Git
sotto il cofano

Fondamentale per il versionamento del codice

Salvatore La Spata (salvatore.la.spata@gmail.com)

Agenda

1. Basi solide
2. Il modello di storage di GIT
    a. Come ragiona GIT
    b. Cosa c'è dietro GIT
    c. Cosa avviene dietro le quinte di una COMMIT

Cos'è **GIT**

Definizione

`Git` è un **sistema di controllo versione** utilizzato per gestire le modifiche apportate ai file in un progetto software.

Con Git, è possibile `tenere traccia delle modifiche` apportate ai file nel tempo, in modo che si possa facilmente tornare indietro a una versione precedente di un file se necessario.

Inoltre, Git consente a più persone di `lavorare sullo stesso progetto contemporaneamente`, facilitando la collaborazione su progetti di grandi dimensioni

Caratteristiche

- **Vedere tutte le modifiche apportate al progetto**, quando sono state apportate e chi le ha fatte.
- Ogni modifica può essere accompagnata da un **messaggio che ne spiega il motivo**.
- **Recuperare le versioni precedenti** dell'intero progetto o di singoli file.
- **Creare rami (branch)**, dove è possibile apportare modifiche in via sperimentale. Questa funzione consente di **lavorare contemporaneamente su diverse serie di modifiche** (ad esempio, funzionalità o correzioni di bug), eventualmente da parte di persone diverse, senza influenzare il ramo principale. In seguito, è possibile unire le modifiche che si desidera mantenere al ramo principale.
- **Assegnare un tag** a una versione, ad esempio per contrassegnare una nuova release.

Attori di GIT

HEAD 

LOCAL 

REMOTE/ORIGIN 

Componenti di GIT

BRANCH 

TAG 

Come si installa

Per installare Git, è necessario seguire i passaggi specifici per il proprio sistema operativo. Di seguito sono riportati i passaggi generali per l'installazione su alcuni sistemi operativi comuni:

`Windows`	`MacOS`	`Linux`
https://git-scm.com/download/win	Git viene già installato di default su macOS, quindi non è necessario fare nulla. Se non avete Git installato,	Sulla maggior parte delle distribuzioni Linux, è possibile installare Git utilizzando il gestore di pacchetti del sistema.

Una volta installato Git, potete verificare che l'installazione sia andata a buon fine digitando `git --version` nel terminale.

Comandi principali

1

git init	Inizializzare un progetto git (crea la cartella .git nascosta)
git clone	scarica una copia di un repository Git in locale.
git add	aggiunge uno o più file al prossimo commit.
git commit	crea una nuova commit (snapshot dei file modificati)
git push	invia le commit locali al repository remoto.
git pull	scarica le commit più recenti dal repository remoto e le integra con la propria copia locale del repository.

Altri comandi

2

git checkout	Cambiare branch
git status	Stato attuale del repository
git merge	Unire due o più storie di sviluppo
git rebase	Riscrive la storia di un branch
git stash	Salva le modifiche che non sono state ancora commitate in un'area temporanea in modo da poter passare a un altro branch.
git cherry-pick	Applica le modifiche di un commit specifico a un branch diverso da quello in cui sono state originariamente fatte.

.gitignore

Il file `.gitignore` è un file speciale utilizzato in un repository Git per specificare i **file e le cartelle** che devono essere **ignorati dal controllo delle versioni**.

Quando si esegue un'operazione come il commit delle modifiche in Git, il contenuto del file .gitignore viene preso in considerazione per determinare **quali file devono essere esclusi**.

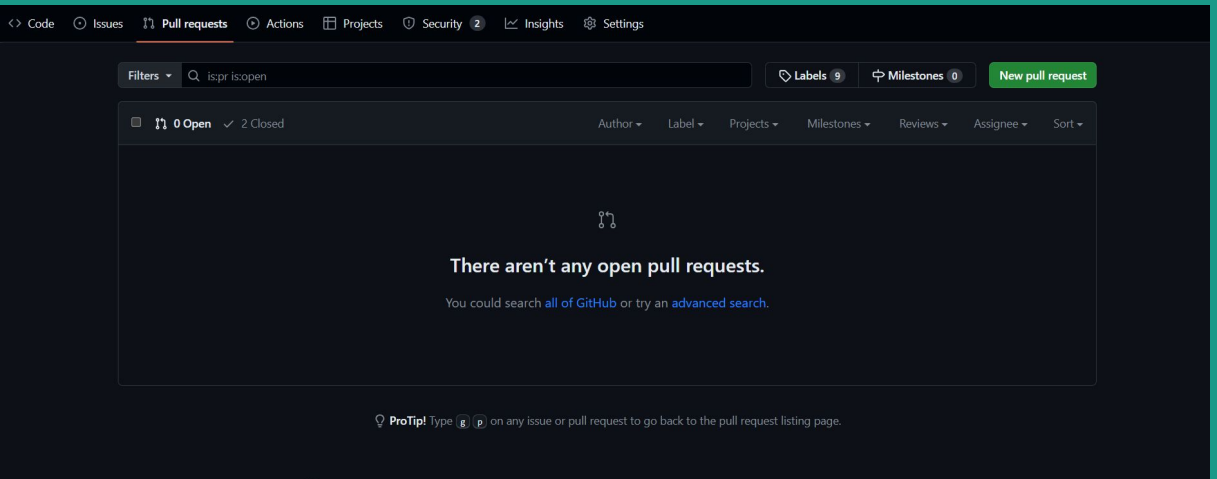
Ma...
GIT e **GITHUB** sono la stessa cosa?!

La differenza è un pò come il porno e pornhub😎. **GITHUB è un hub** che permette di centralizzare i sorgenti software che utilizzano GIT come sistema di versionamento del codice.

#GIT #GITHUB

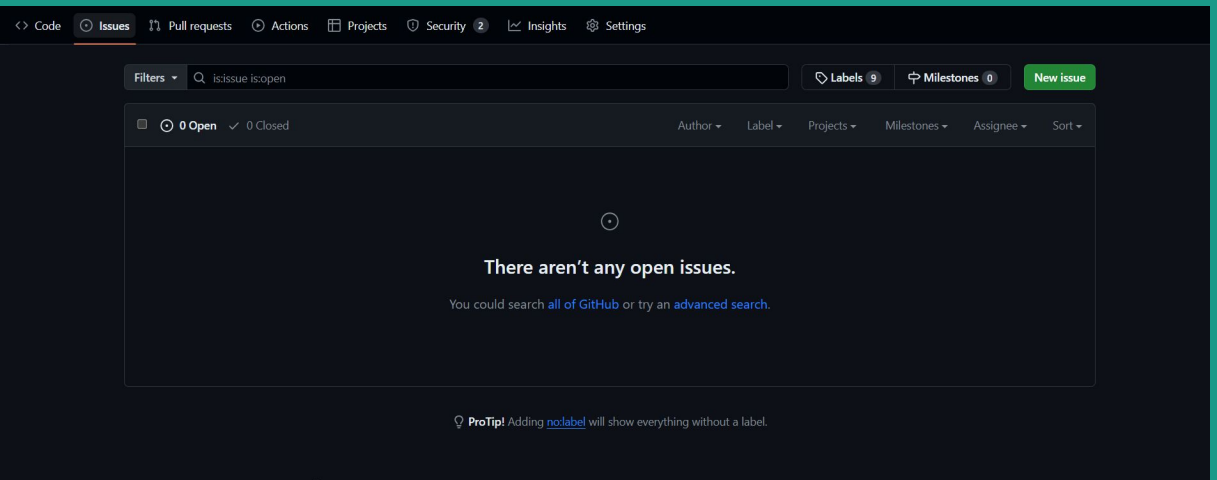
Github - **Pull Request**

Le pull request consentono di comunicare ad altri le modifiche apportate a un ramo di un repository su GitHub. Possono essere soggette ad approvazione.



Github - **Issue**

Utilizzate GitHub Issues per tenere traccia di idee, feedback, compiti o bug per il lavoro su GitHub.



Github - **Actions**

Automatizzate, personalizzate ed eseguite workflow di sviluppo software direttamente nel vostro repository con GitHub Actions.

.github/workflows/lean.yml

```yaml
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

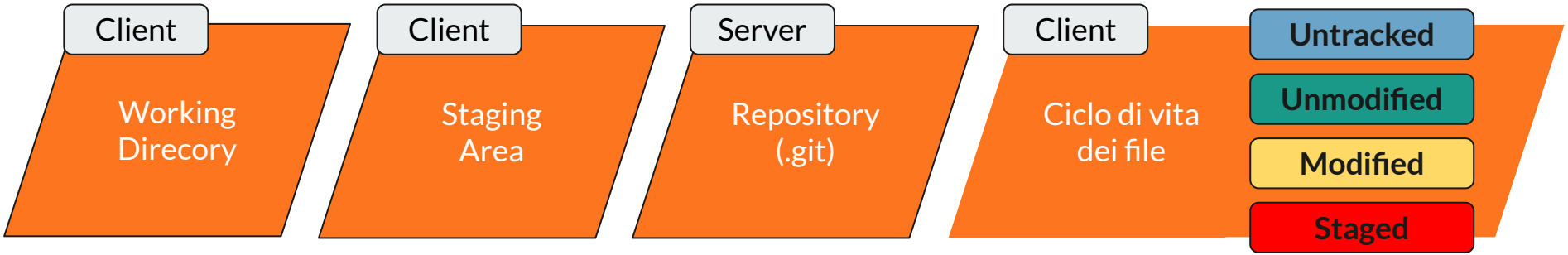
Iniziamo...
i primi passi con **GIT**

La nostra prima commit

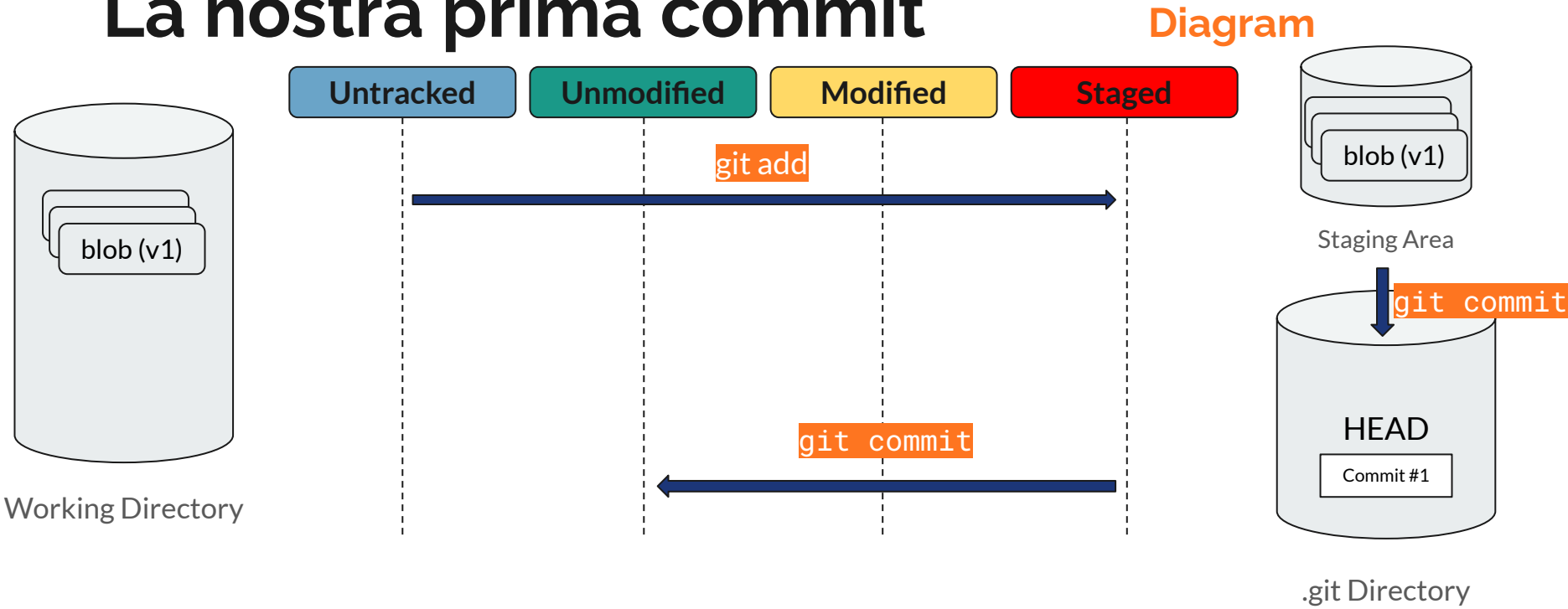
```
> git init

> git add <nome_file>

> git commit -m “<messaggio_di_commit”
```

La nostra prima commit

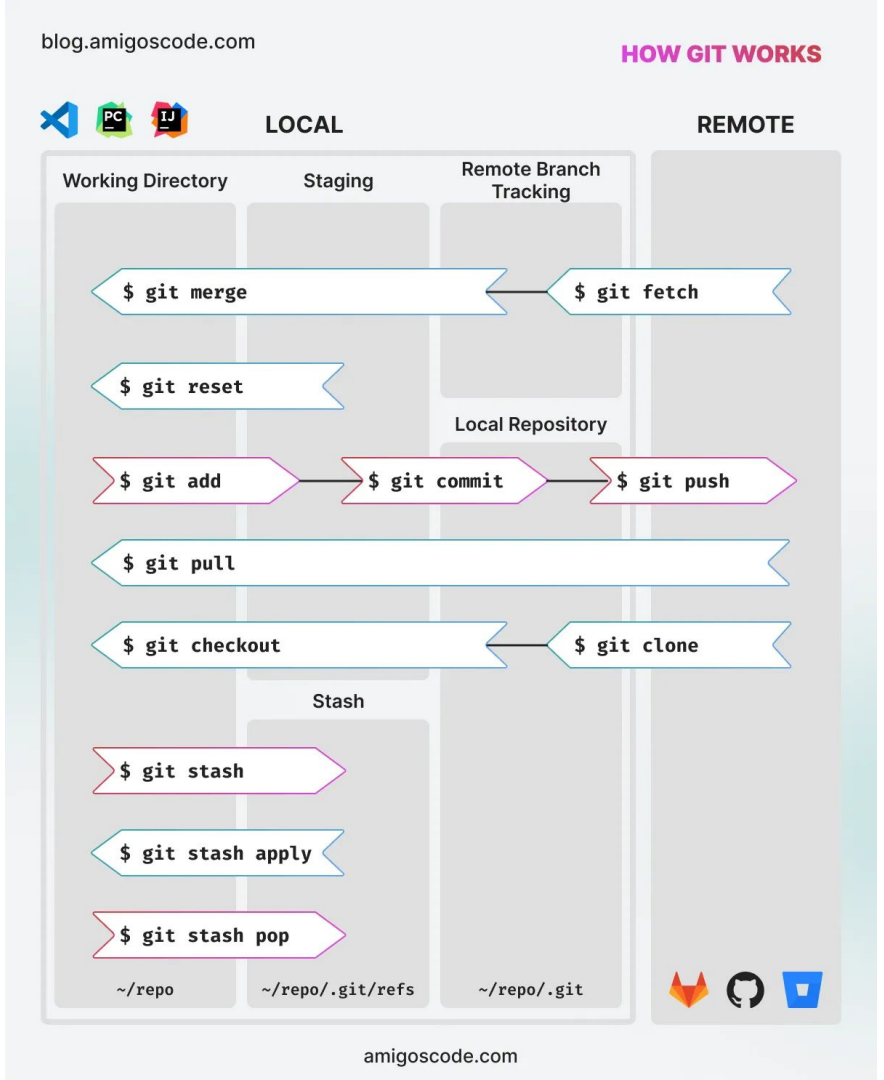
Workflow



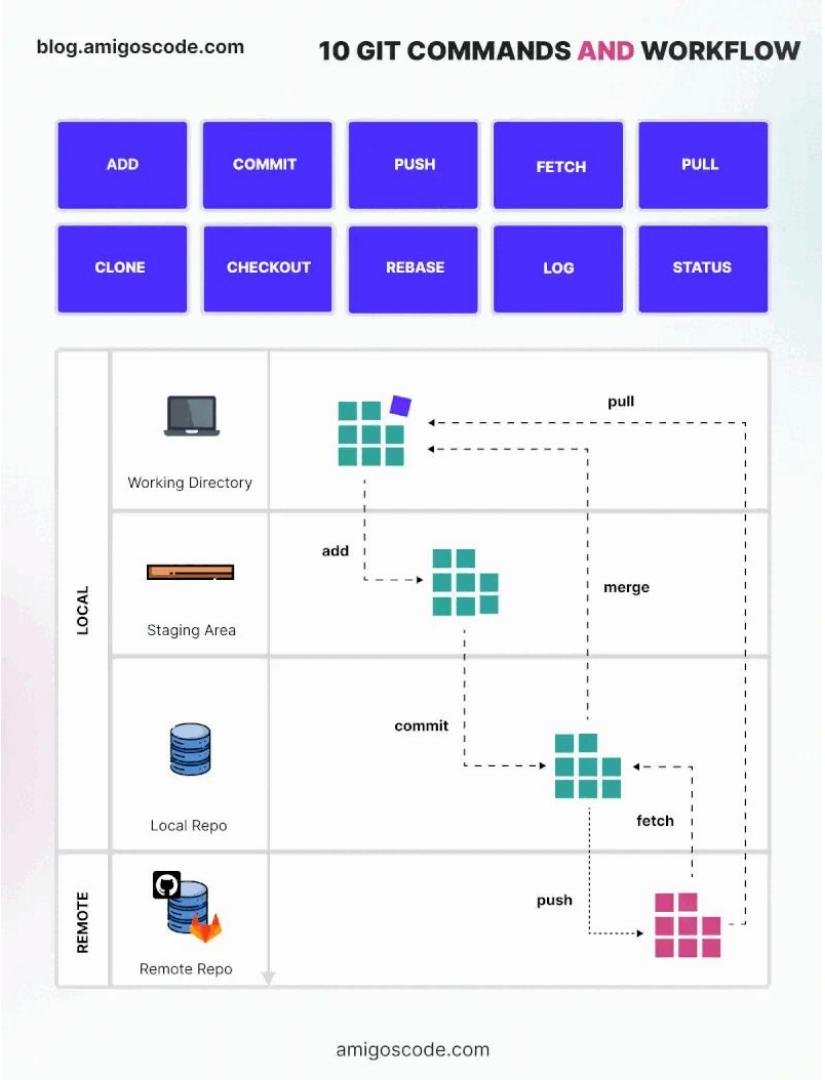
La nostra prima commit



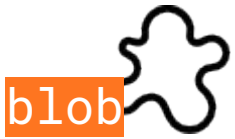
Workflow


blog.amigoscode.com
HOW GIT WORKS
LOCAL
REMOTE
Working Directory
Staging
Remote Branch Tracking
$ git merge
$ git fetch
$ git reset
Local Repository
$ git add
$ git commit
$ git push
$ git pull
$ git checkout
$ git clone
Stash
$ git stash
$ git stash apply
$ git stash pop
~/repo
~/repo/.git/refs
~/repo/.git
amigoscode.com

Workflow


blog.amigoscode.com
10 GIT COMMANDS AND WORKFLOW
ADD
COMMIT
PUSH
FETCH
PULL
CLONE
CHECKOUT
REBASE
LOG
STATUS
LOCAL
REMOTE
Working Directory
Staging Area
Local Repo
Remote Repo
pull
add
merge
commit
fetch
push
amigoscode.com

Il modello di storage



Rappresenta un file versionato (compresso con zlib) nel nostro progetto

obj:blob | 2c6a
annotazione

Un albero che contiene gli oggetti blob e altri alberi. Una sorta di filesystem.

obj:tree | 8e2a
blob 2c6a filename

E' un riferimento all'albero, i relativi file, l'autore della modifica, la data e l'annotazione.

obj:commit | 1a15
tree 8e2a
author timestamp
committer timestamp
annotazione

Puntatore ad una commit o ad un altro puntatore (commit/branch/tag)

ref | HEAD
refs/heads/main

Integrità

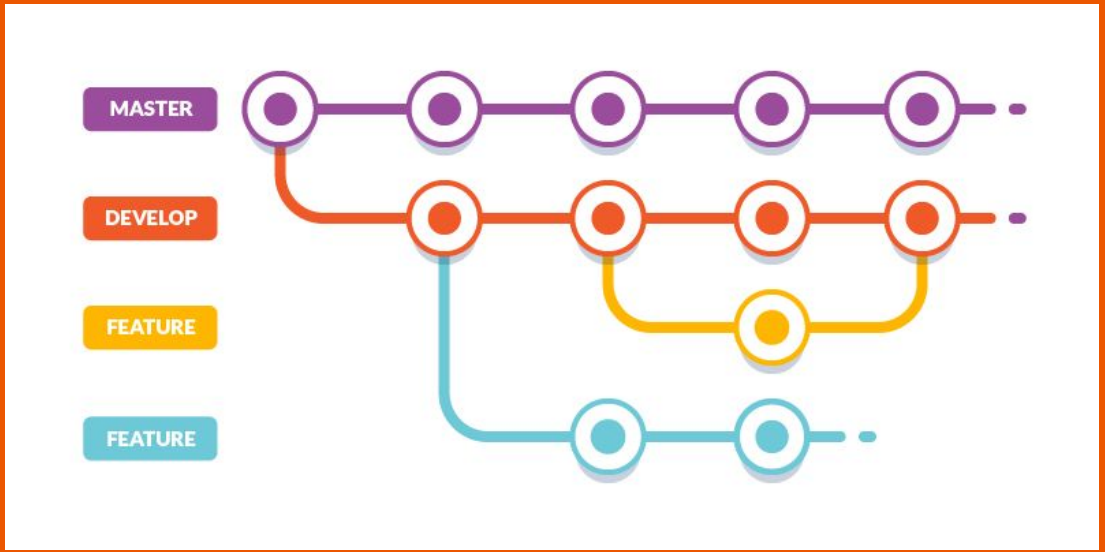
L'integrità viene garantita grazie agli **identificativi (SHA-1)** legati al **contenuto**
Qualsiasi cosa in GIT è controllata tramite checksum

`obj:blob`	**2c6a**
`obj:tree`	**8e2a**
`obj:commit`	**1a15**

SHA-1

2c6a03aa17ddb9053f769147dff01cbb40a81ccd
(30 caratteri esadecimali)

Best Practices...
Like GIT flow



Best Practices

Mantenere le commit piccole e focalizzate

Utilizzare messaggi di commit chiari e concisi

Diramazione frequente dei branch per isolare le modifiche

L’uso delle Pull Request per le Code Reviews

Lavorare su progetti con i membri del team

Possibilità di lavorare su progetti open-source

tips & tricks

Visual Studio Code Extensions:

- Git Graph 
- Git Lens 

Live Demo