

# VCS-GIT-LAB-1

## Installazione e basi

**MARCELLO MISSIROLI**  
**Tecnologia  
e Progettazione**  
per il mondo **digitale**  
e per il **web II**



 digital docet

# Prerequisiti

## COMPUTER DI LABORATORIO

La postazione deve avere Git installato per il sistema operativo utilizzato. Se non lo fosse, può essere richiesto l'intervento dell'amministratore.

Avere accesso all'account.

Conoscenze di base di GIT e VCS

## LAPTOP O PC PERSONALE

Sistema operativo Windows, Linux o Mac

Connessione a internet.

Accesso come amministratore

Conoscenze di base di GIT e VCS

**LA DURATA PREVISTA DI QUESTA ATTIVITÀ È DI 1-2 ORE DI 50 MINUTI.**

“

QUESTO È GIT. TIENE TRACCIA DEL  
LAVORO DEL PROGETTO  
ATTRAVERSO UN BELLISSIMO  
MODELLO TEORICO DI GRAFO  
DISTRIBUITO

FIGO! E COME SI USA?

NON NE HO IDEA. BASTA  
RICORDARSI QUESTI COMANDI  
E USARLI PER SINCRONIZZARSI.  
SE CI SONO ERRORI, SALVA IL  
LAVORO, CANCELLA IL PROGETTO  
E SCARICANE UNA NUOVA VERSIONE



”

1.

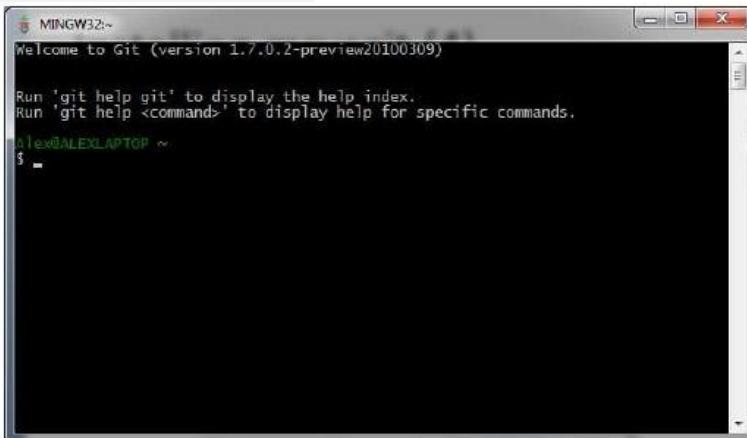
# Configurazione

Di che si tratta?

## PREREQUISITO:

Se non avete ancora installato git sulla vostra postazione, adesso è il momento di farlo.

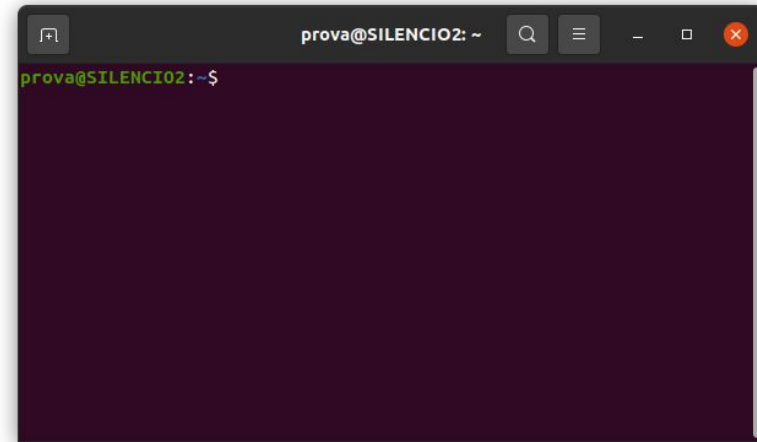
# Apriete un terminale (su Windows: git bash)



```
MINGW32-~
Welcome to Git (version 1.7.0.2-preview20100309)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

alex@ALEXLAPTOP ~
$
```



```
prova@SILENCIO2: ~
prova@SILENCIO2:~$
```

git --version

Una pura verifica che tutto sia OK.  
Dovrebbe restituire una cosa tipo:

```
$prova@pc:~$ git --version  
git version 2.25.1
```

## Identificazione (one-shot)

Indicate il vostro nome e la vostra email.

```
git config --global user.name "Alex York"  
git config --global user.email  
alex@example.com
```

Questo contrassegnerà i vostri lavori da questa postazione. Controllare con `git config --list`



## Default branch name

Cambiate il vostro nome di branch default

```
git config --global init.defaultBranch main
```

Questo semplifica l'interazione con GitHub e GitLab.

“

*Checkpoint #1:*

- ▶ *Sistema Git installato e configurato*

”

# 2.

## Primi passi

Il primo commit non si scorda mai

## Git init

Create una cartella di lavoro (es: demo), entrate, quindi inizializzate il repository con git init.

```
$prova@pc:~$ mkdir demo
$prova@pc:~$ cd demo
$prova@pc2:~/demo$ git init
Inizializzato repository Git vuoto in /home/prova/demo/.git/
$prova@pc2:~/demo$
```

## Git add

Create un file di testo (demo.txt), scrivete qualcosa e salvate. Dare git add . seguito da git status

```
$prova@pc2:~/demo$ git add .
$prova@pc2:~/demo$ git status
Sul branch main

Non ci sono ancora commit

Modifiche di cui verrà eseguito il commit:
  (usa "git rm --cached <file>..." per rimuovere gli elementi
dall'area di staging)
    nuovo file:          demo.txt

$prova@pc2:~/demo$
```

## Git commit

### Committere il lavoro (ricordarsi del -m)

```
$prova@pc2:git commit -m "Il mio primo commit!"  
[master (commit radice) d44487d] Il mio primo commit!  
1 file changed, 2 insertions(+)  
create mode 100644 demo.txt  
$prova@pc2:~/demo$
```

## Controllo finale con git status

```
$prova@pc2:git status  
Sul branch main  
non c'è nulla di cui eseguire il commit, l'albero di lavoro è  
pulito  
$prova@pc2:~/demo$
```

“

## *Checkpoint #2:*

- ▶ *Sapete inizializzare un repo*
- ▶ *Sapete fare un commit*
- ▶ *Sapete controllare la situazione*

”



# 3.

## Rollback

Uno sguardo al passato

## Nuovo commit

Modificate il file demo.txt, salvare e committare con il comando veloce `git commit -am`

```
$prova@pc:git commit -am "Modifiche"  
[main e2e01e8] Modifiche  
1 file changed, 1 insertion(+)  
$prova@pc:~/demo$
```

# Git log

Questo comando permette di vedere la storia delle modifiche

Autore e data

Identificativo  
esadecimale

```
$prova@pc:git log
commit e2e01e84e8594f4b7d894389ea3733267d9f611a (HEAD -> main)
Author: Alex York <alex@example.com>
Date: Thu Sep 2 14:29:21 2021 +0200
```

Modifiche

```
commit d44487d93541dc19d6d77f69123bb94452bd597d
Author: Alex York <alex@example.com>
Date: Thu Sep 2 14:18:00 2021 +0200
```

Il mio primo commit!

```
$prova@pc2:~/demo$
```

Posizione head e  
branch

Commento

## Tornare indietro

Git checkout main.

```
$prova@pc:git reflog  
e2e01e8 (HEAD -> main) HEAD@{0}: commit: Modifiche  
d44487d HEAD@{1}: commit (initial): Il mio primo commit!  
$prova@pc:~/demo$
```

## Ripristiniamo la versione precedente

Il comando è `git checkout`, seguito da un identificativo che può essere esadecimale o relativo, come si vede in `reflog`.

Il risultato è un messaggio minaccioso, ma la modifica è stata fatta (controllare il file)

## Ripristiniamo la versione precedente

```
$prova@pc:git checkout d44487d
```

Nota: eseguo il checkout di 'd44487d'.

Sei nello stato 'HEAD scollegato'. Puoi dare un'occhiata, apportare modifiche sperimentali ed eseguirne il commit, e puoi scartare qualunque commit eseguito in questo stato senza che ciò abbia alcuna influenza sugli altri branch tornando su un branch.

Se vuoi creare un nuovo branch per mantenere i commit creati, puoi farlo (ora o in seguito) usando l'opzione -c con il comando switch. Ad esempio:

```
git switch -c <nome nuovo branch>
```

Oppure puoi annullare quest'operazione con:

```
git switch -
```

Disattiva questo consiglio impostando la variabile di configurazione `advice.detachedHead` a `false`

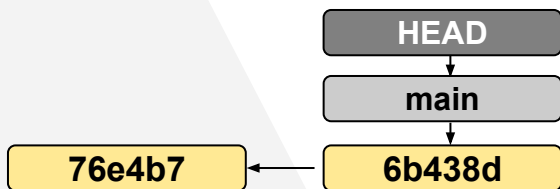
HEAD si trova ora a d44487d Il mio primo commit!

```
$prova@pc:~/demo$
```

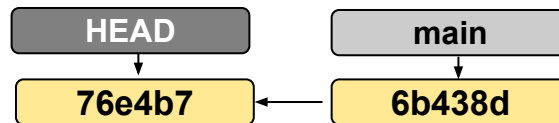
# Detached head



**Prima di checkout 76e4b7**



**Dopo checkout 76e4b7**



*git checkout main riporta HEAD all'ultimo commit. **NON** committate in stato di detached HEAD, rischiate di perdere il lavoro*

## Fate da soli

Create un nuovo file (altrodemo.txt) e aggiungetelo al repo. Che comandi dovete dare?

```
$ git add altrodemo.txt
```

```
$ git commit -m "secondo file"
```



## Ripristiniamo un solo file dal passato

`git checkout` riporta TUTTI i file alla loro condizione precedente, e in più “sgancia la testa”. Se dovete recuperare un solo file potete usare

```
git restore -s [checksum] [nomefile]
```

(Nelle vecchie versioni di git si usava `git checkout [checksum] -- nomefile`)

*(esiste anche il più moderno `git switch` ma lo vedremo più avanti)*

## Ripristiniamo la versione precedente

```
$prova@pc: git restore -s de45455 demo.tx
git status
Sul branch main
Modifiche non nell'area di staging per il commit:
  (usa "git add <file>..." per aggiornare gli elementi di cui sarà eseguito il
commit)
  (usa "git restore <file>..." per scartare le modifiche nella directory di lavoro)
    modificato:          demo.txt

nessuna modifica aggiunta al commit (usa "git add" e/o "git commit -a")

$prova@pc:~/demo$
```

## Variante: revert

`git revert <commit id>` ricostruisce un commit precedente creando un nuovo commit. Questo è tendenzialmente più sicuro in quanto non distrugge la storia delle modifiche. Sul lato negativo, potrebbe appesantire inutilmente il log.

## Pentimenti

Fate un `git add .` e aggiungerete `demo.txt` alla staging area. Se vi pentite, potete dare il comando `git reset` che di fatto è l'undo di `git add`

## Forti pentimenti

Modificate il file `demo.txt` e committate.

Ora modificate il file `altrodemo.txt` e aggiungete “robaccia”, tipo “`khjsèaspifuhwòkfldjzcoaigdàja`”. Quindi salvate.

È sempre possibile cancellare le modifiche non committate e ripristinare l’ultima versione salvata con `git reset --hard`.

Fatelo.

“

### *Checkpoint #3:*

- ▶ *Sapete committare “fast”*
- ▶ *Sapete ripristinare un commit o un file*
- ▶ *Sapete “unstagiare” un file*

”

# 3.

## Ignorare file

Alcune cose è meglio tenerle per sè

## No binaries!

Git si basa sulle differenze di testo con le versioni precedenti.

Per questo motivo non funziona bene con i file binari come .pdf, .png, .docx, .exe, .class: ogni versione viene salvata integralmente, con perdita di spazio e di tempo.



## .gitignore

Per difendersi dai commit involontari di file binari, è possibile utilizzare il file .gitignore (notare il punto). E' un elenco di file e/o percorsi che NON SONO MAI inseriti nella staging area.

## Proviamo

Nel progetto create un file di testo chiamato `.gitignore` e copiate questo →

```
# ignora i file eseguibili e png  
*.exe  
*.png
```

```
# ignora tutti i file nelle  
cartelle indicate temp  
temp/  
img/
```

## Proviamo

Create un file di testo  
nella cartella temp.  
Scaricate un file .png  
qualsiasi dalla rete e  
mettetelo nella cartella.

```
# ignora i file eseguibili e png
```

```
*.exe
```

```
*.png
```

```
# ignora tutti i file nelle  
cartelle indicate temp
```

```
temp/
```

```
img/
```

## Proviamo

Date `git add .` seguito da `git status`. I file “galeotti” non saranno aggiunti - il file `.gitignore` invece sì.

```
$prova@pc:~/demo$ git add .
$prova@pc:~/demo$ git status
Sul branch main
Modifiche di cui verrà eseguito il commit:
  (usa "git restore --staged <file>..." per rimuovere gli elementi dall'area di
  staging)
    nuovo file:          .gitignore

$prova@pc:~/demo$
```

## Forziamo la mano

Lo switch -f (o --force) elimina il controllo di .gitignore.

```
$prova@pc:~/demo$ git add temp/antipatico.txt -f
$prova@pc:~/demo$ git status
Sul branch main
Modifiche di cui verrà eseguito il commit:
  (usa "git restore --staged <file>..." per rimuovere gli elementi dall'area di
  staging)
    nuovo file:          .gitignore
    nuovo file:          temp/antipatico.txt

$prova@pc:~/demo$
```

“

## *Checkpoint 4:*

- ▶ *Sapete ignorare file indesiderati*
- ▶ *Sapete “forzare la mano” se necessario*

”

# Bignamino quotidiano (Cheat sheet)

```
git config --global user.name "[firstname lastname]"
```

set a name that is identifiable for credit when review version history

```
git config --global user.email "[valid-email]"
```

set an email address that will be associated with each history marker

```
git init
```

initialize an existing directory as a Git repository

```
git status
```

show modified files in working directory, staged for your next commit

```
git add [file]
```

add a file as it looks now to your next commit (stage)

```
git reset [file]
```

unstage a file while retaining the changes in working directory

```
git commit -m "[descriptive message]"
```

commit your staged content as a new commit snapshot

```
git checkout
```

switch to another branch and check it out into your working directory

```
git log
```

show all commits in the current branch's history

```
logs/  
*.notes  
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

```
git reset --hard [commit]
```

clear staging area, rewrite working tree from specified commit

## Best Git Cheat Sheet

- ▶ [Marvel](#) - Sintetico
- ▶ [Gitlab cheat sheet \(lungo\)](#) - PDF:
- ▶ [Atlassian Cheatsheet](#)
- ▶ Git-tower
- ▶ git-cheatsheet-visual (Github Project)
- ▶ Zack Rusin: Git cheat sheet
- ▶ Git PDF Rogerdudler



## Test (A1 level)

- ▶ Creare un nuovo progetto git (“nuovo”)
- ▶ Creare un nuovo file (“pippo.txt”) con una riga di testo
- ▶ Committare
- ▶ Aggiungere una nuova riga di testo e committare
- ▶ Aggiungere una nuova riga di testo e committare.
- ▶ Guardare il log (provate aggiungendo - - oneline)
- ▶ Fare in modo che i file con estensione .md NON vengano tracciati
- ▶ Creare il file “pluto.md”, dare “git add .” e commitare.
- ▶ Rollback al secondo commit (il file md deve esserci ancora!)
- ▶ Tornare all’HEAD.
- ▶ Committare il file “pluto.md”

# GRAZIE!

**Torniamo alla teoria!**

**DA QUESTO LINK POTETE SCARICARE UN FILE  
BASH CHE ESEGUE TUTTI I COMANDI VISTI**

**<https://pastebin.com/tJiKzjgN>**

## Credits

Special thanks to all the people who made and released these awesome resources for free:

- ▶ Presentation template by [SlidesCarnival](#)
- ▶ Photographs by [Startupstockphotos](#)
- ▶ Anil Gupta ([www.guptaanil.com](http://www.guptaanil.com))
- ▶ Pete Nicholls ([github.com/Aupajo](https://github.com/Aupajo))
- ▶ Armando Fox

***Questo documento è distribuito con licenza CreativeCommon BY-SA 3.0***

# Presentation design

This presentation uses the following typographies and colors:

- ▶ Titles: **Dosis**
- ▶ Body copy: **Roboto**

You can download the fonts on these pages:

<https://www.fontsquirrel.com/fonts/dosis>

<https://material.google.com/resources/roboto-noto-fonts.html>

- ▶ Orange **#ff8700**

You don't need to keep this slide in your presentation. It's only here to serve you as a design guide if you need to create new slides or download the fonts to edit the presentation in PowerPoint®

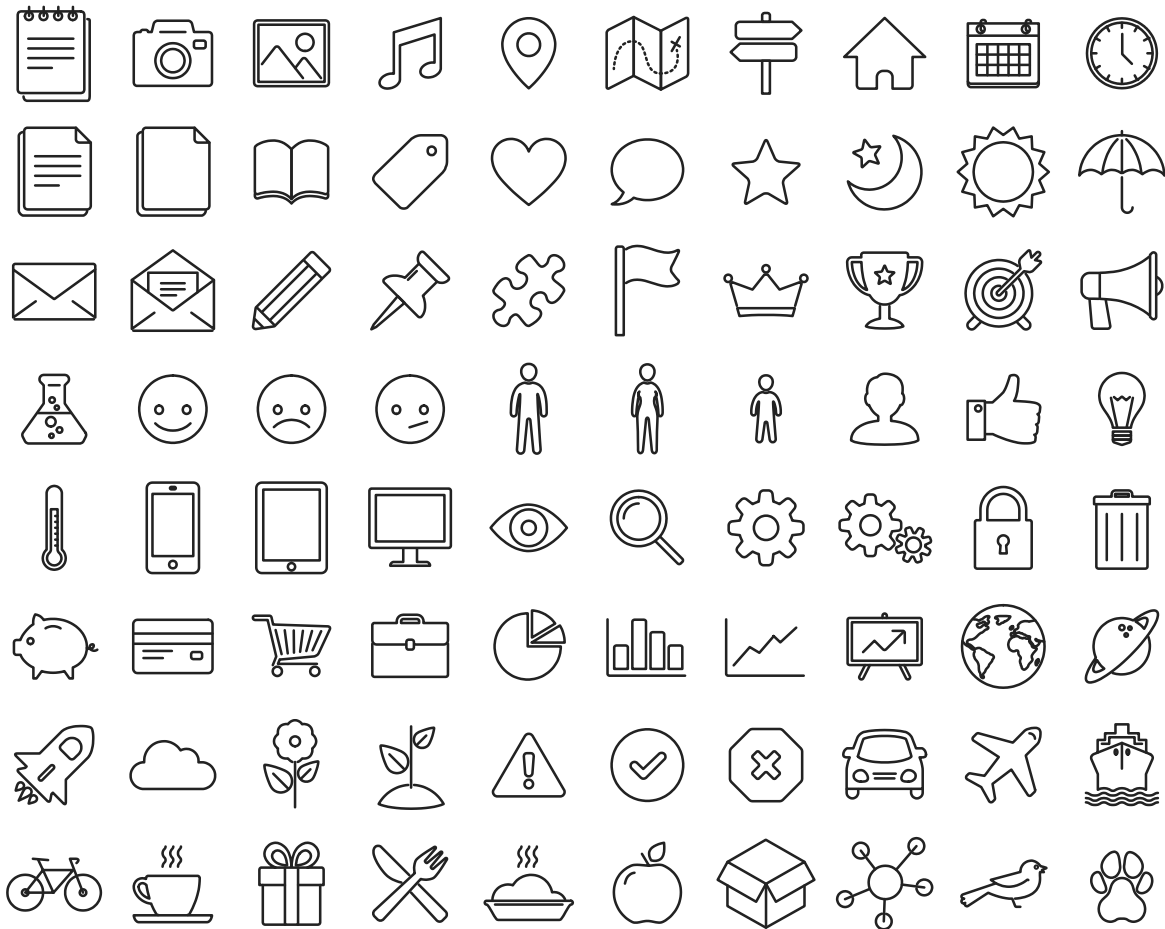
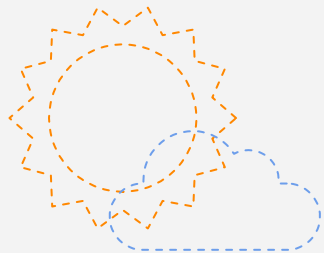
## SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change line color, width and style.

Isn't that nice? :)

Examples:



**Now you can use any emoji as an icon!**

And of course it resizes without losing quality and you can change the color.

How? Follow Google instructions

<https://twitter.com/googledocs/status/730087240156643328>



more...