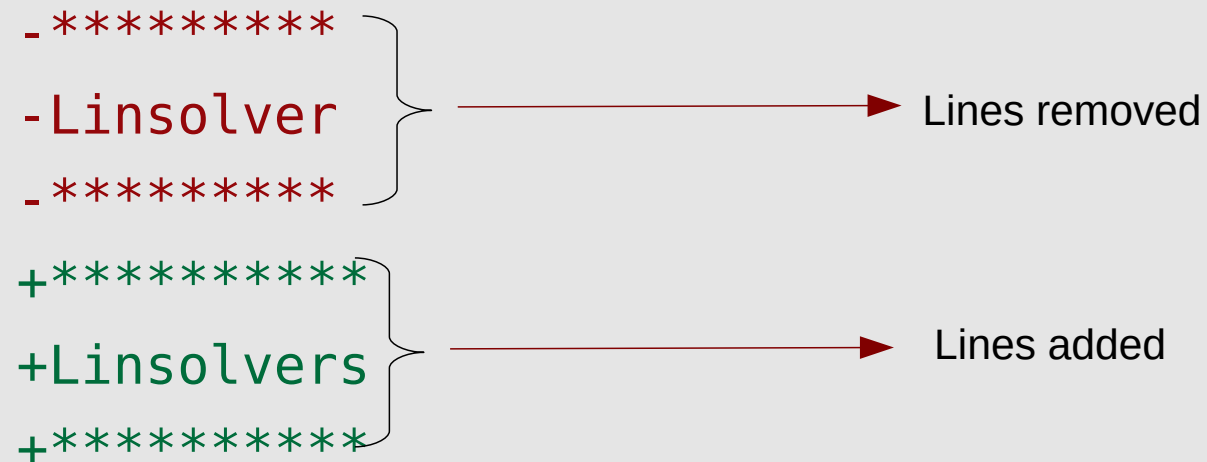
```
git commit -m "Delete unnecessary file"
```

- Corresponding **file in the working directory will be deleted** immediately
- The **removal must be committed** like any other change
- The file will be not present in **future revisions**, but stays part of the previous commits!

Investigating changes

- Changes between **working copy and last checked in / staged version**

```
git diff README.txt
diff --git a/README.txt b/README.txt
index 8eab0a7..770eee5 100644
--- a/README.txt
+++ b/README.txt
@@ -1,6 +1,6 @@
-*****
-Linsolver
-*****
+*****
+Linsolvers
+*****
```

A diagram illustrating the diff output. On the left, the diff text is shown with red lines for removals and green lines for additions. Brackets group the red lines and the green lines. Arrows point from these brackets to the text 'Lines removed' and 'Lines added' respectively.

Lines removed

Lines added

- If no file name is specified, all changes in all version controlled files are shown

```
git diff
```

- Changes between **two committed revisions** using revision hashes

```
git diff 04d386 2a3186 -- README.txt
```

Optional, otherwise
changes in all files shown

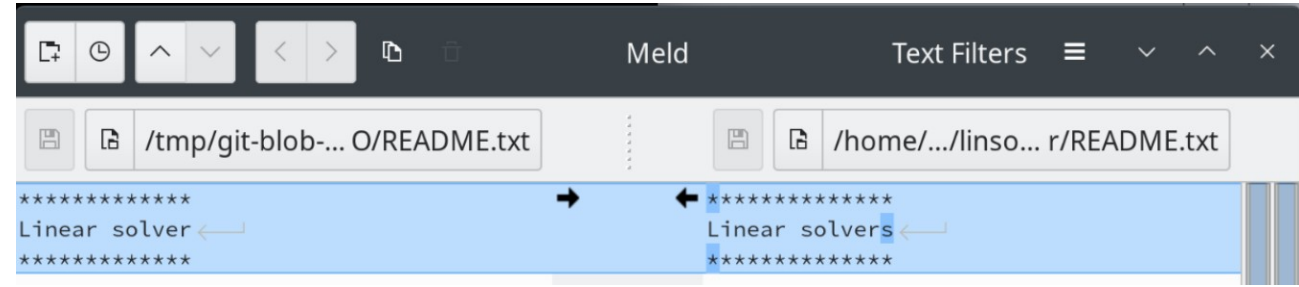
Investigating changes

- The **difftool** sub-command calls the default diff-viewer (kdiff3, meld) to **visualize changes**

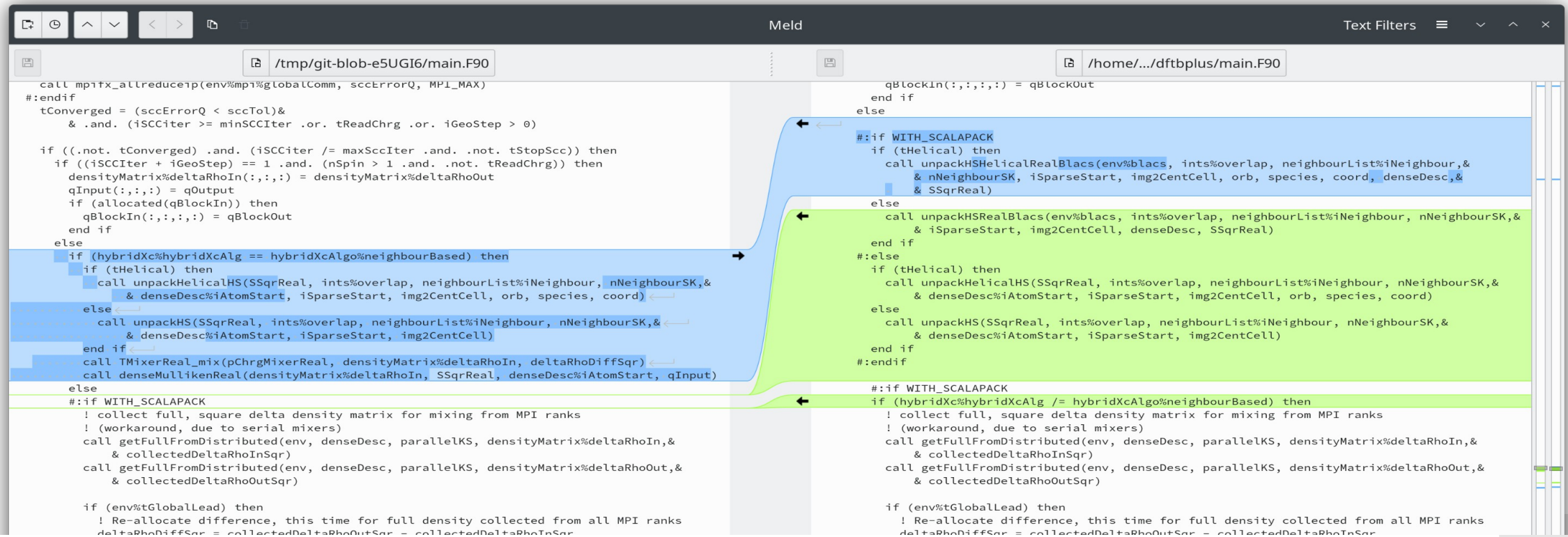
```
git difftool
```

```
Viewing (1/1): 'README.txt'
```

```
Launch 'meld' [Y/n]?
```



A more complex example:



Discard changes in working copy

- **Set working directory back** to last committed / staged version:

```
git status
[...]
```

modified: README.txt

no changes added to commit (use "git add" and/or "git commit -a")

```
git restore README.txt
git status
[...]
```

nothing to commit, working tree clean

Overwrites working copy!

- If you specify a **directory** (e.g. "." = current directory) all files within will be **restored recursively**

Unstage files

- Staged files can be **unstaged**, if they should not be part of the next commit
- Corresponding file in the **work directory** is **not affected** by the operation

```
git status
```

```
On branch main
```

```
Changes to be committed:
```

```
    modified:   README.txt
```

```
git restore --staged README.txt
```

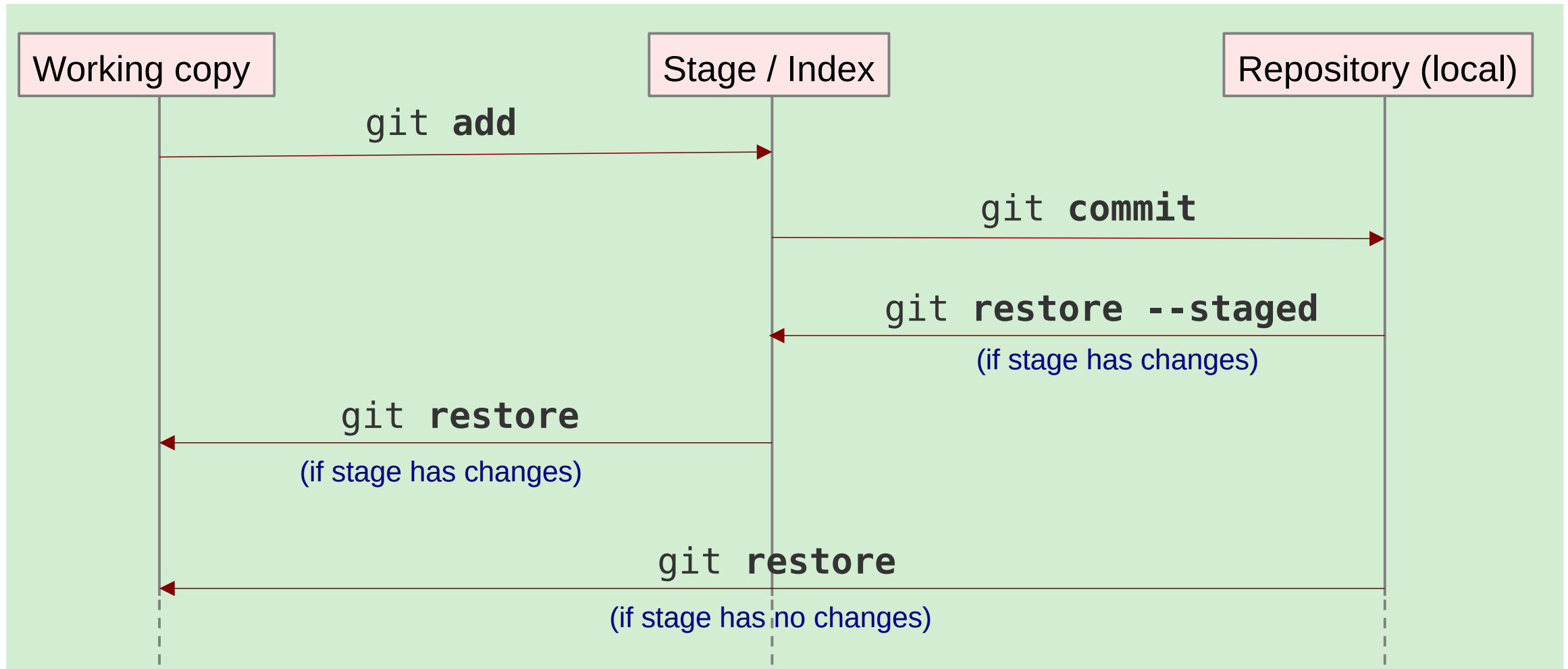
```
git status
```

```
On branch main
```

```
Changes not staged for commit:
```

```
    modified:   README.txt
```

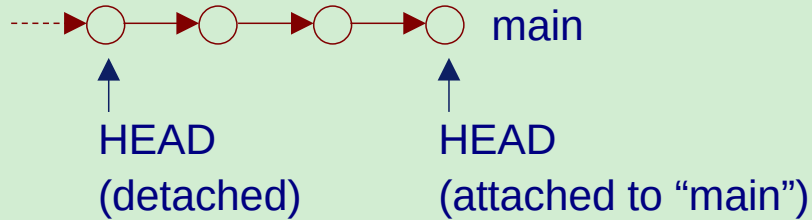
Overview: Git file transfer commands



Time travel: switching to an earlier commit

branch = line of history

HEAD = currently active commit



Attached HEAD

- points to last commit (in current branch)
- next commit appended to this commit

Detached HEAD

- points to earlier commit
- no new commits can be appended in current branch to keep history intact (but new branch can be created from commit)

- **Switching to a specific commit**
(by specifying its hash value)

```
git switch --detach 2a31862  
HEAD is now at 2a31862 ...
```

```
git status  
HEAD detached at 2a31862
```

- You have to **change back to most recent commit** before committing any changes (otherwise history would change!)

```
git switch main  
Switched to branch 'main'
```

- Alternative: new branch from detached HEAD,
committing changes, merging to main

Tagging versions

- **Commits with special importance** (e.g. release) can be **tagged**
- **Annotated tags** are distributed with the repository

tag = symbolic name for a commit

```
git log --oneline
```

```
7d8cf66 (HEAD -> main, origin/main) Add readme and .gitignore  
ddd085a Start linsolver project with stub files
```

```
git tag -a v0.1
```

← HEAD tagged by default if no commit specified

```
git log --oneline
```

```
7d8cf66 (HEAD -> main, tag: v0.1, origin/main) Add readme and .gitignore  
ddd085a Start linsolver project with stub files
```

- Tag names can be used **instead of revision hashes** in all git commands

```
git diff 04d3866 v0.1
```

Git aliases

- Aliases help to **abbreviate often used git commands and options**

```
git config --global alias.ci commit
git config --global alias.sw switch
git config --global alias.swd "switch --detach"
git config --global alias.st status
git config --global alias.gdiff difftool
```

- If an alias is used, the corresponding command / options will be substituted

```
git ci -m "Add quick changes"
git swd 2a31862
git st
git gdiff README.rst
```

Please create these aliases for your account, since the following examples will make use of them!

Graphical git-viewers

- Several graphical git-clients exist to visualize development history:

qgit &

Windows

- gitk is automatically installed with Git

Linux / MacOS

- several viewer installable as packages (qgit, gitk, ...)

The screenshot shows the QGit application window. The title bar indicates the path `/home/aradi/sync/bccms/education/SciPro/2022/vorlesung/linsolver [main] - QGit`. The interface includes a menu bar (File, Edit, View, Actions, Help) and a toolbar with icons for file operations and a search bar. Below the toolbar is a 'Rev list' tab, and a 'Graph' view on the left shows a commit history graph with a red line indicating the current branch. The main area displays a table of commit history:

Graph	Short Log	Commit	Author
	main bccms-webserver/master 1.0 exercise10 Improve user ...	0fda580	Bálint Aradi<aradi@uni-bremen.de>
	Refactor modules into separate package	918c1f9	Bálint Aradi<aradi@uni-bremen.de>
	Add command line options to linsolve	30834ca	Bálint Aradi<aradi@uni-bremen.de>
	Add command line options to geninput	9b374d7	Bálint Aradi<aradi@uni-bremen.de>
	exercise09 Merge branch 'multirhs'	53d2869	Bálint Aradi<aradi@uni-bremen.de>
	Merge branch 'master' into multirhs	18835bf	Bálint Aradi<aradi@uni-bremen.de>
	Merge branch 'performance'	00d579c	Bálint Aradi<aradi@uni-bremen.de>
	Implement Gaussian-elimination via numpy.linalg.solve()	b21cdc8	Bálint Aradi<aradi@uni-bremen.de>
	Add script for generating random input	92c13ed	Bálint Aradi<aradi@uni-bremen.de>
	Allow for solving with multiple right hand side vectors	6c7dd1a	Bálint Aradi<aradi@uni-bremen.de>
	scipro20/master exercise08 Simplify testing	6bc4204	Bálint Aradi<aradi@uni-bremen.de>
	Fix some issues in the API-documentation	5a91871	Bálint Aradi<aradi@uni-bremen.de>
	Add sphinx config to extract API-documentation	3aac18a	Bálint Aradi<aradi@uni-bremen.de>

Below the table, the 'Improve user documentation' commit is selected, showing its details: Author (Bálint Aradi<aradi@uni-bremen.de>), Author date (6/29/20 9:42 PM), and Parent (Refactor modules into separate package). The diff view on the right shows changes to README.rst, geninput, and linsolve.

Tracking remote repositories

- Remote repository must be cloned first to create a local git repository

```
git clone https://github.com/aradi/scipro-demo.git
Cloning into 'scipro-demo'...
```

- Remote repository will be associated with the new local repository (under then name “**origin**”)

```
cd scipro-demo
git remote -v
origin https://github.com/aradi/scipro-demo.git (fetch)
origin https://github.com/aradi/scipro-demo.git (push)
git status
On branch main
Your branch is up to date with 'origin/main'.
```

- Recent changes in the remote repository can be pulled

```
git pull
Updating ddd085a..7d8cf66
```

Note: This only works without side-effects, if the local repository was not modified apart of “git pull” calls.

Some further git-notes

- **Read the manual!**
- **Commit after each non-trivial change**
Rule of thumb: It should be easy for other developers to follow and understand the changes of a commit.
- **One commit: only logically related changes.**
- **Each commit: working project**
- **Git commands** must be executed **in the project directory** (or in a subdirectory of it).
- **Version history** is stored in the **.git sub-directory**.
If it is copied with the project, the version history is copied as well.

Hosting git repositories

- **Raw Git repositories** can be easily hosted on any webserver
- Hosting **Git repositories with user friendly web interfaces** is also possible, but more complex (e.g. running a **GitLab server**)
- Several companies offer “**free**” **git repository hosting** (with some constraints):



GitHub (Microsoft): <https://github.com>



GitLab (GitLab Inc.): <https://gitlab.com>



Bitbucket (Atlassian): <https://bitbucket.org>

See <https://github.com/aradi/scipro-demo> for an example ...