INTRODUCTION TO GIT

Sonny Lion (sonny.lion@obspm.fr)

Observatoire de Paris, LESIA

GIT BASICS

So, what is Git in a nutshell? It is a powerful tool **to manage version control of your work**.

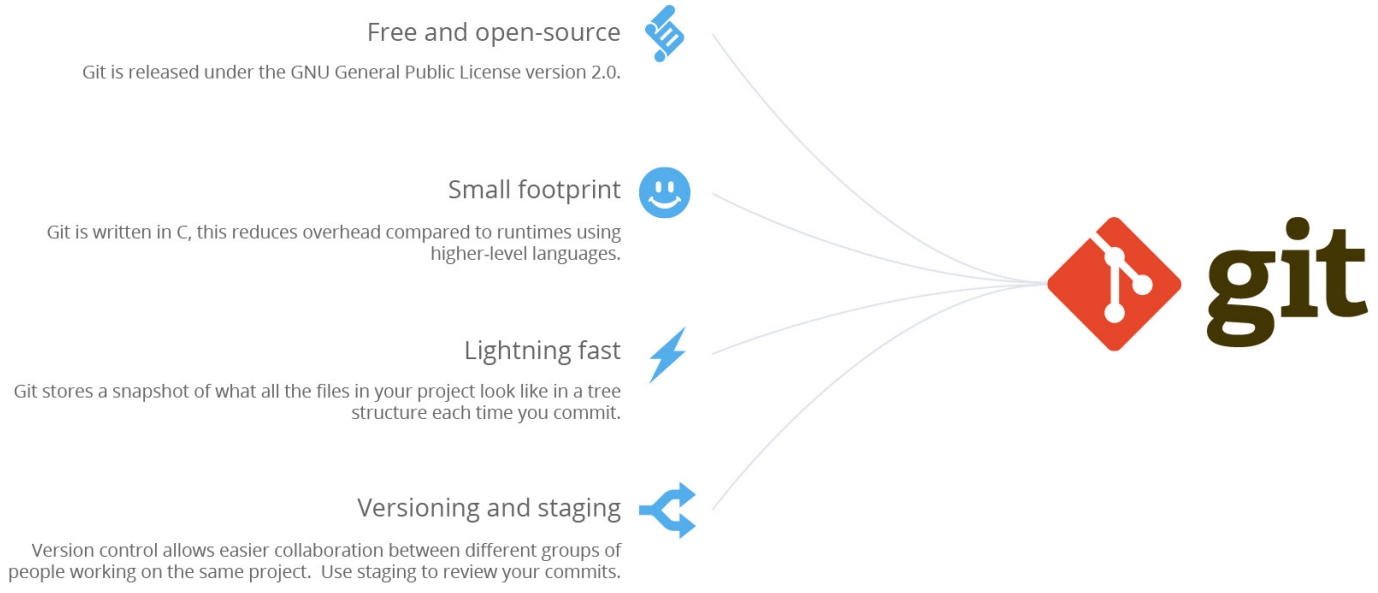


Figure 1. Free, open-source and fast. What else?

WHY GIT?

The major difference between Git and any other VCS (Subversion and friends included)

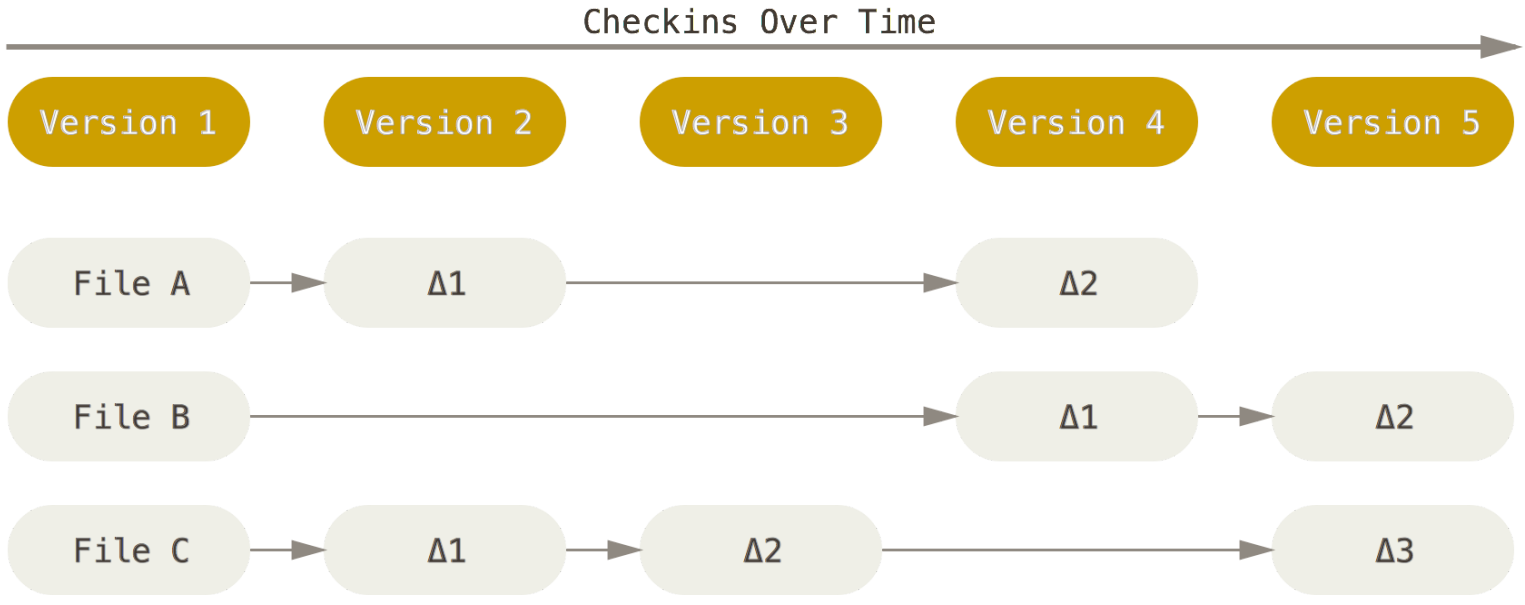


Figure 2. Storing data as changes to a base version of each file.

Git doesn’t think of or store its data this way.

SNAPSHOTS, NOT DIFFERENCES

Instead, Git thinks of its data more like a set of snapshots of a miniature filesystem. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.

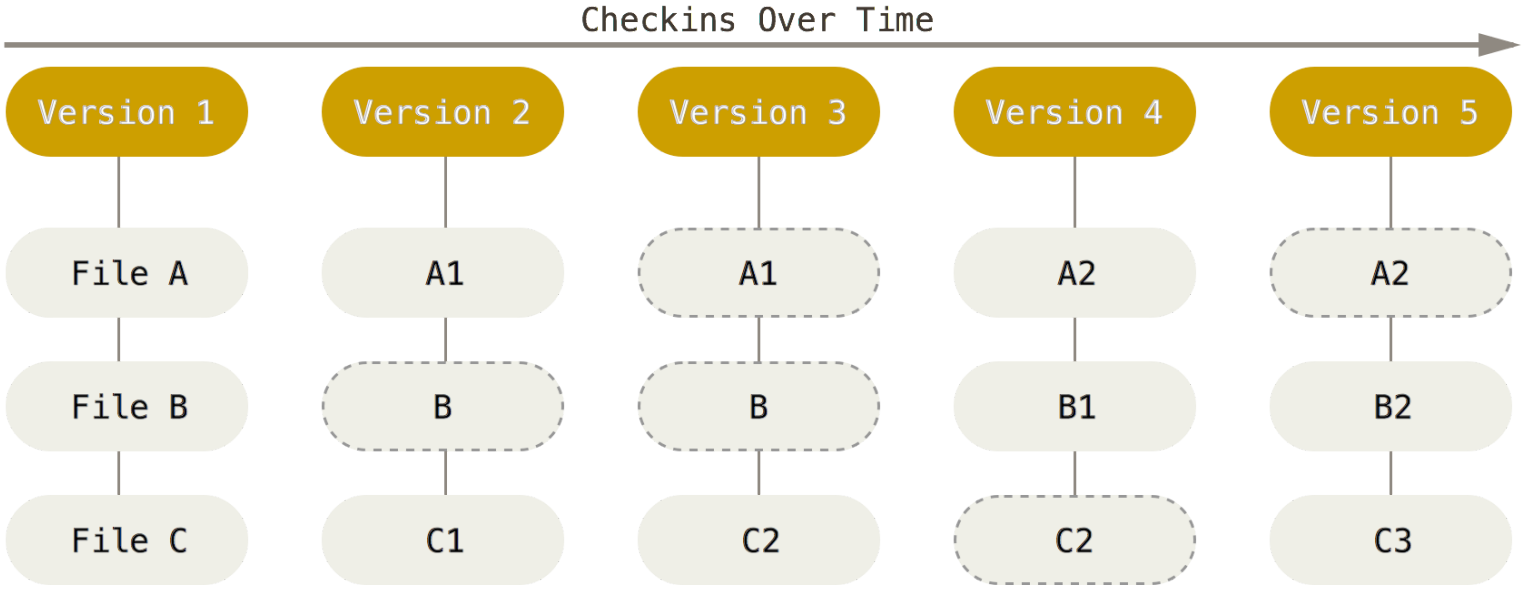


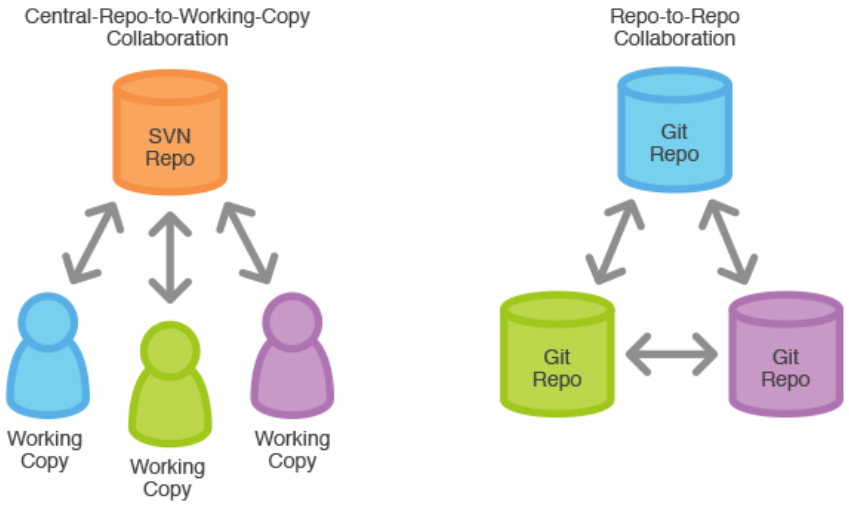
Figure 3. Storing data as snapshots of the project over time.

NEARLY EVERY OPERATION IS LOCAL

Most operations in Git only need local files and resources to operate

Git doesn’t need to go out to the server to get the history and display it for you.

This also means that there is very little you can’t do if you’re offline or off VPN.



GIT HAS INTEGRITY

Everything in Git is check-summed before it is stored and is then referred to by that checksum.

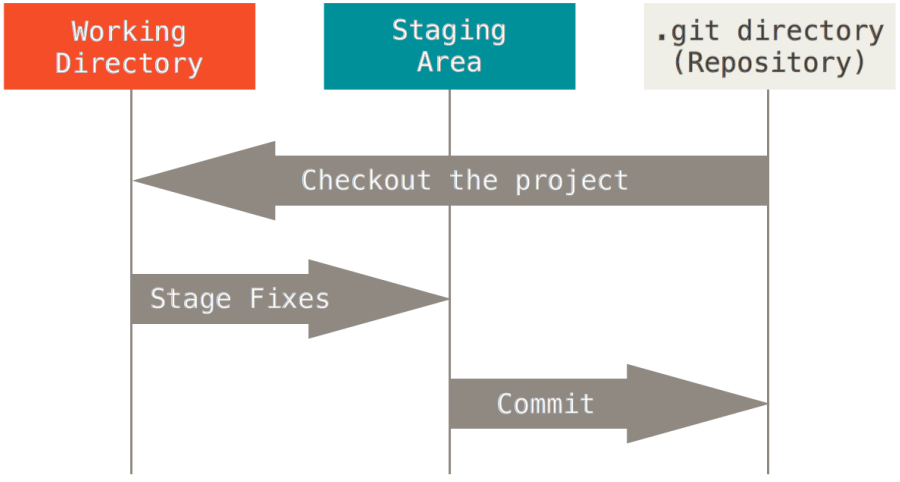
The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. A SHA-1 hash looks something like this:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Using git, we know we can experiment without the danger of severely screwing things up.

THE THREE STATES

Git has three main states that your files can reside in: committed, modified, and staged.The basic Git workflow goes something like this:



1. You modify files in your working directory.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

SETTING UP A GIT REPOSITORY

GIT INIT

The git init command initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

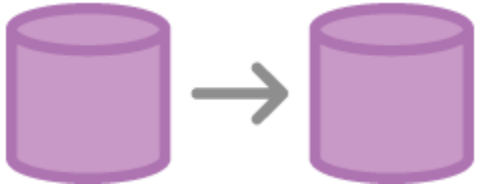
Usage

```
git init
```

SETTING UP A GIT REPOSITORY

GIT CLONE

The git clone command creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository.



Usage

```
git clone repo directory
```

SETTING UP A GIT REPOSITORY

GIT CONFIG

The git config command is a convenient way to set configuration options for your Git installation. You'll typically only need to use this immediately after installing Git on a new development machine.

Usage

```
git config --global user.name name
```

```
git config --global user.email email@toto.net
```

RECORDING SNAPSHOTS

GIT ADD

The git add command moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.

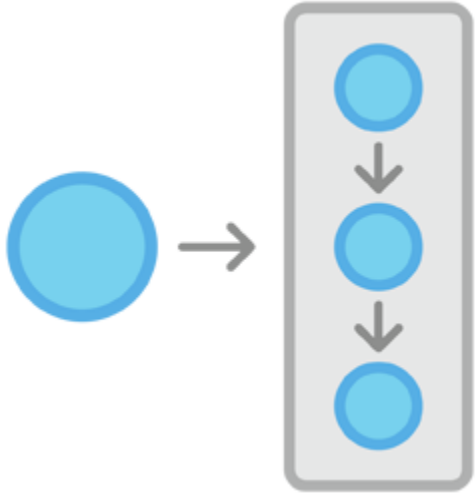
Usage

```
git add file
```

RECORDING SNAPSHOTS

GIT COMMIT

The git commit takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.



Usage

```
git commit -m "message"
```

INSPECTING A GIT REPOSITORY

GIT STATUS

The git status command displays the state of the working directory and the staged snapshot. You'll want to run this in conjunction with git add and git commit to see exactly what's being included in the next snapshot.

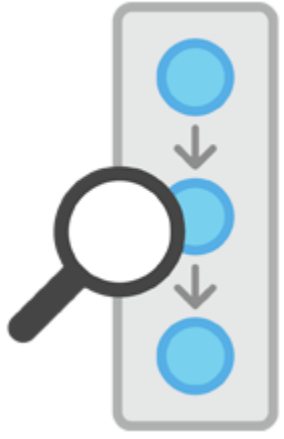
Usage

```
git status
```

INSPECTING A GIT REPOSITORY

GIT LOG

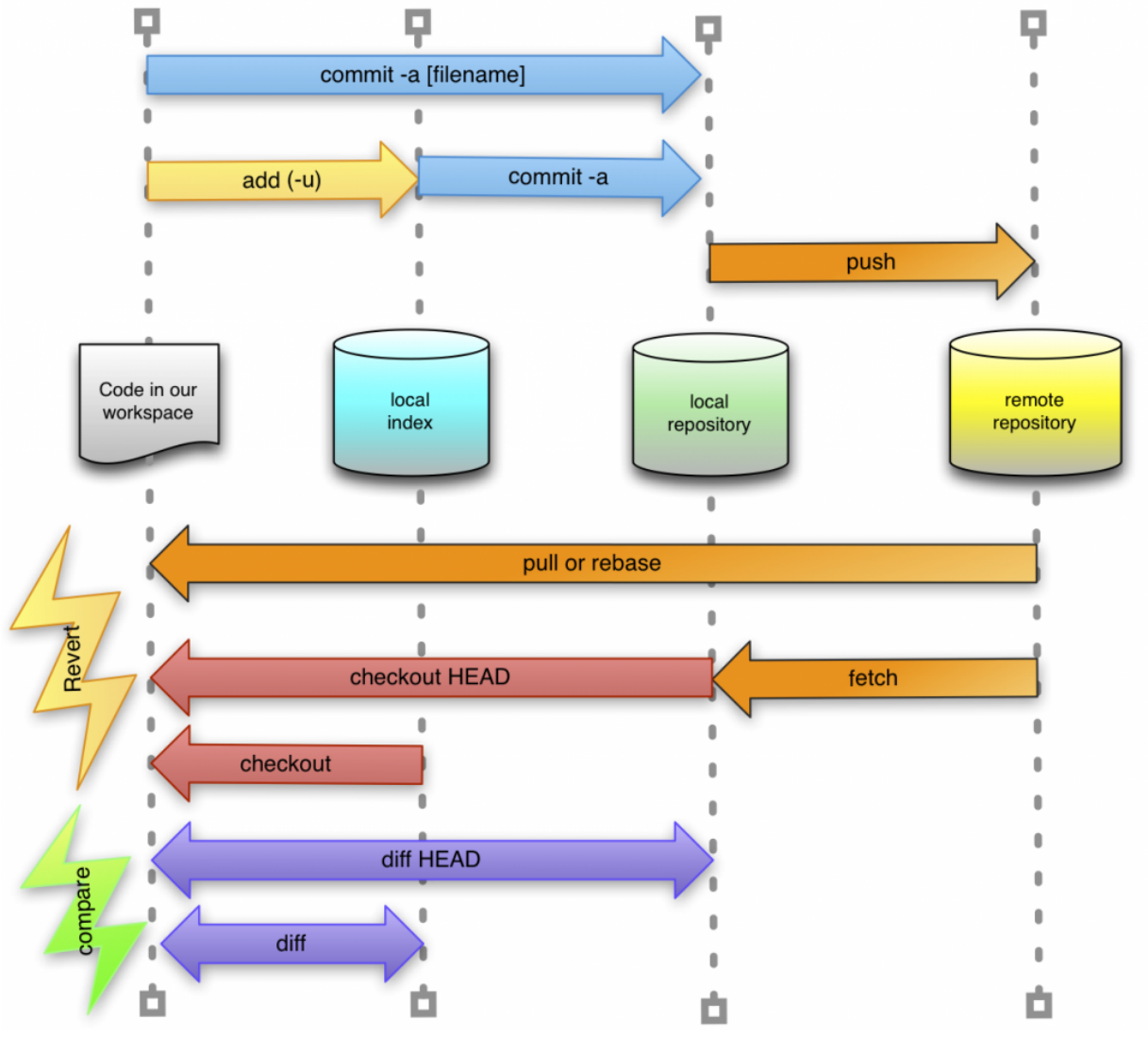
The git log command lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots.



Usage

```
git log -n limit
```

SUMMARY


commit -a [filename]
add (-u)
commit -a
push
Code in our workspace
local index
local repository
remote repository
pull or rebase
Revert
checkout HEAD
fetch
checkout
diff HEAD
compare
diff