Introduzione sistema git per edizioni collaborative

Angelo Mario Del Grosso

angelo.delgrosso@ilc.cnr.it

CNR-ILC-LicoLab

Istituto di Linguistica Computazionale "A. Zampolli",
14th June 2019

Outline

Progress status

What is my work about

Digital and Computational Philology

Analisi, progettazione e sviluppo di componenti software per sistemi di linguistica e filologia digitale/computazionale volti alla produzione, rappresentazione, trattamento, fruizione e interrogazione di testi di tradizione medievale, a stampa e di autori moderni e contemporanei.

Topic of the talk

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

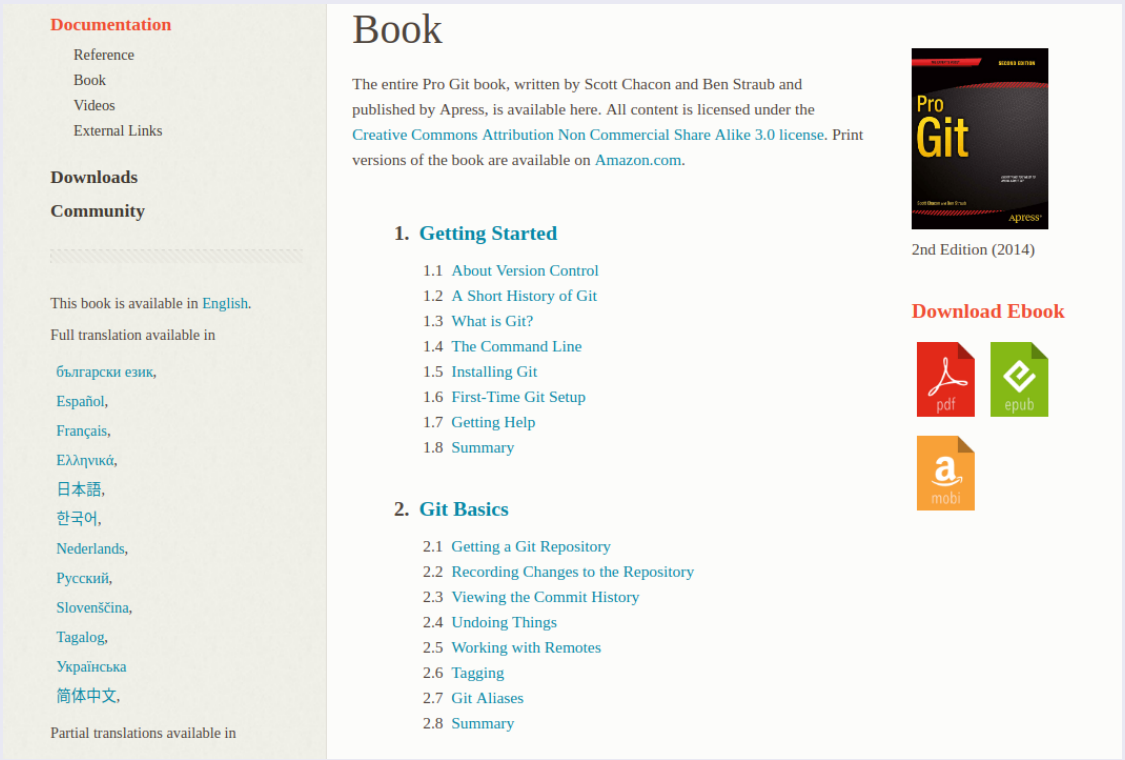
- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

First of all

- Version Control Systems (VCSs)
- Git usage through the main CLI commands
- Cloning, modifying, contributing, diffing, logging
- Working with remotes
- GitHub hosting service (little tips on projects and organization)
- Stashing and Branching model (NO WITHIN THIS TALK)
- Adavanced git tools (NO WITHIN THIS TALK)

Topic of the talk

Book which this workshop is derived from



Documentation

Reference
Book
Videos
External Links

Downloads

Community

This book is available in English.

Full translation available in

български език,
Español,
Français,
Ελληνικά,
日本語,
한국어,
Nederlands,
Русский,
Slovenščina,
Tagalog,
Українська
简体中文,

Partial translations available in

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the Creative Commons Attribution Non Commercial Share Alike 3.0 license. Print versions of the book are available on Amazon.com.

1. Getting Started
   - 1.1 About Version Control
   - 1.2 A Short History of Git
   - 1.3 What is Git?
   - 1.4 The Command Line
   - 1.5 Installing Git
   - 1.6 First-Time Git Setup
   - 1.7 Getting Help
   - 1.8 Summary
2. Git Basics
   - 2.1 Getting a Git Repository
   - 2.2 Recording Changes to the Repository
   - 2.3 Viewing the Commit History
   - 2.4 Undoing Things
   - 2.5 Working with Remotes
   - 2.6 Tagging
   - 2.7 Git Aliases
   - 2.8 Summary

2nd Edition (2014)

Download Ebook

pdf epub mobi

`https://git-scm.com/book/en/v2`

Topic of the workshop

Working Session - Example of using Git

```
MacBookAir-Angelo:git-esercitazione angelo$ git diff
MacBookAir-Angelo:git-esercitazione angelo$ vim myEdition.xml
MacBookAir-Angelo:git-esercitazione angelo$ git diff
diff --git a/myEdition.xml b/myEdition.xml
index 74ba00e..089a9fa 100644
--- a/myEdition.xml
+++ b/myEdition.xml
@@ -1,5 +1,5 @@
 <?xml version="1" encoding="UTF-8"?>
-<!-- comment after tag -->
+<!-- comment after tag and diffing -->
 <TEI>
   <teiHeader>
     <fileDesc>
MacBookAir-Angelo:git-esercitazione angelo$ git diff --word-diff
diff --git a/myEdition.xml b/myEdition.xml
index 74ba00e..089a9fa 100644
--- a/myEdition.xml
+++ b/myEdition.xml
@@ -1,5 +1,5 @@
<?xml version="1" encoding="UTF-8"?>
<!-- comment after tag {+and diffing+} -->
<TEI>
  <teiHeader>
    <fileDesc>
MacBookAir-Angelo:git-esercitazione angelo$
```

Progress status

VCS

Version control (VCS) is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Benefits

- It allows you to revert selected files back to a previous state
- compare changes over time
- who last modified something that might be causing a problem
- ...

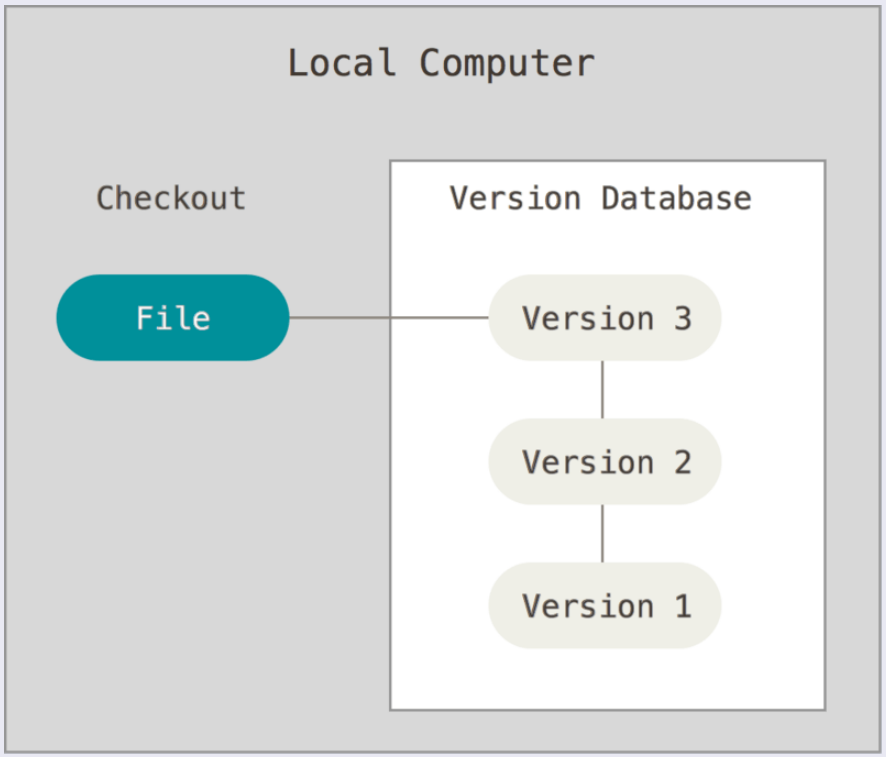
VCS Main feature

Using a VCS also generally means that if you screw things up or lose files, you can easily recover

Different VCS Architectures

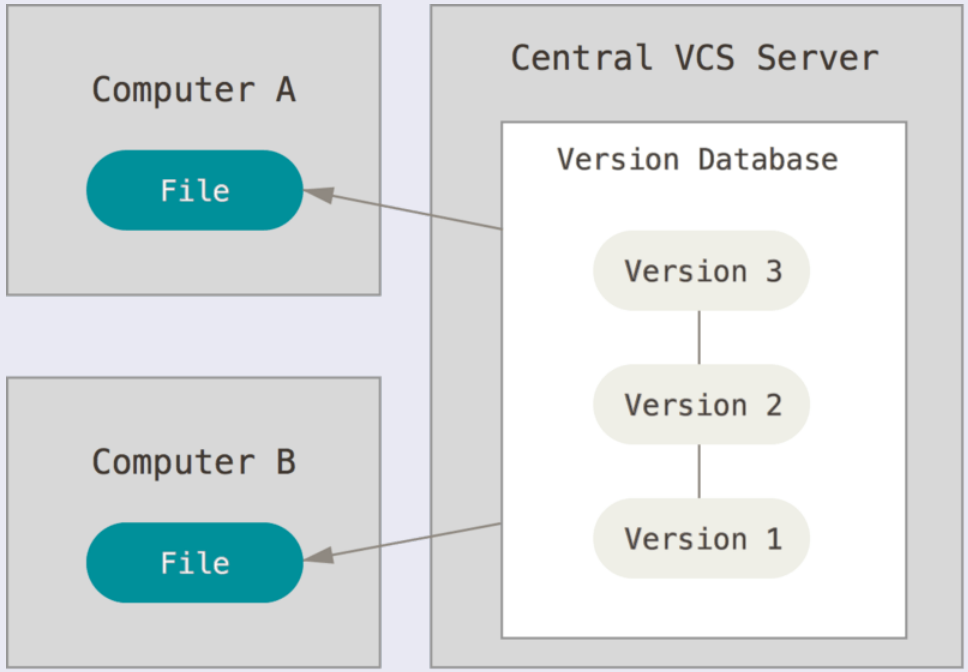
- Local Version Control System (RCS)
- Centralized Version Control System (CVS, SVN)
- Distributed Version Control System (GIT, Mercurial)

Local Version Control System

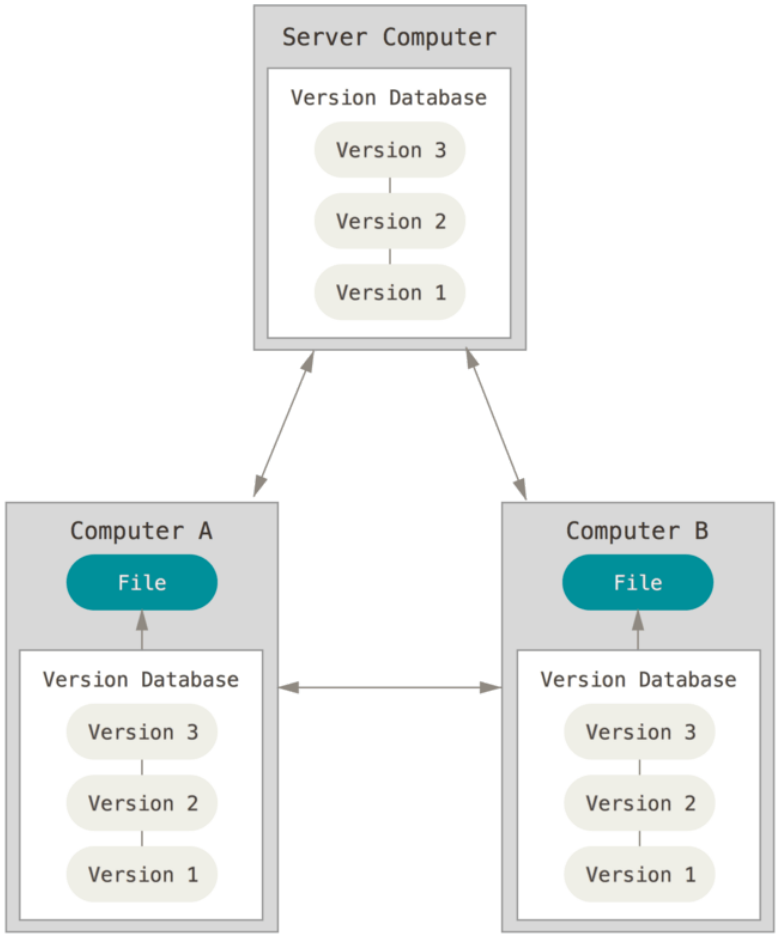


database that kept all the changes to files under control

Centralized Version Control System



Need to collaborate: single server that contains all the files



Client repositories can be copied back up to the server to restore it

GIT DVCS

- Started by Linux community
- Fast and efficient
- Simple design
- non-linear development
- fully distributed
- handle large projects
- easy to use

GIT DVCS

With Git, every time you commit, or save the state of your project, Git basically **takes a picture of what all your files look like** at that moment and stores a **reference to that snapshot**.

GIT DVCS

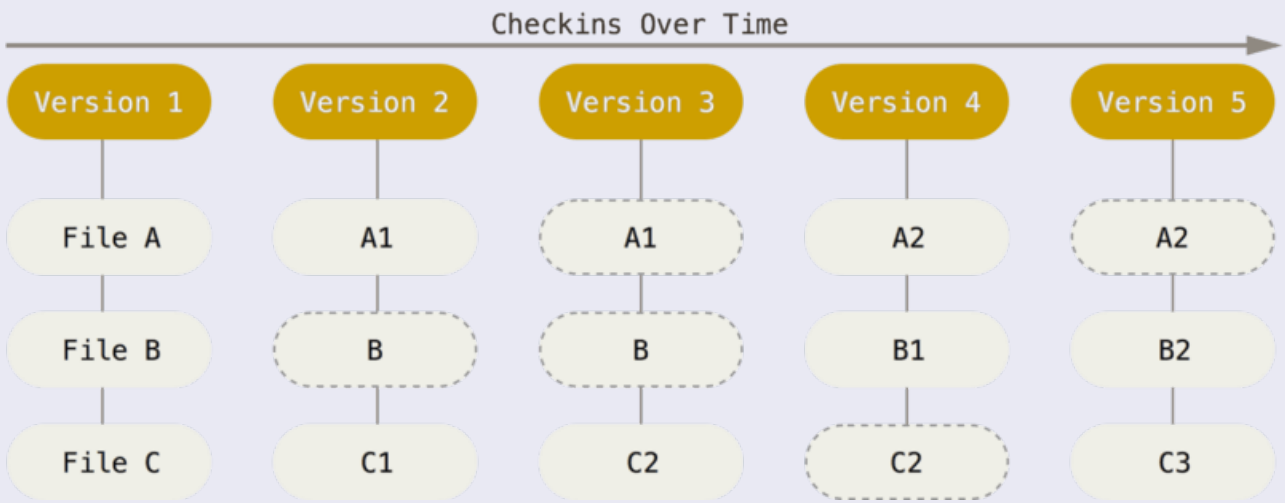
Everything in git is **checksummed before it is stored** and is then referred to by that checksum

GIT DVCS

40-character string composed of hexadecimal characters

a62bc012b405ee47d26b695708063a9f2ffad243

GIT DVCS

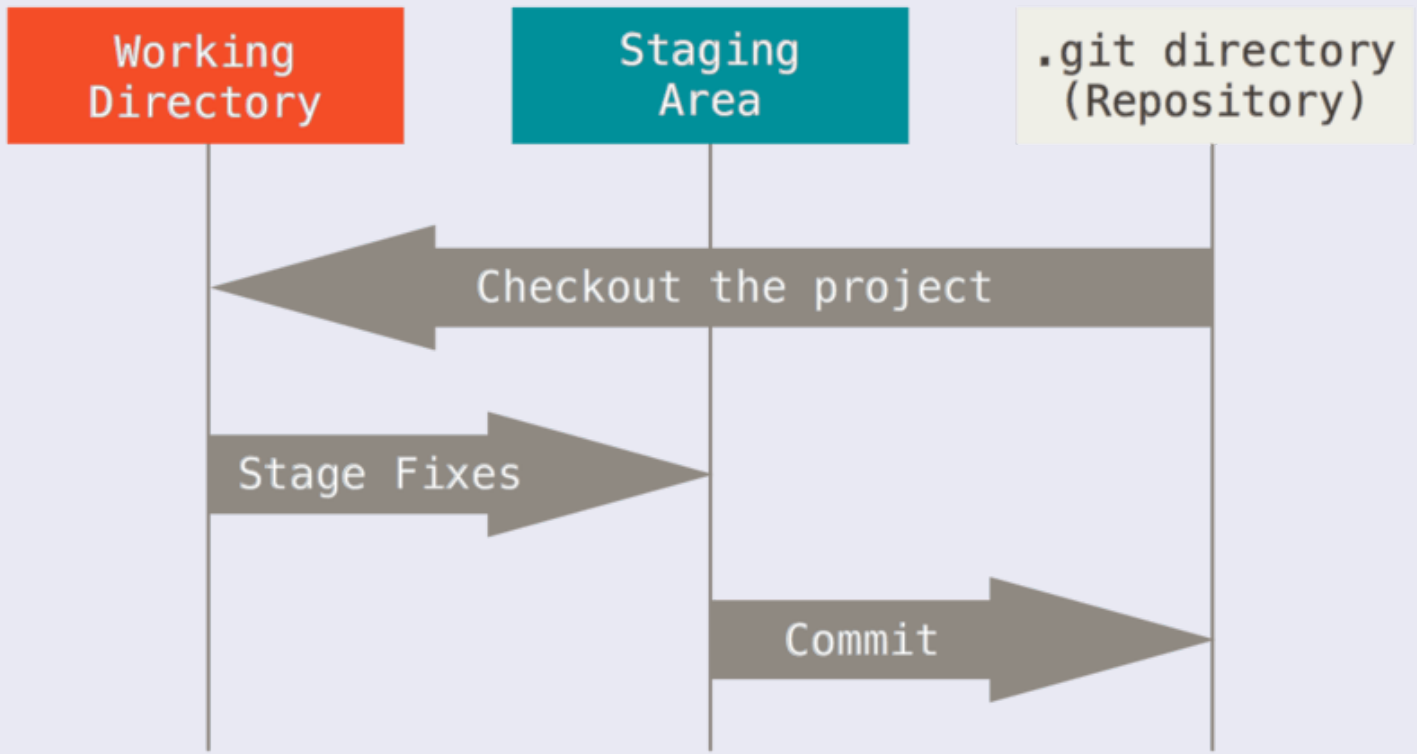


Git has three main states that your files can reside in

GIT DVCS

- committed
- modified
- staged

GIT Areas



Git has three main states that your files can reside in

GIT local workflow

- modify files in your working tree
- stage just those changes you want to be part of your next commit
- do a commit which stores that snapshot permanently to your git directory

Progress status

Git command line environment

The command line is the only place you can run all Git commands.

GUIs environment

GUIs implement only a partial subset of Git functionality for simplicity

Git and GitHub
Getting started with Git


```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clona un repository in una nuova directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Aggiunge il contenuto del file a index
   mv        Sposta o rinomina un file, una directory o un link simbolico
   reset     Ripristina l'HEAD corrente allo stato specificato
   rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   grep      Stampa le righe corrispondenti ad un modello
   log       Mostra log del commit
   show      Mostra vari tipi di oggetti
   status    Show the working tree status

grow, mark and tweak your common history
   branch    Elenca, crea o elimina branch
   checkout  Switch branches or restore working tree files
   commit    Registra modifiche nel repository
   diff      Show changes between commits, commit and working tree, etc
   merge     Unisce due o più cronologie di sviluppo
   rebase    Reapply commits on top of another base tip
   tag       Crea, elenca, elimina o verifica un oggetto tag firmato con GPG

collaborate (see also: git help workflows)
   fetch     Scarica oggetti e ref da un altro repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Aggiorna i ref remoti insieme agli oggetti associati

'git help -g' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
```

Git comes with a tool called **git config** that lets you get and set configuration variables that control all aspects of how Git looks and operates

git config

- system (all users, all repositories)
- global (all repositories, single user)
- local (single repository, single user)

The first thing you should do when you install Git is to set your **user** name and **email** address

git config

- `git config --global user.name "Angelo Mario Del Grosso"`
- `git config --global user.email "angelo.delgrosso@ilc.cnr.it"`

Git and GitHub
Checking Your Settings


```
MacBookAir-Angelo:git-esercitazione angelo$ git config --list
credential.helper=osxkeychain
user.name=angelodel80
user.email=angelodel80@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
MacBookAir-Angelo:git-esercitazione angelo$
MacBookAir-Angelo:git-esercitazione angelo$ git config user.name
angelodel80
MacBookAir-Angelo:git-esercitazione angelo$ git config user.email
angelodel80@gmail.com
MacBookAir-Angelo:git-esercitazione angelo$
```

help while using git

- `git help <verb>`
- `man git-<verb>`
- `git <verb> --help`
- `git <verb> -h`

fundamental capabilities

- configure and initialize a repository
- tracking files
- stage and commit changes
- ignore certain files and file patterns
- undo mistakes
- browse the history and view changes
- push and pull from remote repositories

git repository

- local directory that is not under version control, and turn it into a git repository
- clone an existing Git repository from elsewhere

git repository init

- `git init`
- `git clone <URL> <DIR>`

git repository init

After **init** nothing in the project is tracked yet.
Need to begin tracking those files and do an initial commit.

specify the files you want to track

- `git add <FILE(S)>`
- `git commit -m "<MESSAGE>"`

Git repository with tracked files and an initial commit

git clone repository

Every version of every file for the history of the project is pulled down by default when you run `git clone`

git repository init

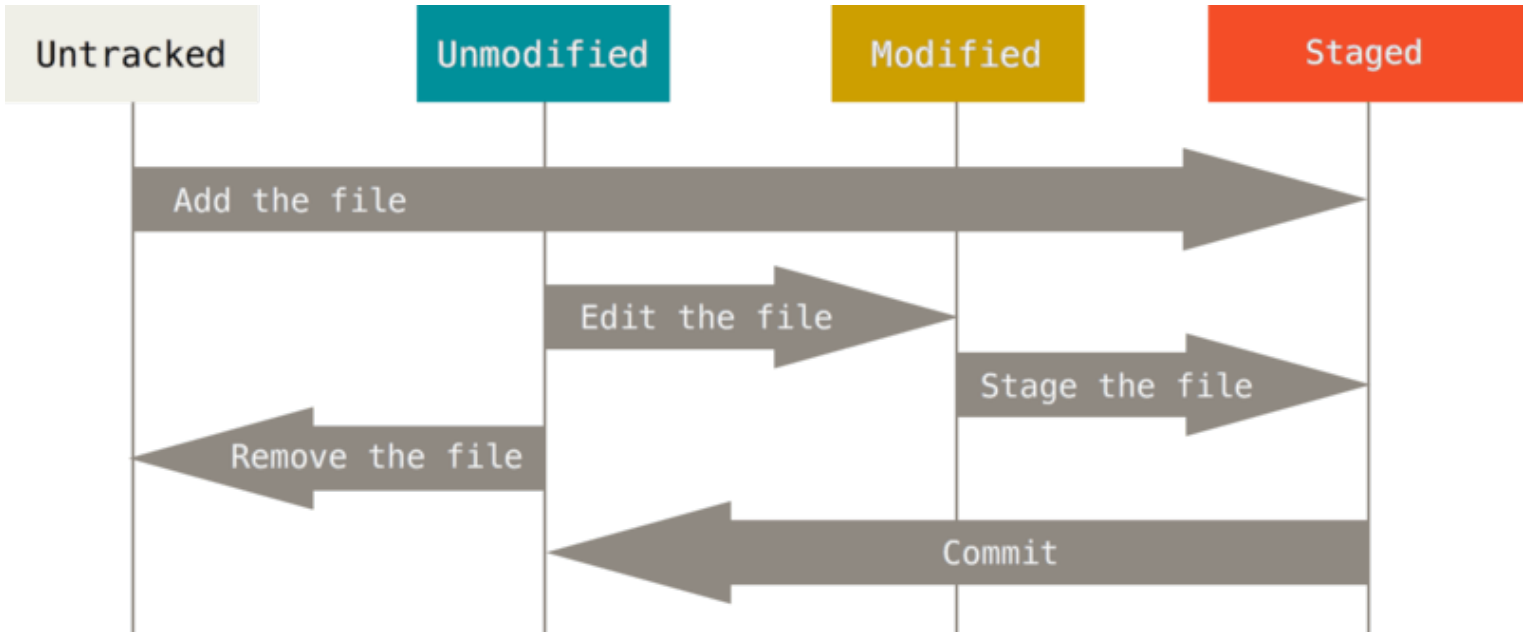
Each file in your working directory can be in one of two states

track files

- tracked
- untracked

Tracked files are files that were in the last snapshot; they can be **unmodified**, **modified**, *or* **staged**

Git and GitHub
track files



To determine which files are in which state: **the git status command**

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

        modified:   main-seminario-git.tex

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

        modified:   includes/git-cli.tex
        modified:   includes/intro.tex

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

        imgs/git-lifecycle-files.png
```

git add

In order to begin tracking a new file, you use the **command git add**

git add

file is now **tracked** and **staged** to be **committed**

The git add command **takes a path name** *for either a file or a directory*

git add

File that is tracked has been modified in the working directory but not yet staged

git add

To stage a modified tracked file, you have to run the **git add command** again.

After git add, the files are staged and will go into your next commit

git add

If you modify a file after you run git add, you have to run git add again to stage the latest version of the file

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

        modified:   main-seminario-git.tex

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

        modified:   includes/git-cli.tex
        modified:   includes/intro.tex
        modified:   main-seminario-git.tex

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

        imgs/git-lifecycle-files.png
        imgs/git-status.png
```

.gitignore file

If you'll have a class of files that you don't want to track

.gitignore file

you can create a file listing patterns to match them named **.gitignore**.

more .gitignore

```
main-seminario-git.aux
main-seminario-git.log
main-seminario-git.nav
main-seminario-git.out
main-seminario-git.pdf
main-seminario-git.snm
main-seminario-git.toc
*~
```

git diff

know exactly what you changed, not just which files were changed
by using the **git diff command**

git diff

- What have you changed but not yet staged (`git diff`)
- what have you staged that you are about to commit (`git diff --staged`)

Git and GitHub
viewing files


```
MacBookAir-Angelo:git-esercitazione angelo$ git diff
MacBookAir-Angelo:git-esercitazione angelo$ vim myEdition.xml
MacBookAir-Angelo:git-esercitazione angelo$ git diff
diff --git a/myEdition.xml b/myEdition.xml
index 74ba00e..089a9fa 100644
--- a/myEdition.xml
+++ b/myEdition.xml
@@ -1,5 +1,5 @@
 <?xml version="1" encoding="UTF-8"?>
-<!-- comment after tag -->
+<!-- comment after tag and diffing -->
 <TEI>
   <teiHeader>
     <fileDesc>
MacBookAir-Angelo:git-esercitazione angelo$ git diff --word-diff
diff --git a/myEdition.xml b/myEdition.xml
index 74ba00e..089a9fa 100644
--- a/myEdition.xml
+++ b/myEdition.xml
@@ -1,5 +1,5 @@
<?xml version="1" encoding="UTF-8"?>
<!-- comment after tag {+and diffing+} -->
<TEI>
  <teiHeader>
    <fileDesc>
MacBookAir-Angelo:git-esercitazione angelo$
```

git commit

Any files you have created or modified that you haven't run git add on since you edited them — won't go into the commit.

git commit

- the simplest way to commit is to type (`git commit`)
- type your commit message inline (`git commit -m "message"`)

git commit

Every time you perform a commit, you're recording a snapshot of your project that you can revert to or compare to later.

```
[master d0295cd] editing git-cli.tex
 7 files changed, 243 insertions(+), 5 deletions(-)
 create mode 100644 imgs/git-add-modify.png
 create mode 100644 imgs/git-lifecycle-files.png
 create mode 100644 imgs/git-status.png
 create mode 100644 imgs/gitignore.png
```

git rm

To remove a file from git, you have to remove it from your tracked files

git rm

- `git rm <FILE>`
- `git rm -f <FILE>`
- `git rm --cached <FILE>`

git mv

If you rename a file in Git, no metadata is stored in Git that tells it you renamed the file

git mv

- `git mv <FILE-FROM> <FILE-TO>`

git mv

- `mv <FILE-FROM> <FILE-TO>`
- `git rm <FILE-FROM>`
- `git add <FILE-TO>`

git log

git log lists the commits made in that repository in reverse chronological order, each commit with its checksum hash string, author's name and email, date, the commit message.

git log

- `git log <options>`

git log

if you want to see some abbreviated stats for each commit, you can use **the –stat option**

git log

- `git log --stat`

Git and GitHub

History of commits

```
commit d0295cd0fac89518d896ced1c110e7b788f1c95c (HEAD -> master)
Author: angelodel80 <angelodel80@gmail.com>
Date:   Thu Jun 13 16:53:20 2019 +0200

    editing git-cli.tex

 imgs/git-add-modify.png      | Bin 0 -> 35902 bytes
 imgs/git-lifecycle-files.png | Bin 0 -> 13727 bytes
 imgs/git-status.png          | Bin 0 -> 32484 bytes
 imgs/gitignore.png           | Bin 0 -> 4243 bytes
 includes/git-cli.tex         | 238 ++++++++++++++++++++++++++++++++++
 includes/intro.tex           |   6 ++--
 main-seminario-git.tex       |   4 +--
 7 files changed, 243 insertions(+), 5 deletions(-)

commit 4c07bb1cae889347bb8a1b73678bacc99484d903 (origin/master)
Author: angelodel80 <angelodel80@gmail.com>
Date:   Thu Jun 13 15:43:39 2019 +0200

    ending the intro.tex part

 imgs/git-areas.png      | Bin 0 -> 18502 bytes
 imgs/snapshots-git.png  | Bin 0 -> 20722 bytes
 includes/intro.tex      | 210 +++++++++++++++++++++++++++++-----------
 main-seminario-git.tex  |   4 ++-
 4 files changed, 44 insertions(+), 170 deletions(-)
```

git log options

Table 2. Common options to `git log`

Option	Description
`-p`	Show the patch introduced with each commit.
`--stat`	Show statistics for files modified in each commit.
`--shortstat`	Display only the changed/insertions/deletions line from the --stat command.
`--name-only`	Show the list of files modified after the commit information.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.
`--relative-date`	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.
`--pretty`	Show commits in an alternate format. Options include oneline, short, full, fuller, and format (where you specify your own format).
`--oneline`	Shorthand for `--pretty=oneline --abbrev-commit` used together.

Table 1. Useful options for `git log --pretty=format`

Option	Description of Output
%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author email
%ad	Author date (format respects the --date=option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

git log limit options

Table 3. Options to limit the output of `git log`

Option	Description
`-<n>`	Show only the last n commits
`--since`, `--after`	Limit the commits to those made after the specified date.
`--until`, `--before`	Limit the commits to those made before the specified date.
`--author`	Only show commits in which the author entry matches the specified string.
`--committer`	Only show commits in which the committer entry matches the specified string.
`--grep`	Only show commits with a commit message containing the string
`-S`	Only show commits adding or removing code matching the string

```
git log --pretty="%h:  %an -- %s" --no-merges
```

git log –pretty

```
d0295cd: angelodel80 -- editing git-cli.tex
4c07bb1: angelodel80 -- ending the intro.tex part
9d23569: angelodel80 -- editing intro.tex
725e96e: angelodel80 -- editing git-cli.tex
075509e: angelodel80 -- added some images
2291b3c: angelodel80 -- adding info on intro
0f1fac7: angelodel80 -- added README file
af65b1f: angelodel80 -- seminaio git dh repo init
```

amend option

If you commit too early and possibly forget to add some files, make the additional changes you forgot, stage them, and **commit again using the –amend option**.
You end up with a single commit — the *second commit replaces the first one*.

git log

- `git commit --amend [-m "MESSAGE"]`

unstage and discard changes

How can you unstage a file or revert it back to what it looked like when you last committed.

git reset and checkout

- `git reset HEAD <FILE>` (unstage file)
- `git checkout -- <FILE>` (discard changes)

Remote Repositories

Remote repositories are versions of your project that are hosted on the Internet

Remote Repositories

Collaborating with others involves managing remote repositories.
This entails **pushing** and **pulling** data to and from remote repositories when you need to share data.

capabilities

- add remote repositories
- remove remotes
- manage various remote branches
- define them as being tracked or not
- pushing, pulling and fetching operations

remote repositories

To see which remote servers you have configured, you can run the **git remote command**

git remote

- `git remote`
- `git remote -v`

remote repositories

To add a new remote Git repository as a shortname you can reference easily, run **git remote add ¡shortname¿ ¡url¿**:

git remote

- `git remote add upstream-edition https://github.com/angelodel80/myEditon`

If you clone a repository, the command automatically adds that remote repository under the name "origin"

remote repositories

to get data from your remote projects, you can run the **git fetch command**.
It's important to note that the git fetch command only downloads the data to your local repository — it doesn't automatically merge it with any of your work or modify what you're currently working on.

git remote

- `git fetch <remote>`

remote repositories

When you have your project at a point that you want to share, you have to **push it upstream**. This pushes any commits you've done back up to the server if you have write access and if nobody has pushed in the meantime.

git remote

- `git push <remote> <branch>`

git remote

- `git remote show <remote>`

remote repositories

```
* remote origin
  Fetch URL: https://github.com/angelodel80/seminarioGit.git
  Push  URL: https://github.com/angelodel80/seminarioGit.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local ref configured for 'git push':
    master pushes to master (local out of date)
```

remote repositories

You can run **git remote rename** to change a remote's shortname, if you want to remove a remote repository you can either use **git remote remove** command or **git remote rm** command.

git remote

- `git remote rename original upstream-edition`
- `git remote remove upstream-edition`

tag specific points

Git has the ability to **tag specific points** in a *repository's history* as being important, e.g. mark release points. Git supports two types of tags: lightweight and annotated.

git tag

- `git tag [-l] [--list] <PATTERN>` (list tags)
- `git tag -a <TAG-NAME> -m "MESSAGGIO"` (create an annotated tag)
- `git show <TAG-NAME>` (show the tag data)
- `git push <REMOTE> <TAG-NAME>` (push tag)
- `git tag -d <TAG-NAME>` (delate locally)
- `git push <REMOTE> --delete <TAG-NAME>` (delate remotelly)

tag specific points

If you want to view the versions of files a tag is pointing to, you can do a git checkout of that tag.
This puts your repository in "detached HEAD" state, which has some ill side effects

git tag

- `git checkout <TAG-NAME>` (View the files in tag version)

Progress status

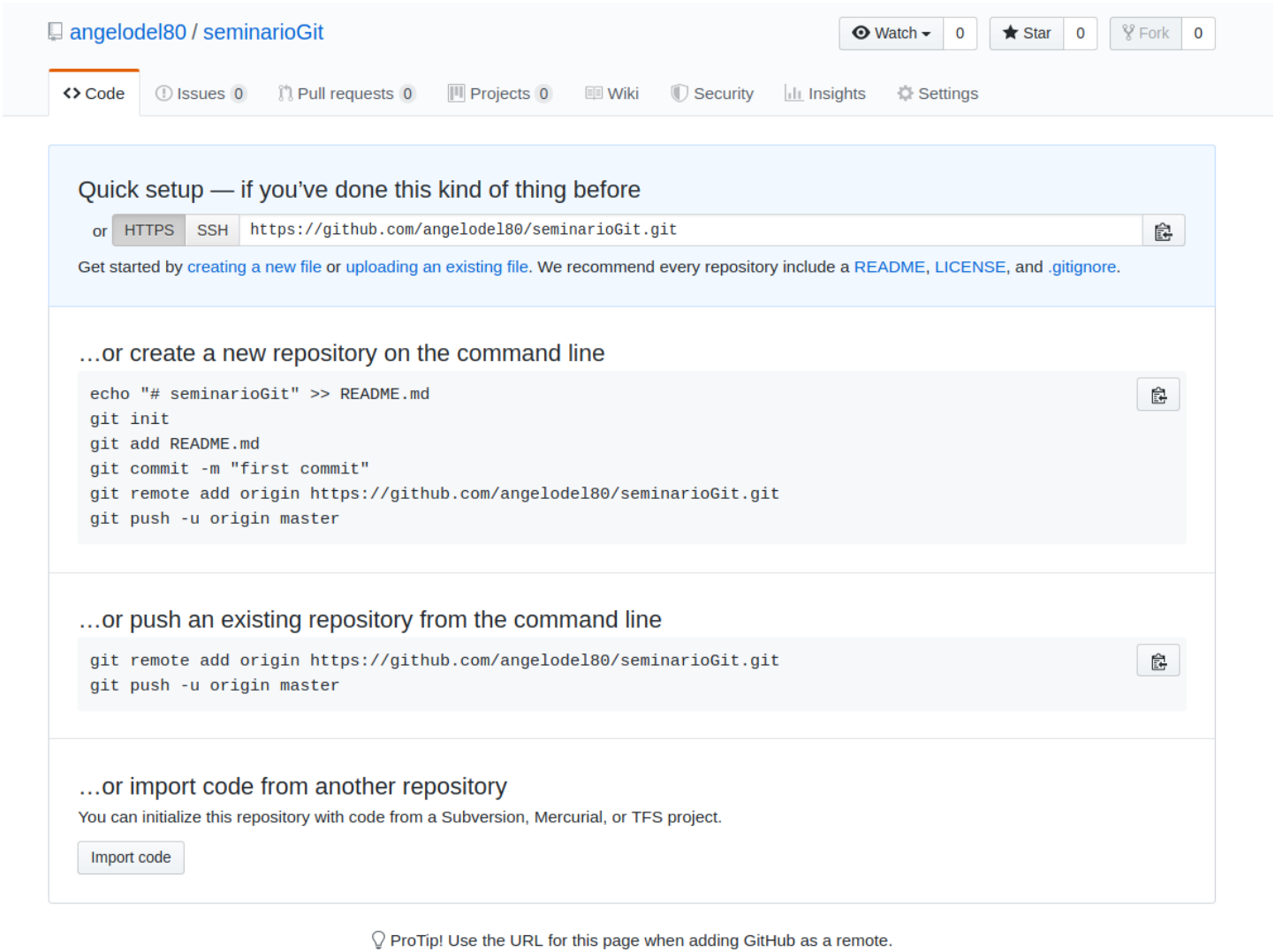
Github platform

GitHub is the largest host for git repositories. It is a central point of collaboration among developers.

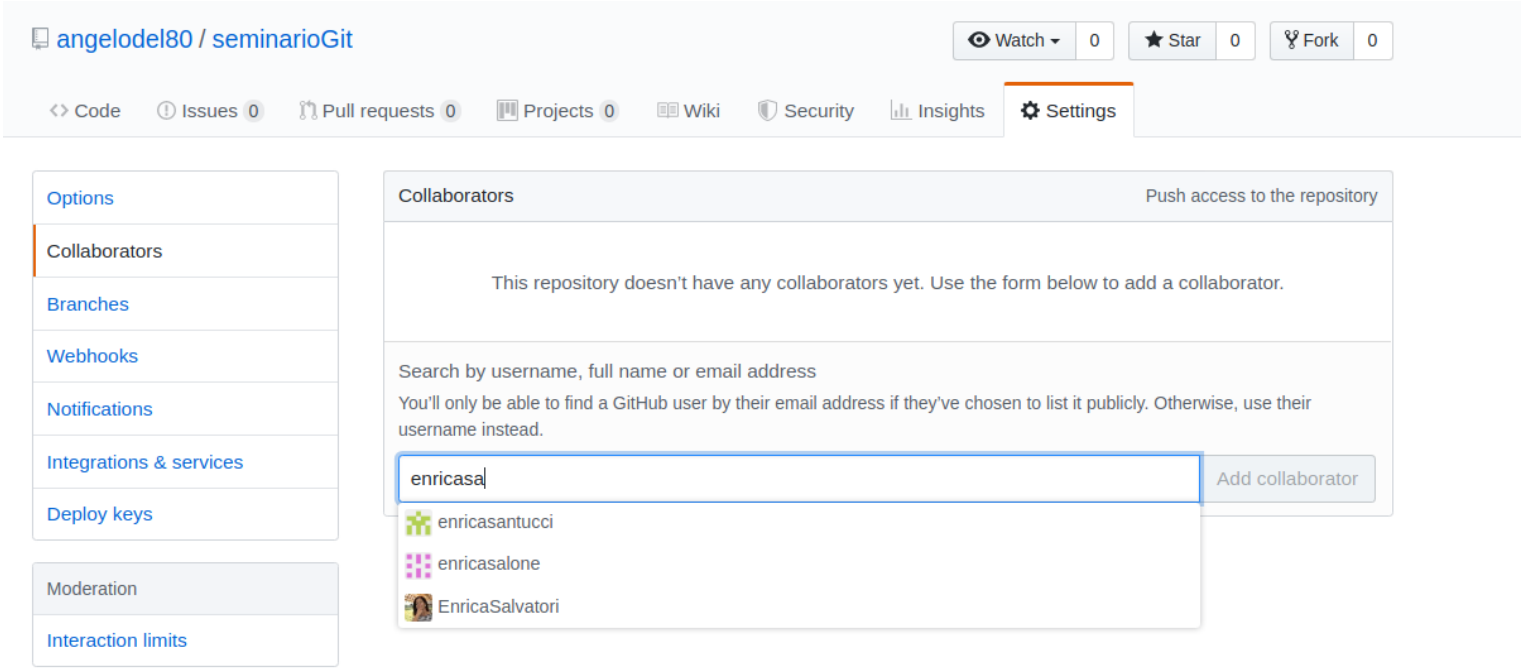
Github capabilities

Git hosting, issue tracking, code review, and other things

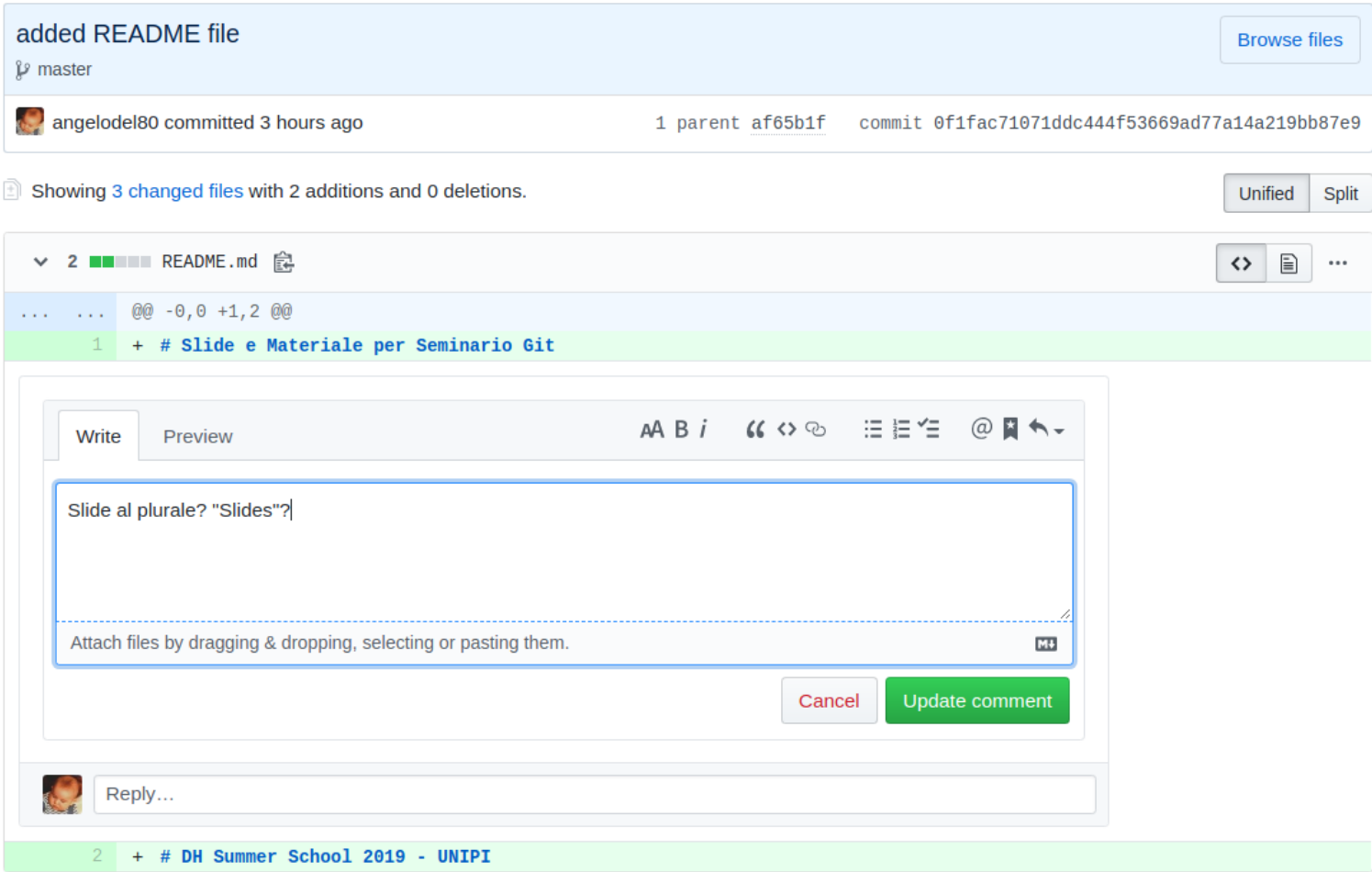
Github
Init a repository



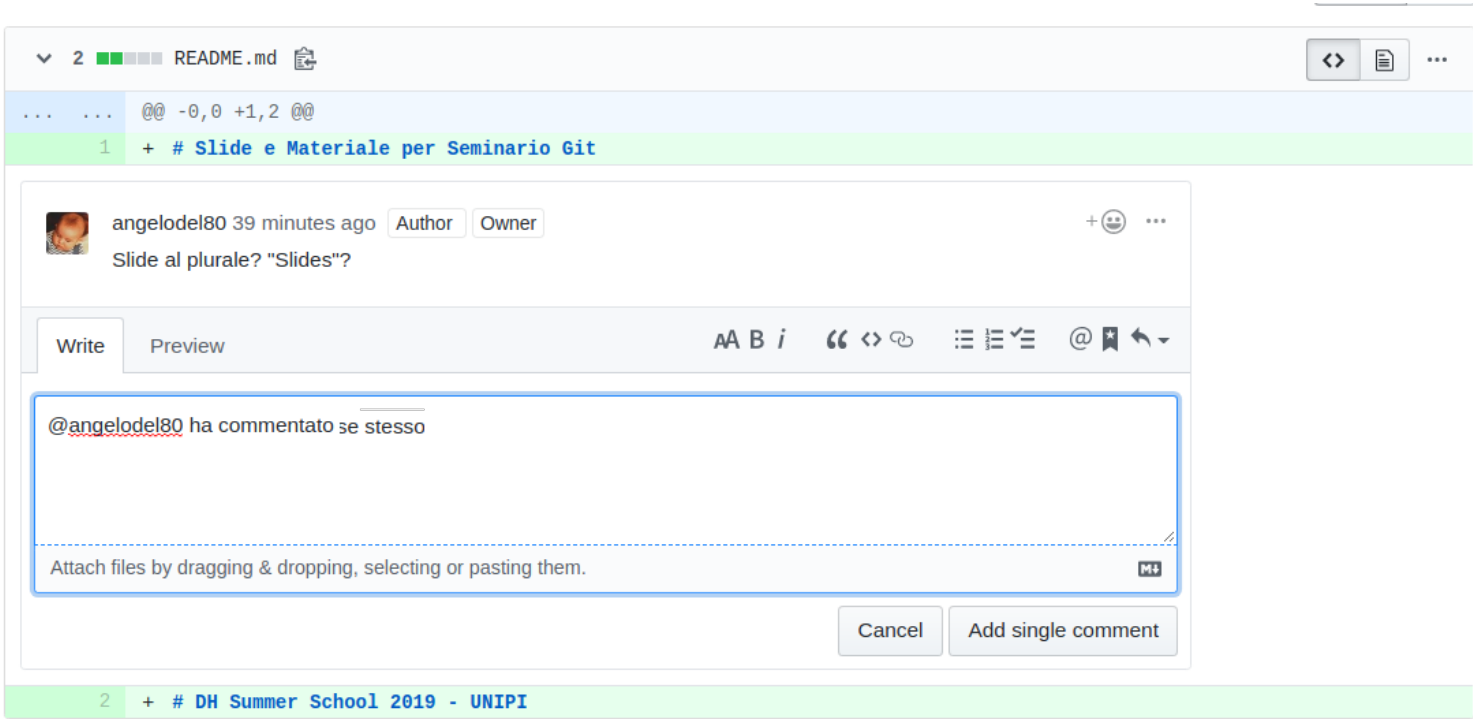
Github
adding collaborators



Github
Comments to content lines



Github
Comments to content lines



Progress status

GIT

- basic understanding of what VCS and git are
- working version of Git on your system
- basic configuration set up

GIT basic command line tools

- all the basic local Git operations
- creating or cloning a repository
- making, staging and committing changes
- viewing the history of the changes
- NO branching model

GIT remote tool

- get a remote git repository up and running
- collaborate with others or share your work
- contributing to a project
- maintaining your own project
- integrating other users' contributions

GITHUB

- gitHub user
- how to create an account
- manage an organization
- create and push to repositories
- contribute to other people's projects
- accept contributions from others

Progress status

References

 Chacon, S., e B. Straub. 2014. Pro Git. Apress.

References

Pro Git book, written by Scott Chacon and Ben Straub, 2nd Edition (2014).
`https://git-scm.com/book/it/v2`