A (very) short introduction to Git

Corso Strumenti Informatici Modulo B
2025/2026

David Goz

What is Git?

- Git is a ***distributed version control system*** (VCS). Version control (also known as revision control, source control, and source code management) is the software engineering practice of controlling, organizing, and tracking different versions in history of computer files, primarily source code text files, but generally any type of file;
- it basically keeps a "non-human-readable" database of the files you put under version control ("track") and provides commands to access and update that database;
- it was originally created by Linus Torvalds for version control during the development of the Linux kernel;
- it is open source, meaning you have the freedom to examine the code and see how it works.

Why version control?

- Understand software changes over time (i.e. support incremental development);
- combine work of multiple collaborators;
- backup;
- compare and revert to earlier versions;
- parallel software versions;
- …

Getting started - Installing Git [1/3]

Installing on Linux:

You can generally do so through the package management tool that comes with your distribution:

- on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:
  - `$ sudo dnf install git-all`
- on a Debian-based distribution, such as Ubuntu, try apt:
  - `$ sudo apt install git-all`

Check the installation typing:

```
$ git --version
```

Getting started - Installing Git [2/3]

Installing on macOS:

The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run `git` from the terminal the very first time.

- `$ git --version`

Getting started - Installing Git [3/3]

Installing on Windows:

There are also a few ways to install Git on Windows:

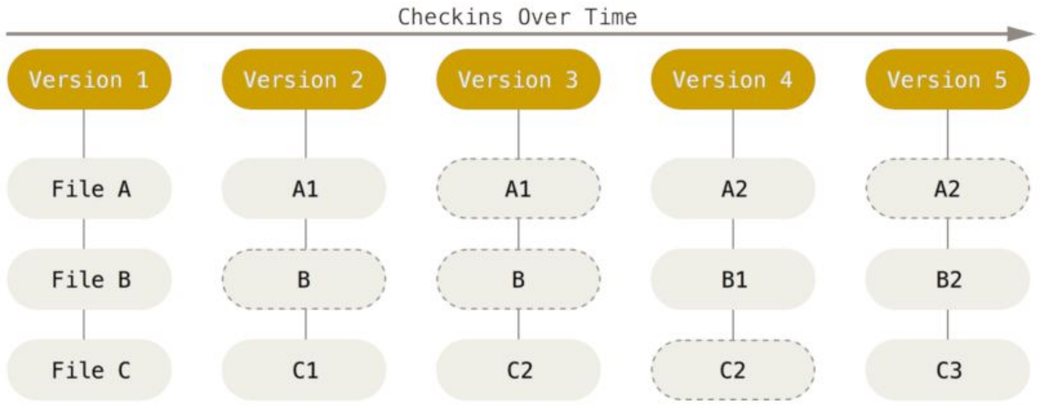
- The most official build is available for download on the Git website.

  Just go to https://git-scm.com/download/win and the download will start automatically.

Resources

- From command line:
  - `$ git help <command>`
- Git reference:
  - https://git-scm.com/docs
- Online book :
  - https://git-scm.com/book/en/v2
- Videos
  - https://git-scm.com/videos

Git concepts



(Git-scm.com, CC BY-NC-SA 3.0)

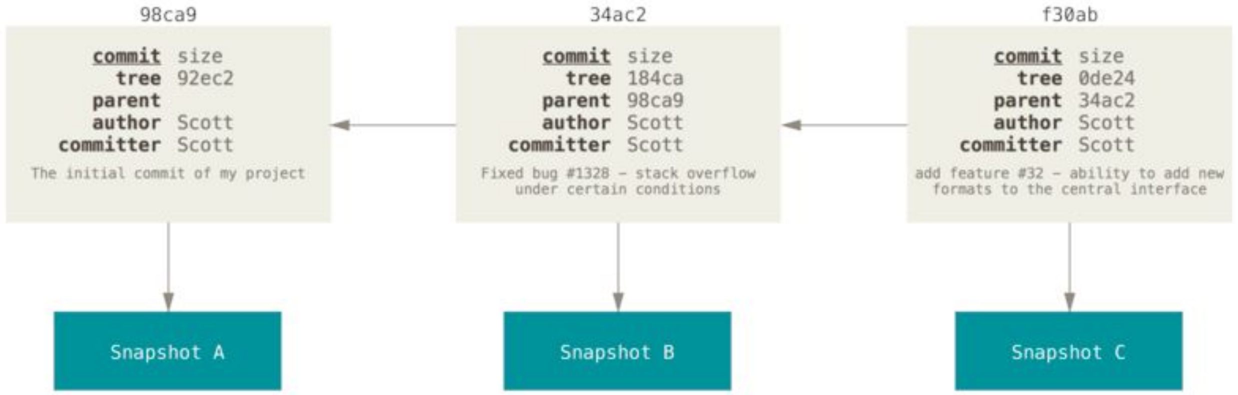
`Git commit:`

The core conceptual unit of work in Git is the commit.

The commit is a snapshot of the files being tracked within your project folder (where the `.git` folder resides).

Any time you change a file, a whole new compressed version of that file is made and stored in that commit. It does this by creating a super compressed Binary Large Object (blob) out of the file, and then keeping track of it by generating a checksum made with the SHA-1 hashing algorithm.

Git concepts



(Git-scm.com, CC BY-NC-SA 3.0)

`Git commit`:

Git is really efficient. If a file does not change between commits, Git does not make a whole new compressed version for storage. Instead, it just refers back to the previous commit. All commits know what commit came directly before it, called its parents (progenitors).

You can easily see this chain of commits when you run:

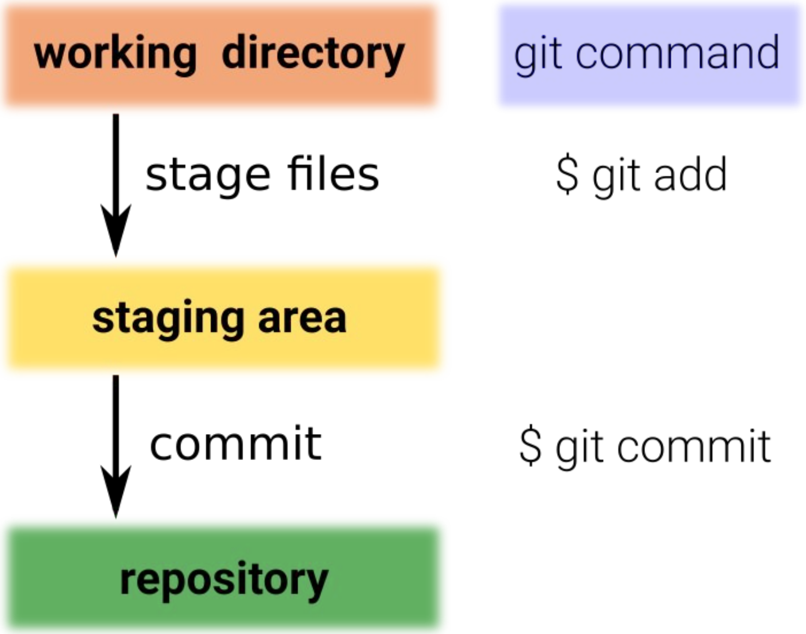
```
$ git log
```

Git concepts

Committing:

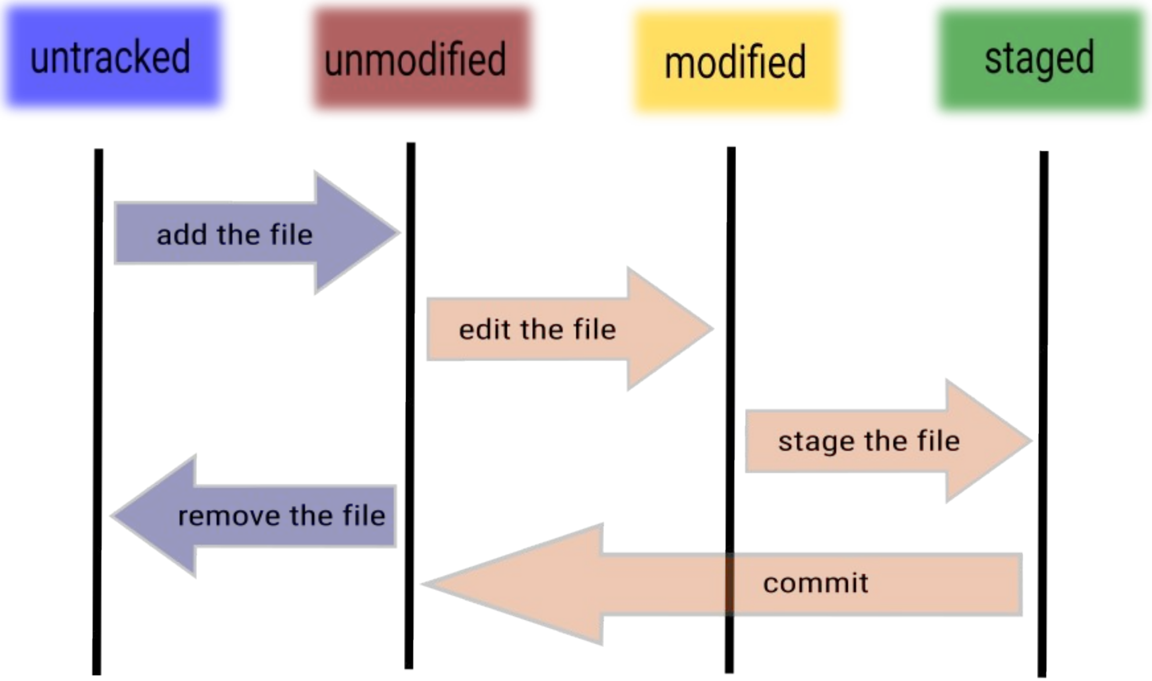
Committing in Git works in two steps:

- **staging phase**: modified or untracked files are "registered" for the next commit. You need to specify all the stage files;
- **commit phase**: staged files are committed along with a commit.



Git concepts

File status lifecycle:



About GitHub

- `GitHub` is a cloud-based platform where you can store, share, and work together with others to write code;
- it uses `Git` software (which has to be installed on your local machine);
- it is commonly used to host open source software development projects;
- services offered:
  - projects on `GitHub` can be accessed and managed using the standard `Git` command-line interface;
  - `GitHub` allows users to browse and download public repositories on the site;
- https://github.com/

About GitHub

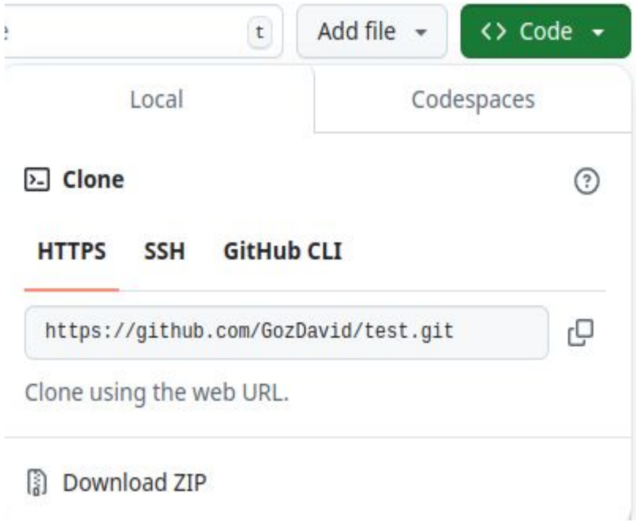
- Create a free account on `https://github.com/`;
- Create a new public repository (e.g. `test`) including a `README.md` file;

How to generate a token on GitHub

- Click on your profile picture and select `Setting`;
- then on the left site scroll down and click on `Developer Settings`;
- click on `Personal access tokens` and select `tokens classic;`
    - `Generate new token;`
    - choose a token name (e.g. "`test token`");
    - select and expiration date (e.g. 30 days);
    - (optional) give a description;
    - select the `Repository access -> Only select repository ->` the previous generated repository (e.g. `test`);
- on `Select scopes`: set the maximum capability only for this test case;
- click on `Generate token;`
- make sure to copy your personal access token now (for example save it on a file) as you will not be able to see this again (otherwise you have to regenerate it).

Clone the first repository

- `<repository> = github.com/GozDavid/test.git`
- `<repository_url> = http://<repository>`
- `<token> = github_pat_XXXXXXXXXXXXXXXXXXXXXXXXXX`
- `<account> = GozDavid`
- Clone your newly created repository with the following Git command:
  - `$ git clone https://<account>:<token>@<repository>`
- You now have a local copy of the repository associated with the generated token.



```
https://github.com/GozDavid/test.git
```

Clone using the web URL.

GitHub remote repository

Repository terminology

TERM	DEFINITION
Branch	A parallel version of your code that is contained within the repository, but does not affect the primary or main branch.
Clone	To download a full copy of a repository's data from GitHub.com, including all versions of every file and folder.
Fork	A new repository that shares code and visibility settings with the original "upstream" repository.
Merge	To take the changes from one branch and apply them to another.
Pull request	A request to merge changes from one branch into another.
Remote	A repository stored on GitHub, not on your computer.

Simple workflow

Work on a `Git` repository usually follows the same scheme:

1. `Retrieve` possible distant (remote) modifications;
2. Perform local modification;
3. `Stage` local modifications;
4. `Commit` local modifications;
5. `Retrieve` possible distant modifications;
6. Manage possible conflicts;
7. `Send` local modifications.

1] Get distant (remote) modifications

After the cloning, distant modifications can be integrated to local repository using the following Git command:

- `$ git pull`

which fetches from the remote (cloned) repository and try to merge into the current branch

2] Perform local modifications

- Create a `hello_world` file in the working directory;
- Edit the `README` file.

3] Stage modifications

- `Unstaged` modifications are modifications that are not tracked yet;
- They can be shown with the Git command:
  - `$ git status`
- To stage a modification (add it to the next commit):
  - `$ git add <file 1> <file 2> … <file n>  ($ git add .)`
- To view again the status of your files in the working directory and staging area:
  - `$ git status`

4] Commit local

- To validate staged modification, we need to create a new commit using the `Git` command:
    - `$ git commit -m “commit name”;`
    - `(first commit - “Author identity unknown” - fix with $ git config)`
- Without the `-m` flag the editor is launched (to set up a default editor (for example nano) use the following command `$ git config --global core.editor "nano"`);
- Commit names are very important:
    - appear in logs, main github page, history, . . .;
    - they should give an overview of the performed modifications;
- use `$ git log` command to see history of the repository.

5/6] Retrieve modifications / manage conflicts

- Retrieve the remote modifications using the command:
  - `$ git pull`
  - distant modifications are applied to local copy;
  - What if local copy also contains modifications?
    - modifications occur at different location -> automatic merge;
    - at the same location -> CONFLICT!;
    - conflicts need to be resolved by hand;
  - When conflicts occur, no push is allowed before they are solved.

7] Send local modifications

- At this point:
  - modifications are locally tracked;
  - but are not available for possible other user (remote repository);
  - changes need to be synchronized with `remote repository`.
- Use `$ git push` command
- Now local and remote repositories are identical.

Conflict workflow

- Find conflict locations;
- resolve conflicts;
- perform a full commit (`$ git commit -a -m ‘message’`);
- retrieve distant modifications (`$ git pull`);
- push local modifications (`$ git push`).

Conflict workflow

- The `$ git diff` command can be used:
  - without argument `$ git diff` lists (all) pending modifications compared to last commit;
  - when conflicts occur (e.g. after `git commit && pull`), conflicting files are modified by `git`, thus they appear in `$ git diff` result;
  - to only get a list of conflicting files use the following command:
    - `$ git diff --name only --diff-filter=U`

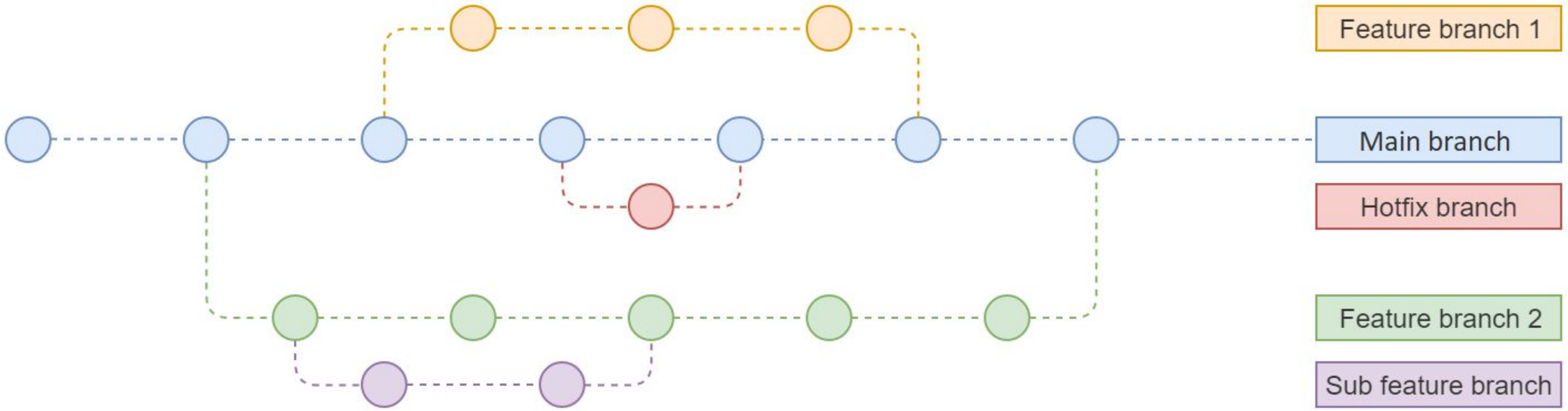
Conflict workflow

A conflict always looks like:

```
<<<<<<< HEAD
Local modification
=======
Distant modification
>>>>>>> master
```

- Chose the modifications you want to keep (can be a mix of both);
- remove all `<<<<<, ===== and >>>>>` markers;
- save files;
- apply it for all conflicts;
- perform a full commit `$ git commit -a -m ‘message’`

Branching model



Branching example containing two feature branches, a hotfix branch, and a sub-feature branch

Resource :

- https://git-scm.com/book/it/v2/Git-Branching-Basic-Branching-and-Merging#r_basic_merging

Branching model

In `Git`, a branch is a new/separate version of the main repository (i.e. edit the project directly without impacting the main branch).

- To create a new branch:
  - `$ git branch <branch name>`
- To show all the existing branches:
  - `$ git branch -l`
- To switch to a specific branch:
  - `$ git switch <branch name>`
- To join two branches together:
  - `$ git switch A (incorporate the changes into branch A)`
  - `$ git merge B  (merge the branch B into A)`

Git detached HEAD

https://www.cloudbees.com/blog/git-detached-head