

Git

A brief introduction to Git



Version control



Why version control?

- **Tracking** *But it worked yesterday...* 😭
- **Storing versions** *Could you recreate this plot for me?*
- **Backup** *I replaced the wrong my_file_2_final_final.py* 😱
- **Exchange** *My colleague sent me a script via mail! Or Mattermost...?*
- **Collaboration** *review_comments.pdf review_feedback-2.docx*

Version control systems

- Manage a sequence of snapshots (e.g. of folders)
- There are many version control systems out there (e.g. Subversion, CVS, Mercurial, Git)
- We will focus on **Git** which is the **de facto standard**
- We will learn how to use Git on the **command line**¹



1. Graphical Git tools often hide what's happening more than they help simplify it.

Expectation management

- You won't master Git in an afternoon; it becomes intuitive only through regular use
- Git may feel unfamiliar at first, but it's a tool that rewards practice
- Excellent documentation and many tutorials are available when you need them

Git is a skill not a button you press



The basic workflow



Creating a repository

`git-init` — Create an empty Git repository or reinitialize an existing one

- Initializes¹ an empty repository in a directory
- Creates a hidden `.git` folder for housekeeping data

```
1 mkdir my_repo
2 cd my_repo
3 git init .
```

1. A repository is only initialized once

The staging area

`git-add` — Add file contents to the index

- Adds a file to the index by creating an **object** in the `.git` folder
- The file is “staged” for the next commit

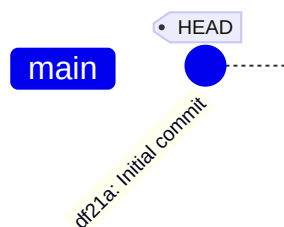
```
1 date > date.txt # Create date.txt with current date
2 git add date.txt
```

Commit

`git-commit` — Record changes to the repository

- Creates a new commit object, which contains the current content of the index
- In addition to the content, a log message and author information are stored

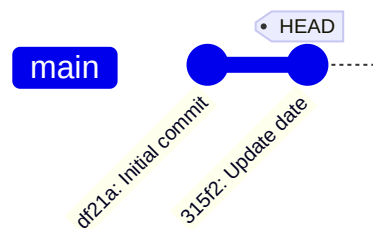
```
1 git commit -m "Initial commit"
```



Commits

- Commits are linked to form a sequence of snapshots
- Each commit is a direct child of the current **HEAD**¹

```
1 date > date.txt # Update content of date.txt
2 git add date.txt
3 git commit -m "Update date"
```



1. **HEAD** is the last commit in the currently checked-out branch

Day-to-day life

- Git has many features that are **extensively documented**
- You will only need a **handful of commands** in your daily work:

```
1 git status # show the working tree status
2 git diff my_file # show what has changed
3 git add my_file # add file contents to the index
4 git commit # record changes to the repository
5 git log # show the commit history
```

Configuration

`git-config` — Get and set repository or global options

- You can configure certain (global) settings for your local Git client
- For example, you can set the username and mail attached to each commit

```
1 git config --global user.name "Your Name"  
2 git config --global user.email "youremail@yourdomain.com"
```

- The concept of authorship is widely used on platforms, such as GitHub or GitLab



Hands-on session

1. Set the user name and email address in your local Git client
2. Create a directory and initialize a Git repository
3. Create a file and commit it to the repo
4. Change the file, inspect the differences, and commit the changes

On Levante: `module load git`



Branches

`git-branch` — List, create, or delete branches

- Branches are names that point to a certain commit
- They are not unique, and they can change over time
- Encapsulate the changes required for a feature/bugfix
- Allow incremental development without impacting other branches



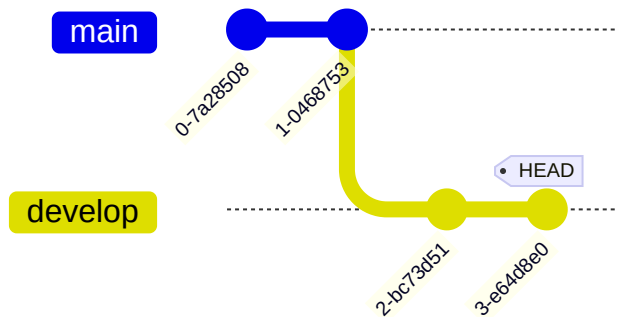
Switching branches

Create a new branch `develop` to work on a feature

```
1 git branch develop
```

Switch to the new branch (changing the `HEAD`)

```
1 git switch develop
```



Show the differences to the `main` branch

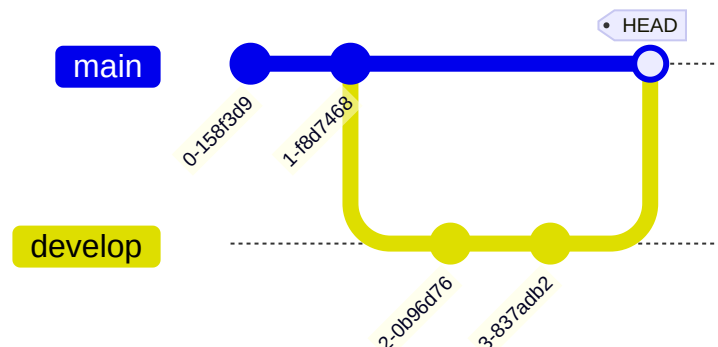
```
1 git diff main
```

Merge two branches

`git-merge` — Join two or more development histories together

- Include changes from another branch into the current one
- Usually creates a “merge commit” with two parent commits

```
1 git switch main
2 git merge develop
```



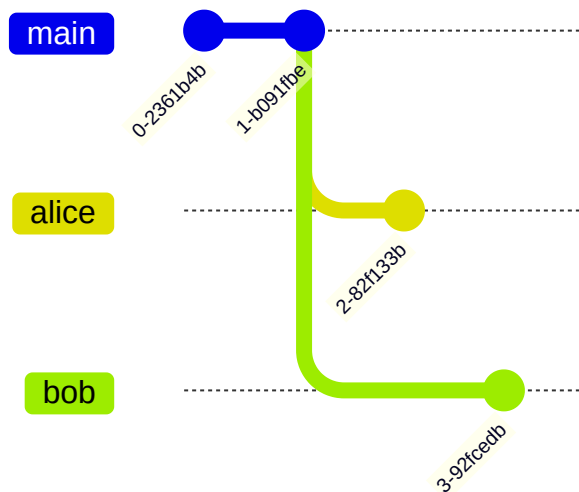
Hands-on session

1. Create a branch
2. Commit something to the branch
3. Merge the branch into `main`
4. Check the log of the `main` branch
5. Delete the branch (`git branch --help`)



Conflicts

Collaboration can lead to disagreements¹



Alice fixes an obvious error in `file.txt`

```
1 -This course is lame
2 +This course is nice!
```

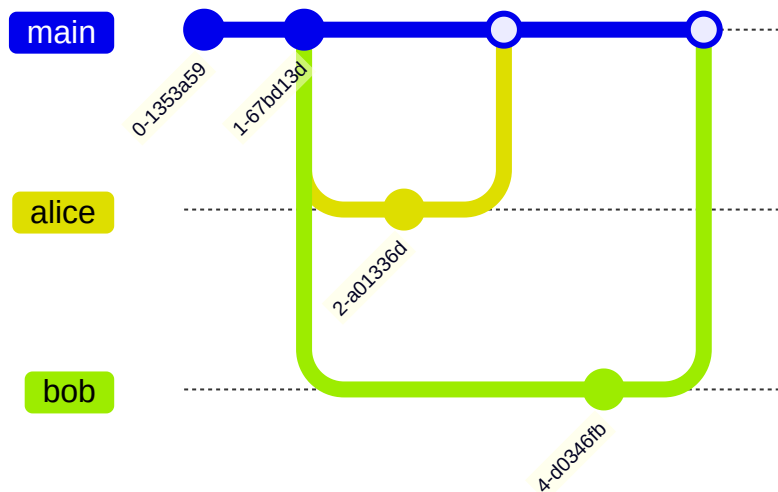
Bob is doing the same

```
1 -This course is lame
2 +This course is awesome!
```

1. Your past-self is one of your most frequent collaborators

Conflicts

This creates a conflict when merging both branches



- 1 Auto-merging file.txt
- 2 CONFLICT (content): Merge conflict in test.txt
- 3 Recorded preimage for 'file.txt'
- 4 Automatic merge failed; fix conflicts and then commit the result.

Solving conflicts

Solving conflicts requires your decision

```
file.txt
```

```
1 <<<<<<< HEAD
2 This course is nice!
3 =====
4 This course is awesome!
5 >>>>>>> bob
```

Solving conflicts

Solving conflicts requires your decision

```
file.txt
```

```
1 This course is awesome!
```

After **resolving the conflict**¹, you have to commit your changes

```
1 git add file.txt
```

```
2 git commit
```

1. All conflict markers need to be removed

Hands-on session

1. Create a `file.txt` in two different branches, each with different content¹
2. Merge both branches into `main` (CONFLICT)
3. Resolve the conflict and commit your changes
 1. Make sure to **commit** the changes in each branch

Best practices



Atomic commits

Commits should only deal with **one task** — one logical unit

- Ensure regular commits to track progress effectively
- Commit changes independently for clarity and easier management
- Commit self-consistent (i.e. working) states



Atomic commits

- A single logical unit is not the same as a file
- You can use Git to interactively add parts of a file

```
1 git add -p hello.txt
```

```
1 diff --git a/hello.txt b/hello.txt
2 index a042389..cd08755 100644
3 --- a/hello.txt
4 +++ b/hello.txt
5 @@ -1 +1 @@
6 -hello world!
7 +Hello world!
8 (1/1) Stage this hunk [y,n,q,a,d,e,p,?]?
```



Commit messages

Write **meaningful** commit messages

- Use the **imperative mood** in the subject line (**what** is done)
- **Limit** the subject line to 50 characters
- Use the body to elaborate **why** changes have been performed



Commit messages

```
1 ice: Fix freeing uninitialized pointers
2
3 Automatically cleaned up pointers need to be initialized before exiting
4 their scope. In this case, they need to be initialized to NULL before
5 any return statement.
6
7 Fixes: 90f821d72e11 ("ice: avoid unnecessary devm_ usage")
8 Signed-off-by: Dan Carpenter <dan.carpenter@linaro.org>
9 Reviewed-by: Jiri Pirko <jiri@nvidia.com>
10 Reviewed-by: Simon Horman <horms@kernel.org>
11 Signed-off-by: Tony Nguyen <anthony.l.nguyen@intel.com>
```

Example from kernel.org



Large (binary) files

You should not commit large (binary) files

- Git stores every version of a file in the repository
- Binary files are hard to meaningfully compare (`diff`)
- Use a `.gitignore` file to exclude file patterns:

```
.gitignore
```

```
1 *.nc  
2 plots/
```



Decentralization



Decentralization

- Git is a **decentralized** version control system
- Each repository contains the full project history
- *Technically* there is no single point of truth¹

1. *Practically* teams agree on one central repo



GitHub and GitLab

- There are many services to host Git repositories
- They offer plenty of additional functionality (merge/pull requests, code review, automated testing, ...)
- GitHub offers the largest user base, while GitLab can be self-hosted

This lecture is hosted on the [DKRZ GitLab](#)



Tracking a remote repository

Add a remote repository¹ to a local repository

```
1 git remote add origin <PATH_TO_REPO>
```

1. You have to create the remote repository

Clone an existing remote repository

```
1 git clone <PATH_TO_REPO>
```

Syncing with a remote repository

- Push your local references to the remote repo

```
1 git push origin <YOUR_BRANCH>
```

- Pull remote changes to your local repo

```
1 git pull origin <YOUR_BRANCH>
```



Syncing with a remote repository

- Push your local references to the remote repo

```
1 git push -u origin <YOUR_BRANCH>
```

- Pull remote changes to your local repo

```
1 git pull
```



Hands-on session

1. Generate a **new** SSH Key pair (ssh-keygen) for use with GitLab
2. Add the public SSH key to your DKRZ GitLab account
3. Configure your SSH client to use the new private key
4. Open your personal Git repo¹ in your web browser
5. Follow the instructions to “Push an existing Git repository”

1. https://gitlab.dkrz.de/gss/students2026/<YOUR_USERNAME>



Configure your local SSH client

- Configure your local SSH client to always use your private key for GitLab
- The key will be added to your SSH agent automatically when needed

```
~/.ssh/config
```

```
1 Host gitlab.dkrz.de
2     Hostname gitlab.dkrz.de
3     User git
4     IdentityFile <path_to_your_private_key>
5     AddKeysToAgent yes
```



Merge/Pull requests

- GitLab and GitHub provide an interface to discuss changes before a merge
- One can request reviews for specific people to get feedback
- **We will use merge requests to collect and review the exercises!**



Merge request on DKRZ GitLab

icon / icon-mpim / Merge requests / 1342

Update libfortran-support version

Code ▾

 Open **Yen-Chen Chen** requested to merge `update_fortran_support` into `master` 2 months ago

Overview **124** Commits **96** Pipelines **165** Changes **80**

5 unresolved threads ^ ▾ ⋮ Add a to do

This MR updates `libfortran-support` to version 1.1.0.

The changes includes:

- Add OpenACC support, move `mo_fortran_tools` to `libfortran-support`: [icon-libraries/libfortran-support!3](#) (merged)
- Adapt unit test for NAG compiler: [icon-libraries/libfortran-support!55](#) (merged)
- Downgrade CMake version: [icon-libraries/libfortran-support!58](#) (merged)
- Repository set up
 - changelog automation: [icon-libraries/libfortran-support!56](#) (merged), [icon-libraries/libfortran-support!65](#) (merged)
 - code coverage: [icon-libraries/libfortran-support!57](#) (merged)
 - repository badges: [icon-libraries/libfortran-support!59](#) (merged)
 - Add unit tests: [icon-libraries/libfortran-support!63](#) (merged)

This MR should be merge after [icon-libraries/libfortran-support!62](#) (merged) is merged

Edited 1 month ago by Yen-Chen Chen



Description

Assignee

 Yen-Chen Chen

Developer

Reviewer

 Sergey Kosukhin

Reviewer

Labels

None

Milestone

None

Time tracking

No estimate or time spent

6 Participants



✓ Merge request pipeline #64349 passed

Merge request pipeline passed for [34977798](#) 1 hour ago



Take-home messages

- Git has a learning curve — the more you use it, the more natural it becomes
- It is the *de facto standard* for version control across science and industry
- Use it for personal version control — and get collaboration features “for free”



Shotgun buffet



I am just gonna throw a bunch of stuff at you. Take what
you might find interesting.
— *Scott Chacon*



Rebase vs merge

`git-rebase` — Reapply commits on top of another base tip

Instead of merging branches, one can also **rebase**



Rebasing retains a linear history by **changing the commit history (!)**

Forks

- A fork is a copy of a repository on server side
- Used to work on public repositories without granting ownership
- Standard names for locally defined remotes:

```
1 origin https://github.com/lkluft/numpy (fetch)
2 origin https://github.com/lkluft/numpy (push)
3 upstream https://github.com/numpy/numpy (fetch)
4 upstream https://github.com/numpy/numpy (push)
```



Git Submodules

- Let you embed a Git repo as a subdirectory of another
- Keep your commits separate
- Avoid them unless you have a *specific, strong reason*
- Heavily used in ICON development

Cloning a repo and it's submoduels

```
1 git clone --recursive <PATH_TO_REPO>
```

Update submodules after changing/updating a branch

```
1 git submodule update
```



Further reading

- [The official git documentation pages](#)
- [Introductions by GitLab](#)
- [Software Carpentry on Git](#)
- [Git Will Finally Make Sense After This \(YouTube\)](#)

