



# Corso Git & GitHub – Lezione 1

Introduzione a git e github

**Elisabetta Ferri, Giorgio Micaglio, Gianmarco Puleo, Michele Tognoni**

Associazione Italiana Studenti di Fisica  
Comitato Locale di Trento

May 2, 2024

# Outline del corso

La principale fonte che abbiamo usato è l'ottimo libro Pro Git, 2nd edition di Scott Chacon e Ben Straub, disponibile gratuitamente in lingua inglese al seguente link:

<https://git-scm.com/book/en/v2>

- Lezione 1: cos'è git, installazione e primi passi.
- Lezione 2: utilizzo di git in locale
- Lezione 3: utilizzo di git in remoto tramite GitHub e collaborazione

# Overview

## 1. Che cos'è git?

- 1.1 Version Control Systems
- 1.2 Git in dettaglio

## 2. Prerequisiti e installazione

- 2.1 Utilizzo del terminale bash
- 2.2 Editor di testo da terminale

## 3. I primi passi con git

- 3.1 Creare un account
- 3.2 Configurare git

# Che cos'è git?

---

# Version Control Systems

- git è un sistema che tiene traccia di tutte le modifiche che vengono effettuate ad un insieme di files detto **repository**. Un tale sistema è detto version control system.
- Questo può avvenire sia in locale (sul vostro PC) che su un server remoto, sia per evitare la perdita di dati che per facilitare la collaborazione di più utenti a progetti.
- **“macchina del tempo”**: si può ritornare facilmente a qualsiasi versione precedente del progetto, senza perdere dati.
- git è usato come VCS in moltissimi ambiti, dalla ricerca in università all’industria del software.

# Esempi

Git è usato in collaborazioni scientifiche e aziende, che rendono disponibili anche diversi codici open-source:

- CERN: <https://github.com/CERN>
- LIGO-Virgo-KAGRA: <https://git.ligo.org/explore/projects/starred>
- Meta: <https://github.com/facebook>
- Python: <https://github.com/python>
- AISF: <https://github.com/ai-sf/>
- Dipartimento di Fisica, Università di Trento:  
<https://gitlab.physics.unitn.it/explore/projects>

Su GitHub si trova di tutto, e si possono anche proporre miglioramenti a progetti già esistenti.

# When everything began...

Linus Torvalds descrisse così il significato del nome git:

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.

# When everything began...

Linus Torvalds descrisse così il significato del nome git:

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.



# When everything began...

Linus Torvalds descrisse così il significato del nome git:

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.

# When everything began...

Linus Torvalds descrisse così il significato del nome git:

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- "goddamn idiotic truckload of sh\*t": when it breaks

# GitHub e GitLab

1. GitHub e GitLab sono due piattaforme che si servono di git e permettono di ospitare le proprie repository su server remoti.
2. Inoltre facilitano le interazioni tra utenti, permettono creazione di Wiki e libri.

## Nota bene

Useremo GitHub per configurare le repository remote nella lezione 3.



# GitHub



# GitLab

# Git in dettaglio

## Idea di base

È come fare delle fotografie a dei file in una cartella, con la possibilità di visualizzare tutte le fotografie del passato in ogni momento.

## Commit

“**commit**” è sia l’atto di “fotografare” dei file (verbo) che la foto stessa (sostantivo). Ogni commit è identificato da git con un numero in formato esadecimale, esempio:

3c552345a5613a94e5f4704a8311919071fc410d

# I possibili stati di un file

Un file in una repo può trovarsi in 4 diversi stati:

- **untracked:** è lo stato di partenza di tutti i nuovi file che create. Un file untracked viene completamente ignorato da git.
- **staged:** significa che il file è identificato da git come “pronto per essere fotografato” con un *commit*
- **committed:** significa che il file è stato “fotografato” (dunque è tracciato da git), e dopo l’ultimo commit non è stato modificato.
- **modified:** significa che il file è stato “fotografato” (dunque è tracciato da git), ma dopo l’ultimo commit è stato modificato.

# Git in dettaglio

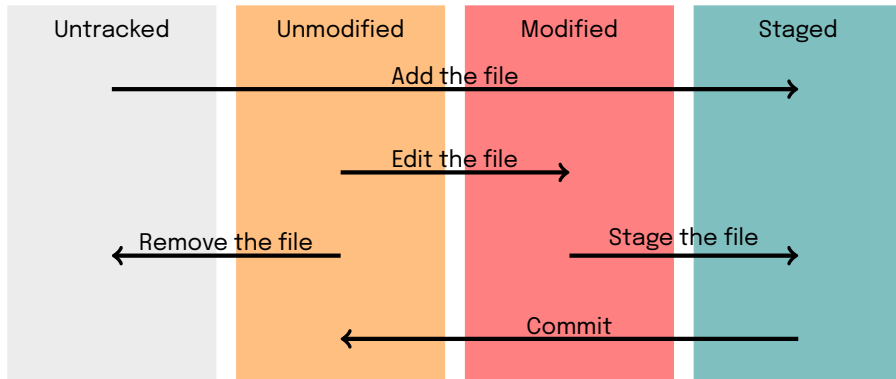


Figure: The lifecycle of the status of your files.

## Git in dettaglio (II)

Un progetto git è composto da tre elementi:

1. **working directory**: la versione del vostro lavoro su cui state lavorando, ovvero l'insieme di file visibili nella cartella in cui lavorate.
2. **.git directory**: è una cartella in cui git salva tutte le versioni del vostro progetto in modo molto intelligente. Contiene tutte le informazioni della vostra repository.
3. **staging area** o **index**: un file all'interno della .git directory, dove sta scritto cosa andrà nel vostro prossimo commit.

# Prerequisiti e installazione

---



# Prerequisiti

Useremo la shell UNIX:

- **Linux e Mac** sono basati su UNIX, non c'è bisogno di fare nulla.
- si prega chi ha **Windows** di installare **Windows Subsystem for Linux** seguendo questa semplice guida:  
<https://learn.microsoft.com/en-us/windows/wsl/install>

Installazione di git da terminale:

- Su Ubuntu/WSL:  
`sudo apt install git-all`
- Su Mac:  
`git --version`, se git non è presente vi dira di installarlo.

# Usare la shell UNIX

Comandi base per navigare nel vostro computer usando la shell:

- ▶ `pwd`: dice in che cartella siete
- ▶ `ls`: stampa una lista di file e cartelle presenti nella cartella in cui siete (di seguito WD).
- ▶ `mkdir xxx` crea una cartella chiamata `xxx`.
- ▶ `touch xxx` crea un file ASCII chiamato `xxx`
- ▶ `cd xxx`: entra nella cartella `xxx`; `cd ..` risale alla cartella genitore. `cd ~` entra nella vostra home directory.
- ▶ `mv source destination`: se `destination` è una cartella, sposta `source` all'interno di essa. Altrimenti rinomina `source` in `destination`
- ▶ `cp source destination`: come `mv`, ma copia anziché rinominare/spostare.
- ▶ `rm xxx`: elimina (irreversibilmente) il file `xxx`. Se è una cartella dovete specificare `-r`.

# Uso di wildcards

La shell permette di manipolare file e cartelle in modo molto efficiente usando le wildcards, ovvero dei caratteri che fungono da segnaposto per altri caratteri:

- \* rappresenta qualsiasi numero di caratteri ignoti. Es. \*.pdf viene espanso a tutti i nomi di file che terminano con .pdf presenti nella WD.
- ? rappresenta esattamente un carattere ignoto. Es. ?.pdf viene espanso a tutti i nomi di file composti da un singolo carattere seguito da .pdf.
- [xyz] rappresenta esattamente uno tra i caratteri inclusi nelle parentesi quadre.

# Parentesi graffe

- {1..10}: si espande in tutti i numeri da 1 a 10, in ordine.
- {10..1..2}, in ordine decrescente, a passi di 2
- Funziona anche con le lettere: {a..z..3}.

## Example

Supponiamo di voler creare 10 cartelle chiamate `simulazione_1`, `simulazione_2`, `simulazione_3`, .... Anziché usare il comando `mkdir` 10 volte, possiamo scrivere

```
mkdir simulazione_{1..10}
```

Un dettagliato manuale di BASH è disponibile qui:  
<https://www.gnu.org/software/bash/manual/>

# Esercizi



# Editor testo da terminale

## Domanda

Avete mai scritto su un file senza usare il mouse?

# Editor testo da terminale

## Domanda

Avete mai scritto su un file senza usare il mouse?

Per farlo, su tutti i terminali bash è presente l'editor `vim`. Sta per *Vi Innervosirà Moltissimo* (non è vero, significa *Vi IMproved*). Per usarlo, dimenticate tutte le volte che avete scritto su un computer:

- `vim prova.txt` apre un file testo chiamato `prova.txt`.
- Per iniziare a scrivere dovete entrare in INSERT MODE, premendo “i”. Qui potete scrivere normalmente.
- Per uscire da INSERT MODE, premere ESC.

# Ancora sugli editor testo

I seguenti comandi funzionano solo una volta usciti dalla insert mode:

- u annulla l'ultima modifica
- :w seguito da INVIO salva.
- :q seguito da INVIO esce.
- :wq seguito da INVIO salva ed esce.

Su vim, tutto funziona diversamente, potete trovare molti comandi qui:  
<https://vim.rtorr.com/>. Se non vi piace, potete usare un editor come gedit.

Installazione: `sudo apt install gedit`

aprire un file: `gedit prova.txt`.

impostarlo come editor predefinito in git:

```
git config --global core.editor "gedit --wait --new-window"
```



# Ancora sugli editor testo

I seguenti comandi funzionano solo una volta usciti dalla insert mode:

- u annulla l'ultima modifica
- :w seguito da INVIO salva.
- :q seguito da INVIO esce.
- :wq seguito da INVIO salva ed esce.

Su vim, tutto funziona diversamente, potete trovare molti comandi qui:  
<https://vim.rtorr.com/>. Se non vi piace, potete usare un editor come gedit.

Installazione: `sudo apt install gedit`

aprire un file: `gedit prova.txt`.

impostarlo come editor predefinito in git:

```
git config --global core.editor "gedit --wait --new-window"
```

**PS: non provate mai a salvare usando CTRL+S in VIM**

# I primi passi con git

---

# Creazione di un account

D'ora in avanti useremo GitHub.

1. Se non lo avete già, create un account: [www.github.com/signup](https://www.github.com/signup).
2. Usate il vostro indirizzo email personale. Se volete che resti completamente privato, fate click sulla vostra foto profilo in alto a destra → Settings → Email e in fondo alla pagina spuntate la voce “Keep my email addresses private”.  
Apparirà un indirizzo del tipo `12345678+username@users.noreply.github.com`.  
Usatelo nel seguito al posto del vostro indirizzo email.

L'account serve per caricare le proprie repository su un server. Lo vedremo nella lezione 3 in dettaglio.

# Configurazione di git

Ora ci concentriamo sulla configurazione locale. git dovrà “parlare” con il server GitHub, perciò dobbiamo dire a git chi siamo: digitare

```
git config --global user.name ilvostrousernamegithub  
git config --global user.email lavostraemail@esempio.com
```

Senza l'opzione `--global`, questi dati verranno settati solo per la repository su cui state lavorando.

# Configurazione di git

Ora ci concentriamo sulla configurazione locale. git dovrà “parlare” con il server GitHub, perciò dobbiamo dire a git chi siamo: digitare

```
git config --global user.name ilvostrousernamegithub  
git config --global user.email lavostraemail@esempio.com
```

Senza l'opzione `--global`, questi dati verranno settati solo per la repository su cui state lavorando.

## Problema

Così come l'username o l'email non bastano per entrare sul vostro account social, queste credenziali *non basteranno a caricare il vostro lavoro su Github*. Per accertarsi che siamo davvero noi GitHub usa le **chiavi ssh**.

# SSH in a nutshell

Immaginate la vostra repository come un diario dei segreti che dovete conservare in uno scaffale dove tutti possono mettere mano (internet). Per mantenere i vostri segreti dovete creare:

- Un lucchetto magico, che tutti potranno vedere, detto **chiave pubblica**.
- Una chiave magica che è l'unica a poter aprire quel lucchetto, e che soltanto voi conserverete. Essa è detta **chiave privata**.

# SSH in a nutshell

Immaginate la vostra repository come un diario dei segreti che dovete conservare in uno scaffale dove tutti possono mettere mano (internet). Per mantenere i vostri segreti dovete creare:

- Un lucchetto magico, che tutti potranno vedere, detto **chiave pubblica**.
- Una chiave magica che è l'unica a poter aprire quel lucchetto, e che soltanto voi conserverete. Essa è detta **chiave privata**.

Dobbiamo fare due cose:

1. Generare la coppia chiave pubblica/chiave privata
2. Caricare la chiave pubblica su GitHub

# (1) Generazione di una coppia di chiavi SSH

Per generare una coppia di chiavi ssh:

```
cd ~/.ssh    #importante  
ssh-keygen -t ed25519 -f nomedellachiave -C your@email.com
```

Quando richiesta una passphrase, lasciare vuoto premendo invio.

Una volta create, usando `ls` vedrete le due chiavi:

`nomedellachiave.pub` ← chiave pubblica

`nomedellachiave` ← chiave privata

**NOTA MOLTO BENE**

**NON CONDIVIDERE MAI CON NESSUNO LA PROPRIA CHIAVE PRIVATA**



## (2) Caricare la chiave pubblica su GitHub

1. Aprite la vostra chiave pubblica con un editor testo e copiate l'intero contenuto.
2. Entrate su GitHub, click in alto a destra→Settings→SSH and GPG keys→New SSH key. Incollate l'intero contenuto della chiave pubblica nel box Key. Inserite un titolo a piacere (es. laptop) e cliccate Add SSH key.

Ora siete pronti per usare github anche in remoto! Riprenderemo questi concetti nella terza lezione.

# Thank you for your attention

**Elisabetta Ferri, Giorgio Micaglio, Gianmarco Puleo, Michele Tognoni**

Associazione Italiana Studenti di Fisica  
Comitato Locale di Trento

May 2, 2024