Git

Version control, debugging, collaborating

Git

- Started in 2005 by Linus Torvalds
- Open-source, holistic, distributed
- Created from necessity
- Able to keep track of giant projects

Git

- Good mood:

  “"global information tracker": [...] Angels sing, and a light suddenly fills the room.

- Bad mood:

  "goddamn idiotic truckload of sh*t": when it breaks.

Version control

- Record of changes
  - File created, deleted
  - File changed at line number *x*
- Creates history
- Creates reproducibility

```
a92b27e Correct GitHub repo link
c29ea11 Merge branch 'main' into feature/day-1-git
c9a1bf6 Typo in index
31e94b2 Merge branch 'develop'
a172f76 Add team information
aac1429 Merge branch 'develop'
bdaf6b9 Forgot a period
90e4072 Add preparation and practical sections
5a5d3f3 Add a landing page
```

Git repositories

- A project is stored in a ***repository***
- Repositories can be shared online (Github or other)
- Shared files but also history

Git Commit

- git commit
- Commit means commit
- Mistakes are okay
- A ‘clean’ history vs. a clear history

A good commit

- One commit =
  - One chunk of functionality
  - Small
  - Clear message:

```
470f552 add csv file class
b12cb7a add analysis function for csv file
6d15e5e fix bug in analysis function
```

A bad commit

- One commit =
  - More than one functionality
  - Big
  - Unclear message:

```
470f552 csv and json support
b12cb7a analyse
6d15e5e fix bug
```

How to commit

- Initialise: make a repository

  ```
  git init
  ```

- Prepare: add files to *staging area*

  ```
  git add <file_name>
  ```

- Commit:

  ```
  git commit –m “message here”
  ```

Why is this important?

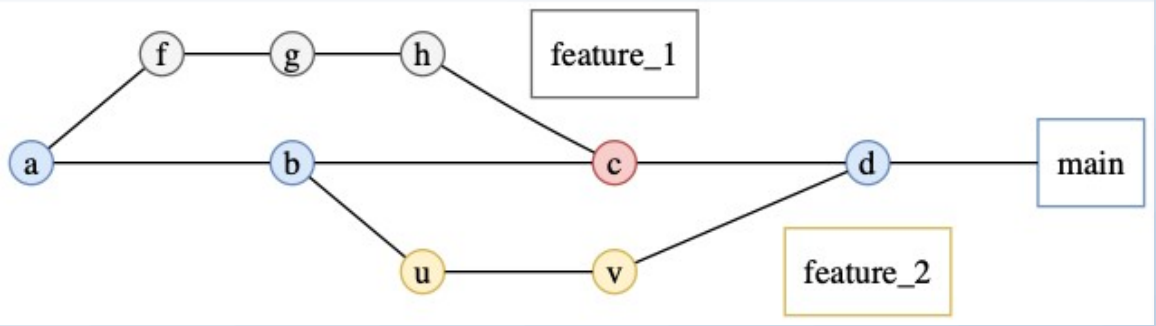
- For others
- For yourself

Branching Out

- Essential for both version control and collaboration:

  git branches

- Copy of the `main` branch
- Make changes, then merge back into the main branch



Branch naming

```
main

develop

feature/<feature_name>

bugfix/<bug_description>

docs/<documentation_topic>
```

Time for module 1 and 2!

Rewriting history

- **Undoing:** `git reset` & `git revert` ← need to know
- **Grafting:** `git cherry-pick` ← nice to know
- **Cleaning up history:** `git rebase` (advanced) ← for the nerds

Reset vs Revert

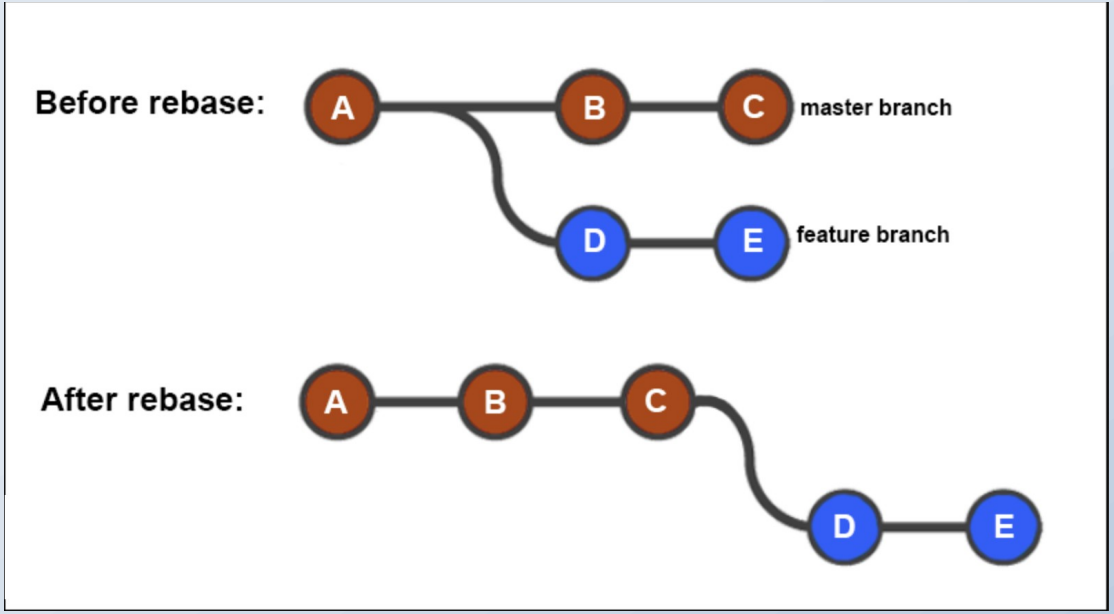
- Reset:
  - different modes
  - Undo without a trace
  - Only for private use
- Revert:
  - Undo with a trace (commit)
  - Changes that have already been shared

Git cherry-pick

- Good in specific circumstances:
  - Bugfixes from other branches needed locally
  - Feature commits from branches that will never be merged
- Bad in most other situations.

Git rebase

- Useful for cleaning up your history:



But use with caution…

Merge Conflicts

- Git tracks changes by line number
- So what if two commits change the same line?
  - You need to choose
- Conflict resolution: edit the file until it is in the state that you want.

Git remote (Github)

- Distributed network
- Sharing code
- Reviewing code
- Commands:
  - `git fetch, git pull , git push`

Fetch, pull, push

- `git fetch`

  gets new info about the remote repo

- `git pull`

  merges the remote commits into local

- `git push`

  updates remote with local commits

Pull Request

- a.k.a. merge request
- History can be viewed
- Comments and feedback from collaborators