

A hand-drawn decorative border in black ink, featuring various loops, swirls, and flourishes that frame the central text. The border is irregular and artistic, with some lines extending outwards from the corners and sides.

Git e GitHub



Obiettivi

Comprendere il concetto di controllo versione: Perché Git è diventato lo standard per la gestione del codice e la collaborazione in team.

Imparare i fondamenti di Git: Repository, commit, branch, merge e altre operazioni essenziali.

Acquisire competenze pratiche: Utilizzare Git da terminale, Github con GUI e comprendere il flusso di lavoro collaborativo.

A hand-drawn decorative border in black ink, featuring various loops, swirls, and flourishes that frame the central text. The border is irregular and artistic, with some lines extending outwards from the corners and sides.

Fondamenti di Git

Cos'è il controllo di versione?

- Un sistema che registra le modifiche a file o insiemi di file nel tempo
- Permette di richiamare versioni specifiche in seguito
- Facilita la collaborazione tra sviluppatori
- Risolve i problemi di conflitti durante il lavoro parallelo
- Storia: dall'approccio manuale ai sistemi moderni

Perché abbiamo bisogno del controllo di versione?

Problema: "Ho salvato il file finale, finale_v2, VERAMENTE_finale, finale_DEFINITIVO..."

Sfide nello sviluppo software:

- Tenere traccia delle modifiche
- Sperimentare senza rischi
- Collaborare con altri sviluppatori
- Ripristinare versioni precedenti
- Documentare il processo di sviluppo

Introduzione a Git

Creato da Linus Torvalds nel 2005 per lo sviluppo del kernel Linux

Sistema di controllo versione **distribuito**

Caratteristiche principali:

- Velocità
- Design semplice
- Supporto per sviluppo non lineare (branching)
- Gestione efficiente di progetti grandi



— www.DucaDaME.it —

Sistemi centralizzati vs distribuiti

Centralizzato (SVN, CVS):

- Un server centrale contiene tutte le versioni
- Gli sviluppatori estraggono solo la versione corrente
- Richiede connessione al server per la maggior parte delle operazioni

Distribuito (Git, Mercurial):

- Ogni sviluppatore ha una copia completa dell'intero repository
- Possibilità di lavorare offline
- Maggiore ridondanza e sicurezza

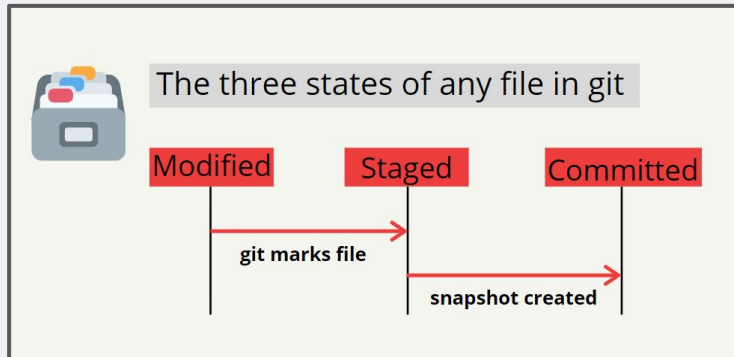
Come funziona Git

I tre stati di Git:

- Working Directory: dove modifichi i file
- Staging Area (Index): dove prepari i cambiamenti da salvare
- Repository: dove sono memorizzati i cambiamenti confermati

Flusso di lavoro base:

1. Modifichi file nella working directory
2. Stage dei file modificati (preparazione)
3. Commit delle modifiche (salvataggio permanente)



Installazione e configurazione di Git

Installazione:

- Windows: <https://git-scm.com/download/win>
- macOS: `brew install git` o <https://git-scm.com/download/mac>
- Linux: `sudo apt-get install git` (Ubuntu/Debian)

Configurazione iniziale:

```
1 git config --global user.name "Il tuo nome"
2 git config --global user.email "tua.email@esempio.com"
3 git config --global init.defaultBranch main
4
5
```

Creare un repository

Inizializzare un nuovo repository:

```
1 mkdir mio-progetto  
2 cd mio-progetto  
3 git init
```

Clonare un repository esistente:

```
1 git clone https://github.com/utente/repository.git
```

Comandi base di Git

- Verificare lo stato:

```
git status
```

- Aggiungere file all'area di staging:


```
git add file.txt # Aggiunge un singolo file
```

```
git add . # Aggiunge tutti i file modificati
```

```
git add src/*.java # Aggiunge tutti i file Java nella cartella src
```

- Verifica le modifiche

```
git diff
```



Comandi base di Git

- Effettuare un commit:

```
git commit -m "Messaggio descrittivo del commit"
```

- Aggiungere file all'area di staging:

```
git log
```

```
git log --oneline          # Formato compatto
```

```
git log --graph --oneline  # Visualizzazione grafica
```

Buone pratiche per i messaggi di commit

- Scrivere messaggi chiari e descrittivi
- Usare l'imperativo presente: "Aggiungi funzionalità" non "Aggiunta funzionalità"
- Prima riga: breve riassunto (max 50 caratteri)
- Corpo: spiegazione dettagliata (opzionale)

Il concetto di branch

- Un branch è un puntatore mobile a un commit
- Permette sviluppo parallelo e isolato
- Il branch principale è `main` (precedentemente `master`)
- Comandi principali:

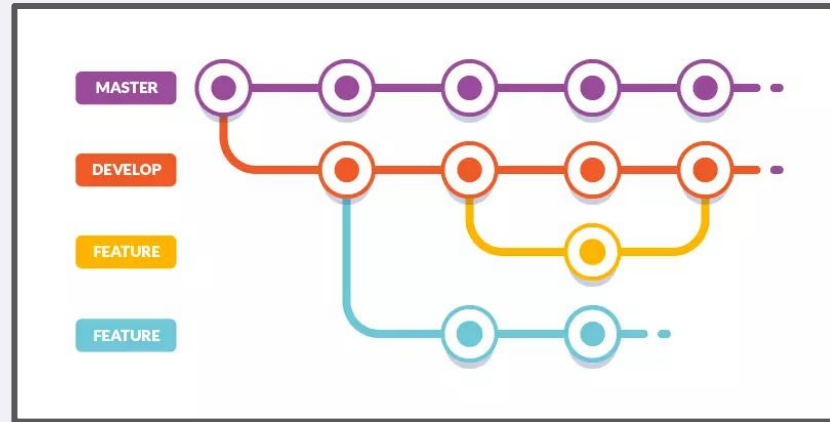
```
1 git branch                # Elenca i branch
2 git branch feature-login  # Crea un nuovo branch
3 git checkout feature-login # Passa a un branch
4 git checkout -b new-feature # Crea e passa a un nuovo branch
```

Merging

- Unire le modifiche da un branch all'altro

```
git checkout main      # Passa al branch di destinazione
```

```
git merge feature-login # Unisce il branch feature-login in main
```



Risolvere i conflitti

- I conflitti si verificano quando le stesse righe di un file sono state modificate in entrambi i branch
- Git marca le aree in conflitto:

```
1 <<<<<< HEAD
2 Modifiche nel branch corrente
3 =====
4
5 Modifiche nel branch che stai unendo
6 >>>>>> feature-branch
```

Risoluzione manuale: modifica il file, rimuovi i marcatori e salva

Dopo la risoluzione:

```
git add file-con-conflicto.txt
```

```
git commit -m "Risolvi conflitto di merge"
```


Ripristinare modifiche

- Annullare modifiche non ancora in staging:

```
git checkout -- file.txt
```

- Rimuovere file dall'area di staging:

```
git reset HEAD file.txt
```

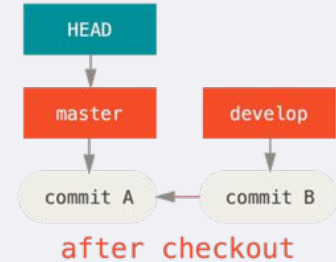
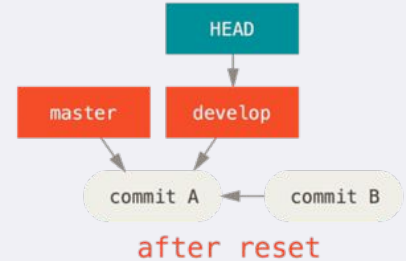
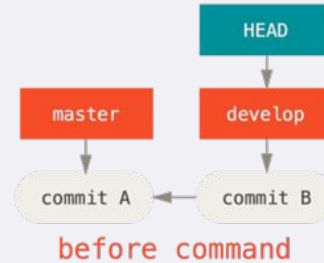
- Modificare l'ultimo commit:

```
git commit --amend -m "Nuovo messaggio"
```

- Ripristinare a un commit precedente:

```
git reset --hard HEAD~1    # Elimina l'ultimo commit
```

```
git revert HEAD             # Crea un nuovo commit che annulla l'ultimo
```



A hand-drawn decorative border in black ink, featuring various loops, swirls, and lines that frame the central text. The border is irregular and artistic, with some lines extending outwards from the corners and sides.

GitHub e collaborazione

Da Git a GitHub

Git: sistema di controllo versione

GitHub: piattaforma di hosting per repository Git

Vantaggi di GitHub:

- Hosting remoto (backup e accessibilità)
- Strumenti di collaborazione
- Issue tracking
- Pull requests
- Integrazione con CI/CD
- Community e social coding



Alternative a GitHub

GitLab: open source, self-hosted o cloud

Bitbucket: integrazione con altri prodotti Atlassian

Azure DevOps: integrazione con l'ecosistema Microsoft


Gitea/Gogs: alternative leggere e self-hosted

Differenze principali: funzionalità, prezzi, privacy, integrazione




GitLab

— www.Duca10.ME —



Creare un account GitHub

- Registrazione su `github.com`
 - Configurazione del profilo
- 

Repository remoti

- Repository remoti

```
1 git remote add origin https://github.com/utente/repository.git
```


Push e Pull

- **Push:** invio delle modifiche locali al repository remoto


```
1 git push -u origin main    # Prima volta con tracking
2 git push                  # Successive volte
```

- **Pull:** recupero e integrazione delle modifiche remote
- **Fetch:** recupero senza integrazione automatica
- git fetch è il comando che dice al tuo git locale di ottenere i meta-dati più recenti dall'originale (ma non fa nessun trasferimento di file). git pull , d'altra parte, fa quello e copia i cambiamenti dal repository remoto.

```
1 git pull                  # Equivale a git fetch + git merge
2 git fetch origin
```

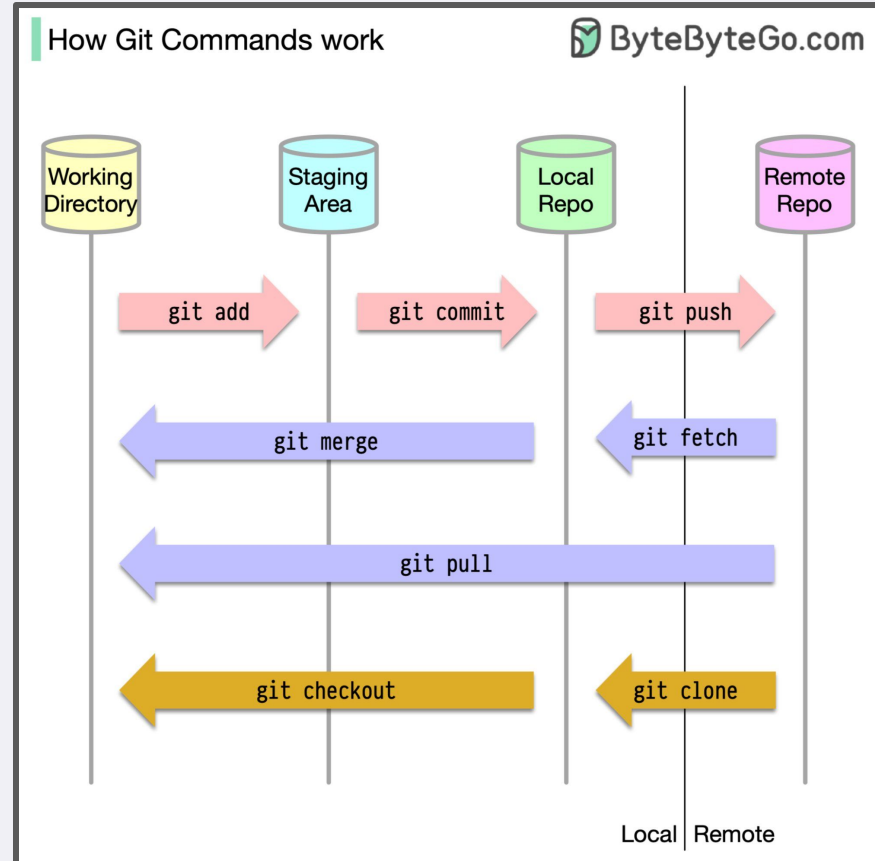


Il modello Fork & Pull Request



1. Fork del repository originale sul proprio account
2. Clone del fork in locale
3. Creazione di un branch per la nuova feature
4. Sviluppo e commit delle modifiche
5. Push del branch sul proprio fork
6. Creazione di una Pull Request verso il repository originale
7. Discussione, review e merge

Il modello Fork & Pull Request





Creare e gestire Pull Request

Creazione:

- Da GitHub: "New pull request"
- Titolo chiaro e descrizione dettagliata
- Riferimento a issues: "Fixes #123"

Gestione:

- Code review
- Commenti in linea
- Richieste di modifica
- Merge finale



Documentare un progetto

README.md: documento principale

- Descrizione del progetto
- Istruzioni di installazione
- Guida rapida all'uso
- Screenshot/demo
- Stato di sviluppo
- Licenza

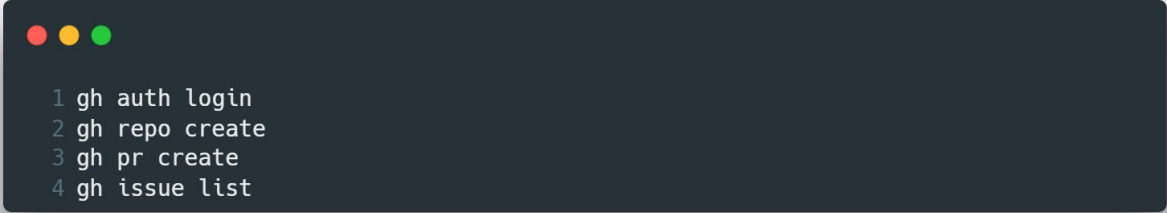
CONTRIBUTING.md: guida per i contributori

LICENSE: licenza del software

CODE_OF_CONDUCT.md: regole di comportamento

GitHub CLI e Desktop

GitHub CLI: interfaccia a riga di comando

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a numbered list of four GitHub CLI commands.

```
1 gh auth login
2 gh repo create
3 gh pr create
4 gh issue list
```

GitHub Desktop: client grafico

- Gestione repository
- Visualizzazione modifiche
- Commit e push semplificati
- Gestione branch visuale



GitHub Actions

Componenti principali:

- Workflows (file YAML in `.github/workflows/`)
- Jobs
- Steps
- Actions

Trigger: push, pull request, schedule, manuale, ecc.

Esempio base di GitHub Actions

```
1 name: CI
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10  build:
11    runs-on: ubuntu-latest
12
13    steps:
14      - uses: actions/checkout@v3
15
16      - name: Setup Node.js
17        uses: actions/setup-node@v3
18        with:
19          node-version: '16'
20
21      - name: Install dependencies
22        run: npm ci
23
24      - name: Run tests
25        run: npm test
```

Casi d'uso comuni per GitHub Actions

Build e test automatici


- Compilazione del codice
- Esecuzione di test unitari e di integrazione
- Analisi statica del codice

Deployment automatico

- Siti web
- Applicazioni mobili
- Container Docker

Automazione di repository

- Gestione di issue e PR
- Generazione di documentazione
- Rilascio di versioni



GitHub Pages

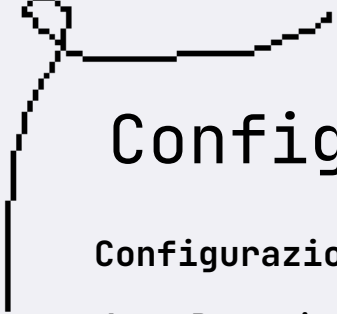
Cos'è: hosting gratuito per siti statici direttamente da un repository GitHub

Tipi di GitHub Pages:

- Pages di utente/organizzazione (username.github.io)
- Pages di progetto (username.github.io/project)

Caratteristiche:

- HTTPS incluso
- Supporto per domini personalizzati
- Ottimizzazione della cache



Configurare GitHub Pages

Configurazione base:

1. Repository pubblico su GitHub
2. Impostazioni → Pages
3. Seleziona branch e cartella (root o /docs)

Soluzioni comuni:

- HTML/CSS/JS statici
- Altri generatori di siti statici (Hugo, Next.js, ecc.) con GitHub Actions

Risorse per continuare ad imparare

Risorse per continuare ad imparare

- **Documentazione ufficiale:**
 - git-scm.com/doc
 - docs.github.com
- **Esercitazioni interattive:**
 - learngitbranching.js.org
 - lab.github.com
- **Libri consigliati:**
 - "Pro Git" di Scott Chacon (gratuito online)
 - "GitHub Actions: Up & Running" di Rosemary Wang
- **Community e forum:**
 - GitHub Community Forum
 - Stack Overflow

Esercizi - Base

1. Creare il primo repository

- a. Crea una nuova directory per un progetto, inizializza un repository Git, crea un file README.md con una breve descrizione del progetto, aggiungilo all'area di staging e fai il primo commit.

2. Visualizzare la storia dei commit

- a. Modifica il README.md aggiungendo una nuova sezione, effettua un secondo commit e poi visualizza la storia dei commit in diversi formati.

3. Annullare le modifiche

- a. Modifica il README.md, visualizza le differenze, annulla le modifiche e verifica che il file sia tornato allo stato precedente.

Esercizi - Intermedi

1. Lavorare con i branch

- a. Crea un nuovo branch chiamato "feature-login", aggiungi un file login.txt con del contenuto, committa le modifiche, torna al branch main e poi unisci (merge) il branch feature-login in main.

2. Risolvere un conflitto di merge

- a. Crea un branch "feature-profile", modifica il README.md aggiungendo una sezione "Profilo utente", torna al branch main, modifica la stessa parte del README.md in modo diverso, prova a fare il merge e risolvi il conflitto.



FINE



— WWW.DRACO.ME —