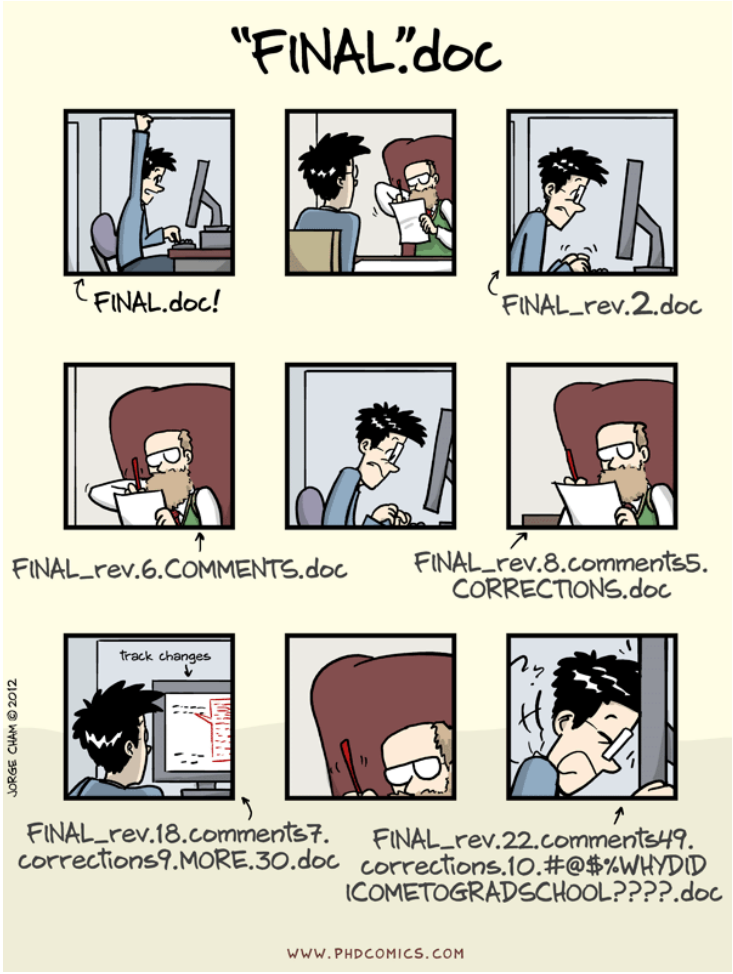
Tópicos em Engenharia Computacional II

Carlos Azevedo

(cdazevedo@ua.pt)

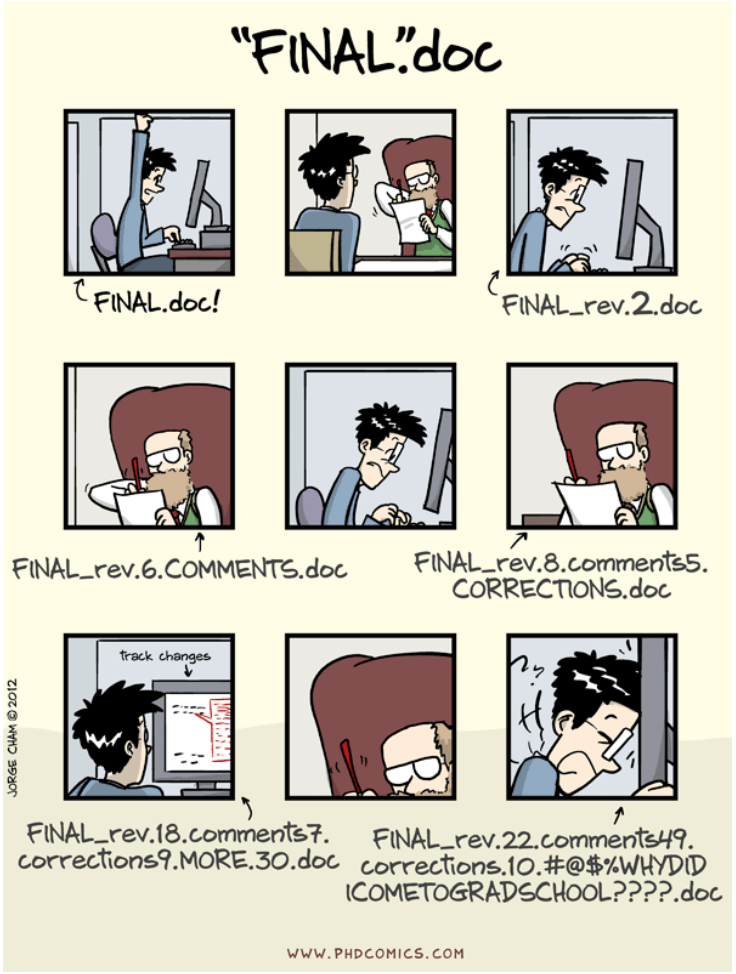
Why Version control?!?

- **Collaboration**
- **Versioning**
- **Rolling Back**
- **Understanding**
- **Scenario** : Multiple students are doing a project together
- **Question**: Why not Google drive, One drive, dropobx, MEGA… ????
- source code management vs file storage management



Why Version control?!?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work



Version control systems

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all “real” projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

What is Git? Why Git?

- Global Information Tracker (acronym name)
- Git is a distributed version control system designed by Linus Torvalds.
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

What is Git? Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion
  - (Depends to whon you ask :))
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons

- Git features:
  - Branches and merging: Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.
  - Small and fast
  - Distributed
  - More! See https://git-scm.com/about

Who uses Git?

Source: https://git-scm.com/

Best way to learn Git?

- Play with it!
- Read about it.
- Use it with other people!
  - Use it in your projects
    - Lab reports
    - Thesys
    - Projects
    - Classes
    - ….

Where to get info and help?

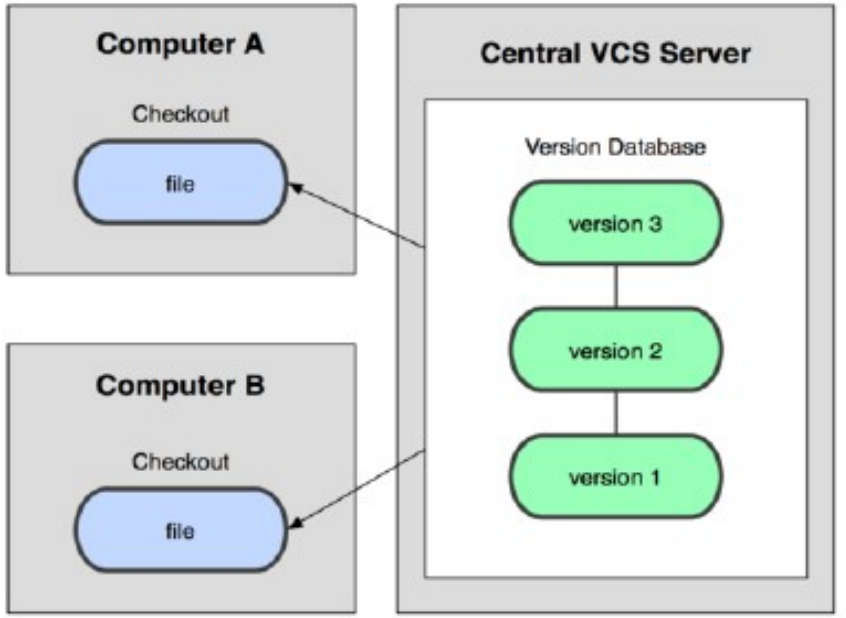
- A simple web search can give you much more information than here
  - Plenty of examples
- Git cheat sheet
  - Git Cheat Sheet (git-cheatsheet.com)
  - Just an example. Many more available
- GitPro eBook. FREE
  - http://git-scm.com/book/en/v2
- Git for computer scientists:
  - https://eagain.net/articles/git-for-computer-scientists/

- Git is primarily a command-line tool
  - – git help verb (where verb = config, add, commit, etc.)
- http://git-scm.com/downloads
  - If you want to set a private server. Note covered in this class
  - Several GUI tools: https://git-scm.com/downloads/guis

Centralize VCS or distributed VCS(Git)?!?

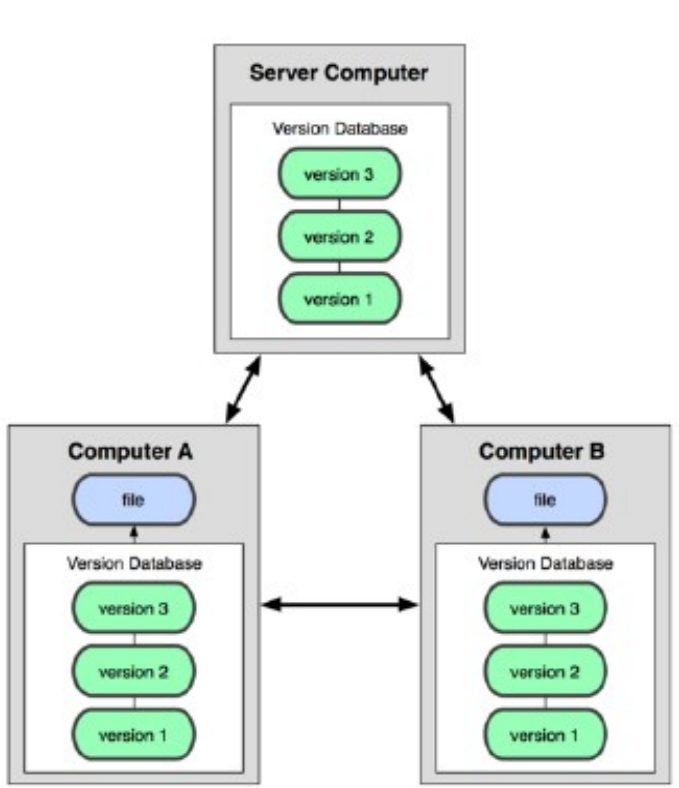
- In Subversion, CVS, etc. a central server repository (repo) holds the "official copy" of the code
  - the server maintains the sole version history of the repo
- User makes "checkouts" to their local copy
  - user makes local modifications
  - User changes are not versioned
- When done, user "check in" back to the server
  - "Check in" increments the repo's version

CENTRALIZED VCS



Centralize VCS or distributed VCS(Git)?!?

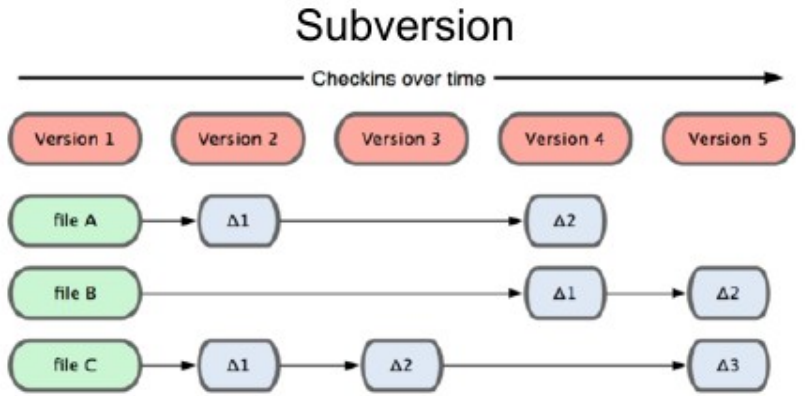
- In distributed VCS (git, for example) user do not "checkout" from a central repo
  - user "clone" it and "pull" changes from it
- Local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo local repo keeps version history
- When you're ready, you can "push" changes back to server

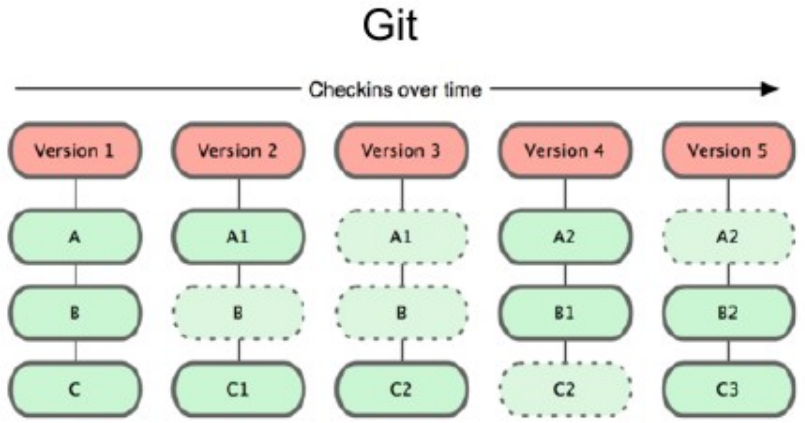


Centralize VCS or distributed VCS(Git)?!?

- Centralized VCS track version data on each individual file.

- Distributed VCS keeps "snapshots" of the entire state of the project.
  - Each "checkin" version of the overall code has a copy of each file in it.
  - Some files change on a given "checkin", some do not.
  - More redundancy, but faster.





Git = GitHub = GitLab = ...???

Question: *Git and GitHub (or GitLab) are the same thing?*

Answer: *No!*

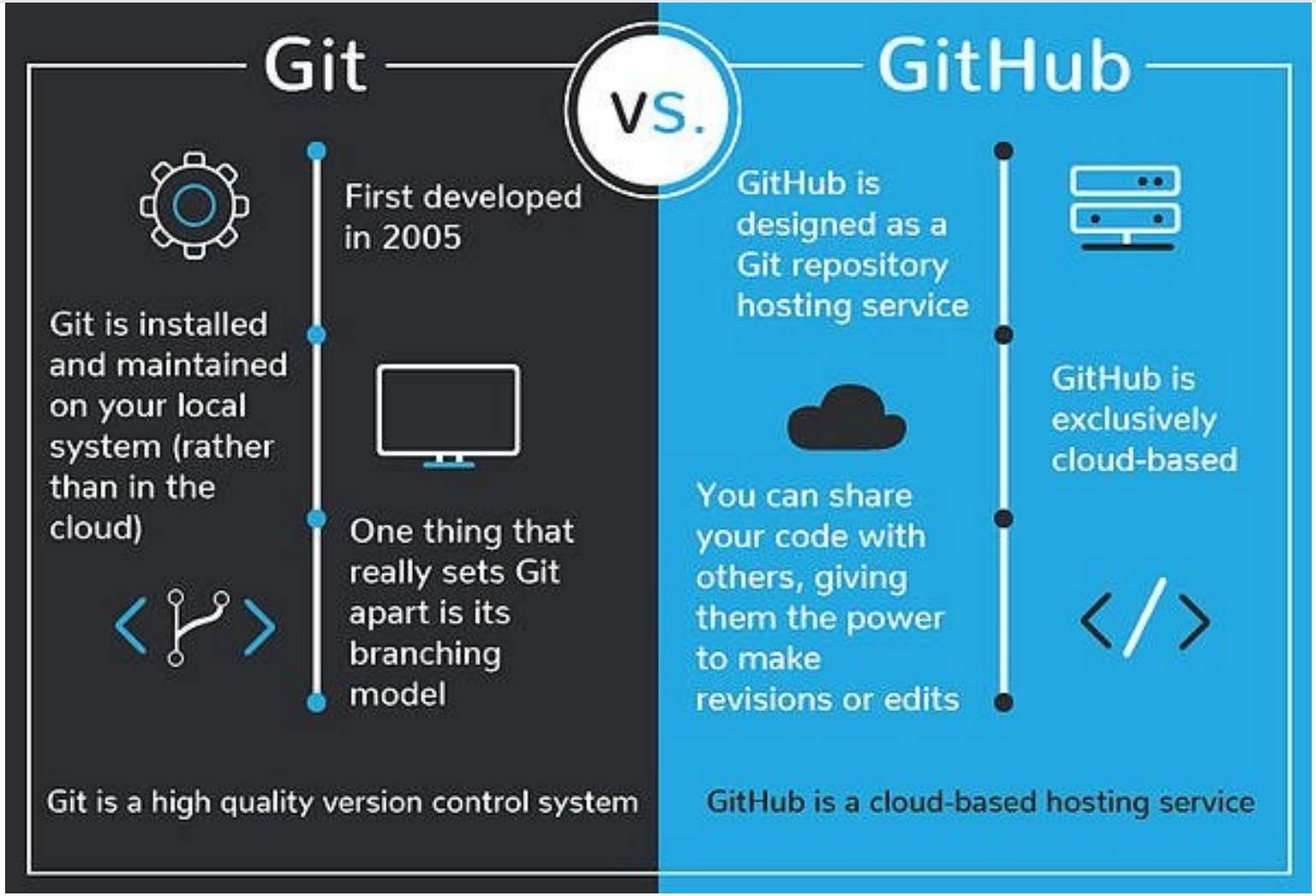
- GitHub or GirLab are sites for online storage of Git repositories.
- Users can get free space for open source projects or can pay for private projects.
- GitHub Education package available at University of Aveiro

Question: *Do I have to use GitHub or GitLab in order to use Git?*

Answer: *No!*

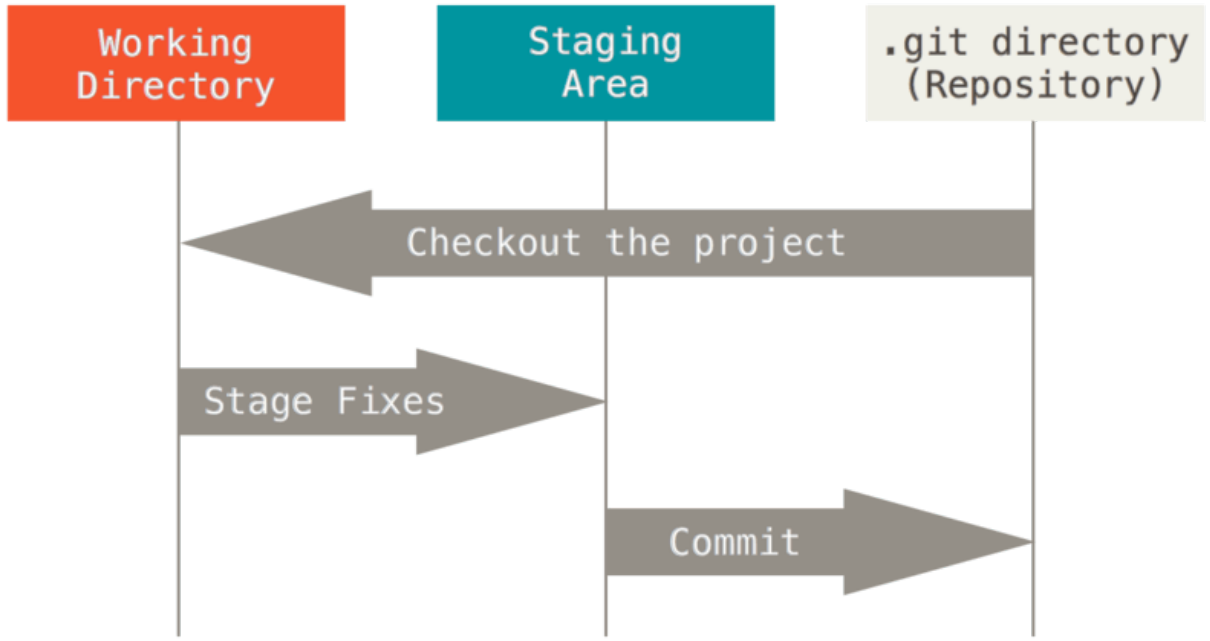
- Users can use Git completely locally for their own purposes
- Someone else can set up a server to share files
- Users can share a repo with other users on the same file system.
- Although:
  - Need to establish a git server (open-source code available)
  - Time consuming and communication safety issues related to internet

$diff Git GitHub



Git jargon and operation logics

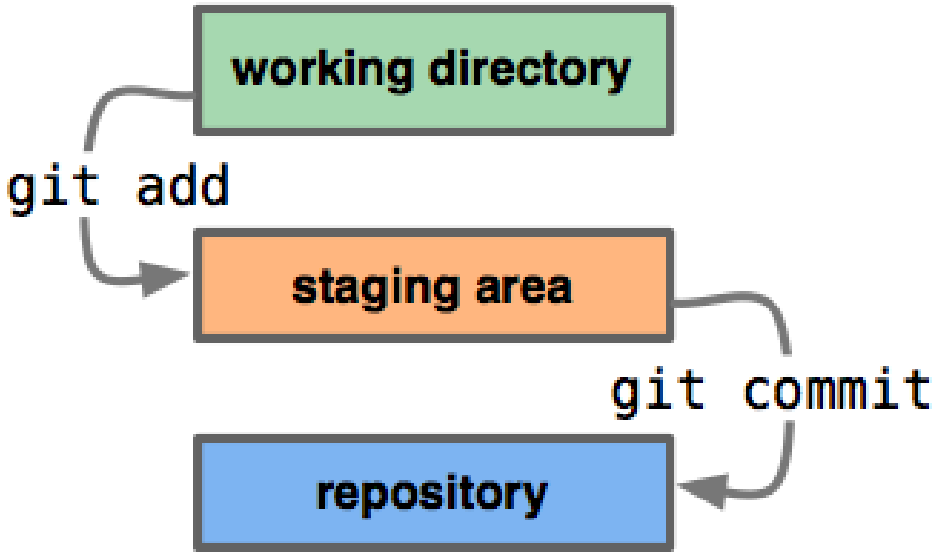
LOCAL OPERATIONS



Unmodified/modified Files

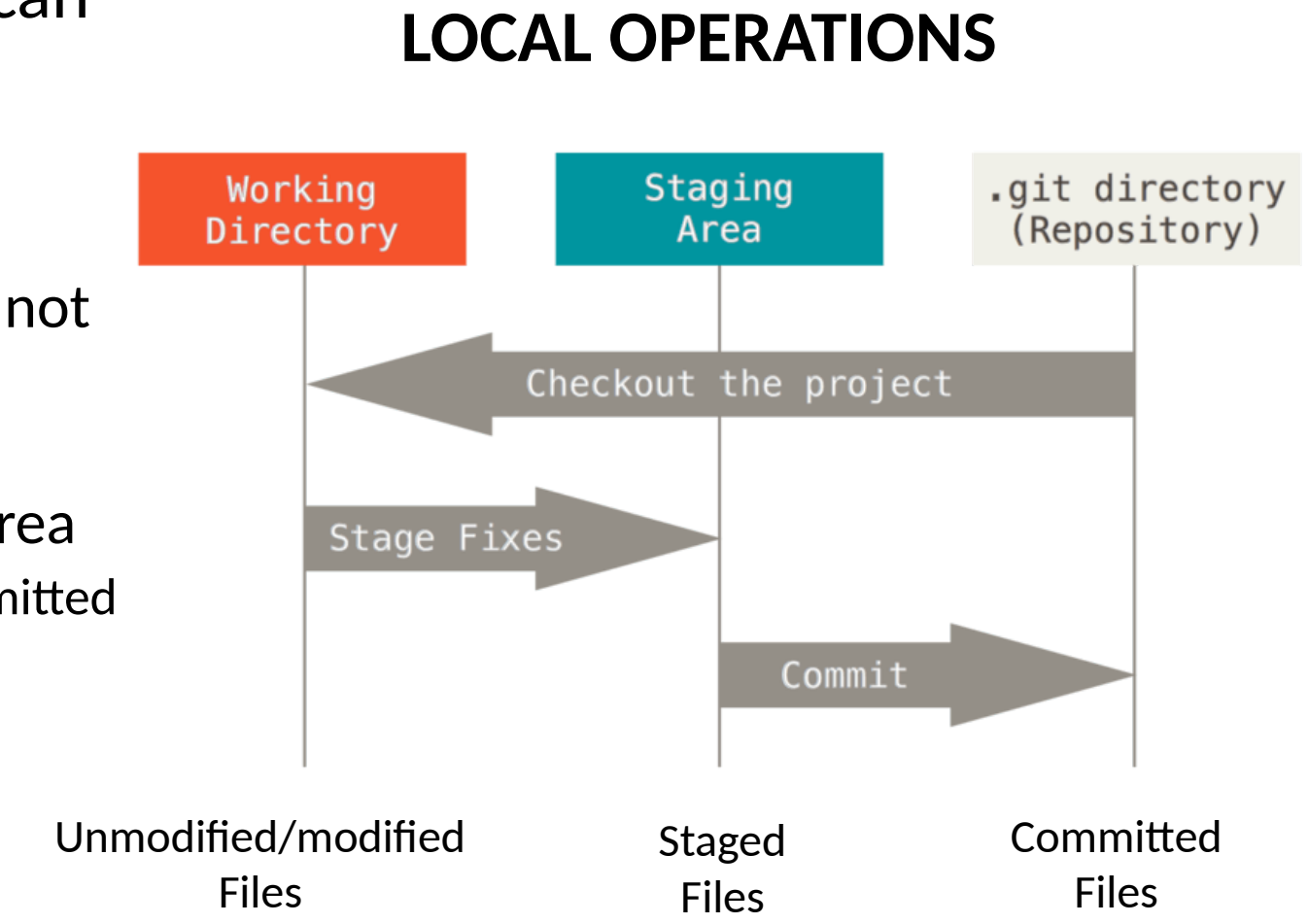
Staged Files

Committed Files



Git jargon and operation logics

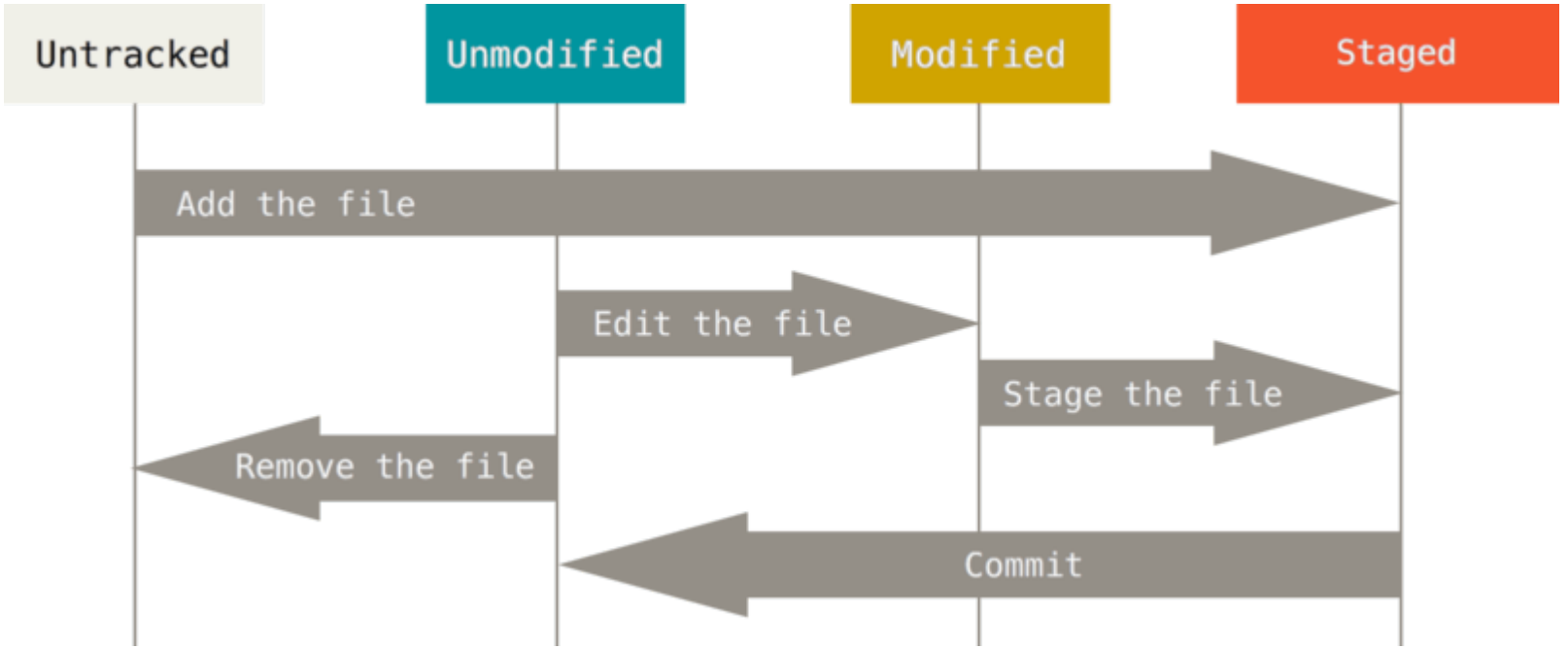
- In your local copy on git, files can be:
  - In your local repo
    - committed
  - Checked out and modified, but not yet committed
    - (working copy)
  - Or, in-between, in a "staging" area
    - Staged files are ready to be committed



Git flow (basic)

- File Life Cycle

LOCAL OPERATIONS



Git commands

- List of (some) useful commands
  - **Git cheat sheet: Your best friend**
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset
  - checkout
  - merge
  - clone
  - pull
  - push
  - (….)

Local repo

Remote repo

Just an example

GitHub

Git Cheat Sheet

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

Install

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git for All Platforms
http://git-scm.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Configure tooling

Configure user information for all local repositories

`$ git config --global user.name "[name]"`
Sets the name you want attached to your commit transactions

`$ git config --global user.email "[email address]"`
Sets the email you want attached to your commit transactions

`$ git config --global color.ui auto`
Enables helpful colorization of command line output

Branches

Branches are an important part of working with Git. Any commits you make will be made on the branch you're currently "checked out" to. Use `git status` to see which branch that is.

`$ git branch [branch-name]`
Creates a new branch

`$ git checkout [branch-name]`
Switches to the specified branch and updates the working directory

`$ git merge [branch]`
Combines the specified branch's history into the current branch. This is usually done in pull requests, but is an important Git operation.

`$ git branch -d [branch-name]`
Deletes the specified branch

Create repositories

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

`$ git init`
Turn an existing directory into a git repository

`$ git clone [url]`
Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

The .gitignore file

Sometimes it may be a good idea to exclude files from being tracked with Git. This is typically done in a special file named `.gitignore`. You can find helpful templates for `.gitignore` files at github.com/github/gitignore.

Synchronize changes

Synchronize your local repository with the remote repository on GitHub.com

`$ git fetch`
Downloads all history from the remote tracking branches

`$ git merge`
Combines remote tracking branch into current local branch

`$ git push`
Uploads all local branch commits to GitHub

`$ git pull`
Updates your current local working branch with all new commits from the corresponding remote branch on GitHub. `git pull` is a combination of `git fetch` and `git merge`

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ git help *[command]*

- get help info about a particular command
- Good friend
- Example:

$ git help *config*

NAME
    git-config - Get and set repository or global options

SYNOPSIS
    git config [<file-option>] [type] [-z|--null] name [value [value_regex]]
    git config [<file-option>] [type] --add name value
    git config [<file-option>] [type] --replace-all name value [value_regex]

(...)

DESCRIPTION
    You can query/set/replace/unset options with this command. The name is actually the section and the key separated by a dot, and the value will be escaped.

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ git config

```
usage: git config [<options>]

Config file location
    --global            use global config file
    --system            use system config file
    --local             use repository config file
    --worktree          use per-worktree config file
    -f, --file <file>   use given config file
    --blob <blob-id>    read config from given blob object

Action
    --get               get value: name [value-regex]
    --get-all           get all values: key [value-regex]
    --get-regexp        get values for regexp: name-regex [value-regex]
    --get-urlmatch      get value specific for the URL: section[.var] URL
    --replace-all       replace all matching variables: name value [value_regex]
    --add               add a new variable: name value
    --unset             remove a variable: name [value-regex]
```

Nothing happens?!?

- Follow the tips from command
- Use "$ git help config"!!!
- Search in web the usage!!
- Use git cheat sheet!!!

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ git config

Configure tooling

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```
Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```
Sets the email you want attached to your commit transactions

Linux systems:

- User-specific configuration file. Also called "global" configuration file.
- File stored in home folder
  - ~/.gitconfig

$ cat ~/.gitconfig

```
[user]
   name = Carlos Azevedo
   email = cdazevedo@ua.pt
```

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ **git init**

- Starts a new local repo inside current directory
  - Unnamed repository
  - Must change descriptions in .git folder

$ **git init [project-name]**

- Creates a folder inside current directory named with the "project-name"

Linux systems:

$ ls project-name/.git
branches
config
Description
HEAD
(...)

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ **git status**

- View the status of your files in the working directory and staging area
- Your new BFF
- Maybe the most useful command
- Use it often

Linux systems:

$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ **git add *files***

- adds file contents to the staging area
- Preparation to versioning before commit
- Single or multiple files can be added
- Wildcards accepted (ex: *$git add *.py*)

Maybe you staged the file and wanted to unstaged it:

- "*git rm --cached <file>...*" to unstage

Linux systems:

$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ **git commit**

- Records a snapshot of the staging area
- Records file snapshots permanently in version history
- A Git object, a snapshot of your entire repository compressed into a SHA
  - SHA stands for "secure hashing algorithm"
- **$ git commit –m "[descriptive message]"**
  - The descriptive message is be short and clear containing the description on what was changed/implemented
  - Fundamental for versioning control and for later software debug in case of problems
  - Do not underestimate the importance of a clear descriptive message
  - Output of changes is printed
- **$git status** should be performed to confirm the commit
- Errors and mistake can and will happen
  - Commits can be erased and reset (see ***reset*** command)

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ git diff

- shows the difference of what is staged and what is modified but unstaged
- Also used to check differences in branches (later topic) and much more

Again, "***git help [command]***" can give you important information

Linux systems:

$ git help diff

git-diff - Show changes between commits, commit and working tree, etc

SYNOPSIS

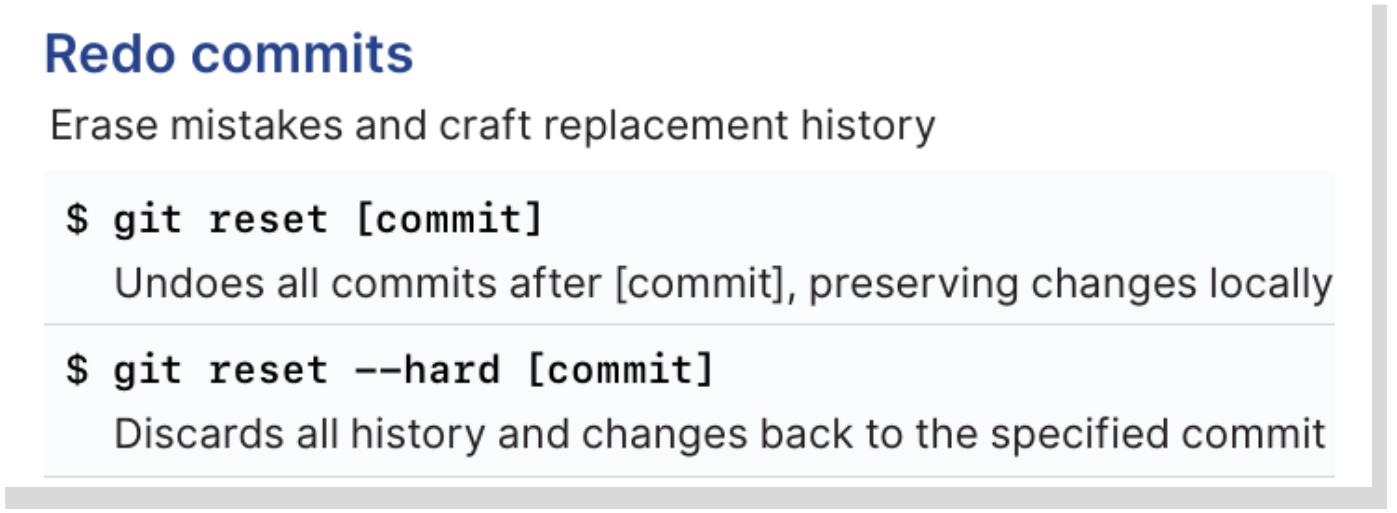
```
git diff [<options>] [<commit>] [--] [<path>...]
git diff [<options>] --cached [<commit>] [--] [<path>...]
git diff [<options>] <commit> <commit> [--] [<path>...]
git diff [<options>] <blob> <blob>
git diff [<options>] --no-index [--] <path> <path>
(...)
```

Git commands

- Some git commands
  - help
  - config
  - init
  - status
  - add
  - commit
  - diff
  - reset

$ **git reset**

- Because errors and mistakes are unavoidable
  - At least for me
  - Is this not one of the great advantages of versioning?
- **CAUTION!** Changing the history namely in remote repos (aka GitHub) can produce nasty effects. Be sure you know what you are doing



Git Ignore

- Should I commit all my files?
  - Short answer: NO!!!!
- Why?
  - Let me give you an example:
    - You will probably compile code
    - Your executable file may not run in all systems (likely to happen)
    - Your compile auxiliar files will also change
    - Imagine you are making calculations: your results should be in the same repo as the code?
  - So why track changes using this files?
    - No sense at all
    - May take lot of space from your online repo
    - Will make the process slower
- How to avoid it?
  - Use "git ignore"
  - How to use it?
    - Use the .gitignore file

Git Ignore

- First step to ignore files:
  - Create a .gitignore file
  - .gitignore is a file that is designed for just this purpose.
  - It’s a good idea to start with an ignore file. If you don’t, you’ll have to go back later and delete the files listed in .gitignore from the repository. By starting with an ignore file, they aren’t added to the repo in the first place.
- In a collaboration:
  - .gitignore is added to your repo so that all repo users have it.
  - What goes into .gitignore will need to be decided by you and your git project collaborators.
  - Be careful – if you add a file to .gitignore after it’s already been tracked, potential issues

Git Ignore

- Can use wildcards (e.g. *.pyc, *.png, Images/*, etc.)
- Do not need to start from scratch
  - A list of recommended .gitignore files:
    - https://github.com/github/gitignore
  - You can start from the recommended list and change it as you wish

Example: C++.ignore file

```
# Prerequisites
*.d
# Compiled Object files
*.slo
*.lo
*.o
*.obj
# Precompiled Headers
*.gch
*.pch
# Compiled Dynamic libraries
*.so
*.dylib
*.dll
# Fortran module files
*.mod
*.smod
# Compiled Static libraries
*.lai
*.la
*.a
*.lib
# Executables
*.exe
*.out
*.app
```