

UNIVERSITÀ DEGLI STUDI DI SALERNO



**Dipartimento di Ingegneria dell'Informazione ed
Elettrica e Matematica applicata**

Corso di Laurea in Ingegneria Informatica

**APPUNTI DI INGEGNERIA DEL SOFTWARE
DI FRANCESCO PIO CIRILLO**

<https://github.com/francescopiocirillo>



"Non ti abbattere mai"

😊 Ehi, un attimo prima di iniziare!

Hai appena aperto una raccolta di appunti che ho deciso di condividere **gratuitamente** su GitHub, se ti sono utili fai **una buona azione digitale**:

-  **Lascia una stellina alla repo:** è gratis, indolore e fa super piacere!
-  **Condividerla con amici**, compagni di corso, o chiunque possa averne bisogno.

Insomma, se questi appunti ti salvano anche solo una giornata di studio... fammelo sapere con una **stellina!**

Grazie di cuore ❤

Version Control Systems and Git

Version Control Systems



I Sistemi di Controllo Versione sono software in grado di gestire gruppi di file e directories nel tempo (anche noti come Revision Control Systems o Source Code Management).

Questi Software:

- ricordano ogni cambiamento e quando e da chi è stato fatto il cambiamento
 - permettono di tornare ad una versione precedente e di comparare diverse versioni
 - permettono di fare la merge (fusione) dei cambiamenti fatti da diversi sviluppatori
- ▼ Pillole di storia

- 1972: **SCCS**
 - IBM Mainframe, and later Unix; required file system sharing
- 1986: **CVS**
 - Client-server with concurrency control (pessimistic concurrency, i.e. file locking)
- 2000: **Subversion**
 - Versioning at the directory (not file) level; atomic commit; support for binary files; optimistic concurrency
- today: **Arch**, **Monotone**, **BitKeeper**, **Git**, **Bazaar**, ...
 - Distributed (as opposed to centralized) repository

Git

▼ Storia e curiosità

Creato da Linus Torvalds per lavorare sul Kernel di Linux nel 2005 dopo delle controversie con l'azienda BitKeeper.

Full docs: <https://git-scm.com/>

Aziende come Google, Netflix, Microsoft e Meta usano Git.

Il nome di Git deriva dal termine dell'inglese britannico "git" che significa sciocco o spregievole.

Il creatore ha scelto il nome con un tocco di sarcasmo per riflettere la sua frustrazione nei confronti degli allora esistenti sistemi di controllo versione.

Quando nel 2005 Linus Torvald ha presentato Git al mondo ha detto:

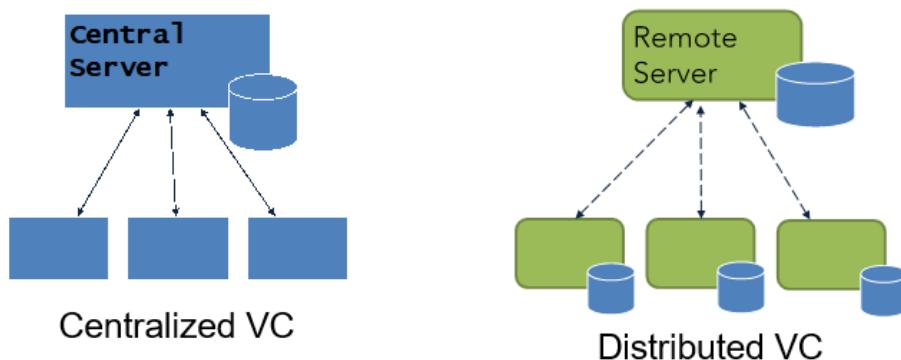
Sono un bastardo egoista, nomino tutti i miei progetti in mio onore. Prima "Linux" e ora "git".

Git è stato creato con in mente i seguenti concetti chiave:

- **velocità**
- **design semplice**
- **forte supporto per lo sviluppo non-lineare** (migliaia di rami (**branches**) paralleli)
- completamente distribuito
- **capace di gestire grandi progetti** come il Kernel di Linux in maniera efficiente (in termini di **velocità e di dimensione dei dati**)

Git è completamente distribuito, questo vuol dire che:

- tutti hanno la cronologie completa
- tutto è fatto **offline**
- **non c'è un'autorità centrale**
- i cambiamenti possono essere condivisi **senza un server**



▼ Installare Git

Git è disponibile per la maggior parte dei sistemi operativi, basta scaricarlo da <https://git-scm.com/downloads> e seguire le istruzioni.

Git è già integrato in molte IDE come NetBeans, Eclipse e VSCode.

Git è compatibile con molti sistemi di controllo versione basati su cloud come GitHub, GitLab, Bitbucket e Azure DevOps.

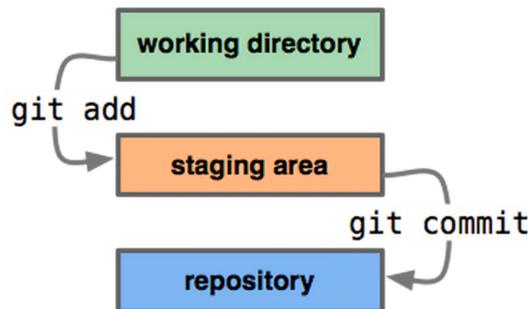
Git: concetti fondamentali

Git Sections

Working Directory: la vera e propria directory nel proprio filesystem dove vengono creati e modificati i propri file.

Staging Area (index): un'area intermedia dove si possono preparare i propri cambiamenti prima di farne il commit. Fa da snapshot di cosa andrà nel prossimo commit.

Repository (head): è dove Git conserva permanentemente la cronologia di tutti i cambiamenti committed. Ogni commit rappresenta uno snapshot del proprio progetto in un certo momento nel tempo.



Git File States

Git ha **tre principali stati** nei quali possono trovarsi i file:

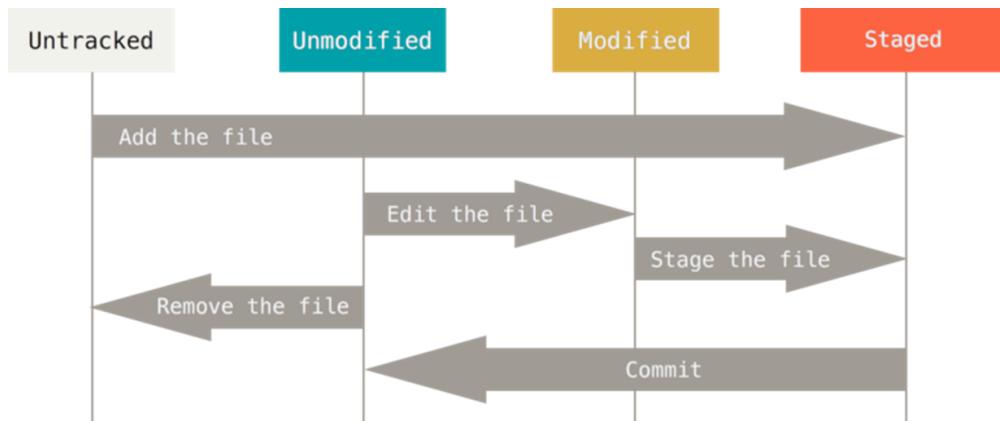
- **Committed:** il file è conservato al sicuro nella repository
- **Modified:** il file è stato cambiato rispetto all'ultima **commit**
 - stiamo lavorando sul file (nella **working directory**), ci saranno ulteriori cambiamenti
- **Staged:** abbiamo terminato le modifiche e messo il file nella **staging area**
 - il file è ora pronto ad essere aggiunto alla repository con la prossima **commit**



I file states di Git rendono un sistema basato sul concetto di **promozione**

Altri possibili stati dei file sono:

- **Untracked:** un file nella working directory i cui cambiamenti sono però non tracciati da Git
 - se si decide di iniziare a tracciare i cambiamenti al file questo va direttamente nello stato **Staged**
- **Unmodified:** un file nella working directory che non è stato modificato rispetto all'ultima commit (in altre parole **Committed**)



Project History (cronologia del progetto)

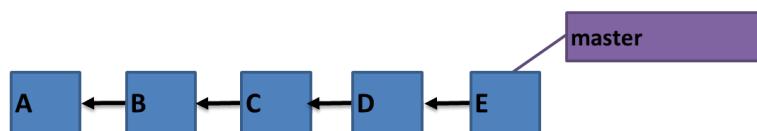
Nella repository la cronologia del progetto è rappresentata da un **grafo**.

I **nodi** del grafo sono creati con una operazione di commit.

A ATTENZIONE: andrebbe fatta la commit solo di "file sorgente", cioè file non ottenuti automaticamente da altri file. Ad esempio in un progetto Java vanno aggiunti i file .java ma non i file oggetto e gli eseguibili come .class

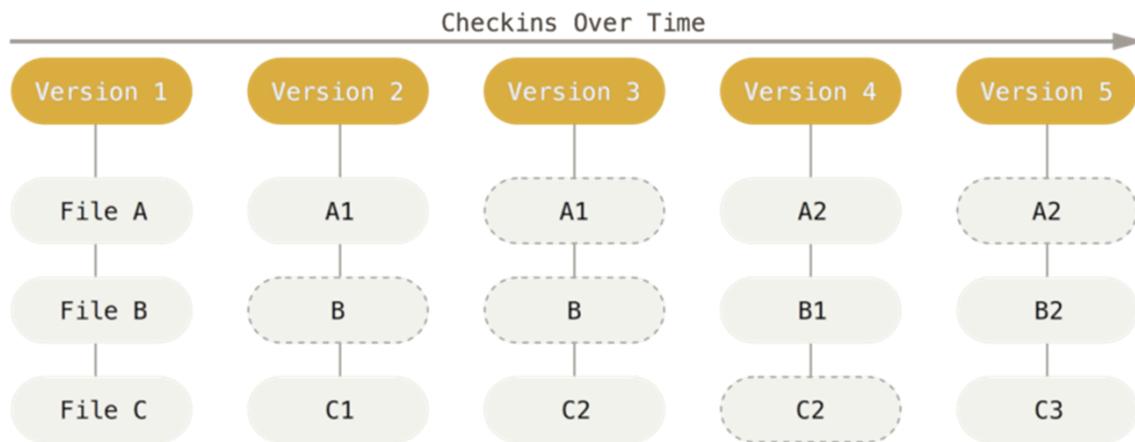
Ogni nodo contiene solo i file che sono stati modificati rispetto ai suoi nodi "genitori", bisognerebbe pensare ad un nodo come ad un **checkpoint** al quale si può facilmente tornare se qualcosa va storto.

In foto una repository con 5 nodi (ottenuta quindi dopo 5 commit).



A è genitore di B che è genitore di C e così via.

Ogni nodo rappresenta uno **snapshot** (una fotografia istantanea) dei file nel particolare momento nel tempo in cui è stata fatta l'operazione di commit che ha creato il nodo.



Ogni nodo conserva solo i file cambiati rispetto al nodo precedente.

Git: comandi principali

- **git init:** permette a git di controllare i file nella directory corrente
 - dopo aver eseguito questo comando si ottiene una repository vuota senza commits né branches (ramificazioni)
 - dopo aver eseguito questo comando si può iniziare ad aggiungere file, effettuare commits e usare altri comandi Git per gestire il proprio progetto sottoposto a controllo di versione
 - **git status:** mostra lo stato di tutti i file nella directory corrente
 - **git add:** mette un file che è sotto il controllo di Git nella staging area
 - **git add <file(s)>:** mette nella staging area i file specificati
 - **git add . :** mette nella staging area tutte le modifiche, aggiunta e cancellazioni attualmente presenti nella working directory
 - **git add <directory>:** effettua la stessa operazione di **git add .** ma per una specifica directory invece che per tutta la working directory
 - **git commit:** salva la staging area nella repository, creando quindi un nuovo nodo
 - **git commit -m “Commit message”:** permette di fare la commit specificando anche un messaggio per descrivere i cambiamenti introdotti
 - **git diff:** mostra le differenze tra la working directory e l'ultima commit
 - **git log:** mostra la cronologia delle commit, mostrandone l'id, l'autore, la data e il messaggio di commit
 - **git reset <file(s)>:** toglie dalla staging area i cambiamenti fatti ai file, è l'opposto di **git add**
 - **git revert <commit id>:** crea una nuova commit che annulla i cambiamenti introdotti da un'altra, è l'opposto di **git commit**
- ▼ Esempio di utilizzo dei comandi

```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first commit'
[master (root-commit) 7106a52] my first commit
 1 file changed, 1 insertion(+)
create mode 100644 README.txt
```

Branches e Merging

In Git le branches permettono di lavorare su diverse versioni del progetto simultaneamente. Sono utili per esplorare nuove idee o per risolvere bug.

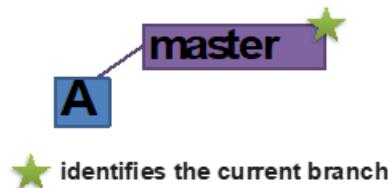
Ogni branch rappresenta una **linea di sviluppo indipendente**.

Si possono **creare** nuove branch, fare lo **switch** tra diverse branch ed effettuare la **merge** (fusione) di più branch.

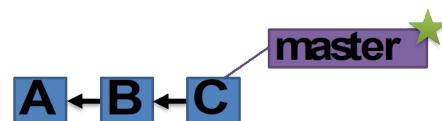
La **current branch** è la branch sulla quale si sta lavorando al momento, tutte le commit sono **allegate** alla branch corrente.

Branch Illustrate

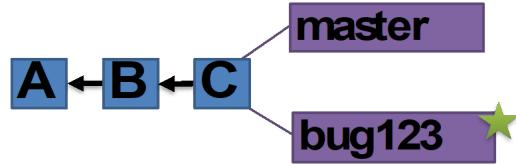
```
> git commit -m 'my first commit'
```



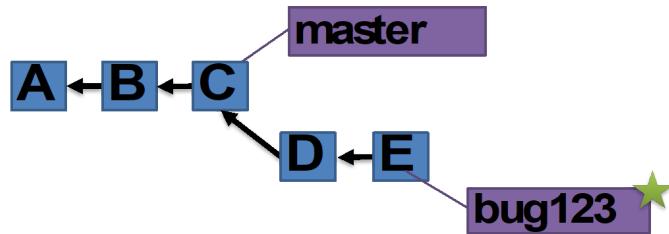
```
> git commit
> ...
> git commit
```



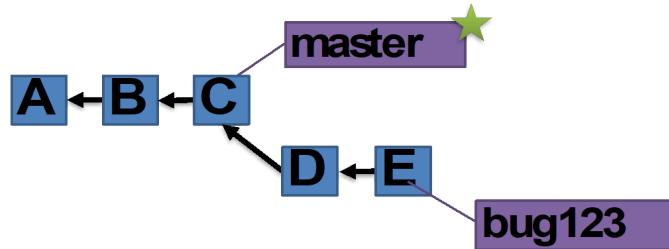
```
> git checkout -b bug123  
//cambia la current branch  
//l'opzione -b crea una nuova branch
```



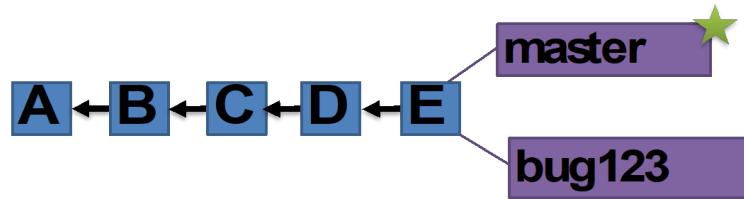
```
> git commit  
> ...  
> git commit
```



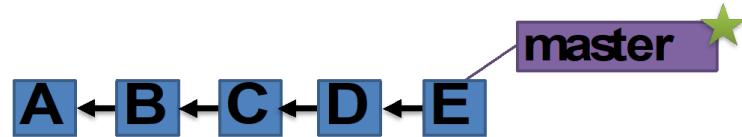
```
> git checkout master  
//si torna al master branch
```



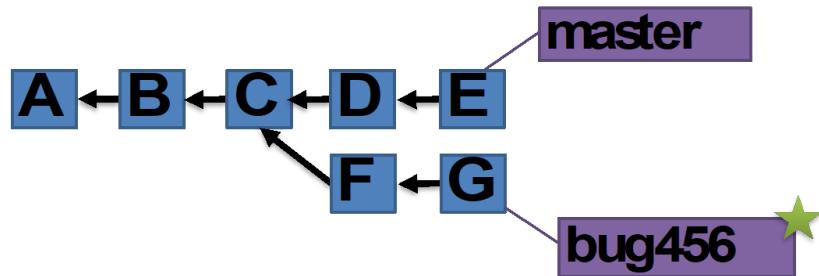
```
> git merge bug123  
/*fonde la current branch con un'altra specificata  
applicando i cambiamenti fatti all'altra branch*/
```



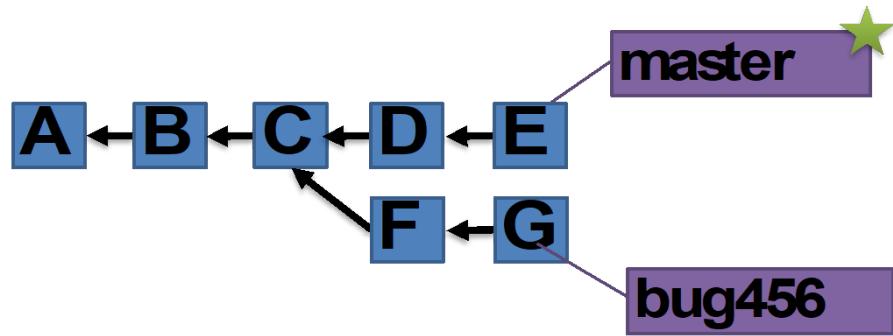
```
> git branch -d bug123  
//crea una nuova branch (o con -d la elimina)
```



```
//dopo un po' di lavoro fatto partendo dal nodo C abbiamo cambiamenti in entrambi
```



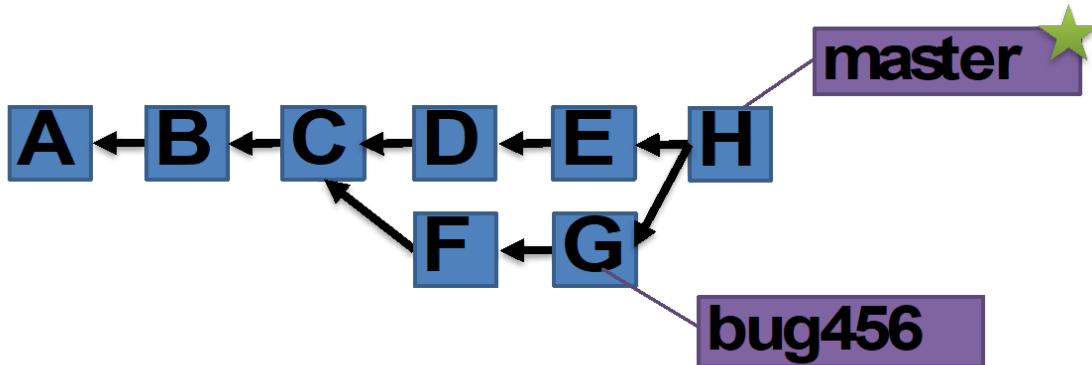
```
> git checkout master  
//permette di tornare al master branch
```



```

> git merge bug456
//potrebbe aggiungere un nuovo nodo per fondere i due branches
//ma solo se non sono rilevati conflitti tra i due rami

```



Gestione dei Conflitti

La merge potrebbe non essere completabile se ci sono conflitti tra due rami ad esempio:

- file di testo che sono diversi in parti che si sovrappongono
- file binari che sono diversi

In questi casi la **merge fallisce e viene fatto un report dei conflitti**.

Lo sviluppatore a questo punto deve:

- modificare il file e decidere quali cambiamenti mantenere o combinare
- effettuare stage e commit dei file con i conflitti risolti

```
index.html styles.css
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>GC Merge Demo</title>
5  <link href="styles.css" rel="stylesheet" type="text/css" />
6  </head>
7  <body>
8  <<<< HEAD
9  <h1>Grand Circus Merge Demo</h1>
10 =====
11 <h1 class="header">Merge Demo</h1>
12 >>>> 1d46372af5a97f8ef05b9eebc82712382cc5f31c
13 <p>
14     Demo the merge.
15 </p>
16 <p class="footer">
17     Grand Circus Detroit
18 </p>
19 </body>
20 </html>
21
```

Dei marker sono aggiunti automaticamente ai file in conflitto in modo da indicare dove sono presenti conflitti

Git e NetBeans

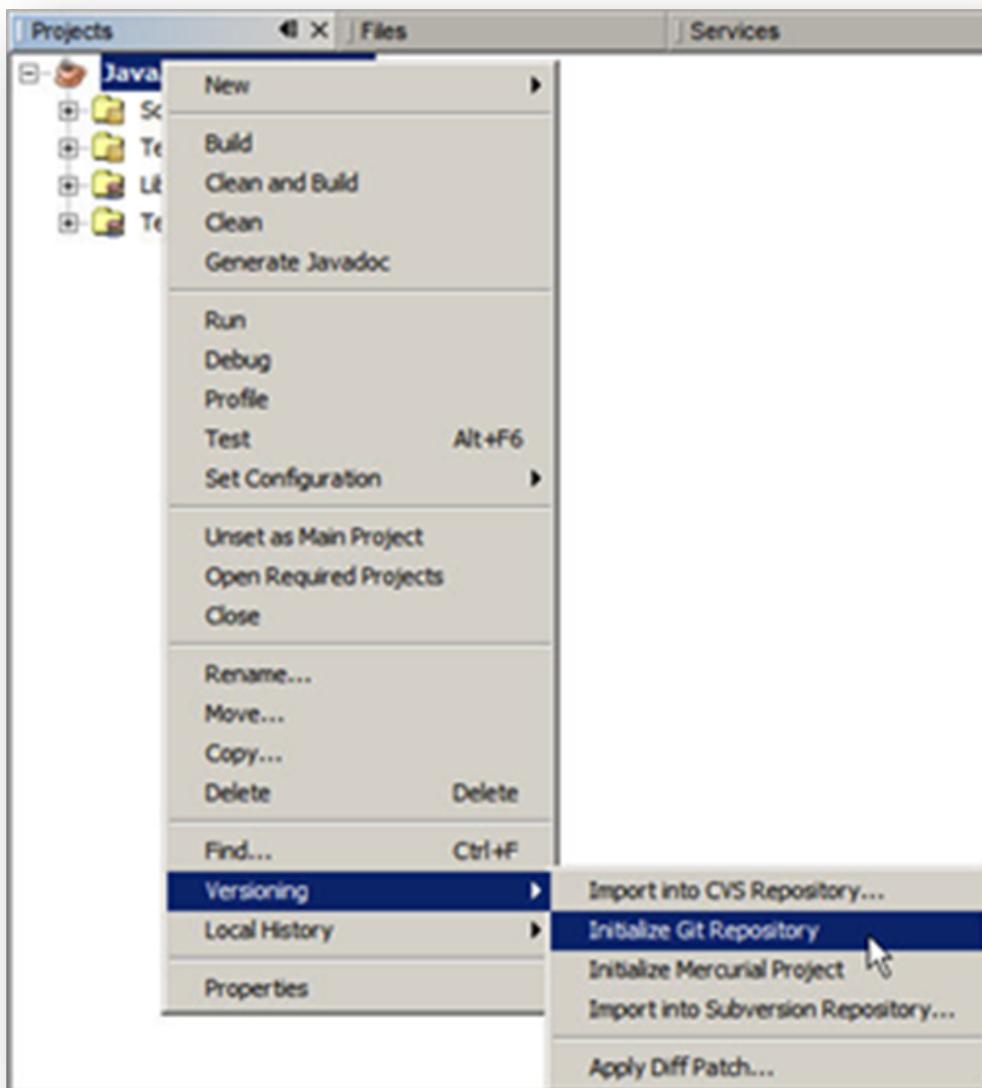
La IDE Apache Netbeans fornisce supporto a Git.

Esempio di utilizzo: <https://netbeans.apache.org/tutorial/main/kb/docs/ide/git/>

Inizializzare una Repository Git in NetBeans

Per inizializzare una Repository Git a partire da un progetto esistente:

- nella finestra Projects seleziona un progetto attualmente non sottoposto a controllo di versione
- fai tasto destro sul nome del progetto
- scegli **Versioning → Initialize Git Repository**

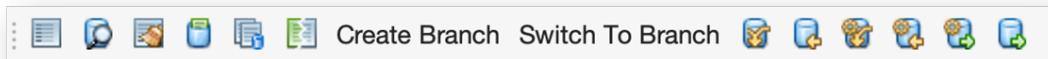


- specifica il percorso della repository e clicca Ok (di default il percorso sarà la cartella del progetto NetBeans)

A questo punto sarà creata una sottocartella **.git** nella cartella specificata, questa è la **Git repository**.

Tutti i file del progetto saranno a questo punto marcati come **added** (aggiunti, saranno in verde) nel **Working Tree** (albero di lavoro).

Può essere utile attivare la **Git toolbar**



A seconda del colore dei file è possibile riconoscere il loro stato corrente

Color	Example	Description
No specific color (black)	Main.java	Indicates that the file has no changes.
Blue	Main.java	Indicates that the file has been locally modified.
Green	Main.java	Indicates that the file has been locally added.
Red	Main.java	Indicates that the file is in a merge conflict.
Gray	Main.java	Indicates that the file is ignored by Git and will not be included in versioning commands (e.g. Update and Commit). Files cannot be ignored if they are versioned.

I colori sono anche usati ai margini dell'Editor del codice per trasferire i seguenti concetti:

- **Blu:** righe che sono state cambiate rispetto alle versioni precedenti
- **Verde:** righe che sono state aggiunte rispetto a versioni precedenti
- **Rosso:** righe che sono state rimosse rispetto a versioni precedenti

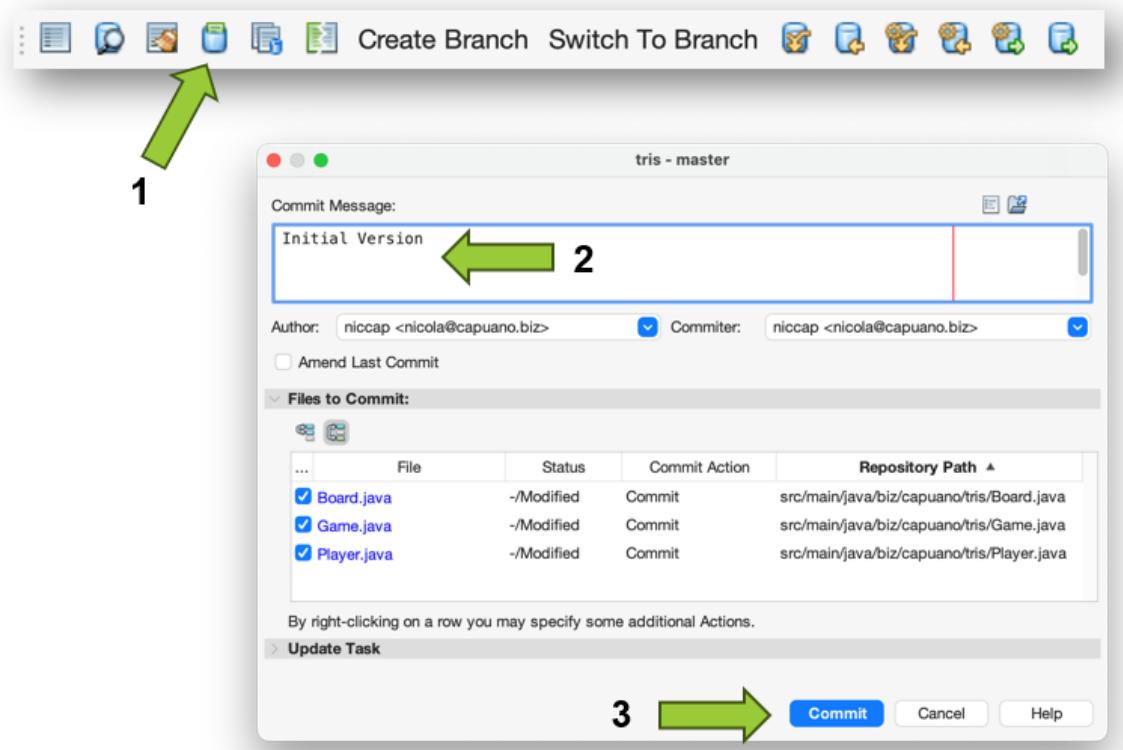
```

5
6
7 1 line deleted
8
9 package javaapplicationforgit;
10 /**
11
12 * @author author@example.com
13 */
14
15 public class JavaApplicationForGitModified {
16 }

```

Commit e Comparing

Per effettuare il **commit** dei cambiamenti si esegue la seguente procedura:



La IDE permette inoltre di **comparare** diverse versioni del codice:

- il Diff Viewer mostra due copie in due pannelli uno di fianco all'altro

- la copia più recente è quella mostrata a destra
- attraverso i colori spiegati in precedenza vengono mostrate le differenze tra le versioni

```

Main.java [ Diff ] ×
File Status Location
Main.java Locally Modified /JavaApplication/src/javaApplication/Main.java
Base (BASE) 3/4 Locally Modified (Based On LOCAL)
package javaApplication;
/*
 * @author larry
 */
public class Main {
    // creates a new instance of Main()
    public Main() {
}

```

The screenshot shows a Java file named Main.java in a diff editor. The left pane is labeled 'Base (BASE)' and the right pane is labeled 'Locally Modified (Based On LOCAL)'. Both panes show the same code. The diff interface uses color coding: blue for deleted text, red for added text, and green for modified text. In this specific view, the code appears identical, but the diff interface highlights differences in authorship ('@author larry' vs '@author peter') and line numbers (14 vs 15).

Info aggiuntive: <https://netbeans.apache.org/tutorial/main/kb/docs/ide/git/>

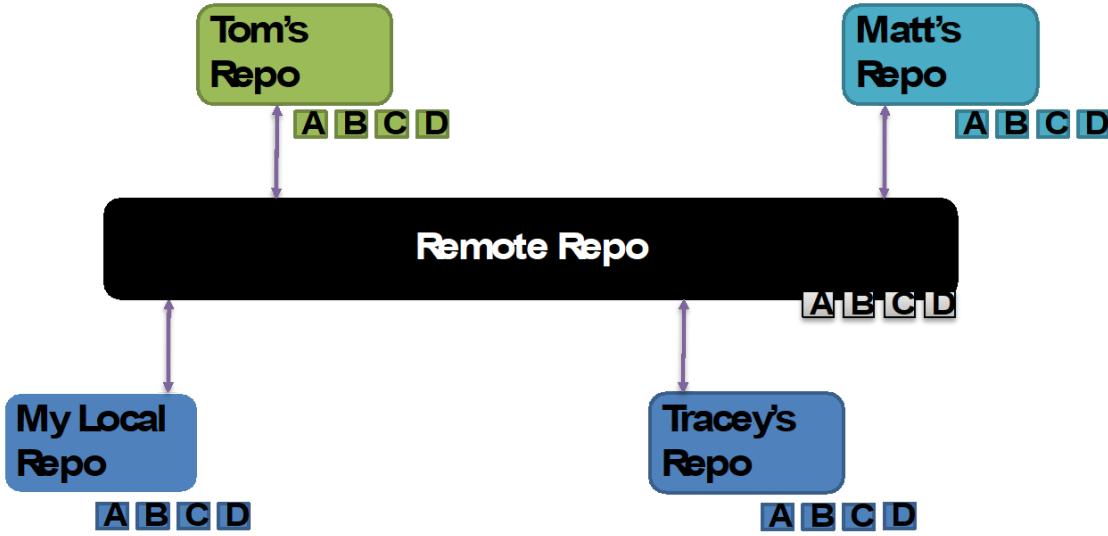
Remotes



Una **Remote** è una repository ospitata da un **server**, questa è identificata da un **URL** e permette lo **sviluppo collaborativo** e il **code sharing**.

Una Remote può essere ospitata da piattaforme cloud come GitHub, GitLab o Bitbucket.

Git ha dei comandi per sincronizzare dei branch nella repository locale con la Remote.



Per impostare una remote si può:

- aggiungere una remote ad una repo locale esistente

```

git remote add origin https://github.com/coolproject
#origin è il nome dato alla remote (origin è una convenzione ma si può scegliere
#una repository locale può anche essere connessa a diverse remote

```

- creare una copia di una repository a partire da una remote

```

git clone https://github.com/coolproject
#comando utile da usare se non si possiede già una copia locale della repository

```

I comandi principali relativi alle remote sono:

```

git fetch origin
#recupera i cambiamenti da una repository remota senza effettuare la merge

```

```

git pull origin <branch name>
#recupera i cambiamenti da una repository remota e ne fa la merge con il branch
#specificato
#pull = fetch + merge

```

```

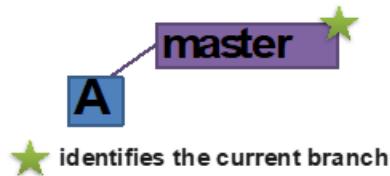
git push origin <branch name>
#si fa la push delle commits locali verso una repository remota in modo da aggiornare
#un branch remoto

```

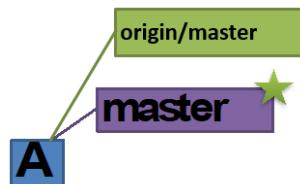
```
#si nota che Git rifiuterà la push se in remoto sono presenti cambiamenti più recenti  
#è una buona pratica fare prima la Pull e poi la Push
```

Remotes Illustrate

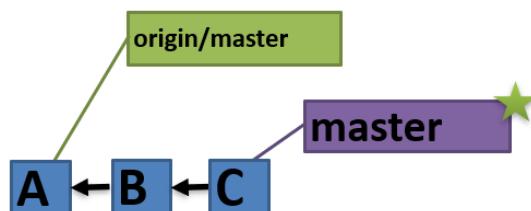
Si parte con la propria repository locale



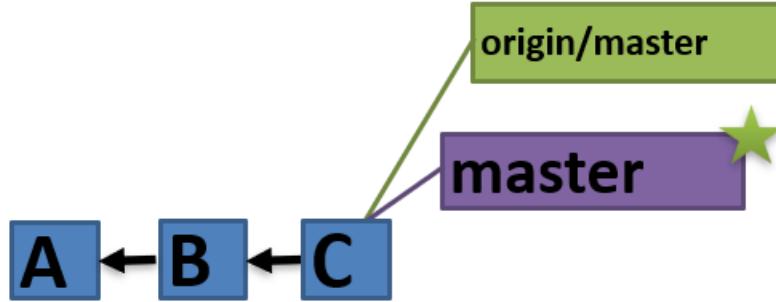
```
> git remote add origin https://github.com/coolproject  
//aggiungiamo una remote
```



```
> git commit  
> ...  
> git commit  
//facciamo la commit di dei cambiamenti alla repository locale
```



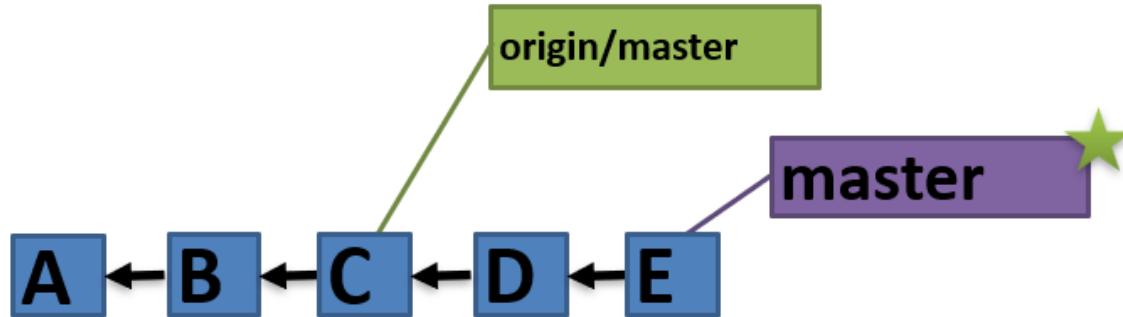
```
> git push origin  
//mandiamo le modifiche alla remote
```



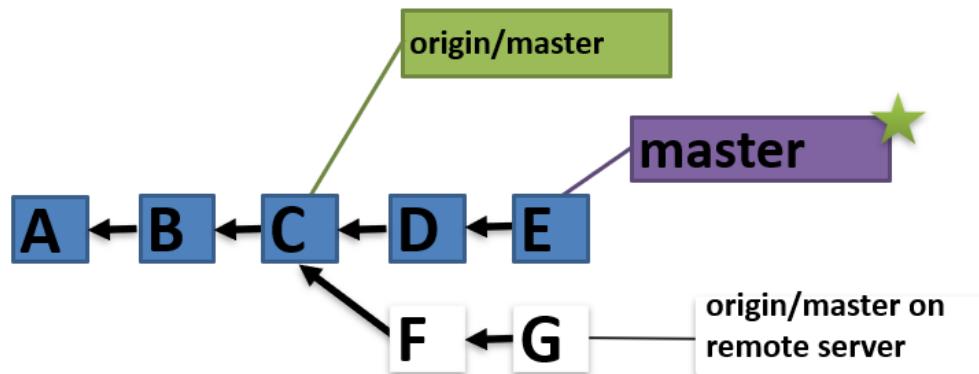
```

> git commit
> ...
> git commit
//facciamo la commit di dei cambiamenti alla repository locale (di nuovo)

```



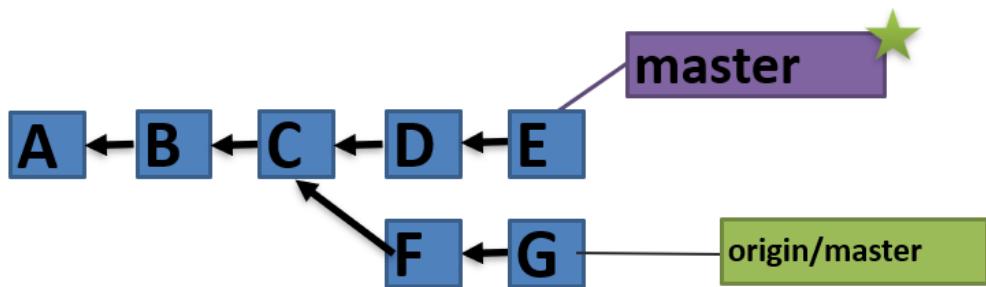
Nel frattempo qualcun altro aggiunge nuovi nodi al master branch remoto, questi cambiamenti in questa fase per noi **non** sono visibili



```

> git fetch origin
//effettuiamo la fetch delle modifiche presenti in remoto

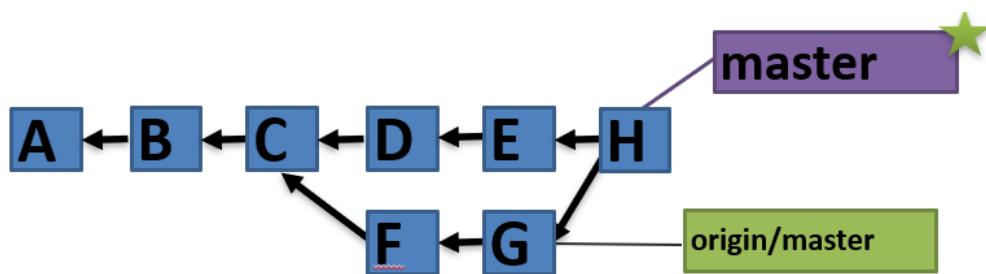
```



```

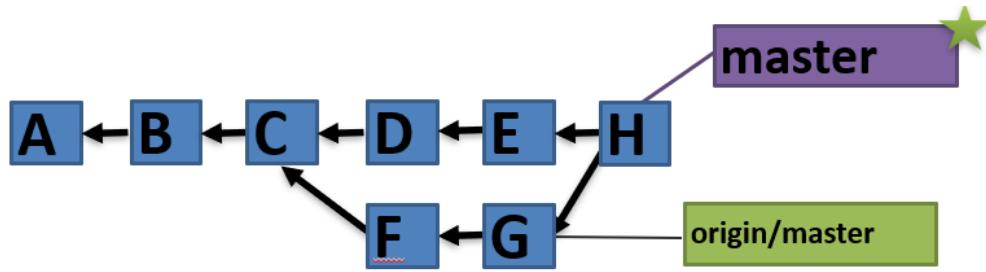
> git merge origin/master
//effettuiamo la merge delle modifiche remote nel nostro branch master
//in caso di conflitto sarà necessario fare conflict management

```



In alternativa alle ultime due operazioni avremmo potuto effettuare la pull ricordando che pull = fetch + merge. Saremmo arrivati allo stesso risultato.

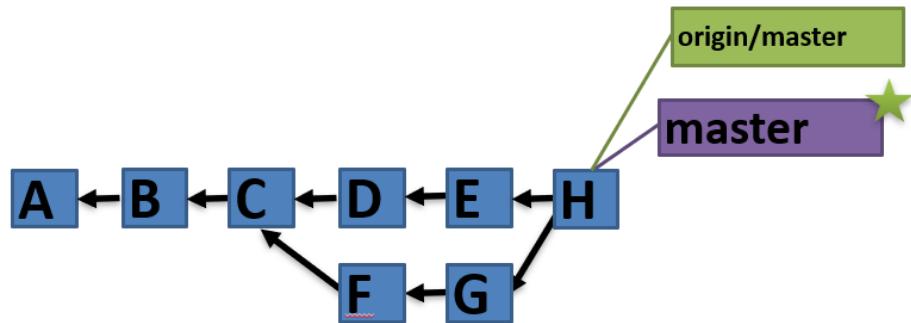
```
> git pull origin
```



```

> git push origin
//ora possiamo aggiornare la versione remota con quella locale fusa (vale a dire

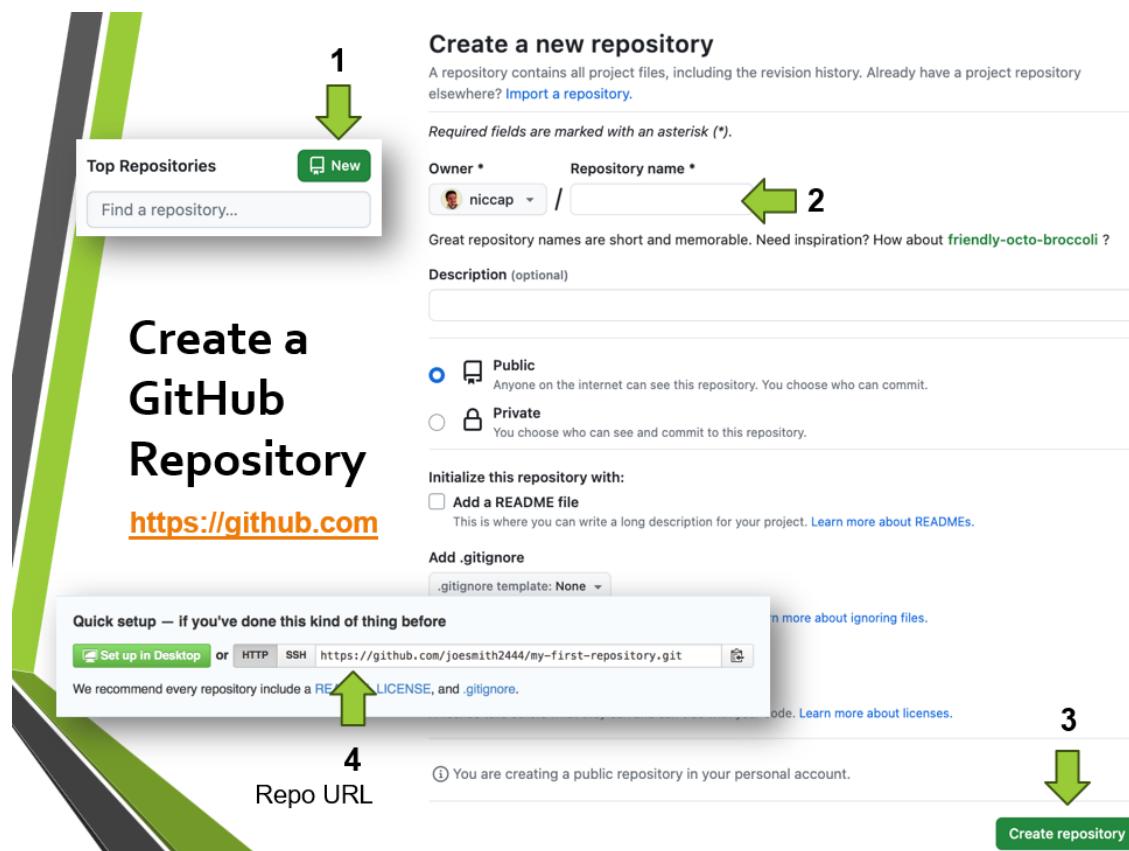
```



Tips per l'uso delle Remote

- mai fare la commit di codice che non compila
- mai fare la commit di codice che non passa tutti i test
- evitare di aggiungere file non sorgente a meno che si è sicuri che non verranno cambiati
- assicurarsi che tutti i file necessari per la compilazione siano nella commit
- evitare ogni tipo di file non necessario
- effettuare Pull/Push spesso (almeno quotidianamente, di più è meglio)

Git, GitHub e NetBeans



Generate a GitHub Access Token

<https://github.com/settings/tokens>

- 3 Copy the generated token



New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

1

What's this token for?

Expiration *

No expiration The token will never expire!

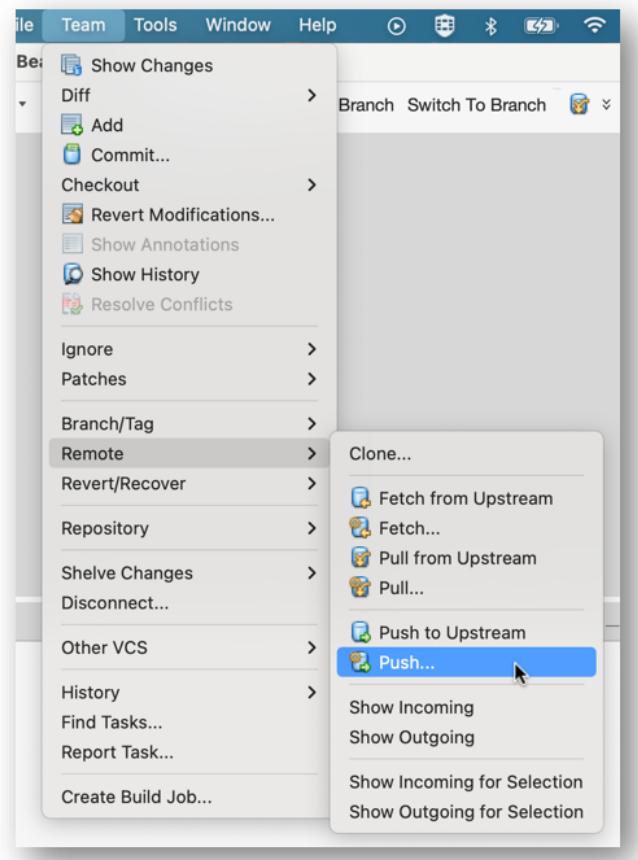
GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. [Learn more](#)

Select scopes

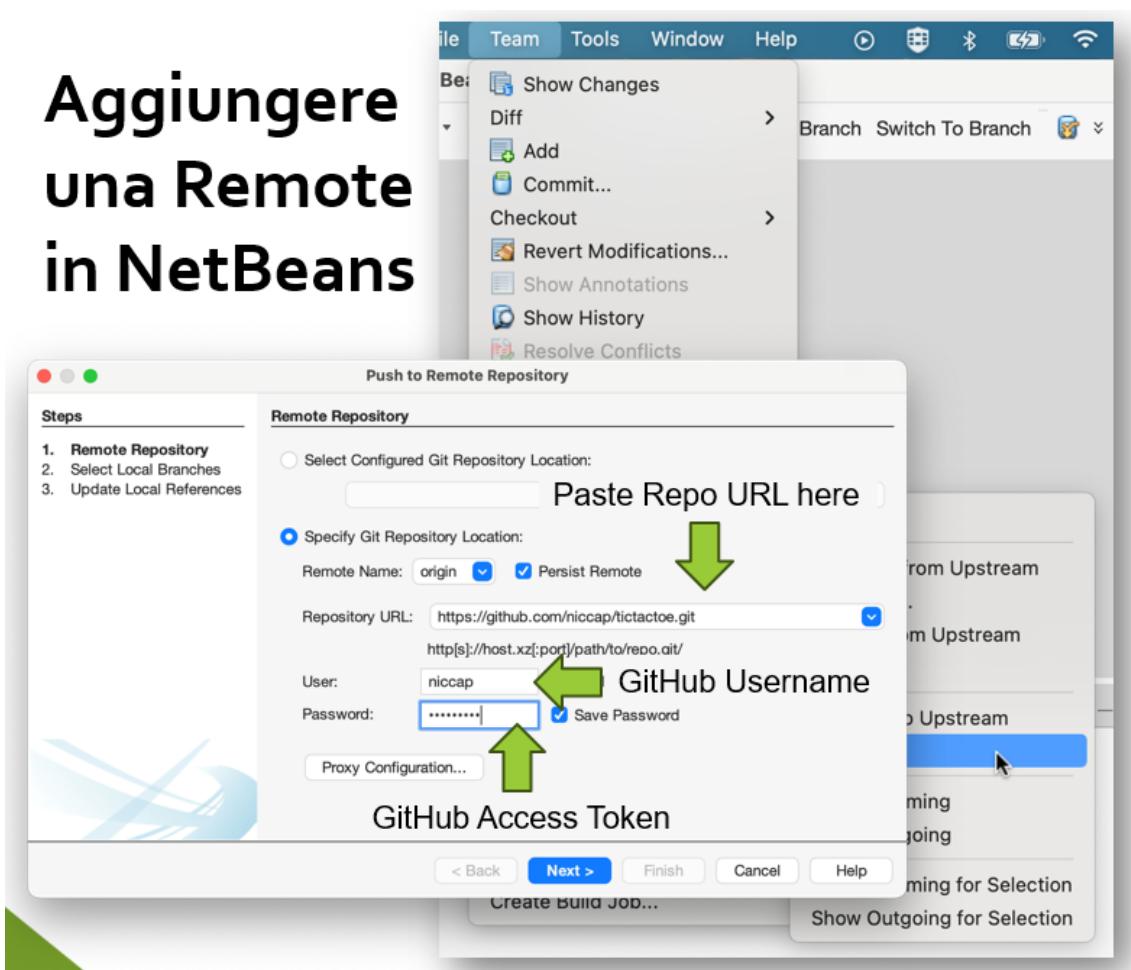
Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

2 repo Full control of private repositories
 repo:status Access commit status
 repo_deployment Access deployment status
 public_repo Access public repositories
 repo:invite Access repository invitations
 security_events Read and write security events

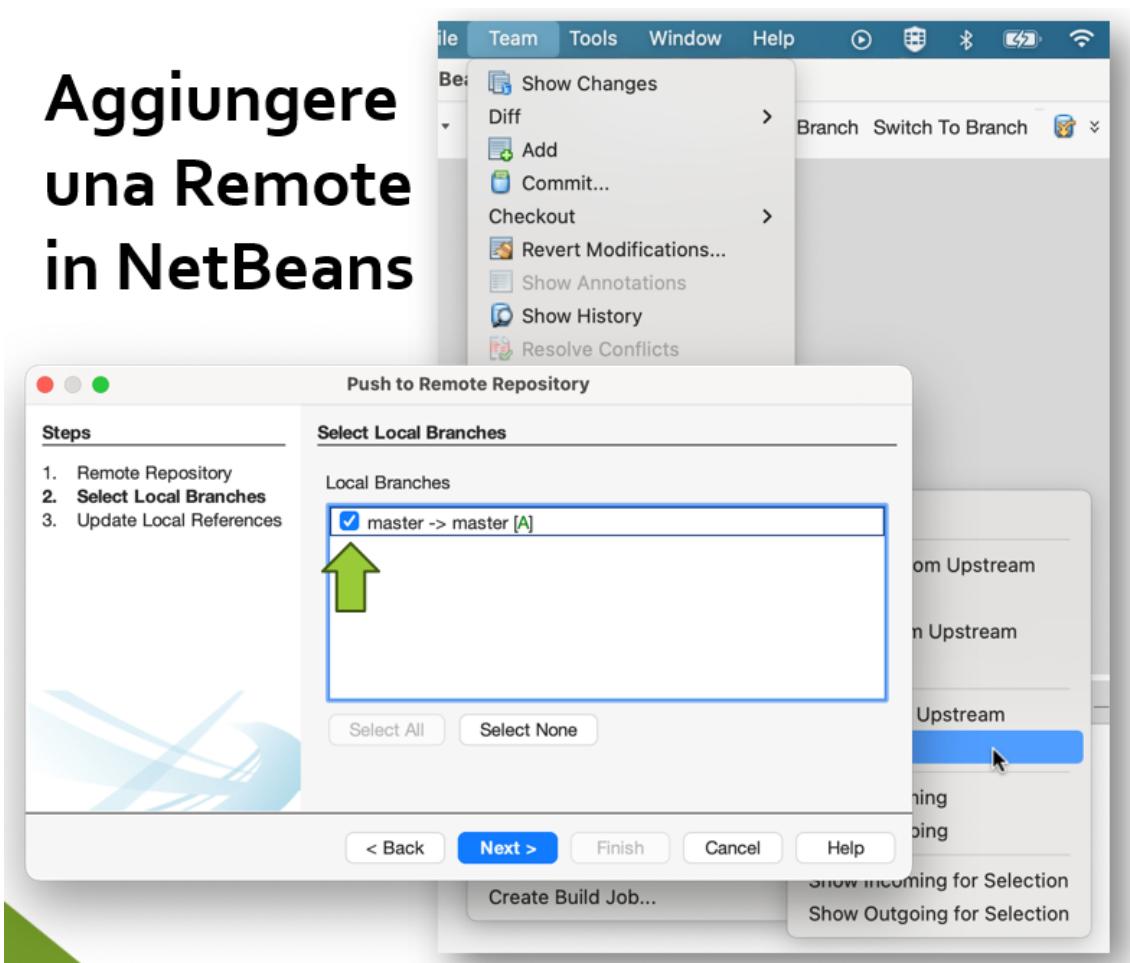
Aggiungere una Remote in NetBeans



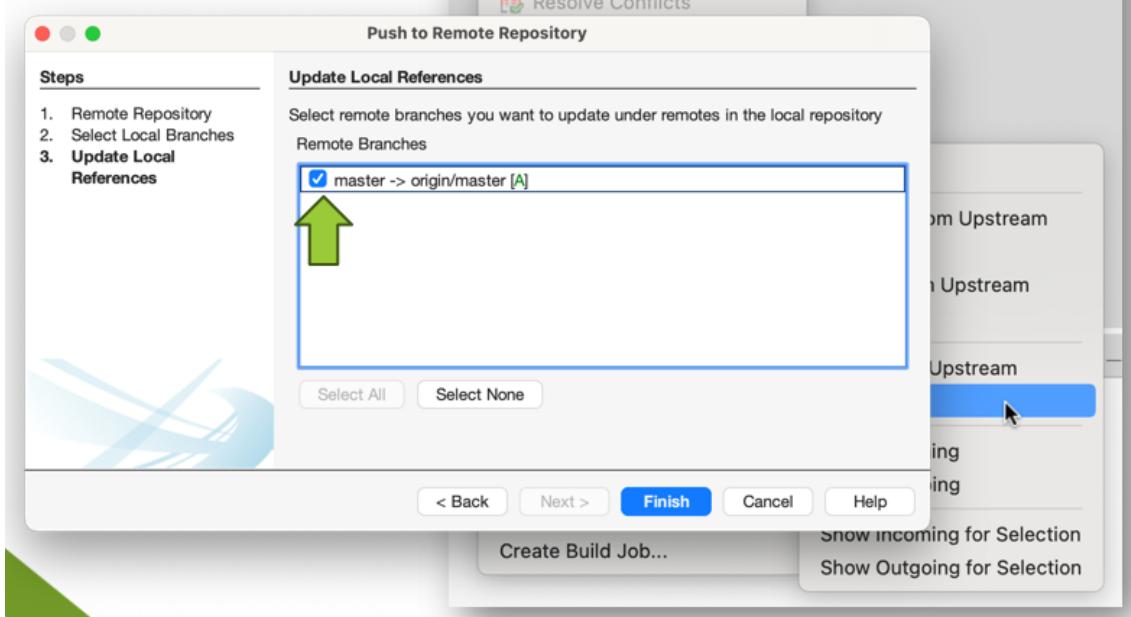
Aggiungere una Remote in NetBeans



Aggiungere una Remote in NetBeans



Aggiungere una Remote in NetBeans



Risorse aggiuntive

Video tutorial sulla configurazione di una Remote GitHub in NetBeans:

<https://www.youtube.com/watch?v=UOIPS-ewFHg>

Documentazione Git:

<https://git-scm.com/doc>

Libro gratuito su Git:

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

DISCLAIMER

Questi appunti sono stati realizzati a scopo puramente educativo e di condivisione della conoscenza. Non hanno alcun fine commerciale e non intendono violare alcun diritto d'autore o di proprietà intellettuale.

I contenuti di questo documento sono una rielaborazione personale di lezioni universitarie, materiali di studio e concetti appresi, espressi in modo originale ove possibile. Tuttavia, potrebbero includere riferimenti a fonti esterne, concetti accademici o traduzioni di materiale didattico fornito dai docenti o presente in libri di testo.

Se ritieni che questo documento contenga materiale di tua proprietà intellettuale e desideri richiederne la modifica o la rimozione, ti invito a contattarmi. Sarò disponibile a risolvere la questione nel minor tempo possibile.

In quanto autore di questi appunti non posso garantire l'accuratezza, la completezza o l'aggiornamento dei contenuti e non mi assumo alcuna responsabilità per eventuali errori, omissioni o danni derivanti dall'uso di queste informazioni. L'uso di questo materiale è a totale discrezione e responsabilità dell'utente.