



I branch in Git

Corso di Laboratorio di Gestione progetto e organizzazione d'impresa

Prof. Dei Rossi, Leonardo Essam

I branch (1)

In Git, o più in generale su un sistema di controllo di versione (VCS), un branch lo si può vedere come uno spazio di lavoro separato nel quale è possibile apportare delle modifiche e testare delle nuove idee senza creare danni nel progetto principale.

Alcuni motivi comuni per creare un branch sono:

- Sviluppo di una nuova funzione;
- Fix di uno o più bug;
- Sperimentare con nuove idee.

Esempio: con o senza Git?

Supponiamo di essere il team a capo dello sviluppo di una Web App per la Provincia Autonoma di Trento e che la PAT richieda al team di sviluppo di modificare il design dell'interfaccia.

In uno scenario senza Git, il programmatore:

1. Crea una copia di tutti i file rilevanti per evitare di compromettere la versione originale;
2. Inizia a lavorare sul progetto e scopre che il codice dipende dal codice presente in altri file, che devono anch'essi essere modificati;
3. Crea copie anche dei file dipendenti, assicurandoti che ogni dipendenza faccia riferimento al nome file corretto;

Esempio: con o senza Git?

4. Emergenza: c'è un errore non correlato in un'altra parte del progetto che deve essere risolto al più presto;
5. Salva tutti i suoi file, prendendo nota dei nomi delle copie su cui stava lavorando;
6. Lavora sull'errore non correlato e aggiorna il codice per risolverlo;
7. Torna al progetto e finisce il lavoro iniziato;
8. Copia il codice o rinomina i file, in modo che il design aggiornato sia nella versione definitiva.

Esempio: con o senza Git?

Due settimane dopo, lo sviluppatore si rende conto che l'errore non correlato non è stato risolto nella nuova versione del design perché ha copiato i file prima della correzione!

Esempio: con o senza Git?

In uno scenario con Git, invece:

1. Con un nuovo ramo denominato `new-design` e modifica il codice direttamente senza influire sul ramo principale;
2. Emergenza: c'è un errore non correlato in un'altra parte del progetto che deve essere risolto al più presto;
3. Crea un nuovo ramo dal progetto principale chiamato `error-fix`;
4. Corregge l'errore non correlato e unisce il ramo `error-fix` con il ramo principale;
5. Torna al ramo del nuovo design e lì finisce il lavoro;
6. Unisce il ramo `new-design` con il ramo `main`.

Esempio: con o senza Git?

Riassumendo:

- I branch consentono di lavorare su parti diverse di un progetto senza influire sul ramo principale;
- Una volta completato il lavoro, un branch può essere unito al progetto principale;
- Si può anche passare da un branch all'altro e lavorare su progetti diversi senza che interferiscano tra loro.

Il branching in Git è molto leggero e veloce!

Creare un nuovo branch

Supponiamo di voler aggiungere una nuova funzionalità al nostro progetto. Possiamo creare un nuovo ramo per essa.

Vale quindi il comando:

```
git branch <nome ramo>
```

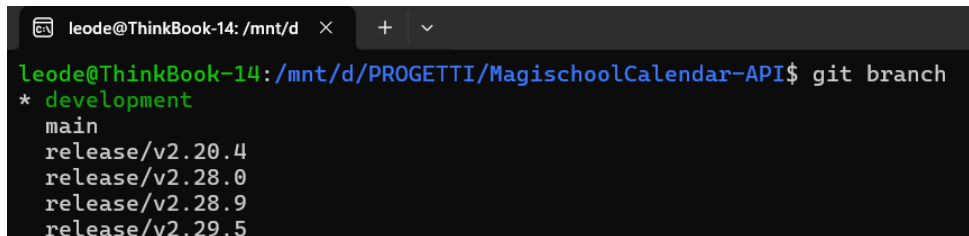
Ora è stato creato un nuovo ramo chiamato "<nome ramo>"¹.

¹Sostituire *<nome ramo>* con il nome che si vuole dare al branch.

Visualizzare i branch (1)

Per vedere tutti i branch presenti in un repository, vale il comando:

```
git branch
```

A screenshot of a terminal window with a dark background. The title bar shows 'leode@ThinkBook-14: /mnt/d' and a search icon. The prompt is 'leode@ThinkBook-14:/mnt/d/PROGETTI/MagischoolCalendar-API\$'. The command 'git branch' has been executed, resulting in the following output: '* development', 'main', 'release/v2.20.4', 'release/v2.28.0', 'release/v2.28.9', and 'release/v2.29.5'.

```
leode@ThinkBook-14: /mnt/d  ×  +  v  
leode@ThinkBook-14:/mnt/d/PROGETTI/MagischoolCalendar-API$ git branch  
* development  
main  
release/v2.20.4  
release/v2.28.0  
release/v2.28.9  
release/v2.29.5
```

Esempio tratto dal progetto di quinta superiore del prof:)

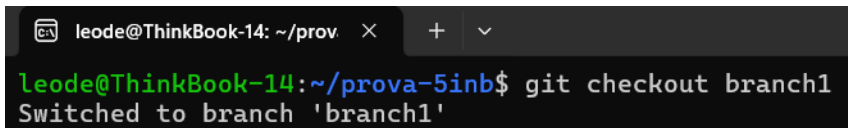
Visualizzare i branch (2)

Per spostarsi da un branch all'altro, vale il comando:

```
git checkout <nome ramo>
```

Dove <nome ramo> è il nome del branch sul quale ci si vuole spostare.

Ad esempio:

A screenshot of a terminal window with a dark background. The title bar at the top shows a window icon, the text 'leode@ThinkBook-14: ~/prov', and window control buttons (close, maximize, minimize). The terminal content shows a prompt 'leode@ThinkBook-14:~/prova-5inb\$' followed by the command 'git checkout branch1'. The output of the command is 'Switched to branch 'branch1''.

```
leode@ThinkBook-14: ~/prov × + ∨  
leode@ThinkBook-14:~/prova-5inb$ git checkout branch1  
Switched to branch 'branch1'
```

Lavorare nel branch

Ora tutte le modifiche verranno fatte all'interno del branch in cui ci si è spostati. Per verificare i file modificati vale sempre il comando: `git status`

Anche se non si è più nel branch principale (main o master) valgono sempre le stesse regole per i commit:

```
git add <file 1> <file 2> <...>  
git commit -m "<messaggio>"
```

Viaggio tra le dimensioni (1)

La scorsa volta abbiamo visto come fare i viaggi nel tempo, oggi vedremo come viaggiare tra le dimensioni ;)

Con il comando:

```
git checkout master
```

Attenzione: nelle versioni più recenti di Git il ramo principale è chiamato `main` (lunga storia...).

Viaggio tra le dimensioni (2)

Quando si viaggia in due mondi paralleli vi è la remota possibilità che, anche in momenti diversi, si vada a modificare qualcosa di quel mondo.

In Git questo si traduce nello scenario in cui, in due branch diversi, si va a modificare lo stesso file.

Provate a svolgere il breve esercizio pubblicato in Classroom e poi proseguiamo con il viaggio tra le dimensioni!

Viaggio tra le dimensioni (3)

Dopo essere passati dal ramo `branch2` al ramo principale e aver eseguito il comando `ls -l`, cosa notiamo?

Notiamo che `file3.txt` non è più presente ma `file4.txt` invece sì.

Ma perché è successo?

Dalle puntate precedenti... (1)

Se si osserva bene la sequenza di comandi, il file `file4.txt` è l'unico file di cui non è stato fatto l'*add* e il *commit*!

Quindi Git non è a conoscenza della sua esistenza e di conseguenza le regole dei viaggi inter-dimensionali (o, più seriamente, dei cambi di branch) non si applicano!

```
leode@ThinkBook-14:~/prova-5inb3$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file4.txt

nothing added to commit but untracked files present (use "git add" to track)
```

L'unione di due mondi (1)

Una delle operazioni principali di Git è l'operazione di **merge**.

Il merge in Git è l'operazione dove un ramo X viene ricongiunto a un ramo Y , riprendendo l'esempio della Supercell ad esempio:

- Branch brawler-mrp e game-footbrawl;
- Branch master.

Supponiamo ora di voler unire i due branch al master.



L'unione di due mondi (2)

Dalla console di Git, ci si posiziona nel branch di destinazione:

```
git checkout master
```

Ora segue il merge dei branch:

```
git merge brawler-mrp
```

```
git merge game-footbrawl
```

L'unione di due mondi (3)

Ora Git domanderà di scrivere il perché dell'operazione di *merge*, questo perché viene generato un nuovo *commit* e ciò che viene scritto sarà il messaggio del commit!

```
leode@ThinkBook-14:~/prova-5inb3$ git log
commit e53cb5113af98a79fc0f1c698cd2578f371a2096 (HEAD -> master)
Merge: a63bc77 1da9977
Author: Leonardo Essam <leode@casamika.local>
Date:   Sun Oct 5 15:58:29 2025 +0200

    Merge branch 'branch3'
```

