Lesson 02
Version Control System

DEVOPS - ITI 4
HEI 2021-2022

What is a Version Control System ?

We should switch to Git...
Not sure...
...SVN does the job quite well, you know...
Arf... Keep it simple, guys...
Look, our new colleague has a very innovative versioning system...
Not bad!
/ <root>
Name
Ext
www
www_02122013
www_bu
www_DONOTDELETE
www_old
It's something...
CommitStrip.com

Version control system (VCS)

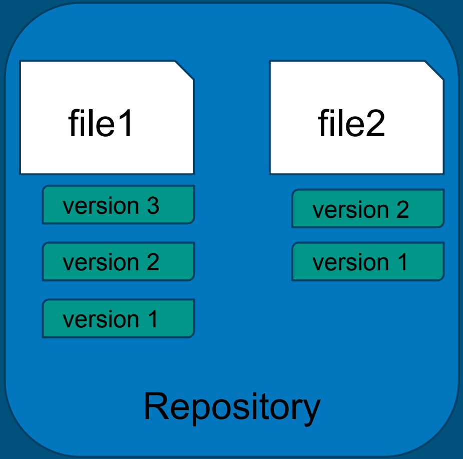
Overview

- Keeps records of your changes
- Enables for collaborative development without the fear of overwriting the work on another person
- Enables you to knows who made what changes and when
- Enables you to revert any changes and go back to a previous state

Version of a file

- For each change of a file, a new version is created in repository

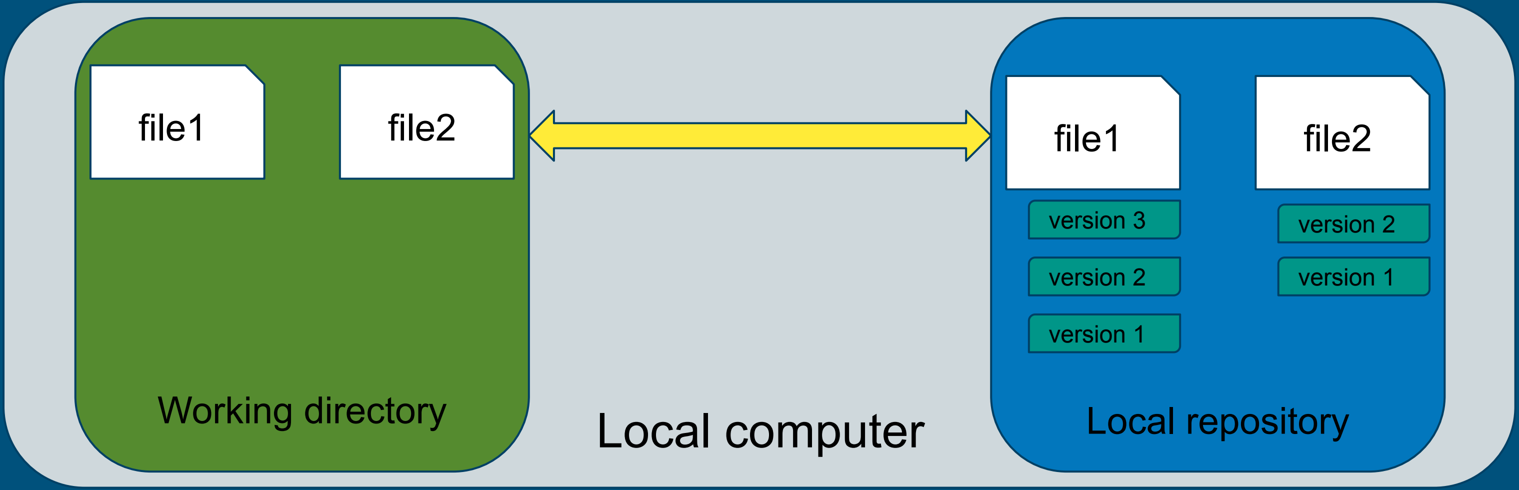
- This version is called a revision and is associated with the following informations:
  - When the change has been done
  - What has been changed in the file
  - Who has made the change



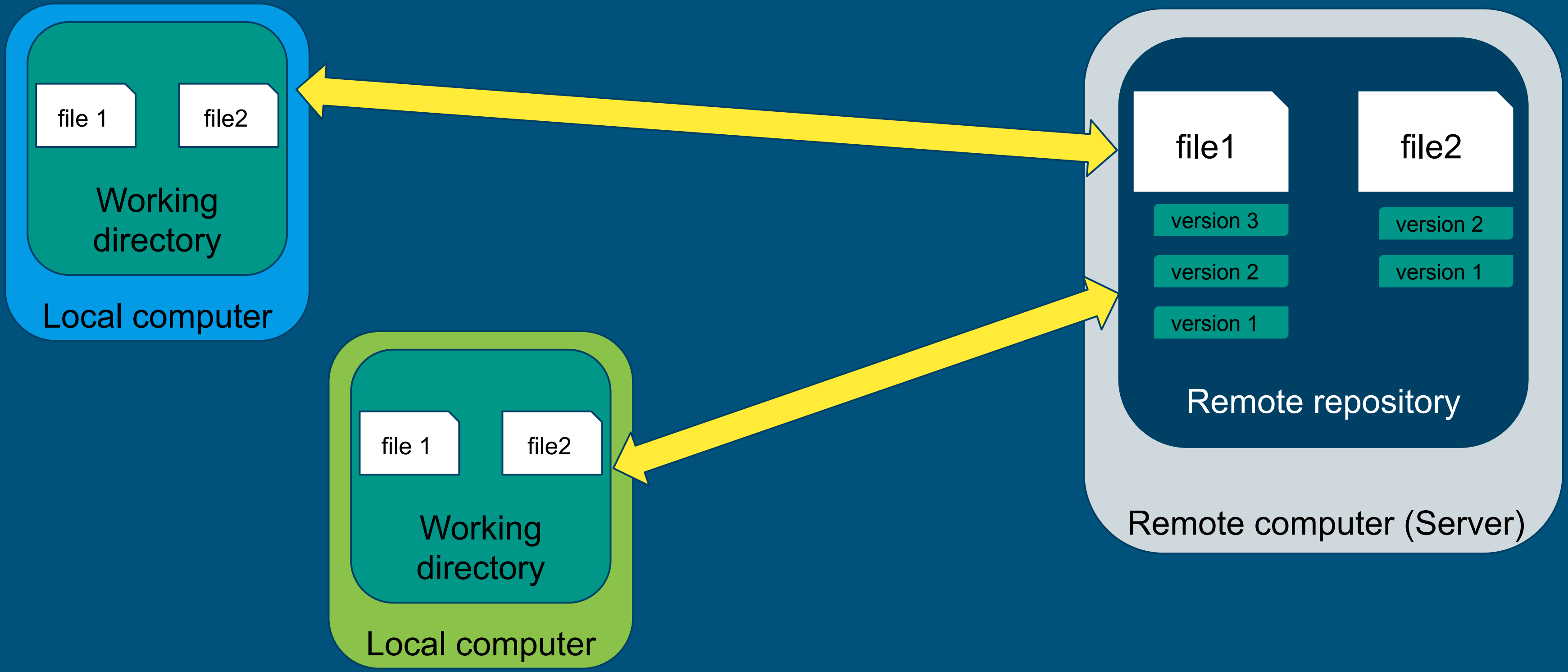
Type of VCS

Local VCS

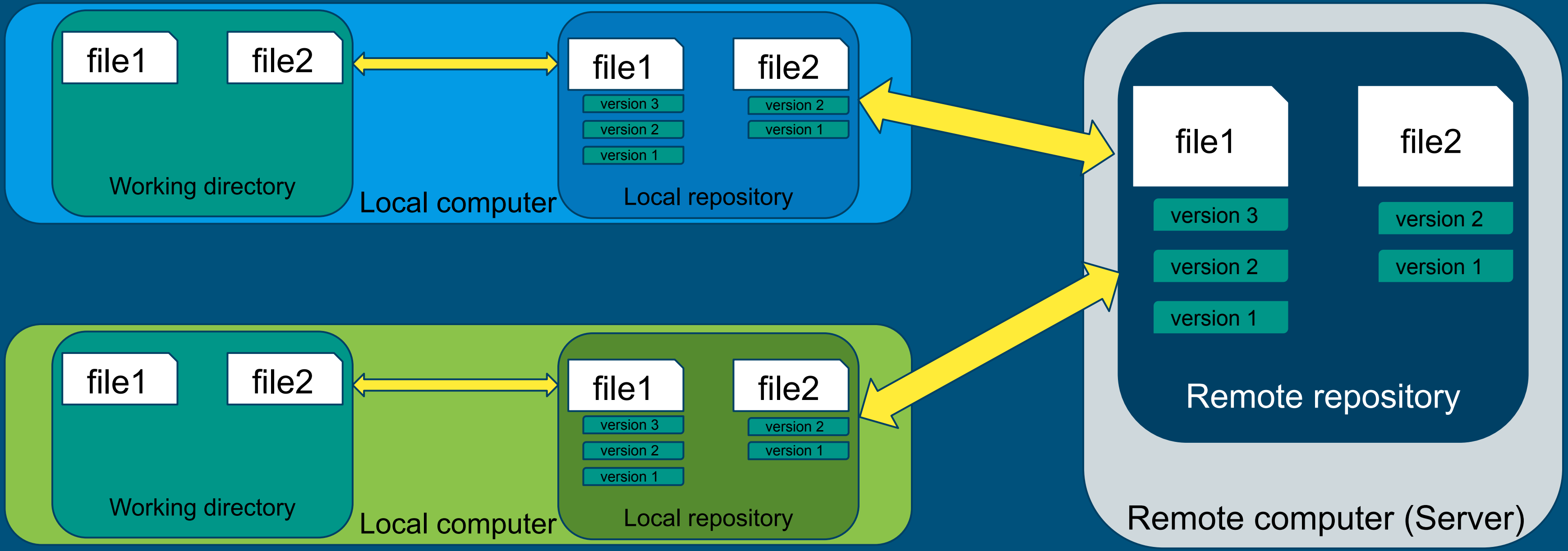
- A VCS is local if the changes are made on the computer that stored the history of changes :



Centralized VCS



Distributed VCS



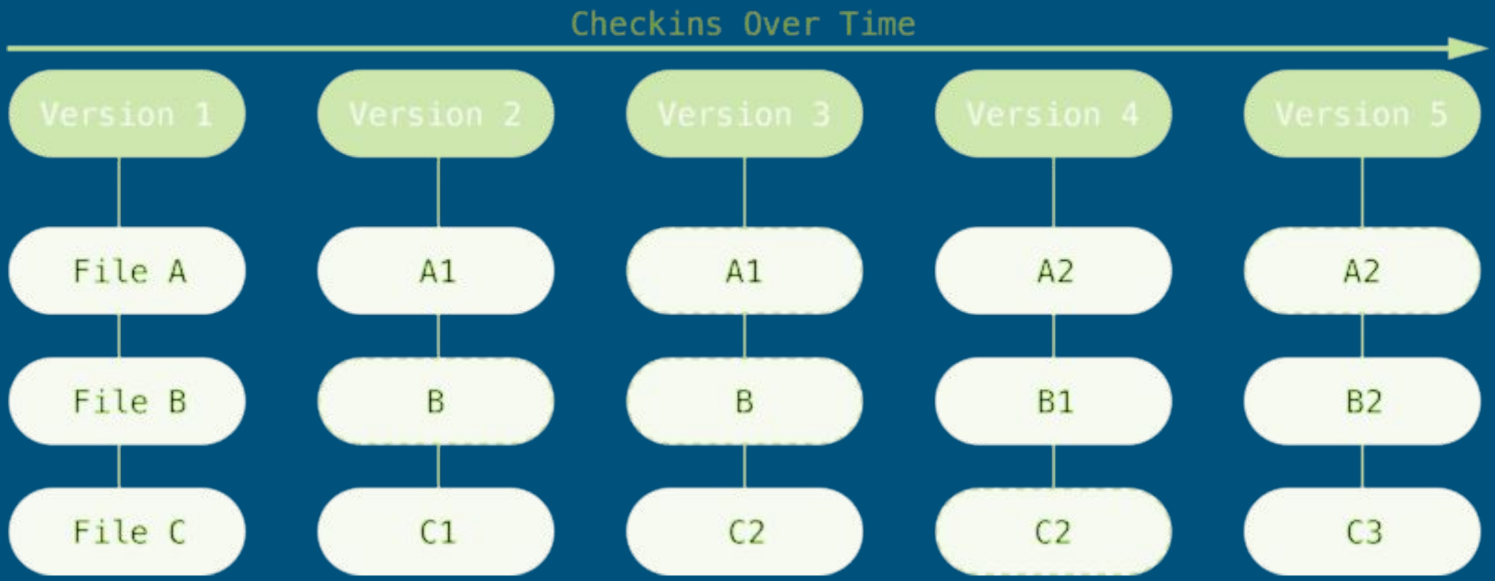
What is Git ?

Git Overview

- Distributed VCS
- Users keep entire code and history on their local system
- Created by Linus Torvald in 2005

Snapshots

- Git version (commit) is a snapshot of every files of the project :

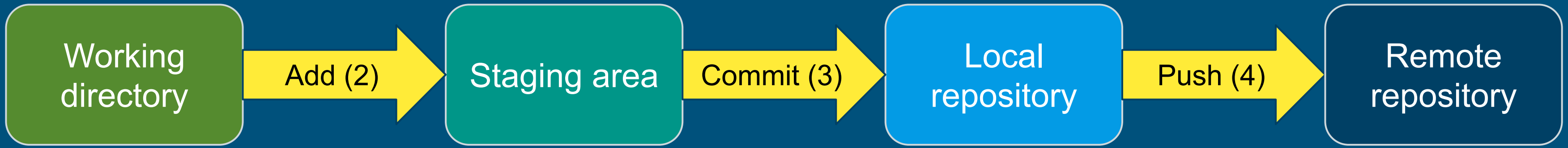


Status of a file

- Index contains all the files taken into account by the git repository
- New file is not in the index automatically
- 3 differents states exist for files in the index :
    - Unmodified :Files that are up to date with the local repository ;
    - Modified : Files that have been modified ;
    - Staged : Files that have been modified and that are ready to be send to the local repository

Basic workflow

1. The user modifies some files in the working directory ;
2. The user add modified files in the staging area ;
3. The user commits and create a new snapshot in the local repository from all the files in the staging area.
4. The user pushes the new snapshot from the local repository to remote repository



How to use git ?

Hey did you see the code the new guy wrote?
Oh, no. Show me...
It's incredible, I've never seen code like this...
It's just sooo...
Bad ?
Yeah...
I didn't want to call him out in front of everyone, so I reverted his commit and I asked him to see me, because it was just... awful
You made the right call. It's better not to do that kind of thing in public
Exactly
Well, I mean I did post the code on Reddit...
Obviously
CommitStrip.com

Use of Git

Git command

- Git is a command line tool that can be used :
  - Directly on unix system (Linux / Mac OS) :
  - by using Git Bash on Windows

- Git commands follows this pattern : git verb options
  - verb : action to perform
  - options : parameters for the action

Git configuration

- Git configuration is controlled by the .gitconfig file ($HOME/.gitconfig)

- Important params :
    - [user] name
    - [user] email
    - [http] proxy :
        - to configure proxy
        - host:port

```
[user]
        name = Denis Cauchois
        email = denis.cauchois@junia.com
[http]
        proxy = http://username:password@proxy-domain:port
```

.gitignore

- To specify intentionally untracked files to ignore
- File in the root of your repository
- Each line specifies a pattern to ignore

```
# Eclipse
.classpath
.project
.settings/

# Intellij
.idea/
*.iml
*.iws

# Mac
.DS_Store

# Maven
log/
target/
```

Initializing a project

Git init

- *git init* create a git repository in local repository

```
$ cd ~/workspace/hei-devops/lesson-03
$ git init
Initialized empty Git repository in ~/workspace/hei-devops/lesson-03
$ ls -a
./  ../  .git/
```

Git clone

- *git clone* start a git repository based on an existing project

```
$ cd ~/workspace/hei-devops/
$ git clone https://gitlab.com/hei-devops/lesson-03.git
Cloning into 'lesson-03'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.

$ ls
lesson-03/
```

Retrieving information

Git status

- *git status* shows the status of files

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        newFile.txt

nothing added to commit but untracked files present
(use "git add" to track)
```

Git log

- *git log* shows all commits done on the project :

```
$ git log
commit 0bad6f9289b75a44dec7e7597a772d7dbd295f4c (HEAD -> master,
origin/master, origin/HEAD)
Author: Denis Cauchois <denis.cauchois@junia.com>
Date:   Fri Aug 24 14:51:48 2018 +0000

    Ajout de README.md
```

Interaction with repositories

Git fetch

- *git fetch* download changes from remote repository to local repository

```
$ git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 2 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (2/2), done.
From https://gitlab.com/hei-devops/lesson-03
   fd6e37a..46c76b1  master     -> origin/master
```

Git pull

- *git pull* transfers changes from remote repository to local repository and workspace (*git fetch && git merge*)

```
$ git pull
Updating 46c76b1..dabc8d0
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
```

Git add

- *git add* is used to add file in the staging area :

```
$ git add newFile.txt

$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   newFile.txt
```

Git commit

- *git commit* transfers changes from staging area to local git repository
- *git commit -a* transfers changes from working area to local git repository
- Each commit must be accompanied by a message (-m option)

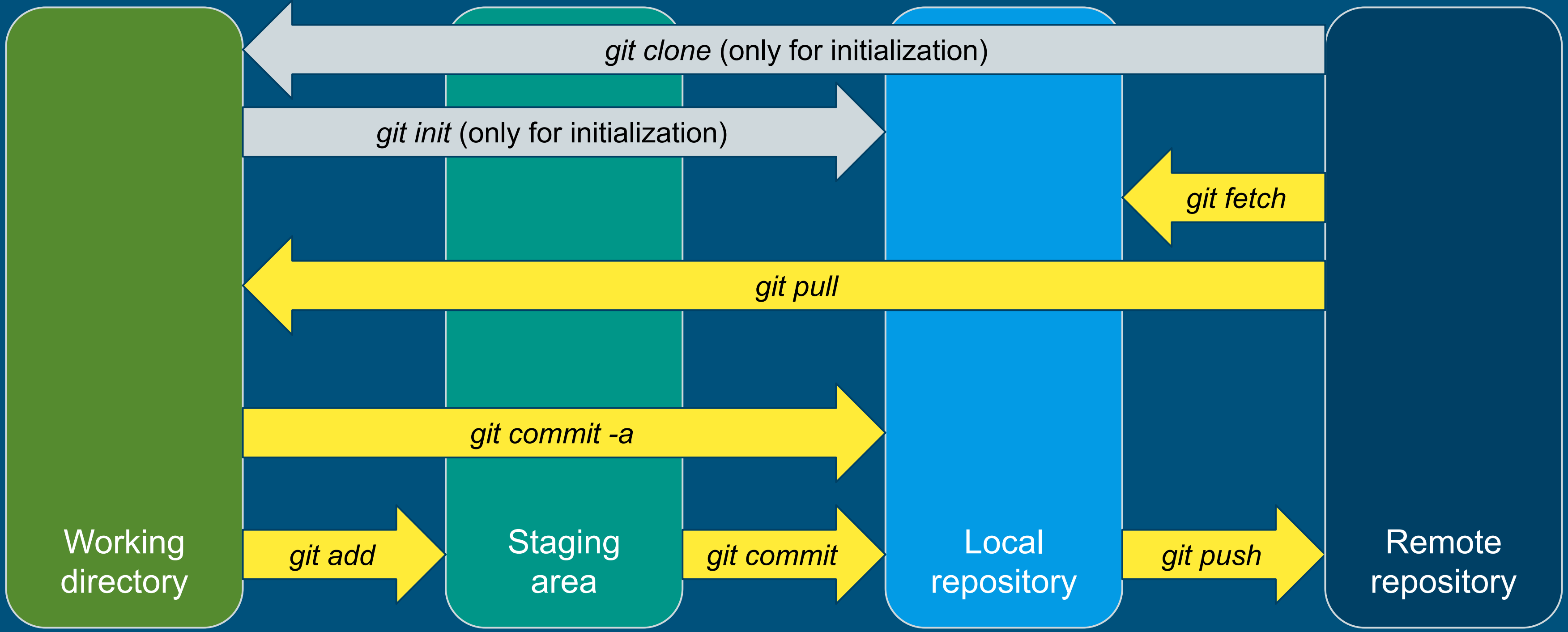
```
$ git commit -m "Add newFile.txt"
[master fd6e37a] Add newFile.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFile.txt
```

Git push

- *git push* upload changes from local git repository to remote git repository
- Each push must transfer a stable version to not block other developers
- Each push must be preceded by a fetch (else git refuses push action)

```
$ git push
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 258 bytes | 129.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://gitlab.com/hei-devops/lesson-03.git
   0bad6f9..fd6e37a  master -> master
```

To sum up


git clone (only for initialization)
git init (only for initialization)
git fetch
git pull
git commit -a
Working directory
git add
Staging area
git commit
Local repository
git push
Remote repository

How to take power with git ?

Branches management

Overview

- Branch is a pointer to one commit (*snapshot of every files of the project*)
- Each branch is independent of other branches
- For each commit on the branch, the branch points to the new commit.
- Default branch name is master

Goals

- Allows multiple users to work in parallel without interfering with each other
- Allows one user to work on multiple features in parallel

Git branch

- *git branch* creates a new branch

```
$ git branch test
```

- *git branch -d* delete an existing branch (if user is not on the branch)

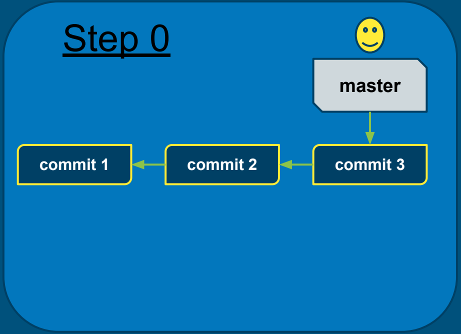
```
$ git branch -d test
Deleted branch test (was ea47f29).
```

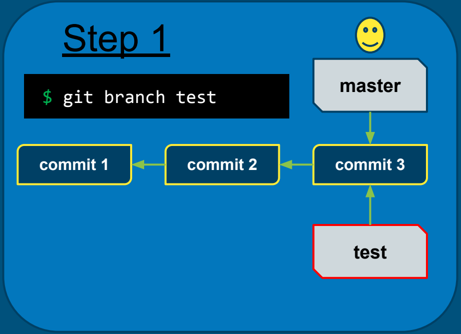
Git checkout

- *git checkout* changes the current branch

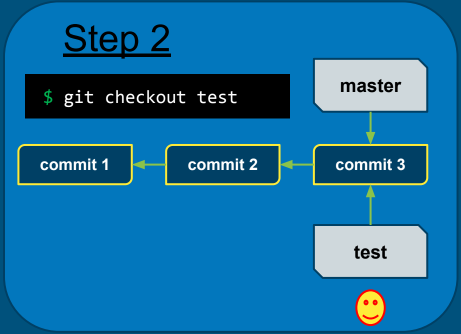
```
$ git checkout test
Switched to branch 'test'
```

Branches : Practical case

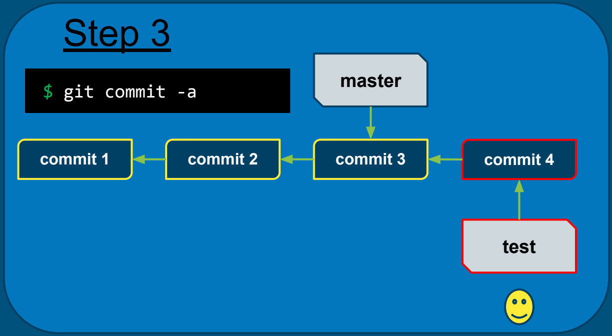




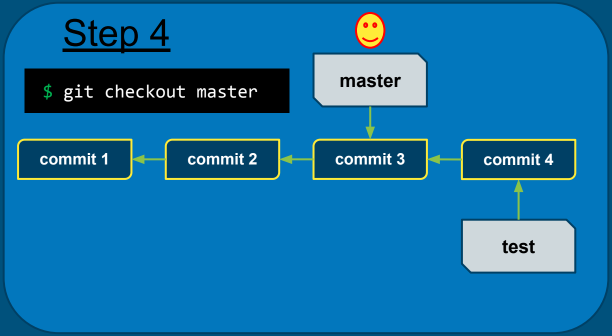
```
$ git branch test
```



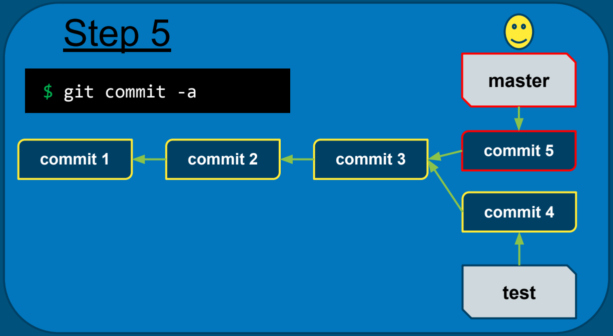
```
$ git checkout test
```



```
$ git commit -a
```



```
$ git checkout master
```



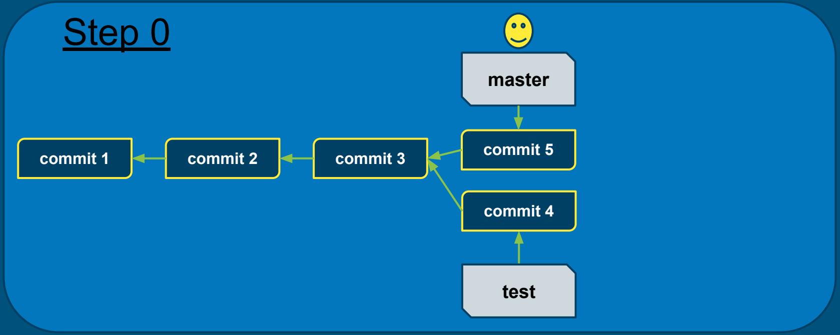
```
$ git commit -a
```

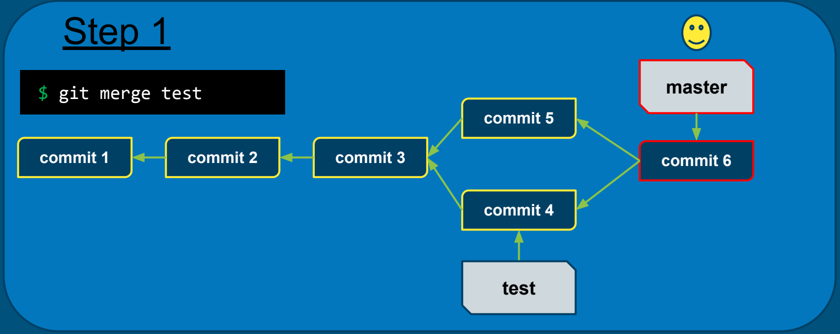
Merge management

Git merge

- *git merge <branch>* merges changes from one branch to current branch

```
$ $ git merge test
Updating ea47f29..9fa7c26
Fast-forward
 test | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test
```





Conflict

- Conflicts are ambiguous situations that GIT is not able to process automatically
- Developer must choose what to do :
  - Keep changes 1
  - Keep changes 2
  - Keep a little bit of both
- Conflict are visible in files :

```
<<<<<<< HEAD

Changes 1

=======

Changes 2

>>>>>>> branch-a
```

Merge conflict

- *git merge* may fail due to conflict that can not be resolved

```
$ git merge test
Auto-merging test
CONFLICT (content): Merge conflict in test
Automatic merge failed; fix conflicts and then commit the result.
```

- Procedure for resolving a merge conflict :
  1. Fix conflicts with an appropriate tool
  2. Commit changes

git mergetool

- *git mergetool* opens a tools to help the user to resolve the conflict.

```
$ git mergetool
Merging:
test

Normal merge conflict for 'test':
  {local}: modified file
  {remote}: modified file
4 fichiers à éditer
```

- To configure vimdiff as tool :

```
$ git config --global merge.tool vimdiff
```

- To configure kdiff3 (need installation before) as a tool :

```
$ git config --global --add merge.tool kdiff3

$ git config --global --add mergetool.kdiff3.path
"C:/Program Files/KDiff3/kdiff3.exe"

$ git config --global --add
mergetool.kdiff3.trustExitCode false
```

Question

What's going to happen ?

1. git branch test
2. git checkout test
3. touch file.txt (create a file)
4. git add file.txt
5. git commit -m “create file.txt”

Working directory

Local Repository

Remote Repository

Thank you for you attention !

Links :

- Sources
  - http://www.commitstrip.com/
  - https://giphy.com/
  - https://git-scm.com/