# Computer Labs: Version Control with Subversion

## 2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

September 20, 2020

# The Problem (by João Cardoso, jcard@fe.up.pt)

```
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it enters in graphics mode, let's make a backup

```
$copy video_gr.c video_gr.v1.c
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it maps graphic memory, let's make another backup

```
$copy video_gr.c video_gr.v2.c
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it draws a pixel, let's make another backup

```
$copy video_gr.c video_gr.v3.c
$edit video_gr.c, make, run, edit, make, run, ...
```

Oops! Does not leave graphics mode, let's retrieve the backup

```
$copy video_gr.v2.c video_grc.c
```

Hmm! This is not the version I want. Shouldn't it be v3? ...
Oops, deleted the last version !@£#%/#*&$@

# The Solution? Git [1]

- Git is a version control system that is able to:
  - Keep several versions of an entire (development) directory tree, or **project**
  - Restore any of the versions it keeps in a consistent way
- Furthermore, it:
  - supports **concurrent access** to the different files or directories in the tree **by several users**;
  - keeps a log of the changes performed to each file/directory that **can be used to document/keep track of the main changes** between versions
  - allows to create new **branches**, i.e. to keep track of the evolution of multiple directory trees that have a common ancestor (i.e. a tree of directory trees)

[1] This transparency is exactly equal to a similar one I wrote for SVN, another version control system, we have used in LCOM.

# Git: Local Repository and Working Directory

Local Repository This is a directory that keeps the **different versions** of the project

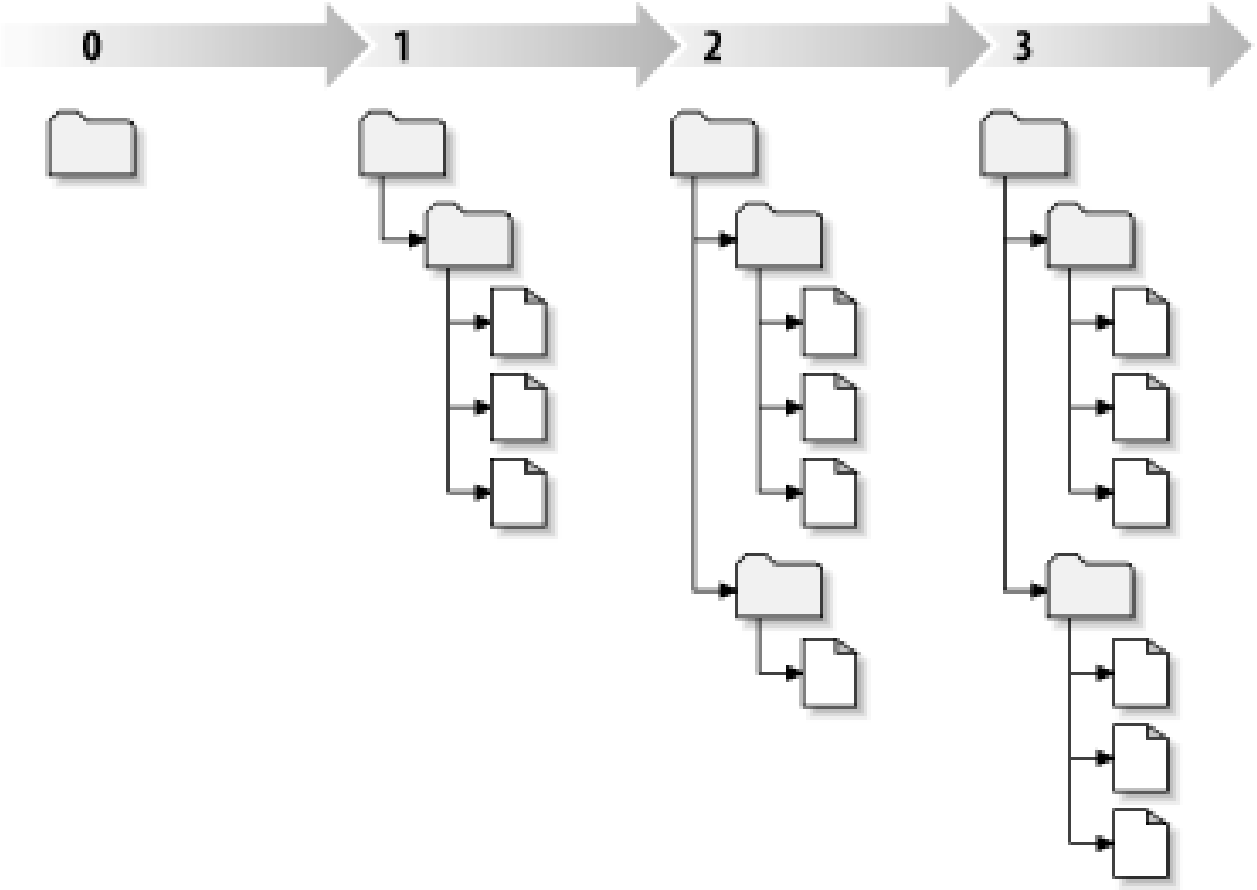
- The term "local" stems from being a subdirectory (`.git/`) of the working directory

Working directory This is a directory that originates on **one version** of the project in the corresponding local repository.

- A file in the working directory may be different from the respective file of the version from which it was derived
  - Actually, a file in the working directory may not exist in the original version.
  - and a file in the original version may have been removed from the working directory
- Other programs, like editors and compilers, do not need to be “version-control-aware”

# Git: Versions

Version A new version is created by **committing** to the local repository the changes done in a working directory

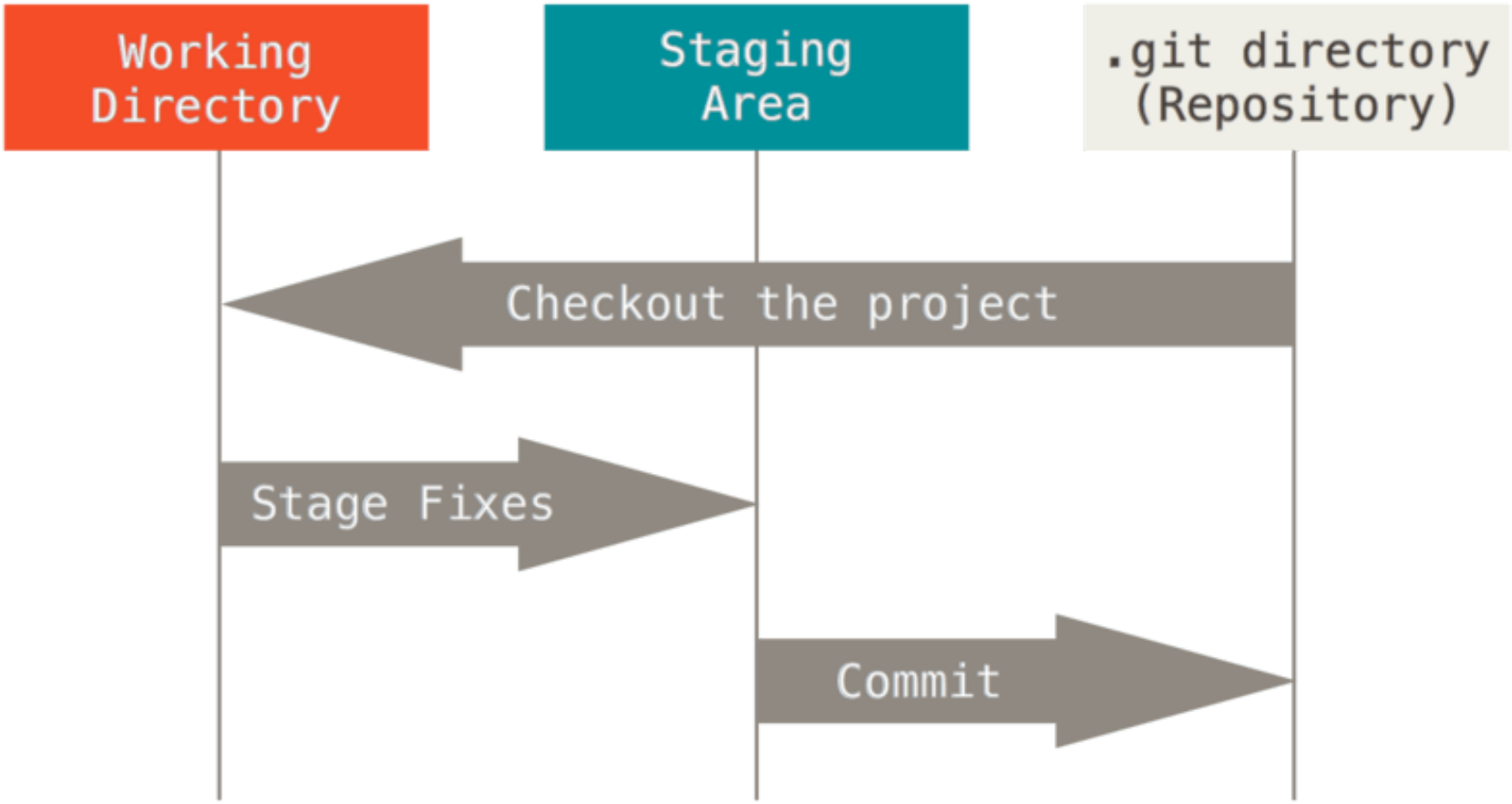



Source: Ben-Collins-Sussman et al. Version Control with Subversion

This image is taken from SVN's documentation; in Git, version numbers are hash values, not integers assigned in increasing order starting at 0.

# Git: Staging Area (or Index)

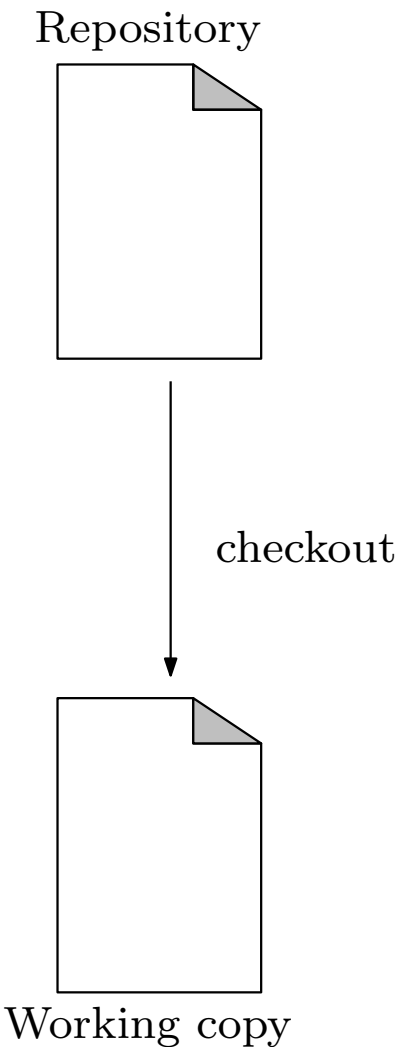
Staging Area is the set of modified files that will be added to the local repository upon commit




Source: Scott Chacon and Ben Straub, Pro Git book

# Git: Basic Usage

- Generate a working copy using **checkout**

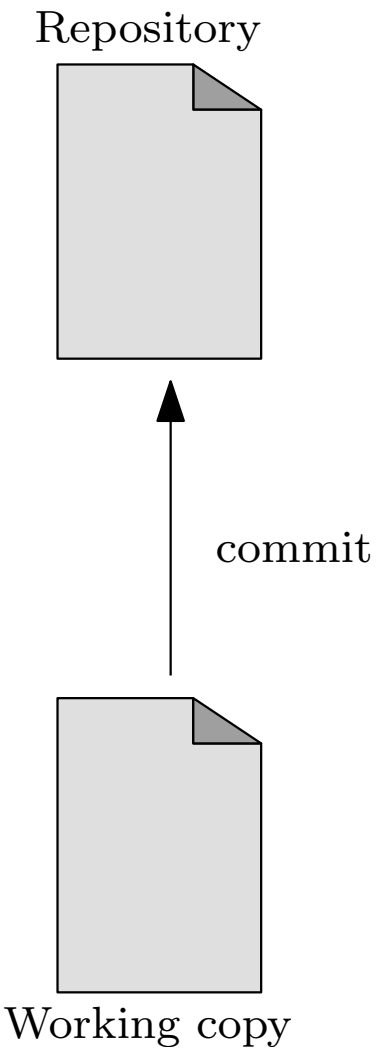


# Git: Basic Usage

- Change the working copy with your favorite editor

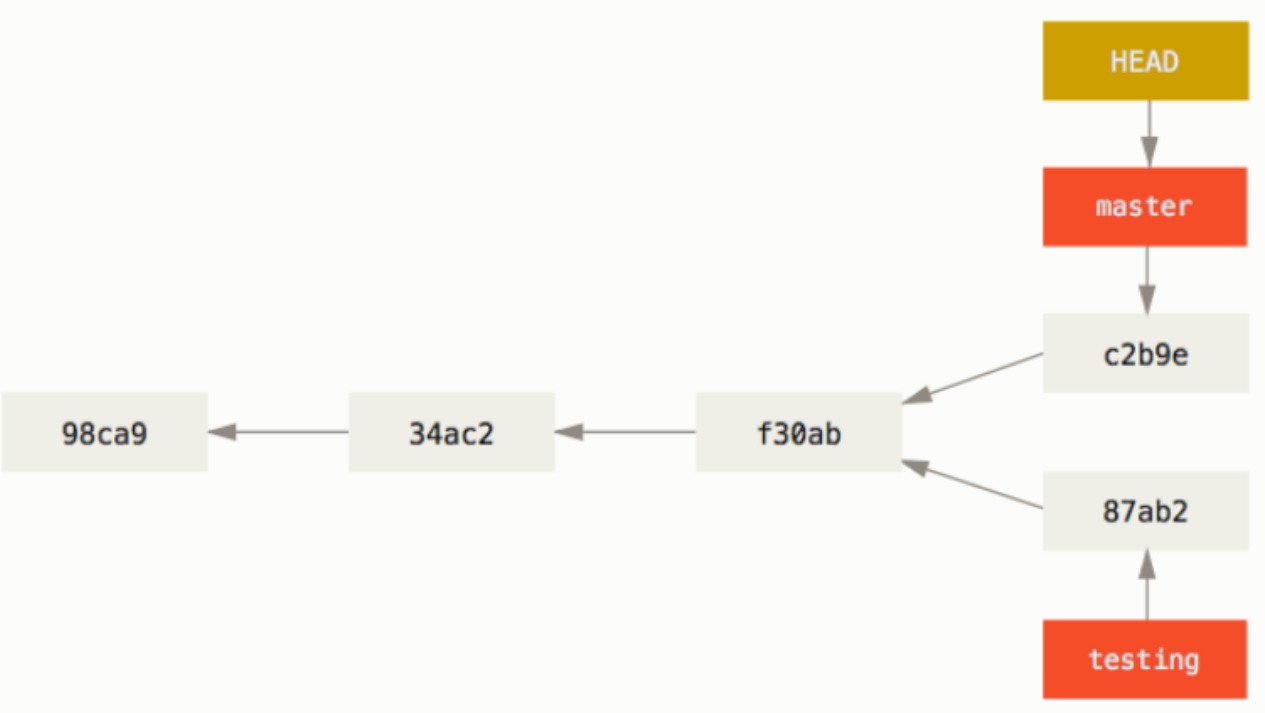
# Git: Basic Usage

- Publish your changes on the local repository with **commit**
  - Git allows you to stage changes and commit with a single command



# Git: Branches

- In Git, like in SVN, the versions kept, i.e. the **commit history**, in a repository may form a tree:

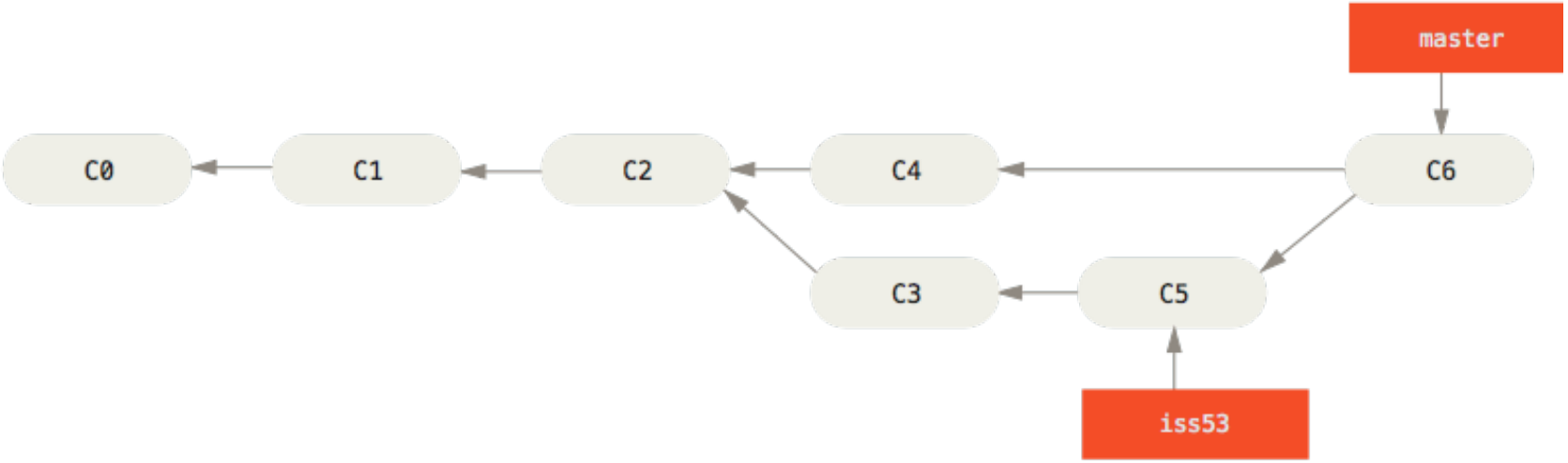



Source: Scott Chacon and Ben Straub, Pro Git book

- This is useful for trying out new ideas without hindering code evolution
- In Git, a **branch** is just a pointer to one version/commit.
  - `master` is the name of the default branch
- `HEAD` is just a reference to the current branch (the one checked out to the working directory)

# Git: Merging

- Merging is the inverse operation of branching:
  - It joins together two or more branches, by creating a new commit

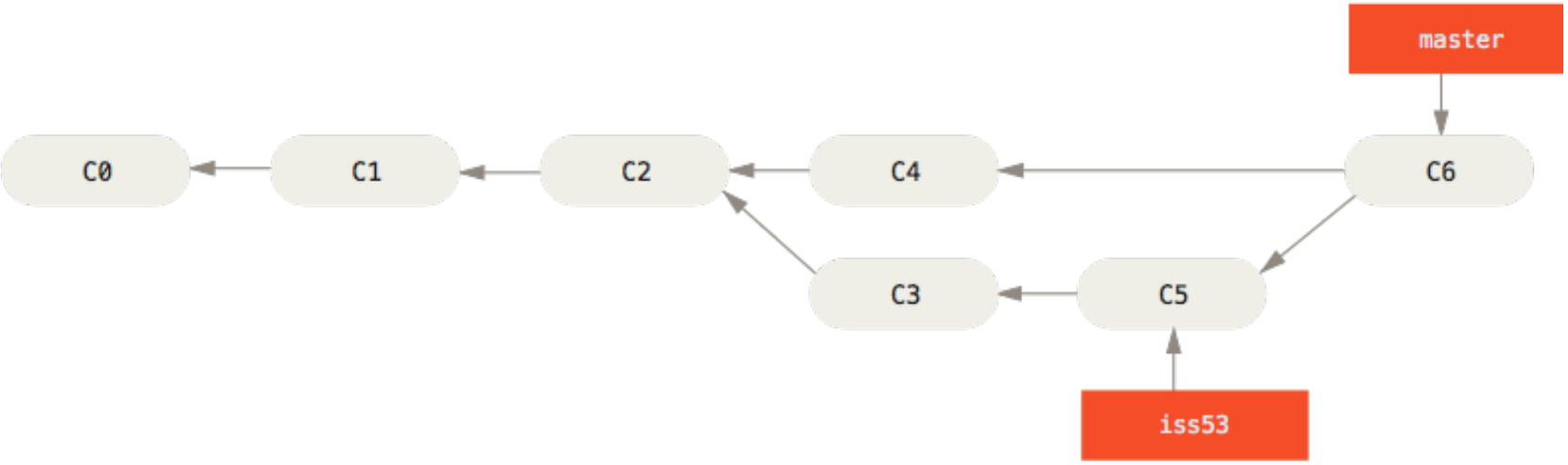



Source: Scott Chacon and Ben Straub, Pro Git book

  - In this case, the new commit, `C6`, belongs to the `master` version, not the `iss53` branch
  - The `iss53` branch can be deleted, but this does not delete commits `C3` and `C5`
  - With merge the commit history of a project becomes a DAG

# Git: Merging Conflicts

- When merging branches, changes to one or more files in different branches may **conflict** with one another:




Source: Scott Chacon and Ben Straub, Pro Git book

- Git tries to resolve these conflicts automatically
- If it is unable to do it, it is up to you to do it
  - This can be messy: avoid adding conflicts if possible.

# Git: Bare and Remote Repositories

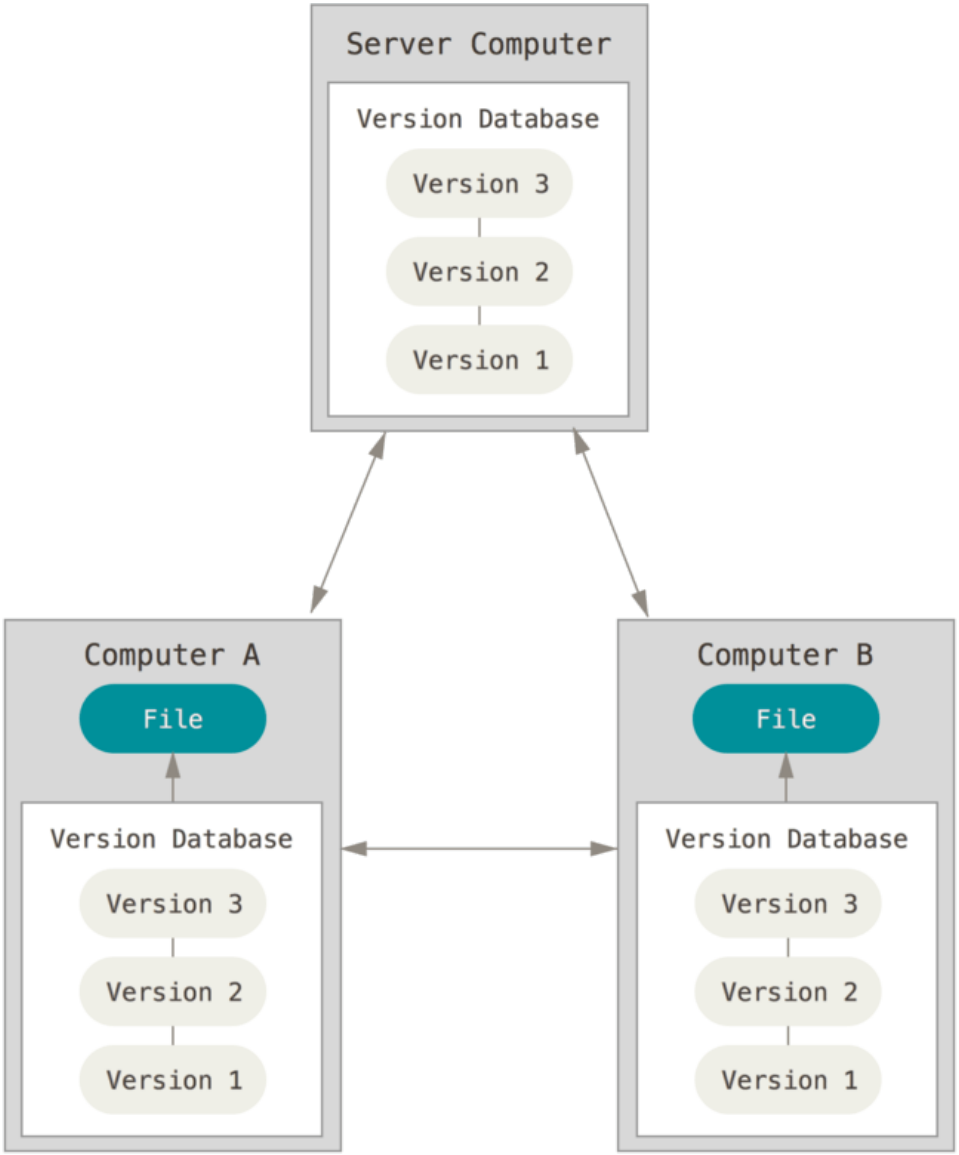
**Remote repositories** a project's files may be kept also in repositories other than the local repository

- E.g., each member of a project's team may have its own working directory, and respective local repository, in its own computer
- Remote repositories are supposed to be synchronized, i.e. have the same project versions, most of the time
  - Synchronization between remote repositories is achieved by merging corresponding branches
  - You should coordinate changes to your project with your teammate to avoid merging conflicts

**Bare repository** is a special Git remote repository

- It has no associated working directory
- It is usually kept in a server computer
  - Bare repositories facilitate keeping remote (non-bare) repositories synchronized

# Git: Repositories




Source: Scott Chacon and Ben Straub, Pro Git book

# Git: (Some) Useful Commands

init create a project, including working dir and local repository

checkout restore working directory

restore restore specified working directory files

add add files to the staging area

- If you add a file in a subdirectory that subidrectory is automatically added

rm remove file/directory both from the working directory and the from the index

reset undo previous operations (that may have affected the index or even the local repository)

mv rename a file/directory in the repository

status show the working dir status

diff show changes between commits, commit and working dir ...

commit record changes (in the staging area) to the repository

log show the commit log

# Git: (Some) Commands For Working with Remotes

clone create a local repository (including working directory) by cloning a remote repository

remote manage (references to) remote repositories

- a repository may have references to more than one remote repository
- the reference may be to a remote repository that is not the one from which the local repository was cloned

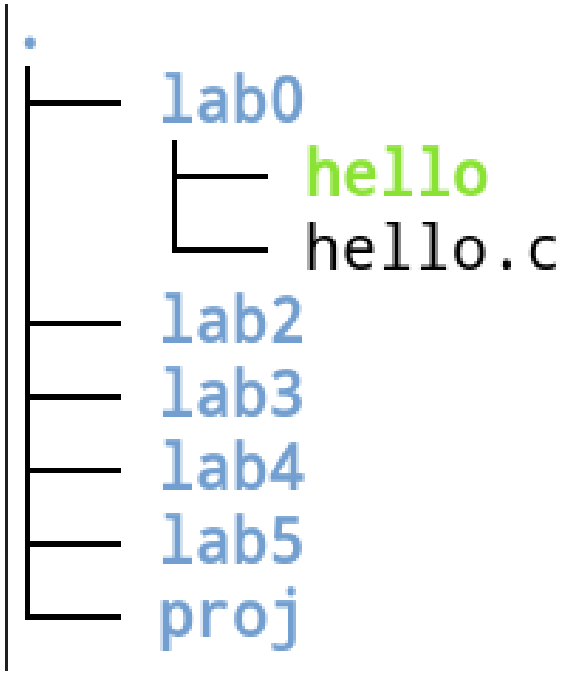
pull synchronize the local repository with a remote repository, by fetching new versions from the remote

- Usually, the remote is a bare repository
- It may require user intervention, if there are conflicts git is unable to resolve

push synchronize/update remote repositories by sending new versions

# Git and LCOM

- Each group must create a project at the Git Service at FEUP (`git.fe.up.pt`) to keep its LCOM work:
  - Both the labs
  - And the project
- To facilitate your life, and ours, you must keep it structured as shown in the picture on the right
  - You can create the structure incrementally
- You may create subdirectories under the "top level" directories
  - This is unlikely to help in the labs
  - But may be useful in the project




- The code for the integration project must be under the `proj/` directory

# Git: Advantages [2]

- It provides automatic backup
- It makes it easy to restore a previous version
- It is supported by most IDEs including Eclipse
  - But you can also use a command line client, even in Minix
- Users can work on any computer
- Members of a team can work simultaneously and independently on the same project
- It logs **who** did/committed **what** and **when**
  - By using appropriate messages or comments, it is also possible to know **why**
- It is possible to try a new approach, and continue development on the older one, if you are not happy with the outcome

[2] This transparency is equal to another I wrote for SVN

# Further Reading

- Search the web for the right tutorial for you on Git
- *Scott Chacon and Ben Straub*, Pro Git book