Git

- *distributed* version control system (you can use it without central server)
- free and open source (GPLv2)
- was created by Linus Torvalds in 2005
- most widespread system in use today
- command line tool: `git`
- various GUIs exist
- different server solutions offer collaboration features: GitHub, GitLab, Bitbucket

Basic Concept of Git

- git stores *snapshots* of data
- it uses the SHA-1 hash algorithm to create *checksums* for the data
- this includes the complete *history*, thus, allows to compare it
- git is *distributed*, i.e. all necessary information is local to your computer and a remote server is only required to collaborate with others

Git Terminology (basics)

repository	a directory containing files and the corresponding history of the project
commit	one snapshot of data, i.e. a single point in Git history. *to commit* is the action of storing a new snapshot
branch	a series of commits defining the history. The most recent commit is the *tip* of the branch
master	is the usual name of the main branch of a repository
to merge	means bringing together two different branches sharing a common history. A successful merge results in a commit with the tips of the merged branches as parents (except if it was fast-forward).
checkout	the action of switching to a articular revision, or getting a single file from it

see also `https://git-scm.com/docs/gitglossary`

Local Advantages of working with Git

You program code, test it, and when satisfied *commit* it to the repository.

The advantages are:

- Each commit has a *message*, *author* and *date* to identify it.
- \+ You can come back to any commit at any time
- \+ You can compare (*diff*) two commits, i.e. show which lines in which files have changed.
- \+ For a single file, you can show who changed which line and when (*blame*).
- \+ You can undo (*revert*) changes done in any commit in the past.

Git generally only adds data

- nearly all actions add data to the repository
- once something was added it is hard to take out again
- it is difficult to loose data

Don't hesitate to try out stuff, nearly all Git commands are save and will not remove any data by default (can often be overridden by an option).

Do not add stuff to the repository which should not be there, e.g. a 1GB simulation data file.

How Git stores data

Assume you have a Git repository in the directory `~/femTU`

- all data Git stores is contained in `~/femTU/.git`
- data is stores as a collection of *objects*
- an *object* is identified by its SHA-1, thus, cannot be changed

Working Tree

The *working tree* comprises all regular files in your repository.

Files can be in different states

committed	the data is safely stored in the local Git database
modified	file contains changes, which are not committed yet
staged	means marked to go into the next commit (i.e. it was added to the *index* or *staging area*)
untracked	means that Git does not know the file yet

The working tree may be

clean	i.e. does not contain changes, or
dirty	e.g. containing new or changed files.

HEAD	denotes with which commit Git compares data, and where the next commit will be placed

Committing changes

Committing is 2 step process

1. *add* files to the index (= staging area) to mark them for commit
2. *commit* all *staged* files, which includes writing a commit message

- a commit is the smallest changeset Git operates on (e.g. can be reverted)
- commit often
- commit messages should reflect what the changes are about
- make changes by topic and commit them
- do not combine unrelated changes in one commit

Branching and merging

- Branches are most useful when working on larger projects within a team
- Branches allow to collect a series of successive commits under a common name

Scenario where you might want to use a branch . . .

You plan a new feature, but it will need substantial changes and will possibly break a few things. . .

- Create a new branch and commit to it whenever you reach important points.

+ you have your working version safely committed to your original *master* branch
+ commits can be reverted and gone back to
+ you can always compare to the working version on *master*
+ you can merge your changes into master once satisfied

How does a merge work?

what Git does if you merge one branch into another . . .

1. use checksums to compare the branches, which include the complete (reachable) history, to determine where branches diverged (=common ancestor)
2. determine the changesets introduced between common ancestor and the head of each branch
3. compare the two changesets
   - changes only present in one changeset are incorporated into the new merge commit
   - if both branches introduce changes in the same location (line), there is a conflict, which must be resolved manually

\+ a merge is automatic if there are no conflicts

\+ Git marks out all conflicts to resolve

Collaborating with others

- The true power of Git is revealed when collaborating with (many) others on the same code
- Git allows you to interact with an arbitrary number of *remote* repositories
- the checksumming mechanism allows Git to determine up to which point repositories are equal
- there a are many web-services offering to host Git repositories, e.g. github.com, bitbucket.org, codebasehq.com, gitlab, . . .
- they often offer additional features like: project wiki, issue tracker, testing environment, . . .

Git Terminology (collaboration)

remote (repository) is used to track the same project but is located somewhere else

tracking means following, e.g. a local branch might have an *upstream-tracking branch* defined which it follows

upstream and downstream define in which way data flows between two repositories

origin denotes the default upstream repository

fetch Fetching a branch means to get the branch's head ref from a remote repository, to find out which objects are missing from the local object database, and to get them, too.

pull Pulling a branch means to fetch it and merge it.

push means putting data into the remote repository

The git command

- Git is a command line tool
- GUIs exist, but might only support a subset of functionality
- git has subcommands with different options, e.g. `git commit`, `git clone`, `git branch`, `git help`, `...`

Get a *cheat sheet* as quick command reference!

Frequently used Git commands

not causing any action . . .

git status	shows the status of files in the working tree
git log	show the commit history
git diff	show differences between files
git help	get help . . .

associated to the respective action . . .

git add	add a file to the index (for committing)
git commit	create a commit
git branch	List, create, or delete branches
git checkout	Switch branches or restore working tree files
git push/pull/fetch	interact with remote repository

Clone

To *clone* a remote repository means to copy it, i.e. to make a local copy of it available.

- use the command `git clone <url>`
- this automatically sets up tracking information, e.g. between *master* (local repo) and *origin/master* (upstream remote)
- git knows different data exchange protocols: file (local), http(s), ssh, ...
- protocol is determined by remote `<url>`, e.g.
  `ssh://[user@]server/project.git`,
  `https://example.com/gitproject.git`
- authentication differs depending on protocol

Using the *ssh* protocol is recommended: create and upload you key to gitlab!

Getting started with Git

Best done by working with it: You know the theory now, find out how to apply it. . .

Recommended references

- Git's help system `git help <command>`
- The git book `https://git-scm.com/book/en/v2`
- cheat sheet: `https://about.gitlab.com/images/press/git-cheat-sheet.pdf`
- `http://git.mdmt.tuwien.ac.at/ftoth/gitlab-intro`
- Gitlab help pages: `http://git.mdmt.tuwien.ac.at/help#getting-started-with-gitlab`
- ask google about specific questions

Try the *git-game*: `https://github.com/git-game/git-game`

.gitignore

- Git can be told to ignore certain files
- Ignored files are not shown as *untracked* and will not be added
- This is done by `.gitignore` files in the repository
- The files support pattern matching

Use gitignore files, e.g. for auto-generated stuff, binary file types, large data, etc., and track the gitignore with Git.

example `.gitignore` file

```
# this is a comment line
# ignore all PDF files
*.pdf
# don't ignore a special file (negate pattern with !)
!special.pdf
```

Changing Git history

- you can change the last commit by `git commit --amend`
- you can *rebase* commits, i.e. put them somewhere else

Never change history already shared with others

If they have based their work on top of it, they will have to re-resolve all

see also: `https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History`

Git Terminology (advanced)

object the unit of storage in Git. Uniquely identified by its SHA-1 hash. Cannot be changed

cherry-picking means to choose a subset of changes out of a series of changes (typically commits) and record them as a new series of changes on top of a different codebase.

detached HEAD means you have checked out a commit not at the tip of a branch

fast-forward is a special merge in which only one branch contains changes (does not require a merge commit).

submodule separate repository inside a repository.

Rebase

One can *rebase* commits from one branch on top of another, e.g. using `git checkout feature` and `git rebase master` we rebase branch *feature* on top of *master*.

original

```
A--B--C     master
    \
     D--E   feature
```

after rebase

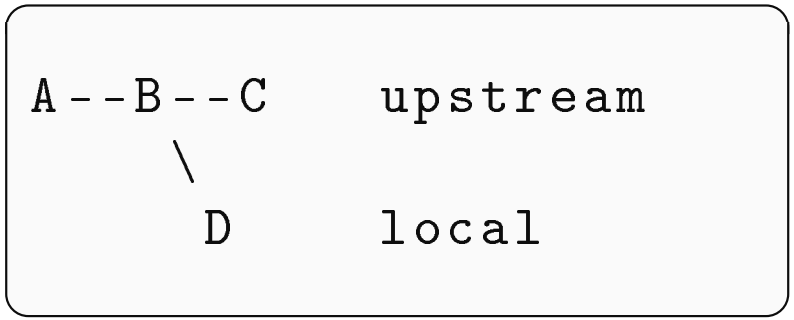
```
A--B--C--D'--E'    master
    \
     D--E          feature
```

Use the `-i` option to do an *interactive* rebase allowing you to

- combine commits
- change commit messages
- reorder commits

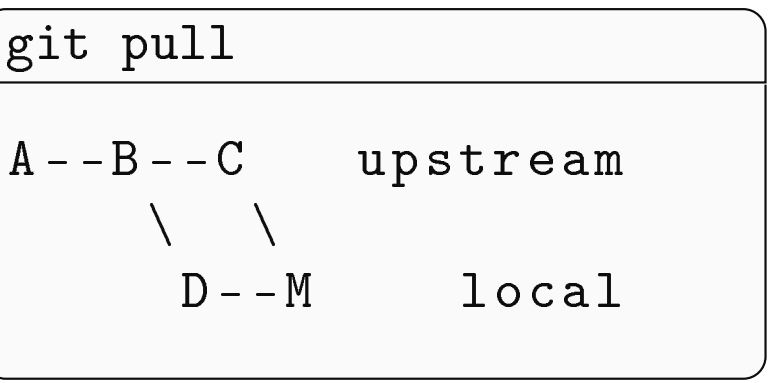
pull methods

Imagine you want to get an upstream change *C* into your local repo, where you have already made a commit *D*.

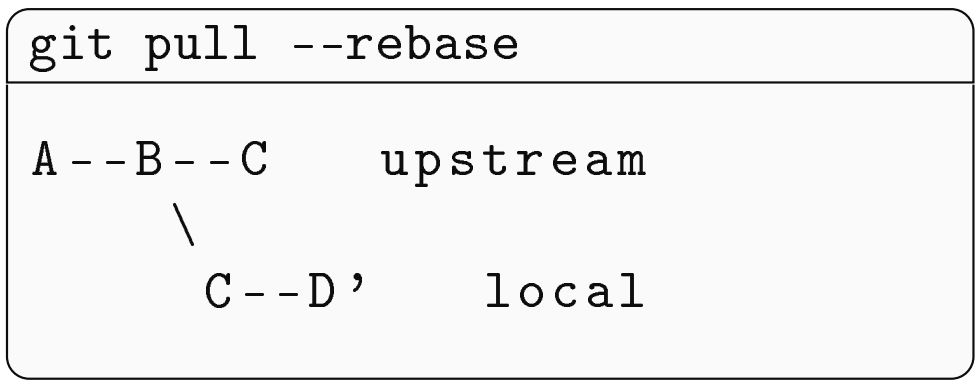


Git allows to pull with *merge* (=default) or *rebase*:

`git pull`



`git pull --rebase`



Pull with rebase to avoid unnecessary merge commits!

stash

- git refuses operations if they would overwrite uncommitted changes, e.g. switch branches, pull, ...
- sometimes you want do not want to commit (e.g. it's all a bit messy), but want to switch branches
- use `git stash` to store your modifications
- can be (re-)applied later using `git stash apply`
- the stash is a stack, i.e. can contain several layers of dirty work

tag

- a git *tag* is a reference to any object in the repository
- Typically it points to a *commit*
- Often used to denote release version, e.g. `v1.0`