Introduction to Git

Git: A distributed version control system

Advantages:

- Every repository has a full version history
- Most operations run locally
- Reliable data handling, ensuring integrity and availability
- Efficient data management for versions and branches
- Scalable collaboration mechanisms for large teams and complex projects

Caveats:

- Need to learn and understand the underlying model
- Not built for binary files or large media files

Learning objectives

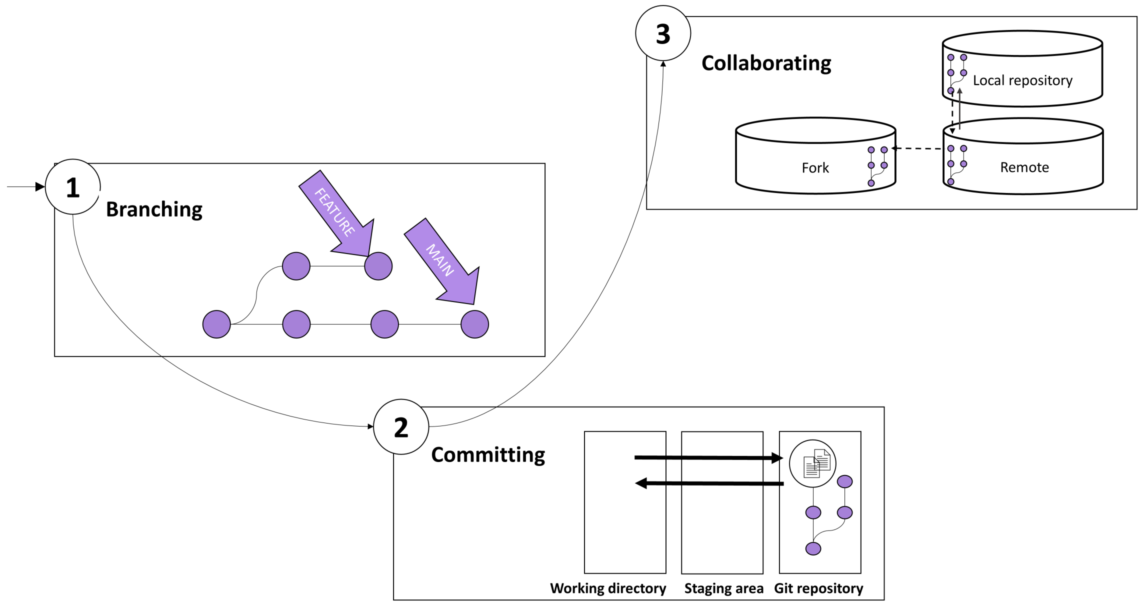
Understand and use git to develop software in teams.

Part 1: Branching
Part 2: Committing
Part 3: Collaborating

Each part starts with the **concepts** before the **practice** session.

In the practice sessions:

- Form groups of two to three students
- Work through the exercises
- Create a *cheat sheet* summarizing the key commands

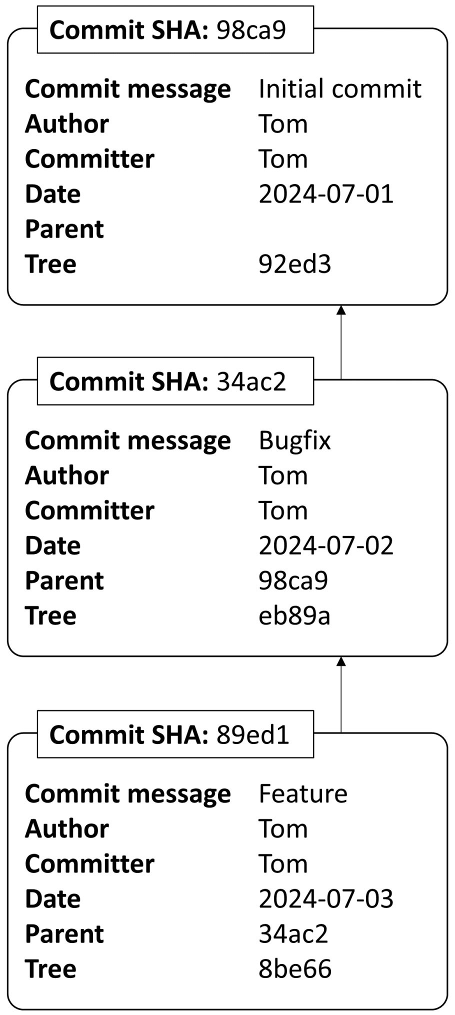


> * Note: This session is based on our unique and peer-reviewed approach.

Part 1: Branching

Commits

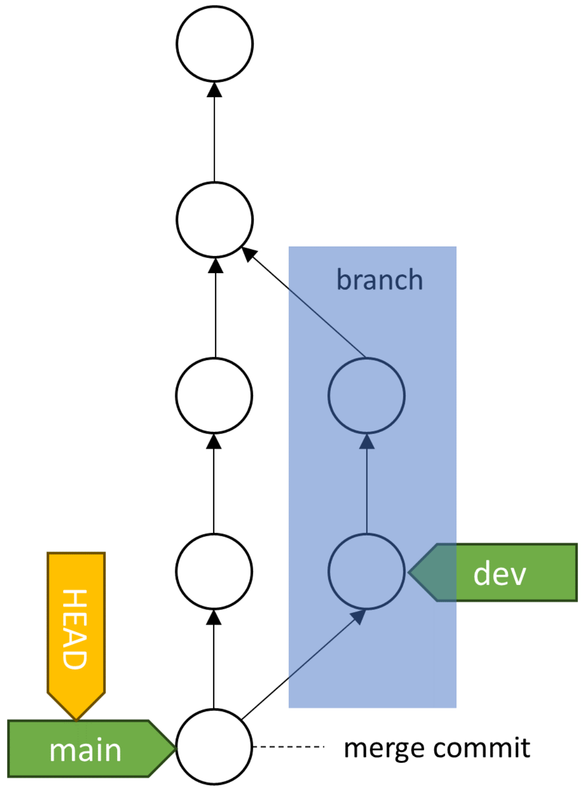
- A **commit** refers to a snapshot (version) of the whole project directory, including the meta data and files
- Commits are identified by the **SHA** fingerprint of their meta data and content*, e.g., `98ca9`
- Commits are created in a sequence, with every commit pointing to its **parent** commit(s)
- The **tree** object contains all files (and non-empty directories); it is identified by a SHA hash
- Commits are created by the **git commit** command



* If any of the meta data or content changes, the SHA will be completely different.

The DAG, branches, and HEAD

- Commits form a **directed acyclic Graph (DAG)**, i.e., all commits can have one or more children, and one or more parents (except for the first commit, which has no parent). Closed directed cycles are not allowed.
- With the **git branch <branch-name>** command, a separate line of commits can be started, i.e., one where different lines of commits are developed from the same parent. The branch pointer typically points at the latest commit in the line.
- With the **git switch <branch-name>** command, we can select the branch on which we want to work. Switch effectively moves the HEAD pointer, which points to a particular branch and indicates where new commits are be added.
- With the **git merge <other-branch>** command, separate lines of commits can be brought together, i.e., creating a commit with two parents. The *merge commit* integrates the contents from the *<other-branch>* into the branch that is currently selected. The *<other-branch>* is not changed.
- Per default, Git sets up a branch named "main".

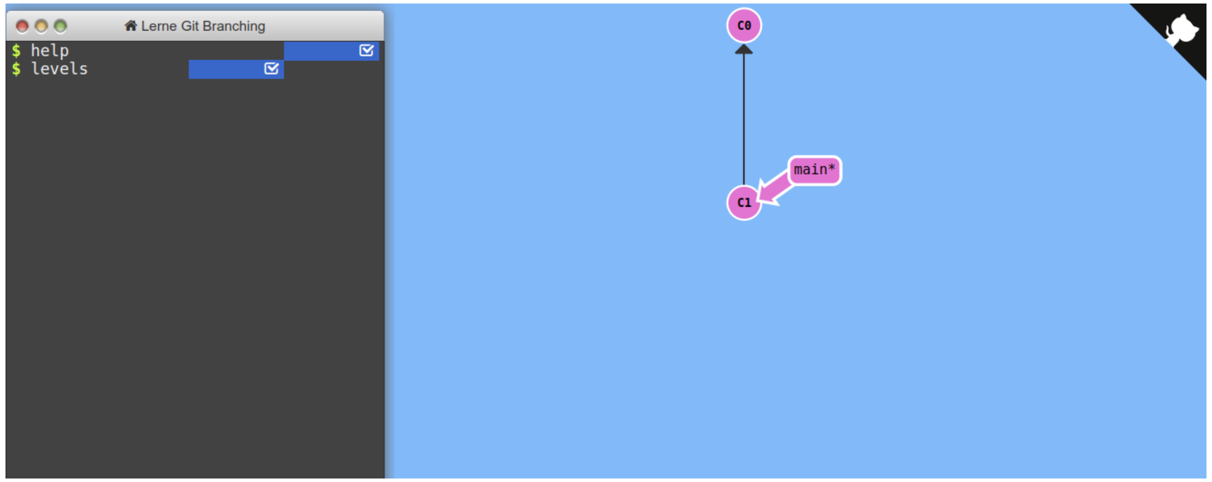


Note: Arrows point from children to parent commits.

Practice: Branching

To practice git branching, we use the learn-git-branching tutorial.

Complete the first two levels on branching, merging, and navigating in the git tree.



```
$ help
$ levels
```

NOTE: You can type "undo" when you made a mistake.

Part 2: Committing

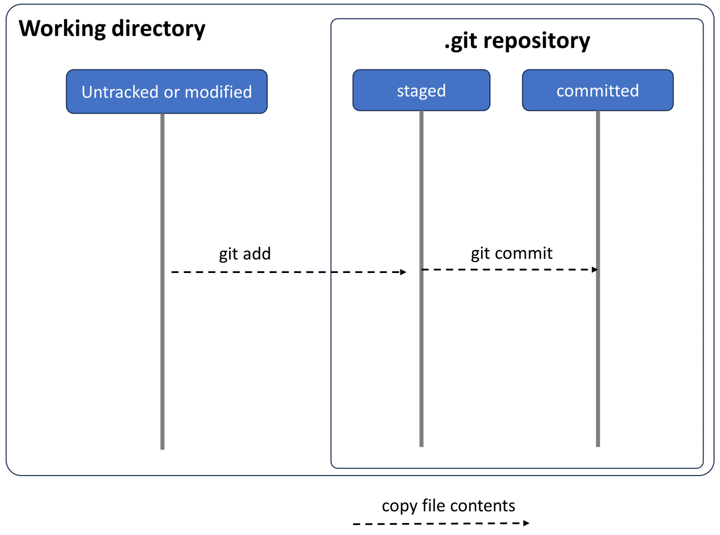
The working directory and .git repository

All working file contents reside in the working directory; staged and committed file contents are stored in the `.git` directory (a subfolder of the working directory).

Git allows us to stage (select) specific file contents for the next commit.

- With **git add <file-name>**, contents of an *untracked or modified* file are copied to the `.git` repository and added to the staging area, i.e., explicitly marked for inclusion in the next commit.
- With **git commit**, *staged* files contents are included in a *commit*.

The **git init** command creates the `.git` directory.

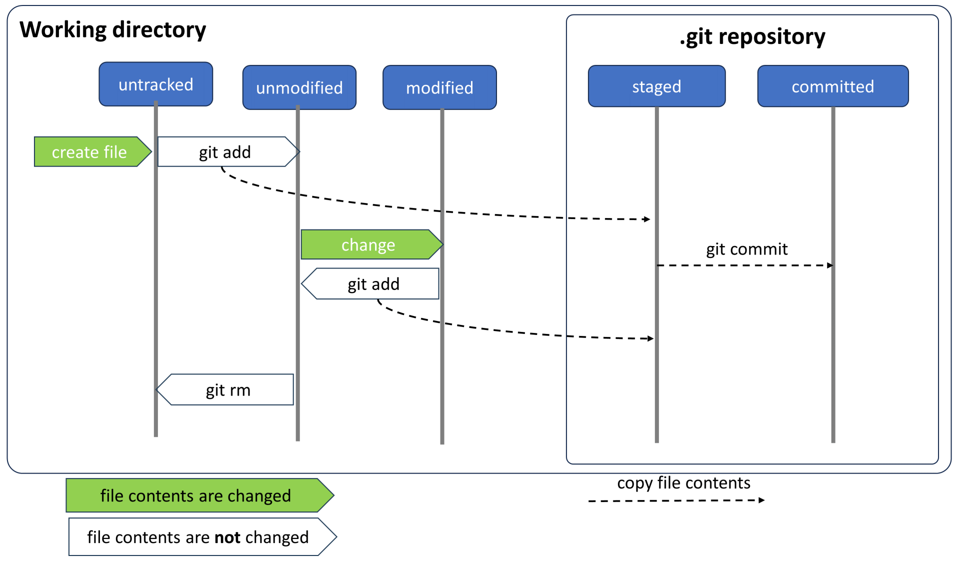


The three states of a file

Files in the working directory can reside in three states:

- New files are initially **untracked**, i.e., Git does not include new files in commits without explicit instruction.
- With *git add*, file contents are staged and the file is tracked. Given that the file in the working directory is identical with the staged file contents, the file is **unmodified**.
- When users change a file, it becomes **modified**, i.e., the file in the working directory differs from the file contents in the staging area.
- With *git add*, the file contents are staged again, and the file becomes **unmodified**.
- With *git rm*, files are no longer tracked.

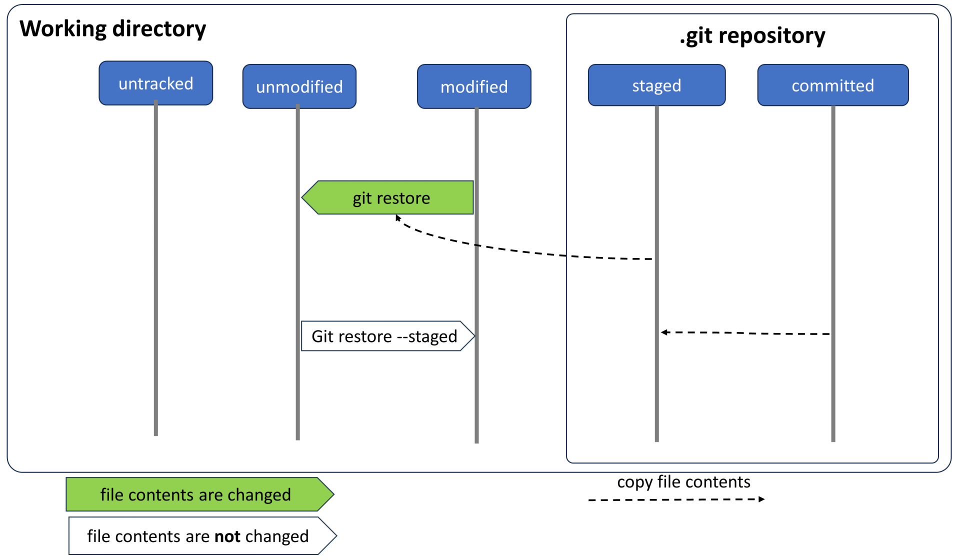
Note: *git add* and *git rm* do not change the contents of the file in the working directory.



Resetting changes

To undo changes that are not yet committed, it is important to understand whether they are staged or unstaged:

- If changes are not yet staged, the file is currently *modified*. A **git restore <file-name>** replaces the file in the working directory with the staged version. As a result, the file is *unmodified* because it corresponds to the staged file.
- If the file is currently *unmodified*, a **git restore --staged <file-name>**, Git discards the staged changes by using the last committed version. The file contents in the working directory do not change, but the file becomes *modified* because it differs from the staged version.

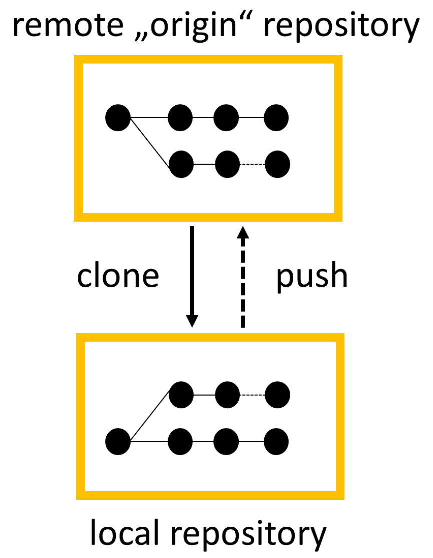


Part 3: Collaborating

Collaborating

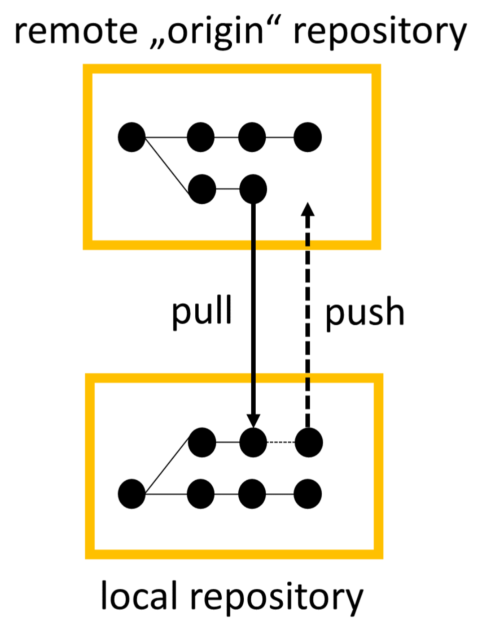
- The distributed model of Git means that every repository has a full version history, (almost) all operations can be completed locally, and every repository can be developed autonomously.
- To collaborate, a *remote* repository is needed, initially named "origin"
- If the remote repository exists, the **git clone** command retrieves a local copy
- To create a remote repository (named "origin"), and push a specific branch:

```
git remote add origin REMOTE-URL
git push origin main
```



Collaborating on branches

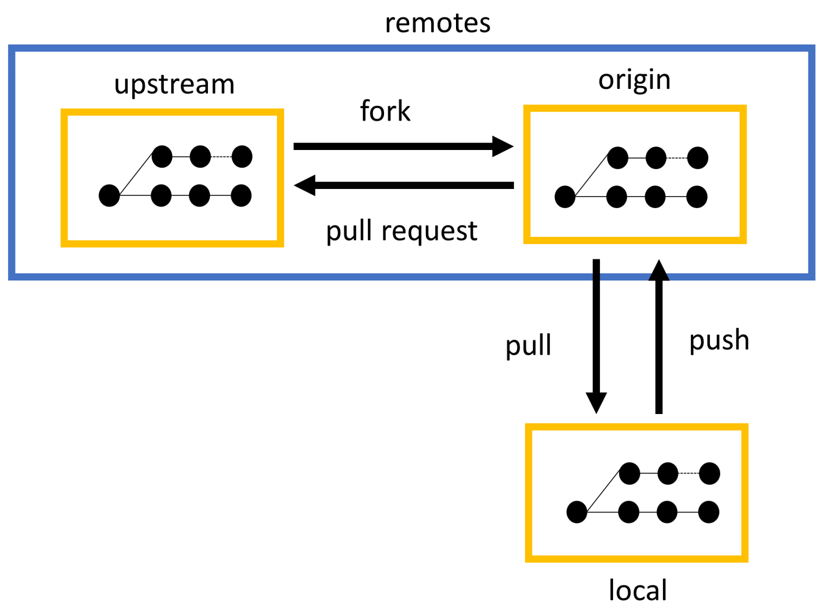
- To retrieve changes, use the **git pull** command
- To share changes, use the **git push** command
- Most remote operations, including pull, push, pull requests refer to branches
- In some cases, **branches must be selected explicitly**, and in other cases, git automatically selects branches, i.e., it remembers the typical branch to pull or push



Collaborating with forks

This model works if you are a maintainer of the remote/origin, i.e., if you have write access.

- In Open-Source projects, write-access is restricted to a few maintainers
- At the same time, it should be possible to integrate contributions from the community
- **Forks** are remote copies of the upstream repository
- Contributors can create forks at any time and push changes
- Contributors can open a **pull request** to signal to maintainers that code from the fork can be merged
- Pull requests are used for code review, and improvements before code is accepted or rejected


remotes
upstream
fork
origin
pull request
pull
push
local