git - Part I
Basics & Local Repositories

Stefan Pranger

stefan.pranger@student.tugraz.at

02. November 2020

Overview

What is git?

Quoting Wikipedia:

git is a *distributed version-control system* for tracking changes in source code during software development. ... goals include speed, *data integrity*, and support for *distributed*, *non-linear workflows*.

Examples of distributed version-control systems: *CVS*, *BitKeeper*, *SVN*, *git*, ...

What is git?

Quoting Wikipedia:

git is a *distributed version-control system* for tracking changes in source code during software development. ... goals include speed, *data integrity*, and support for *distributed*, *non-linear workflows*.

Examples of distributed version-control systems: *CVS*, *BitKeeper*, *SVN*, *git*, ...

git $\neq$ github, gitlab, etc.

What do we use git for?

You may generally use git for

- software projects,
  - in teams of developers or
  - even if you are working on a project on your own,
- making these accessable to the public,
- keeping track of the work on the project,
- trace and fix bugs,

What do we use git for?

You may generally use git for

- software projects,
  - in teams of developers or
  - even if you are working on a project on your own,
- making these accessable to the public,
- keeping track of the work on the project,
- trace and fix bugs,

but also for

- theses,
- storing configuration files,
- storing password files (`$> pass`),
- etc.

When are you going to use git?

- In almost every programming practical!
- But!
  - different platforms and
  - different hand-in requirements.

When are you going to use git?

- In almost every programming practical!
- But!
  - different platforms and
  - different hand-in requirements.
- Maybe in your everyday work!

When are you going to use git?

- In almost every programming practical!
- But!
  - different platforms and
  - different hand-in requirements.
- Maybe in your everyday work!

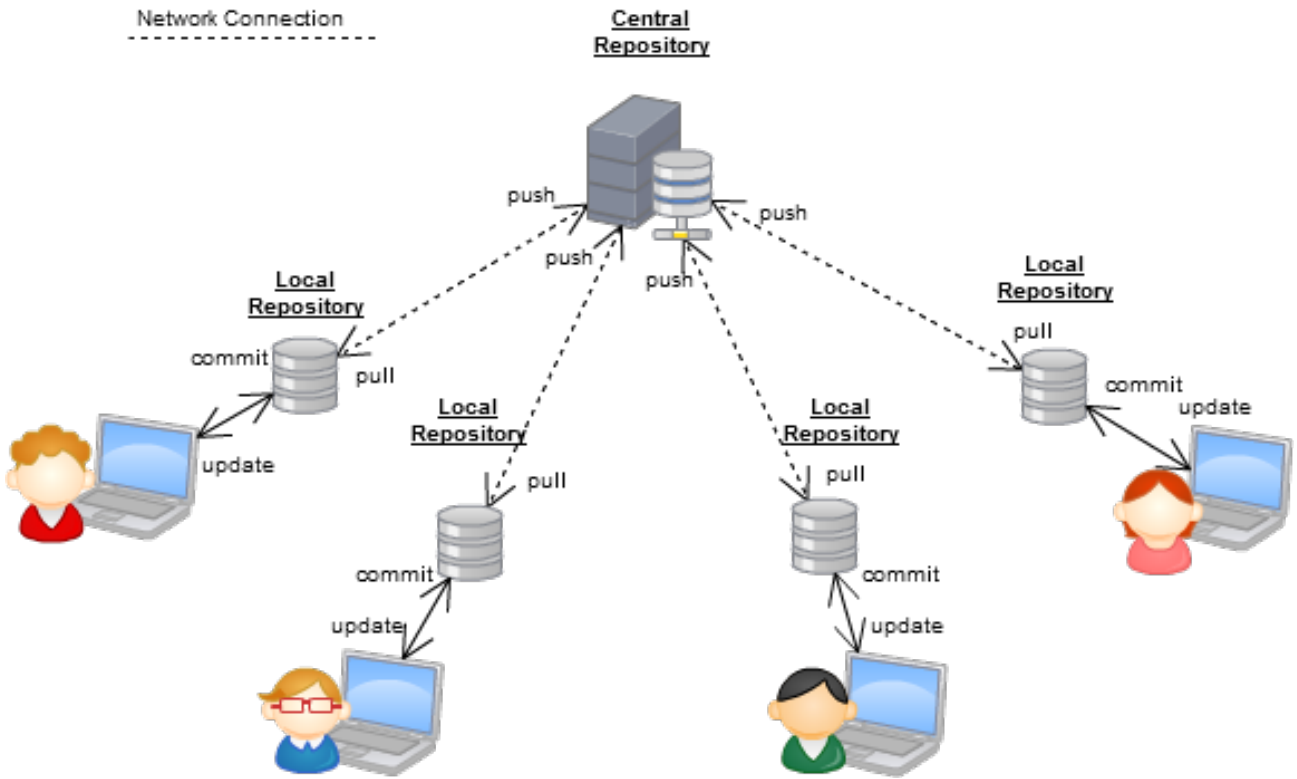
Why attend this course?

- Get to understand background,
- cover basic usage commands and
- preparation (*and hopefully hours of saved time and struggles!*).

Road Map

- *Today*: Local Repository and git basics
- 03. 11.: Branches, Remotes, non-linear workflow and working in a team
- 04. 11.: Miscellaneous: Tagging, Reflog, .gitconfig, Best Practices and Workflows, Hooks, Background, etc. . .

Distributed Systems



Credit: *https://jordankasper.com/lessons-learned-teaching-git/*

Similar to *DropBox*, *Google Drive*, *iCloud*, etc., but working very differently in the background!

Local Repository

- Almost all of your work happens here
- Keeps a history of your project by
- tracking any changes you make
- Divided into four states of stages

Repositories

The *heart* of a git project.

```
test>$ ls
drwxrwxr-x 40 4096 Sep 26 18:42 ..
drwxrwxr-x  3 4096 Sep 26 18:42 .
drwxrwxr-x  7 4096 Sep 26 18:42 .git
```

Repositories

The *heart* of a git project.

```
test>$ ls
drwxrwxr-x 40 4096 Sep 26 18:42 ..
drwxrwxr-x  3 4096 Sep 26 18:42 .
drwxrwxr-x  7 4096 Sep 26 18:42 .git
```

How can we *initialize* a git repository?

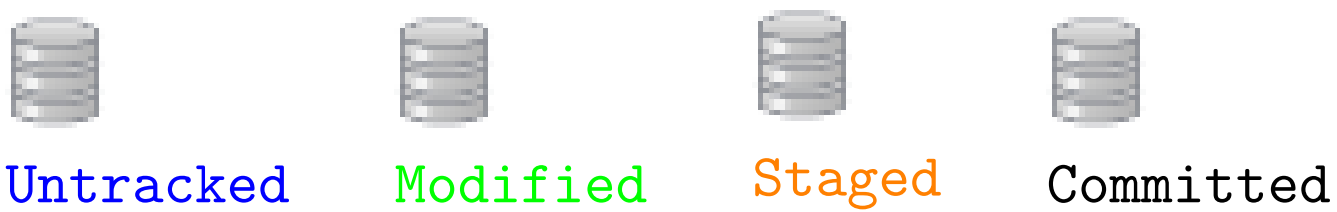
1. `$> cd <dir> && git init`
2. `$> git init <dir>`
3. `$> git clone <git-url>`

.git

```
test/.git$ tree .git
.git
├── branches
├── config
├── description
├── HEAD
├── hooks
├── info
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

Stages

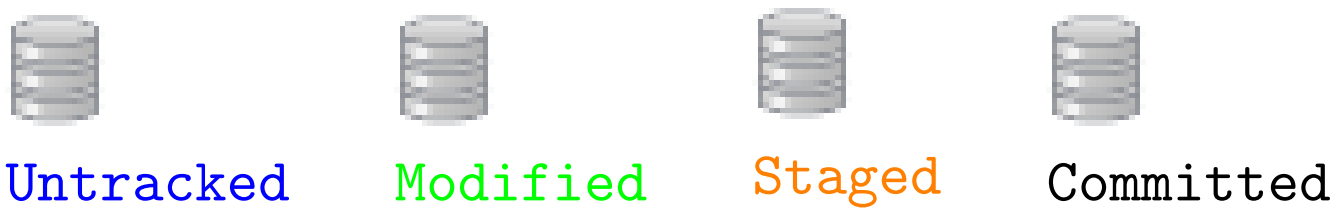
- Every file in your directory is either:
  1. `Untracked`
  2. `Modified`
  3. `Staged`
  4. `Committed`
  5. `(Unmodified or Ignored)`
- You move files between these stages
- git keeps track of moves from `Staged` to `Committed` (i.e. your *commits*)



Commits

What is a *commit*?

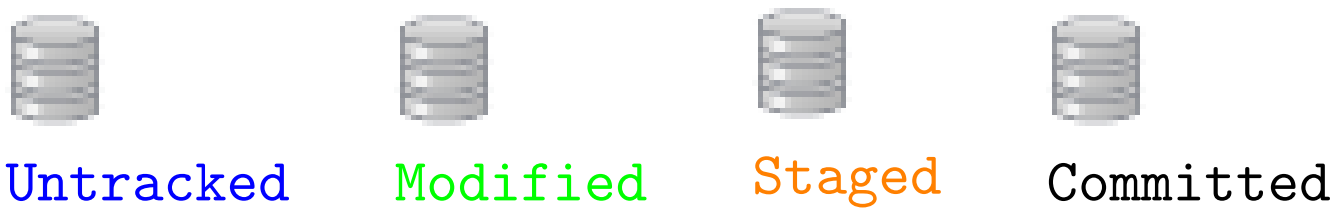
- One entry in the history of you project
- Saves your staged changes in your local repository
- Messaging interface for you and your team



Commits

What is a *commit*?

- Author (and Commiter)
- Message
- Date
- Committed Files



Commits

What is a *commit*?

- Author (and Commiter)
- Message
- Date
- Committed Files

Used for computation of `b0eead...`

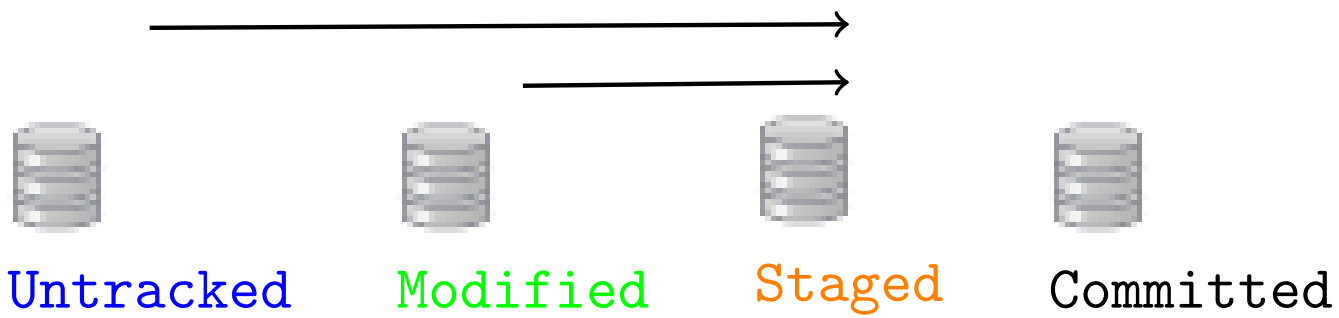
Commits internally

- Internally git manages these stages via three trees:
- `Untracked` and `Modified` belong to the *worktree*,
- The staging area (`Staged`) is also called the *index* and
- *HEAD* will point to your last commit.

Adding and Staging

Move files from Untracked/Modified to Staged:

- `$> git add <file,dir>`,
- `$> git add -u`, (*only from* Modified)
- `$> git add -p <file,dir>`



Committing

After doing some work on the project:

- check all changes via `$> git diff` or `$> git diff --staged`
- `$> git commit -m "<commit message>"`
- Better: `$> git commit`

git then adds a commit to your *log* and stores a *snapshot* of modified objects.

Status

- `$> git status`
- Shows you the current status of you working directory.
- Which files are *staged*, *modified*, etc.

Logs

- `$> git log`
- *By default*, shows you the commit history of you current branch.

Logs

- `$> git log`
- *By default*, shows you the commit history of you current branch.
- Since git is all about versioning and information ... we want this to be prettier!
- `$> git log --oneline`
- `$> git log --graph`
- ... and combined: `$> git log --oneline --graph`
- A bit more advanced:

```
$> git log --graph --pretty=format:'...'  ...
```

How to undo things

- git generally only adds data and information, but it is possible to undo, amend and even change the history!
- Obviously changing commited history is not considered to be best practive.
- You can also drop changes you have made completely.

git restore

- Moving things back from the index.
- `$> git restore --staged <file>`

git restore

- Moving things back from the index.
- `$> git restore --staged <file>`
- `$> git restore --staged --worktree <file>`
- `--staged` and `--worktree` tell git which steps to take

Amending the last commit

- `$> git commit --amend`
- This amends your last commit by adding the currently staged files.

Amending the last commit

- `$> git commit --amend`
- This amends your last commit by adding the currently staged files.
- Two important use cases:
  1. Fixing typos in the commit message and
  2. if you have forgotten to add files:

```
$> git commit
$> git add ...
$> git commit --amend
```

git reset

- git will only add files when you amend your last commit.
- What if you want to remove a file from a commit?

git reset

- `$> git reset (--soft|--mixed|--hard) <commit>`
- Three different options, `--mixed` being the default.
- `--soft` Undo all commits back until `<commit>` and keeps all files staged.
- This discards `$> git commit`.

git reset

- `$> git reset (--soft|--mixed|--hard) <commit>`
- Three different options, `--mixed` being the default.

- `--mixed` Undo all commits back until `<commit>` and unstages files.
- This discards `$> git commit` and `$> git add`

git reset

- `$> git reset (--soft|--mixed|--hard) <commit>`
- Three different options, `--mixed` being the default.

- `--hard` Undo all commits back until `<commit>` and **discards** all changes made.
- This discards `$> git commit`, `$> git add` and any changes to the files.

Untracking Files

- In case you have accidentally added a file:
- Use `$> git rm --cached` to untrack it.
- The `--cached` option will keep it in your working directory.

Untracking Files

- In case you have accidentally added a file:
- Use `$> git rm --cached` to untrack it.
- The `--cached` option will keep it in your working directory.
- If you want to permanently ignore the file:
  - `$> touch .gitignore`
  - `$> echo <filename> >> .gitignore`

Untracking Files

- In case you have accidentally added a file:
- Use `$> git rm --cached` to untrack it.
- The `--cached` option will keep it in your working directory.
- If you want to permanently ignore the file:
  - `$> touch .gitignore`
  - `$> echo <filename> >> .gitignore`
- Add `*.<filetype>` to exclude all files of the given type.

Summary

- What is version control?
- What is a distributed system?
- What is a local repository?
- What are the different stages a file may be in?
- What is a commit?
- What is a snapshot?
- How can we move files between stages?
- What is the git log?
- How can we fix errors?
- What does the `.gitignore` file do?