



...demystified

Mikael Capelle — mikael.capelle@irt-saintexupery.com

2020–2021

Git introduction

Goal of this presentation:

- no more magic command — understand each **git** command you use;
- never delete a local **git** repository if you screwed — hidden secrets of git storage;
- discover new commands and good practice when using **git** as a version-control system.

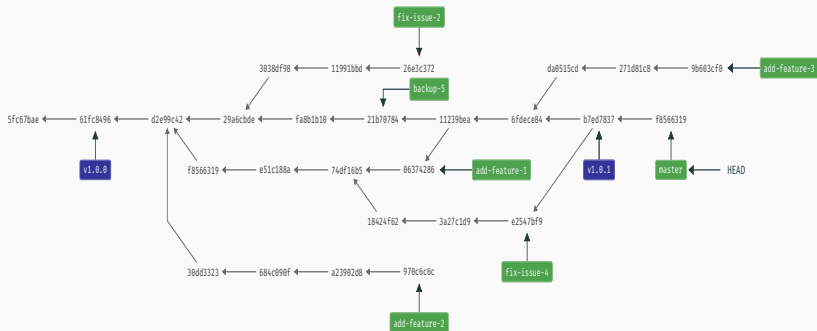


`git` = version-control system

- First commit (!) in April 2005 by Linus Torvalds.
 - Maintained since July 2005 mainly by Junio Hamano (Google).
 - Stable release: 2.19.1 (version < 2.13 no longer supported).
- Git repository of git itself:
 - <https://git.kernel.org/pub/scm/git/git.git/>
 - <https://github.com/git/git> (publish only)
- Documentation and useful resources:
<https://git-scm.com/>

Git in one picture

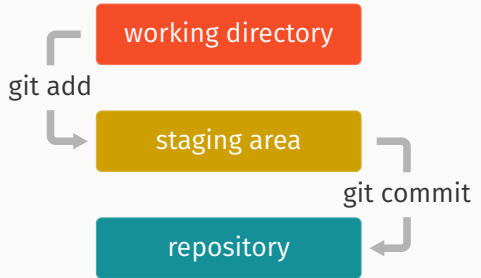
git is **graph-based** —



Git architecture

git manages 3 different zones locally:

- the **working directory**;
- the **staging area** (or **index**);
- the **repository**.



Git architecture — In reality...

stash

Temporary zone to backup files when performing other **git** operations.

working directory

Your actual working directory. This is what other softwares see.

index

Work to add to your next commit (snapshot). Also called the **staging area**.

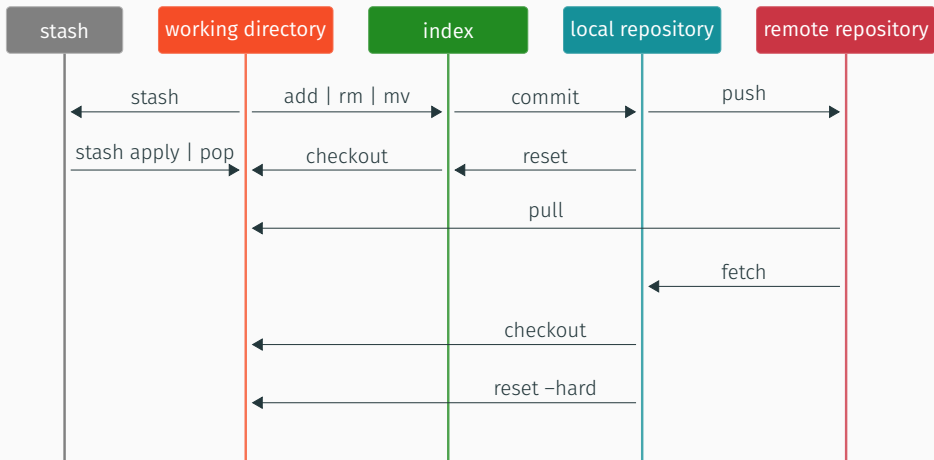
local repository

Local version of **remote repository**. Contains all the (synchronized) history of your project.

remote repository

Remote location of your repository (e.g., Github). Also called **upstream**.

Git architecture — In reality...



Where's my HEAD?

HEAD is a **symbolic reference** to (usually) the current branch —

- HEAD is stored in `.git/HEAD`;
- HEAD can be **detached** and points directly to a commit;
- HEAD is (indirectly) used to determine the parent(s) of new commits;
- by default, **HEAD** points to the `refs/heads/master` reference;
- there are many “HEAD”s: HEAD, FETCH_HEAD, ORIG_HEAD, ...

Git commands — An introduction

`git config [--system|--global|--local] <name> <value>` — Update `git` configuration in the repository (default behaviour), globally for the current user or system-wide.

```
git config --global core.editor emacs
git config core.autocrlf true
```

References: <https://git-scm.com/docs/>

Cheatsheet:

<http://ndpsoftware.com/git-cheatsheet.html>

Windows users? Use `Git Bash`, included when downloading the official `git` at <https://git-scm.com/download/win>.

Git commands — An introduction

`git init [--bare]` — Initialize an empty local repository not associated with any upstream.

`git clone <url>` — Initialize a local repository from the given remote repository at `<url>`. The default name of the remote within the local repository is **origin** and the default branch **master**.

`git remote -v` — List remotes associated with the local repository.

`git remote add <remote> <url>` — Add a remote to the local repository with the name `<remote>` and pointing to the given `<url>`.

`git remote remove <remote>` — Remove the specified remote from the local repository.

`git remote set-url <remote> <url>` — Point the specified remote to the specified URL.

Git commands — An introduction

`git status [-s]` — Display the state of the local repository.
`-s` gives you a shorter version.

`git diff [--cached] <commit>` — Display the difference between the working directory (or the index with the `--cached` option) and the specified commit (or the index if no commit is specified).

`git log [--oneline]` — Display the history of commits of the current branch. The option `--oneline` gives a shorter output.

`git log --oneline --left-right <branch1> <branch2>`

Git commands — An introduction

git pull — Pull the upstream associated with the current branch, updating the **local repository**, the **index** and the **working directory**. Might fail if there are conflicts between the local state and the remote one.

git push — Push the current branch to its associated remote (**git 2+**), or all the branches with associated remotes and matching branch (**git 1**).

git push <remote> <local-branch>:<remote-branch> — Update the remote branch of the given **<remote>** to match the specified local branch.

git fetch — Update the local repository but not the index nor the working directory.

Git commands — An introduction

`git add [-u] <files...>` — Add the given file to the index, i.e., **stage** the file. The option `-u` update all files already in the index to match the working repository (but not newly created files).

`git rm [--cached] <files...>` — Remove a file from **both** the index and the working directory. The option `--cached` allow to remove a file only from the index.

`git mv <files...> <file>` — Rename file or move files to directory in both the working directory and the index (similar to the `mv` command).

`git checkout <files...>` — Update files in the working directory to match their counterparts in the index.

Git commands — An introduction

`git commit [-m <msg>]` — Create a new commit (snapshot) using the index with the specified message. If no message is specified, the default `git` editor is opened.

`git commit --amend` — Update the last commit with changes from the index.

`git reset [<files...>]` — Update the specified files (or all files) in the index to match their counterparts in the current local repository.

`git reset --mixed <commit>` — Update the index to match the specified commit.

Git commands — An introduction

`git checkout <branch>` — Switch to the specified branch, updating the index and the working tree.

`git checkout <commit>` — Switch to the specified commit, updating both the index and the working tree and entering a detached HEAD state.

`git reset --hard [<commit>]` — Reset the index and working tree to match the specified commit (default to the current HEAD). Discard all changes not already committed!

Git commands — An introduction

merge — Merging introduces the changes from a different branch into the current one, and create a new commit representing the “merge”.

git merge [--no-commit] [-m <msg>] <branch> — Merge the change from the given **<branch>** into the current branch. The **--no-commit** performs the merge but does not commit the result. The **-m** option can be used to override the default commit message.

git merge --abort — Abort the current merge process.

git merge --continue — Continue the current merge process after resolving conflicts.

Git commands — An introduction

rebase — Rebasing modifies the history in order to insert commits from a different branches before the commits of the current branch.

git rebase <branch> — Rebase the given branch into the current branch.

git rebase --abort — Abort the current rebase process.

git rebase --continue — Continue the current rebase process after resolving conflicts.

cherry-pick — Cherry-picking applies the change from one or more commits to the current branch, creating new commits.

git cherry-pick <commits...> — Cherry-pick the given commit(s) on top of the current branch.

git cherry-pick --abort — Abort the current cherry-pick process.

git cherry-pick --continue — Continue the current cherry-pick process after resolving conflicts.

Git commands — An introduction

stash — Saves and restores local changes by storing them in the stash stack.

git stash — Record local changes by creating a new stash on top of all previous ones.

git stash apply — Apply the changes from the stash on top of the stack to your current working directory.

git stash drop — Drop the stash on top of the stack.

git stash pop — Equivalent to **apply** then **drop**.

Resolving conflicts

Conflicts occurs when a file has two versions that must be merged, e.g., after a **merge**, a **rebase**, a **cherry-pick** or a **stash**.

```
<<<<<< HEAD
Version 2
=====
Version 3
>>>>>> Other branch
```

Resolving conflicts

1. Abort the current process, e.g., with the `--abort` option that most commands have.
2. Retrieve one version of the file:

```
git checkout --theirs|--ours <files...>
```

The `--theirs` and `--ours` do not have the same meaning for `merge` or `rebase`.

3. Modify the file manually to resolve the conflict. → Do it! Do it! Do it!
4. Use a dedicated tool, e.g., `git mergetool`, Magit.

Git objects

git stores **objects** within the `.git/objects` directory (local):

- **blob** objects;
- **tree** objects;
- **commit** objects;
- (annotated) **tag** objects.

Each object represented by its SHA-1 checksum (20 bytes, 40 hexadecimal characters), e.g:

```
.git/objects/98/7c3be764396c5a315e2c5ea536d8956aba82bc
```

Once created, **objects never change**.

Git objects — blob

blob objects —

- a **blob** contains the content of a “file” (a binary array of data);
- identical contents means identical objects due to SHA-1 naming:
 - two identical files are represented by a single **blob** object;
- a **blob** has no metadata associated directly with it:
 - names of files are stored within **tree** objects and within the **index**;
- **blob** objects are usually created when (revision of) files are added to the repository (**git add**) or files are compared (**git diff**).

➔ **git does not store delta** between file revisions.

tree objects —

- a **tree** is similar to a directory, it contains:
 - references to **blob** objects (files);
 - references to other **tree** objects (sub-directories);
- a **tree** associates names and modes to **blob** and **tree** objects it references;
- a **tree** object has no “name” by itself.

Git objects — tree

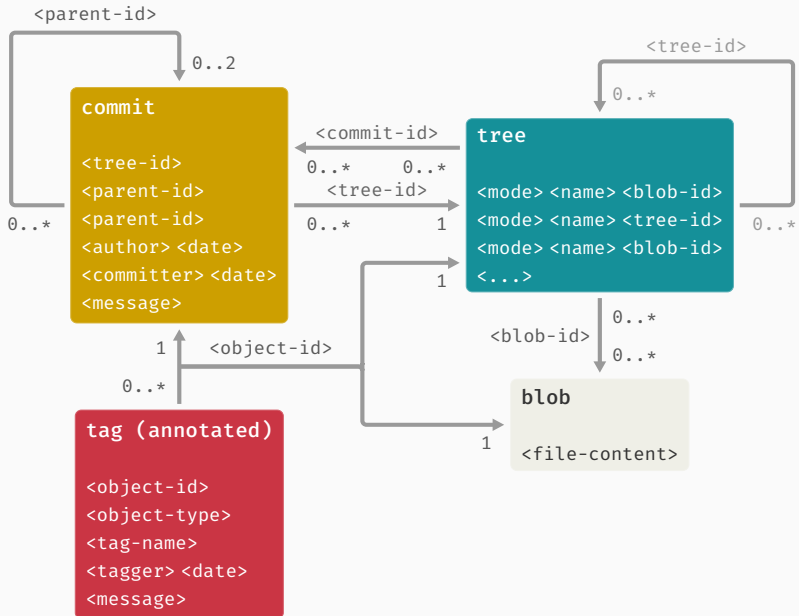
tree objects — possible modes:

- `040000`: directory
 - `100644`: regular non-executable file
 - `100755`: regular executable file
 - `120000`: symbolic link
 - `160000`: gitlink (submodule)
- } **tree**
- } **blob**
- } **commit**

commit objects —

- **commit** objects are the building blocks of **git**;
- a **commit** object contains a snapshot of the working tree (a **tree** object) with associated metadata: author, committer, date, ...;
- **commit** objects are linked together by a parent-child relationship, creating a revision tree;
- references (**branches**, **HEAD**, **tags**) target **commit** objects using their SHA-1 checksums (names).

Git objects — Relation between objects



Git objects — Inspecting objects

```
# List files in the index.
```

```
$ git ls-files --stage
```

```
100644 fa49b077972391ad58037050f2a75f74e3671e92 0      foo.txt
100644 96ac8f82e27c18f4a736ebb277fb0aa9648b711f 0      test.tx
```

```
# Display the content of the given blob.
```

```
$ git cat-file -p 96ac8f82e27c18f4a736ebb277fb0aa9648b711f
version 4
```

```
$ git cat-file -p HEAD
```

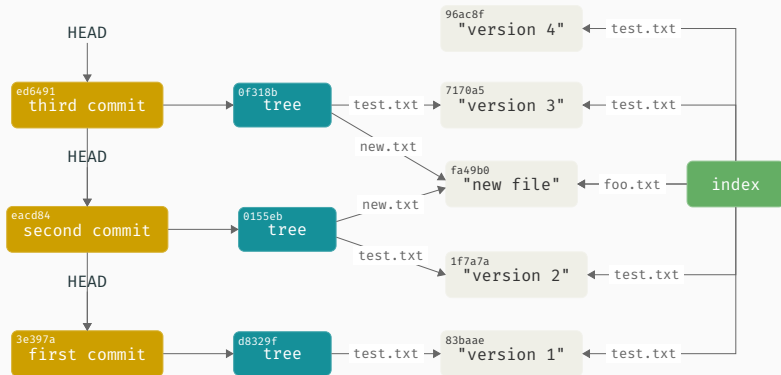
```
tree 0f318b9fb1845be79439afc88c7b76dfa2ff8d91
parent eacd8426cd48c7e14f80b1650110439dbb13a7df
author Mikaël Capelle <mikael.capelle@irt-saintexupery.com> 1541587661 +0000
committer Mikaël Capelle <mikael.capelle@irt-saintexupery.com> 1541587661 +0000

third commit
```

```
$ git cat-file -p HEAD^{tree}
```

```
100644 blob fa49b077972391ad58037050f2a75f74e3671e92      new.txt
100644 blob 7170a5278f42ea12d4b6de8ed1305af8c393e756      test.tx
```

Git — Example



Working directory

bar.txt
"some file"

test.txt
"version 12345"

new.txtfoo.txt
"new file"

```
git init
echo "version 1" > test.txt
git add test.txt
git commit -m "first commit"
echo "version 2" > test.txt
echo "some file" > bar.txt
echo "new file" > new.txt
git add test.txt
git add new.txt
git commit -m "second commit"
echo "version 3" > test.txt
git commit
```

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    new.txt -> foo.txt
        modified:   test.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working dir)

        modified:   test.txt

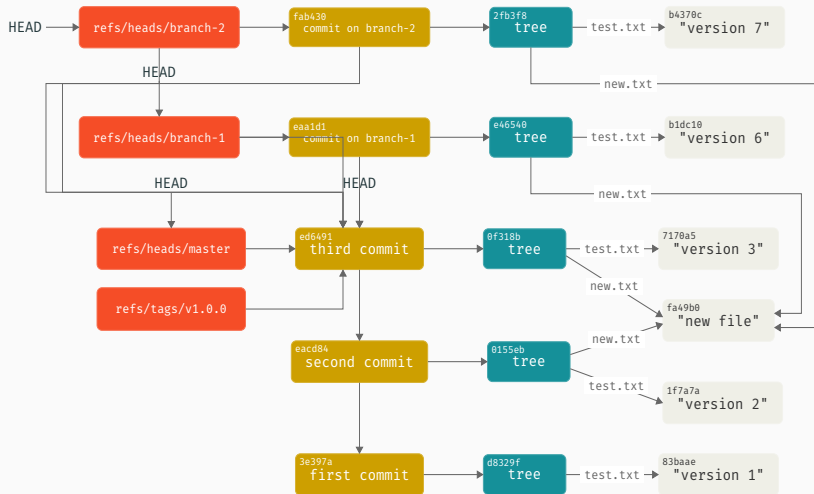
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        bar.txt
```

reference = human-readable name for a SHA-1 hash —

- **references** are stored under `.git/refs`;
- there are three main types of references:
 - **heads** — head of local branches;
 - **remotes** — head of remote branches;
 - **tags**.

References



```
git tag v1.0.0
git checkout -b branch-1
echo "version 6" > test.txt
&& git commit -am "commit on branch-1"
git checkout master
git checkout -b branch-2
echo "version 7" > test.txt
&& git commit -am "commit on branch-2"
```



```
$ git show-ref --head
```

```
eea1d1794759413c31d26fe0b66c6a9d73142d7f HEAD
```

```
eea1d1794759413c31d26fe0b66c6a9d73142d7f refs/heads/branch-1
```

```
fab4302c8ab48a509b3f55bb22c8ad790cffcdde refs/heads/branch-2
```

```
ed64911b6d93a10de9beb6c6ea03c58afbf75e03 refs/heads/master
```

```
eea1d1794759413c31d26fe0b66c6a9d73142d7f refs/remotes/origin/branch-1
```

```
ed64911b6d93a10de9beb6c6ea03c58afbf75e03 refs/remotes/origin/master
```

```
fab4302c8ab48a509b3f55bb22c8ad790cffcdde refs/remotes/public/branch-2
```

```
eea1d1794759413c31d26fe0b66c6a9d73142d7f refs/remotes/public/master
```

```
ed64911b6d93a10de9beb6c6ea03c58afbf75e03 refs/tags/v1.0.0
```

Remotes

remote = “hosted” version of the repository —

- a **remote** is simply a **git** repository somewhere else:
 - a **remote** is often a **bare** repository (see `--bare` for the `init` and `clone` commands), i.e., a repository without a working directory;
- **git** handles **four** protocols to communicate with remotes:
 - **local** protocol — `file://` — for remotes that are on an accessible filesystem;
 - **http(s)** protocol — `http(s)://` — authenticated and unauthenticated access;;
 - **ssh** protocol — `ssh://` — easy to set-up, but does not allow unauthenticated access;
 - **git** protocol — `git://`.

remote = “hosted” version of the repository —

- a **remote** is identified by its name:
 - the default **remote** after `clone` is **origin**;
- **remotes** are often configured to prevent hazardous behaviours:

```
$ git config --system receive.fsckObjects true
$ git config --system receive.denyNonFastForwards true
$ git config --system receive.denyDeletes true
```

- in order to enable branch-specific control, **hooks** must be used.

remote = “hosted” version of the repository —

```
$ git init --bare
Initialized empty Git repository in /data/git/mikael/tutogit.git
$ git status
fatal: This operation must be run in a work tree
$ ls
config  description  HEAD  hooks  info  objects  refs
```

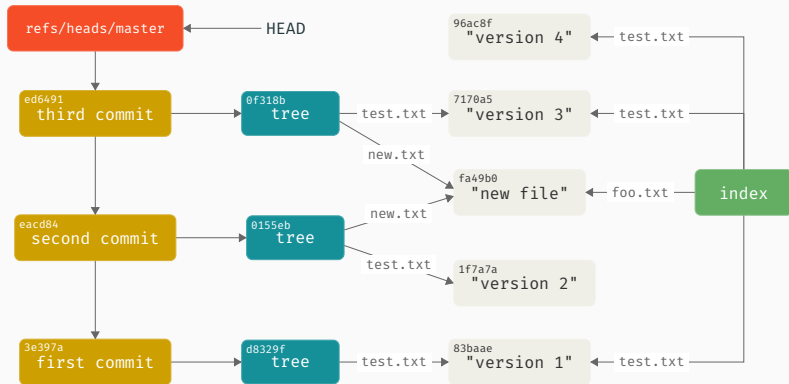
git commands

- Most commands are non-destructive:
 - most objects can be retrieved from the repository, even though it may be hard to find their names;
 - objects may have been added to the repository without a **git add**;
 - only a few commands erase files in the working directory.
- Some commands have very different behaviors when a path is specified at the end, e.g., **reset** or **checkout**.
- Some commands have a **-p|--patch** option to apply the command hunk by hunk, e.g., you can add a file partially to the index using **git add -p**.
- Most “hazardous”/destructive commands have a **-n|--dry-run** flag to perform a dry run of the command, i.e., printing what the command would do without actually doing anything.

```
git reset [-q] [<tree-ish>] [--] [<paths> ... ]
```

- reset the **index** entry for <paths> so that it points to the objects for <paths> in the <tree-ish> revision;
- **does not update** any files in the working directory;
- if a file did not exists in the <tree-ish> revision, remove the file from the **index**.

Git — Example



Working directory

bar.txt
"some file"

test.txt
"version 5"

foo.txt
"new file"

```
git reset HEAD -- test.txt
```

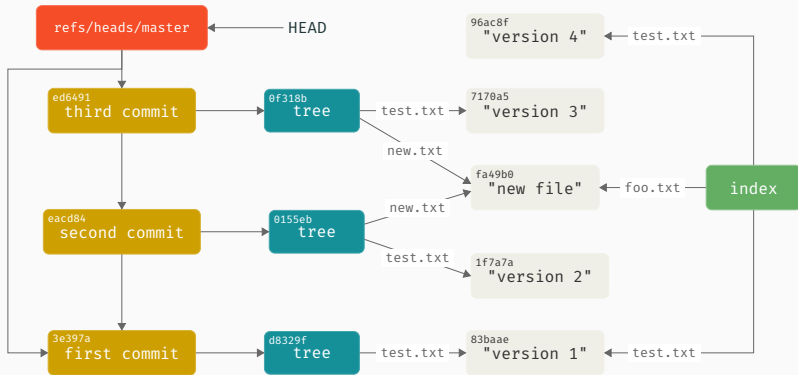
```
git reset 3e397a -- test.txt
```

git reset — V2

```
git reset [--soft | --mixed [-N] | --hard | --merge  
| --keep] [-q] [<commit>]
```

- reset the current branch head to the specified **<commit>**, reset the **index** and reset **files** in the working tree:
 - **--soft** — reset the **head** but does not reset the index or files in the working tree;
 - **--mixed** (default) — reset the **head** and the **index** but does not reset files in the working tree;
 - **--hard** — reset the **head**, the **index** and all tracked **files** in the working tree (i.e., discard local changes for tracked files);
 - **--merge**, **--keep** — reset the **head**, the **index**, and some **files** in the working tree, depending on their states.

Git — Example



Working directory

bar.txt
"some file"

test.txt
"version 51"

foo.txt
"new file"

```
git reset --soft 3e397a
```

```
git reset [--mixed] 3e397a
```

```
git reset --hard 3e397a
```

reset vs. checkout

				HEAD ¹	Index	Workdir	"Safe"? ²
Commit Level							
reset	--soft	[commit]		REF	✗	✗	✓
reset		[commit]		REF	✓	✗	✓
reset	--hard	[commit]		REF	✓	✓	✗
checkout		[commit]		HEAD	✓	✓	✓
File Level							
reset		[commit]	<paths>	—	✓	✗	✓
checkout		[commit]	<paths>	—	✓	✓	✗

¹Indicates if the command moves the reference (branch), REF, or only the HEAD.

²Indicates if the command is safe for the working directory.