Version control with Git

Stefan Richter (University College London)

Git is a distributed version control program.

Why version control?

- Combine work of multiple collaborators
- Understand changes
- Support incremental development
- Compare and revert to earlier versions
- Backup
- Parallel versions
- Document development (for other developers and yourself, not for users)

→ **version control is awesome. Use it all the time.**

Practical introduction by example

We're going to walk you through an example. The things we show you here will teach you all you need to know to collaborate on your team project using Git.

Setting up

Create a Github account and create a public repository called `myrepo`. Then go to a clean folder on your computer and do:

```
>>> git clone https://github.com/<github_username>/myrepo.git
>>> cd myrepo

>>> scp -r user-guest@172.20.75.174:my_project/* .
```

At this point you could set some configurations.

```
>>> git config core.editor "emacs -nw" # or your favourite light-weight editor
>>> git config color.ui true

>>> git config --list   # check that it worked!
```

To make settings for all repositories on your computer, add the flag `--global` after `git config`.

You can also set your user name and email like this:

```
>>> git config user.name "Stefan Richter"
>>> git config user.email "stefan.richter@foo.bar"
```

Monitoring

Your first best friend in Git is the command `status`:

```
>>> git status
```

It shows you the files in the repository, both tracked and untracked by Git. Use this command all the time to know what's going on.

Your second best friend is `diff`. It shows you changes (differences) between versions. Without arguments, it shows all changes made to tracked files in the repository since the last commit.

```
>>> git diff
>>> git diff <path/to/file>
```

(`git diff` can also be used to show differences between arbitrary revisions. You can google it.)

Use

```
>>> git log
```

to see the commit history on your current branch. I use `git log -<number>` a lot to only show the `<number>` last commits, e.g.

```
>>> git log -3
```

Committing

Committing in Git works in two steps. First modified or untracked files are "registered" for the next commit by using `add`. This is called *staging*. The *staged* files are then committed with `commit`:

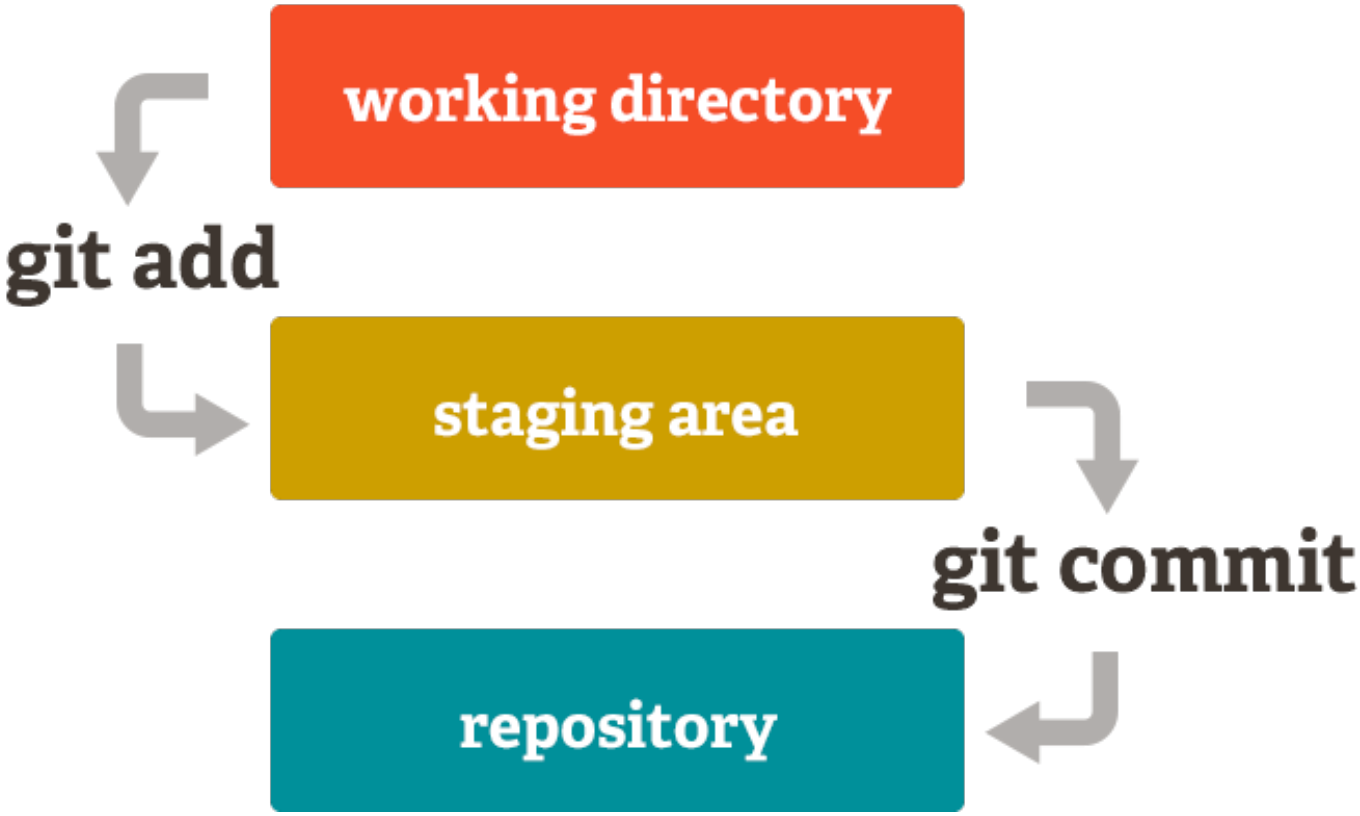

Image from https://git-scm.com (license)

```
>>> git add <path/to/file>   # file is now staged for commit
```

```
>>> git commit
```

Then write a commit message. We'll give you hints for what is a good message.

Good commit messages matter! Here are some good recommendations (weekend reading for you?).

Branches

To check which branch you are on:

```
>>> git branch     # see where we are!
>>> git branch -a  # what's the difference?
```

Create a new branch:

```
>>> git branch dev1   # dev1 is the name of the branch
```

Switch to the branch using `checkout`:

```
>>> git checkout dev1
>>> git branch   # see where we are!
```

To merge my changes into another branch (let's say, `master`):

```
>>> git checkout master
>>> git merge dev1
```

Working with remote repositories: sharing

See what our remote is:

```
>>> git remote     # what's our remote
>>> git remote -v  # some more info
```

To update the local repository (*pull* changes):

```
>>> git pull
```

To update the remote repository (*push* changes):

```
>>> git push origin master
```

A common workflow that your team could adopt:

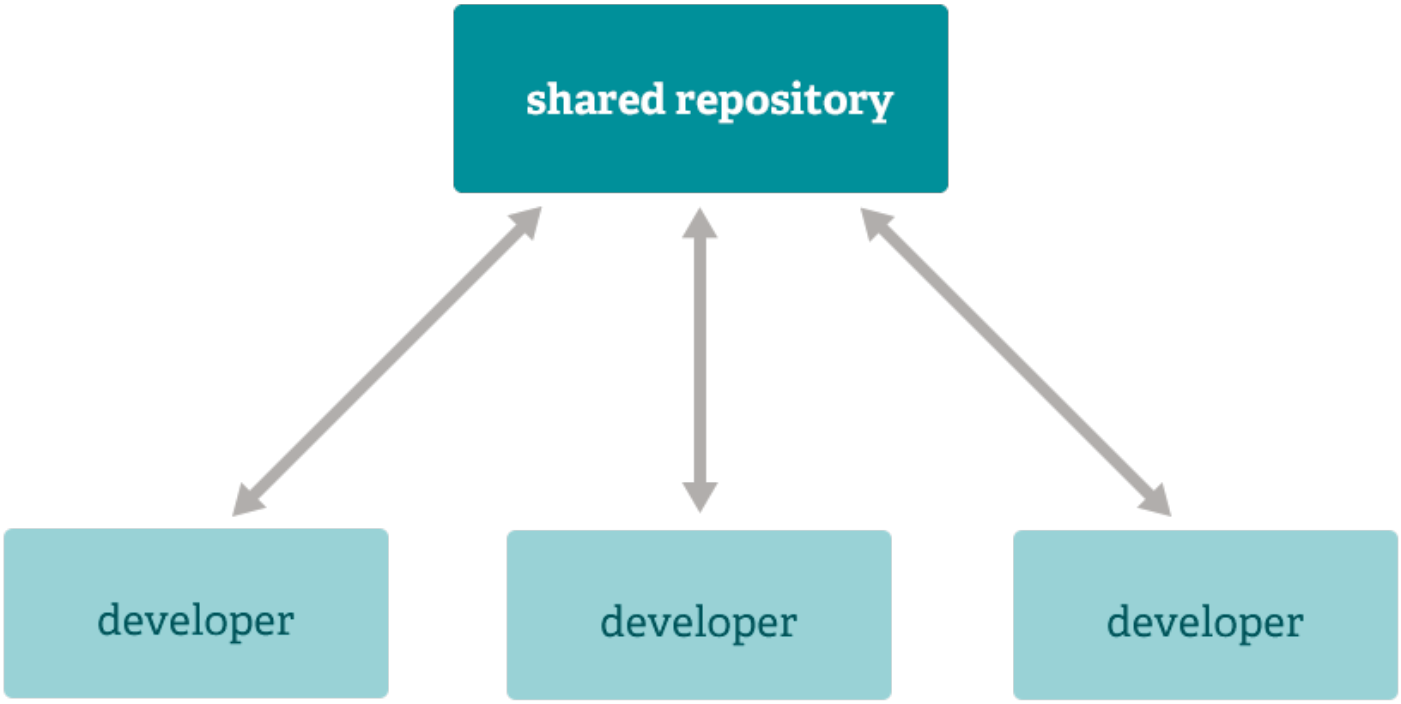

Image from https://git-scm.com (license).

Git's not perfect…

Git is used widely and has many powerful features, but it also has terrible downsides. You might already have noticed that it's quite difficult to use…

Here is a good post about problematic things in Git: https://stevebennett.me/2012/02/24/10-things-i-hate-about-git

There are good **alternatives** to Git: Mercurial (`hg`), which is better, and Bazaar (`bzr`), which I know nothing about.