

# GIT

## Java Foundation

Presented by

*Valerio Cammarota*

IBM Client Innovation Center - Italy

**14/11/2023**

IBM Client Innovation Center  
**Italy**

# Agenda

- 1 VCS (Version Control System)
- 2 GIT
- 3 Installazione Git e comandi base
- 4 Esercizi

- 5 GIT repository in Eclipse
- 6 Synchronize with repository
- 7 Esercitazione

# Version Control System

I sistemi di controllo della versione sono strumenti **che aiutano i team a gestire le modifiche** al codice sorgente nel tempo.

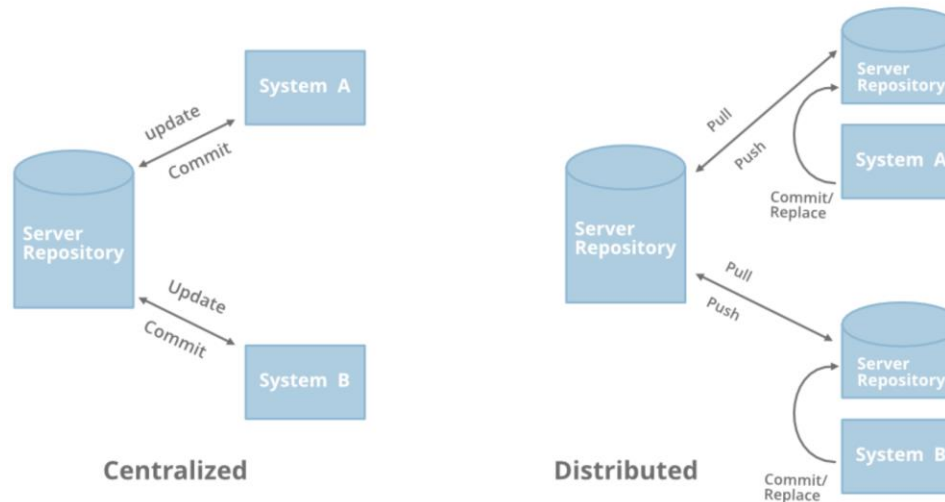
I sistemi di controllo della versione portano numerosi vantaggi e sono strumenti indispensabili non solo per qualsiasi progetto che includa un team numeroso, ma anche per i progetti che hanno un unico manutentore:

- ❖ **Monitoraggio e tracciamento** di tutte le modifiche apportate al codice;
- ❖ Possibilità di **annullamento e rollback** modifiche per recuperare una versione precedente;
- ❖ Gestione di **sviluppo in rami paralleli** e monitoraggio dei conflitti;

# Version Control System

I sistemi di controllo della versione possono essere centralizzati o distribuiti.

- ❖ **Centralizzato:** quando uno sviluppatore modifica il codice sorgente nella sua macchina locale, tali modifiche vengono salvate nel server centrale;
- ❖ **Distribuito:** le versioni possono essere salvate nel repository remoto e nei repository locali delle macchine locali.



In questo corso lavoreremo con Git (**sistema di controllo distribuito**).

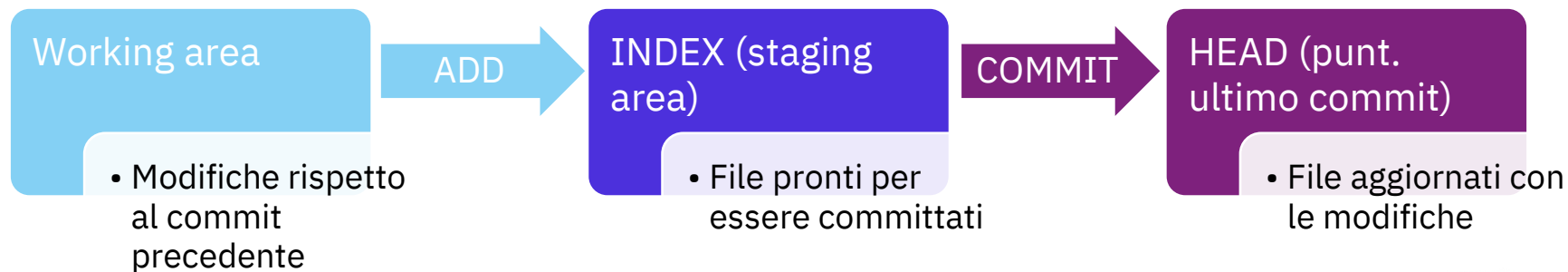
# GIT

## Controllo di versione distribuita

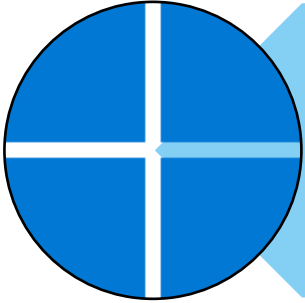
Nei sistemi di controllo distribuiti (o decentralizzati) ogni sviluppatore, scarica una copia del **repository centrale** sul proprio computer, questa copia prende in nome di **repository locale**.



Ogni copia rappresenta un **back-up completo** del repository.



# Installazione e comandi base



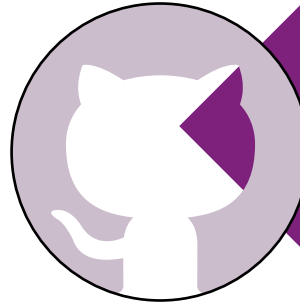
## WINDOWS

- <https://git-scm.com/download/win>
- Selezionare Standalone Installer 64bit



## macOS

- <https://git-scm.com/download/mac>
- Eseguire il comando: *brew install git*
- Se il comando non fosse disponibile seguire la guida: <https://brew.sh/>

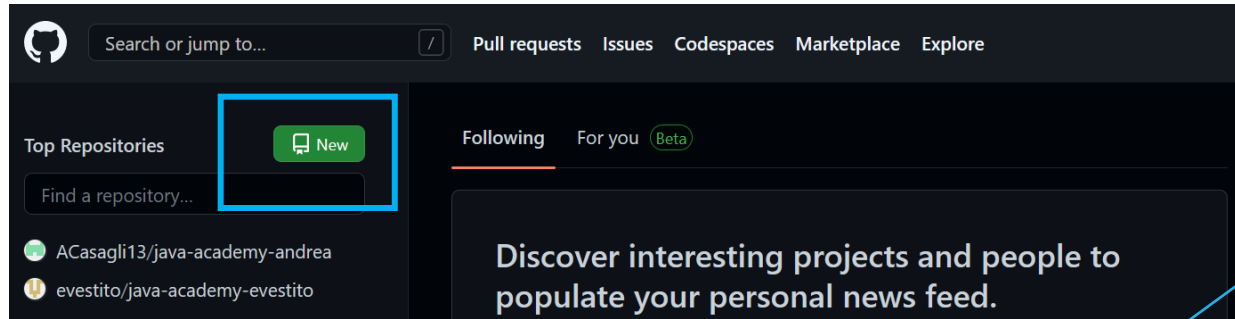


**GitHub:** è uno strumento web di version control Git

- Creazione account gratuito
- <https://github.com/>
- Download <https://desktop.github.com/>

# Installazione e comandi base: Repository

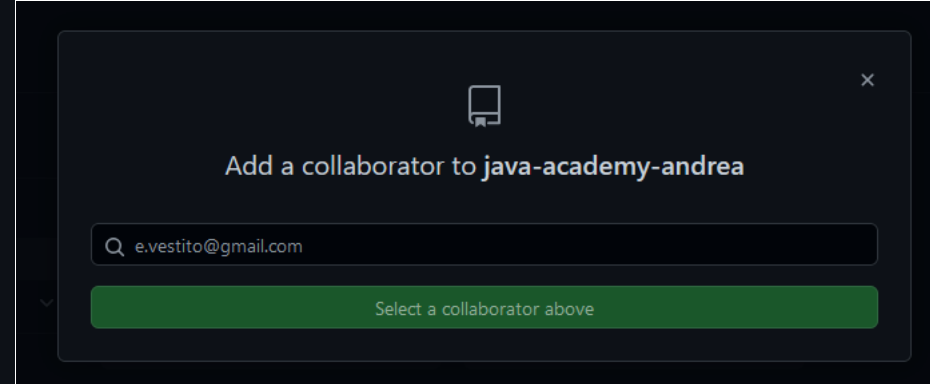
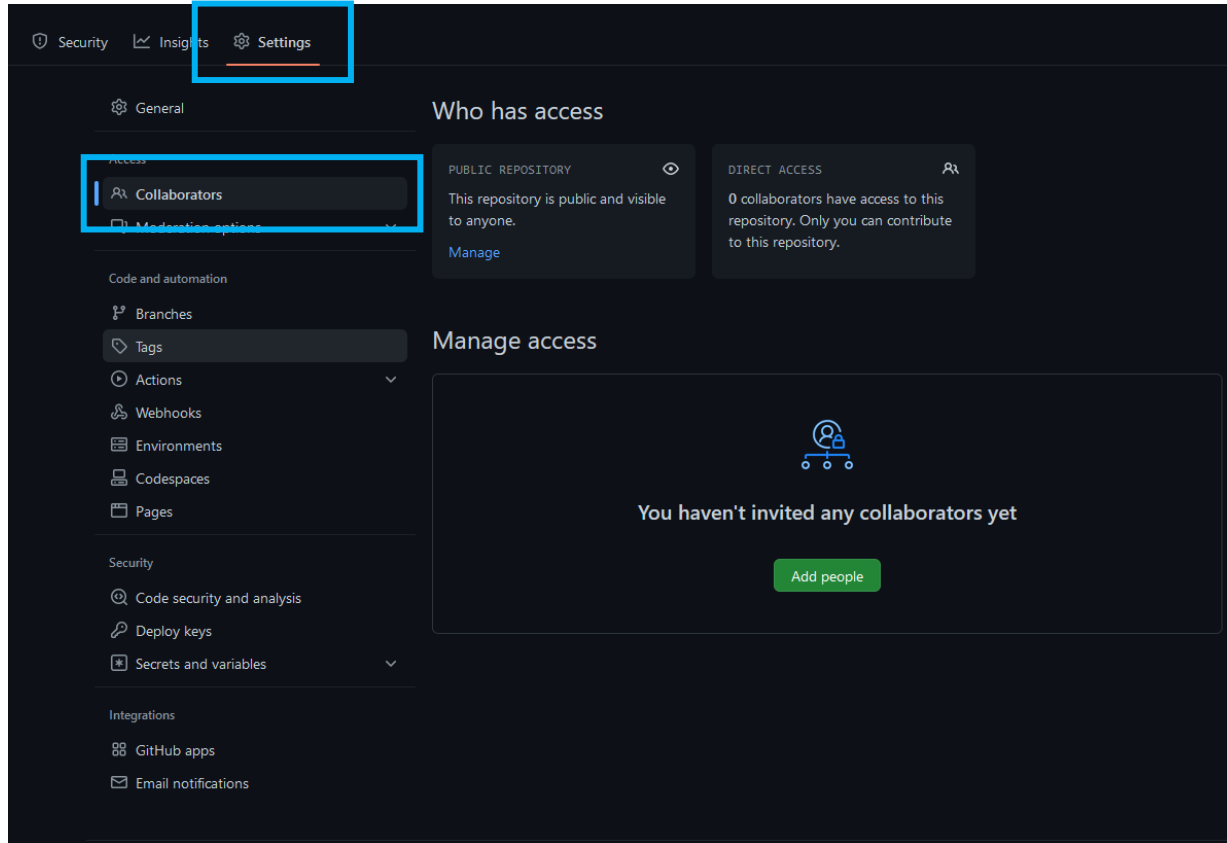
Creamo un nuovo repository:

A screenshot of the 'Create a new repository' form on GitHub. Several fields are highlighted with red rectangular boxes, and blue arrows point from the text on the left to these fields. The highlighted fields are: 'Repository name' (containing 'java-academy-andrea'), 'Public' (selected under 'Initialize this repository with'), 'Add a README file' (checked checkbox), and the 'Description' field (containing 'Great repository name...'). The form also includes fields for 'Owner' (ACasagli13), 'Description' (optional), 'Public/Private' selection, 'Initialize this repository with' section, 'Add a README file' checkbox, 'Choose which files not to track from a list of templates', 'Choose a license', and 'This will set main as the default branch'. A green 'Create repository' button is at the bottom.

- ❖ Il nome del repository deve essere univoco;
- ❖ Possiamo lasciare l'impostazione Public di default;
- ❖ Spuntare la checkbox per creare un file README. All'interno del file README dovremmo inserire una descrizione corposa del Progetto;

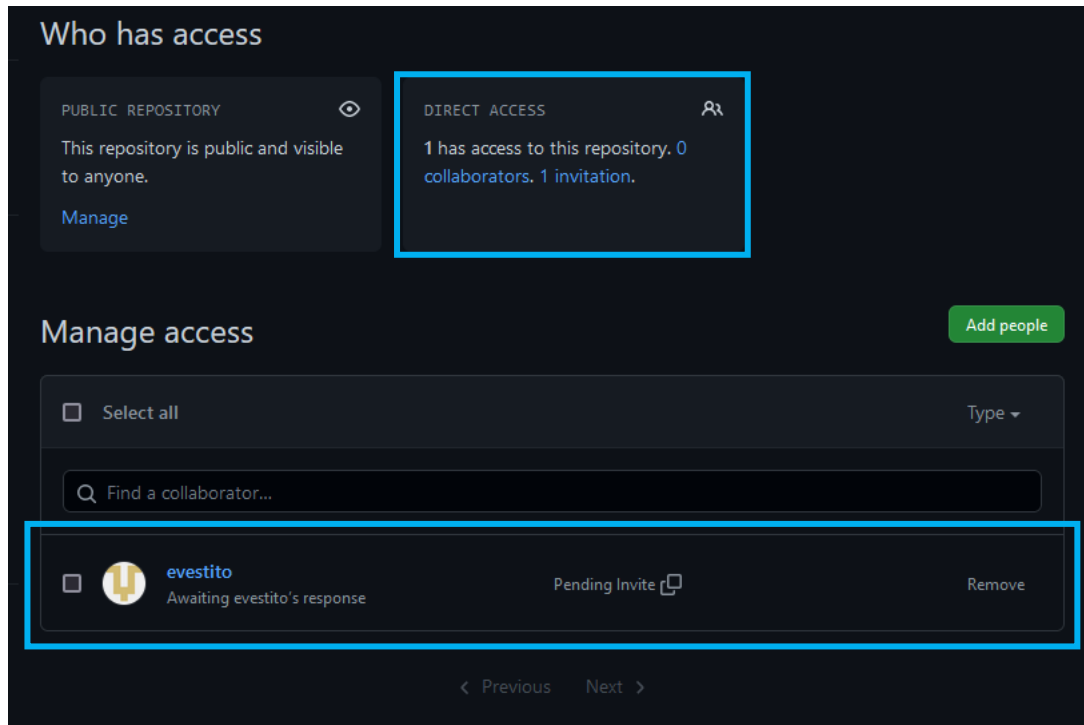
# Installazione e comandi base: Collaborators

Aggiungiamo un utente al nostro repository:





# Installazione e comandi base : Collaborators



**Who has access**

**PUBLIC REPOSITORY**

This repository is public and visible to anyone.

[Manage](#)

**DIRECT ACCESS**

1 has access to this repository. 0 collaborators. 1 invitation.

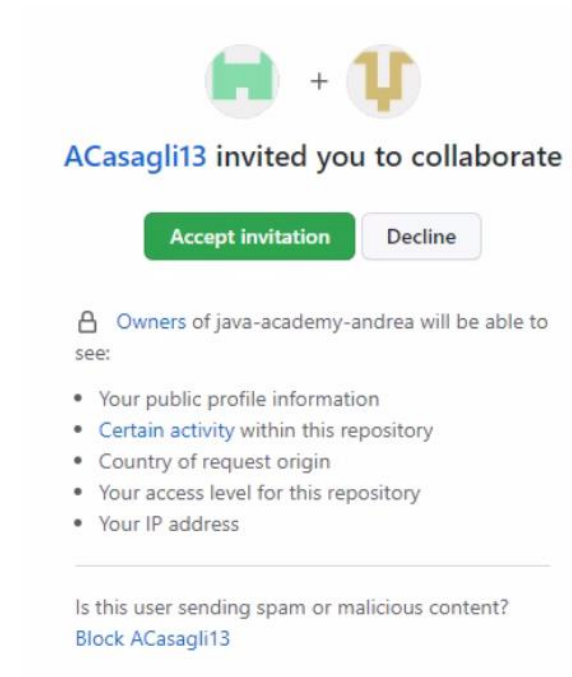
**Manage access** [Add people](#)

☐ Select all Type ▾

☐ **evestito** Pending Invite [Remove](#)

Awaiting evestito's response

[< Previous](#) [Next >](#)



+

**ACasagli13** invited you to collaborate

[Accept invitation](#) [Decline](#)

Owners of java-academy-andrea will be able to see:

- Your public profile information
- [Certain activity](#) within this repository
- Country of request origin
- Your access level for this repository
- Your IP address

Is this user sending spam or malicious content?  
[Block ACasagli13](#)

# Installazione e comandi base: Generazione Token

Per poter interagire con il repository server, è necessario generare un token di accesso personale.

Dal menu in alto a destra selezionare:

- ❖ Developer settings > Tokens (classic) > Generate new token (classic);
- ❖ Selezionare l'opzione **repo**;
- ❖ Fare click sul pulsante **Generate Token**.

# Installazione e comandi base: Git Bash

Apriamo il terminale '*Git Bash*' e configuriamo git con le **nostre credenziali di GitHub**:

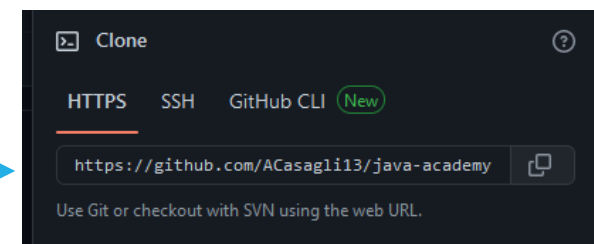
- ❖ `git config --global user.name '<Nome_GitHub>'`
- ❖ `git config --global user.email <Email_GitHub>`

```
GMX+065643758@IBM-PW03193V MINGW64 ~ (master)
$ git config --global user.name 'evestito'

GMX+065643758@IBM-PW03193V MINGW64 ~ (master)
$ git config --global user.email evestito@gmail.com
```

Ci sono due modi per istanziare un progetto

- Inizializziamo **un repository non esistente**:
  - ❖ `git init`
- Inizializziamo **un repository esistente** su un server git:
  - ❖ `git clone <serverURL>`



GIT

# Git branching

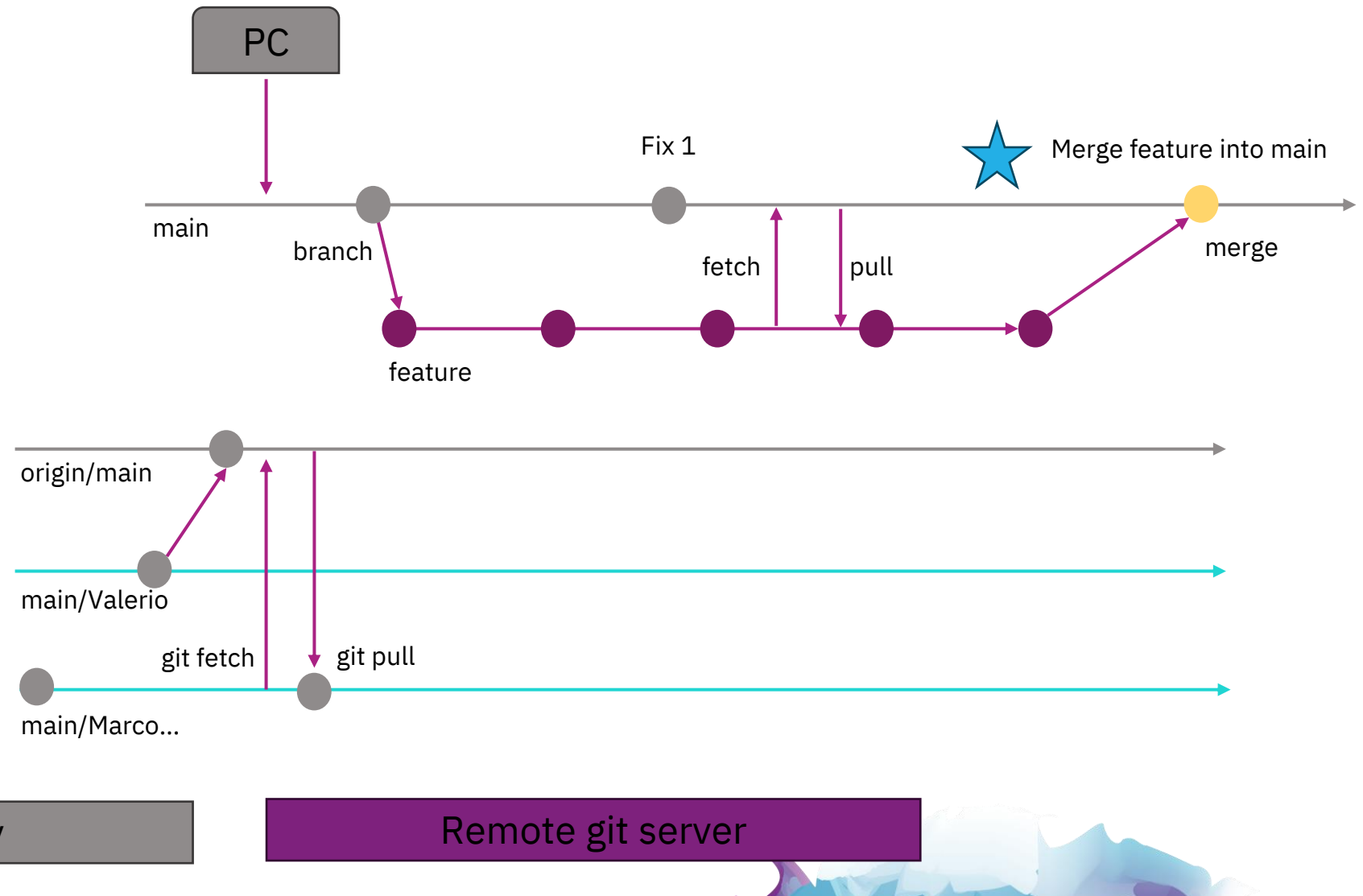
Gestione delle version software

<major>.<minor>.<rev>

Revision: 17.0.1

Minor version: 17.0.2

Major version: 18.0.0





# Installazione e comandi base : Git Bash

Posizioniamoci sulla cartella del repository appena scaricata:

Comando per mostrare i **commit in ordine cronologico inverso**:

❖ `git log`

**Hash:** identificatore univoco

**Repository:** remote e/o locale in cui è presente il commit

```
$ git log
commit b5683a1b3a81b24758e57d8af4ae5c8b896bfdee (HEAD -> main, origin/main, origin/HEAD)
Author: Valerio Cammarota <valerio.cammarota-cic@ibm.com>
Date: Tue Nov 14 09:34:49 2023 +0100

    slides are here :-)! Enjoy
```

**Informazioni:** autore, data e messaggio di commit

GIT

# Comandi base : git status

Consente di **visualizzare lo stato** della working area e staging area:

❖ `git status`

## NAME

git-status - Show the working tree status

## SYNOPSIS

```
git status [<options>] [--] [<pathspec>...]
```

# Comandi base : git fetch

Consente di **consente lo stato** di sincronizzare il local repository con il remote repository:

❖ `git fetch`

## NAME

git-fetch - Download objects and refs from another repository

## SYNOPSIS

```
git fetch [<options>] [<repository> [<refspec>...]]  
git fetch [<options>] <group>  
git fetch --multiple [<options>] [(<repository> | <group>)...]  
git fetch --all [<options>]
```



# Comandi base : git pull

Consente di **aggiornare** il local repository prendendo tutte le modifiche sincronizzate con **git fetch**, dal il remote repository:

❖ `git pull`

## NAME

git-pull - Fetch from and integrate with another repository or a local branch

## SYNOPSIS

```
git pull [<options>] [<repository> [<refspec>...]]
```

# Comandi base : git add

Consente di **aggiungere** un file alla staging area:

❖ `git add <nome-file>`

## NAME

git-add - Add file contents to the index

## SYNOPSIS

```
git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive |  
-i] [--patch | -p]  
        [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update |  
-u]] [--sparse]  
        [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-  
missing] [--renormalize]  
        [--chmod=(+|-)x] [--pathspec-from-file=<file>] [--pathspec-file-  
nul]]  
        [--] [<pathspec>...]
```

GIT

# Cbase : git commit

Consente di aggiungere un file alla staging area:

❖ `git commit -m <commento>`

## NAME

git-commit - Record changes to the repository

## SYNOPSIS

```
git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
           [--dry-run] [(--c | -C | --squash) <commit> | --fixup
           [(amend|reword):]<commit>)]
           [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
           [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
           [--date=<date>] [--cleanup=<mode>] [--no-]status
           [-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]
           [--trailer <token>[(=|:)<value>)]... [-S[<keyid>]]
           [--] [<pathspec>...]
```

# Comandi base : git push

Consente di **rendere disponibile** la nostra modifica a tutti sul remote repository:

❖ git push

## NAME

git-push - Update remote refs along with associated objects

## SYNOPSIS

```
git push [--all | --branches | --mirror | --tags] [--follow-tags] [--atomic] [-n | --dry-run] [--receive-pack=<git-receive-pack>]
        [--repo=<repository>] [-f | --force] [-d | --delete] [--prune]
[-q | --quiet] [-v | --verbose]
        [-u | --set-upstream] [-o <string> | --push-option=<string>]
        [--[no-]signed|--signed=(true|false|if-asked)]
        [--force-with-lease[=<refname>[:<expect>]]] [--force-if-
includes]]
        [--no-verify] [<repository> [<refspec>...]]
```

# Comandi base : git diff

Per vedere **i cambiamenti dei singoli files (presenti nella working area)** rispetto alla staging area o al repository remoto digitiamo:

❖ `git diff`

Per vedere **i cambiamenti dei singoli files (presenti nella staging area)** rispetto al repository remoto digitiamo:

❖ `git diff --staged`

# Concetti base: branch

Il branch rappresenta una **linea indipendente di sviluppo**. Un repository può contenere più branch, che equivalgono a più versioni del repository.

Nuovi branch possono essere realizzati per lo **sviluppo di nuove funzionalità** o per lavorare in parallelo sullo stesso progetto **senza impatti sul ramo principale** (chiamato **main**).

Una volta creato un nuovo branch tutti i commit fatti da quel momento in poi saranno fatti su questo nuovo branch e le modifiche saranno riportate sul ramo principale solo nel momento in cui verrà fatta un'azione di «merge».

Nell'esempio a fianco:

- ❖ **Ramo ROSA:** branch principale → MAIN;
- ❖ **Ramo CELESTE:** primo branch staccato dal main;
- ❖ **Ramo GIALLO:** secondo branch staccato, è stato effettuato il merge sul MAIN dopo il completamento degli sviluppo;



# Comandi base: git branch

Di seguito alcuni comandi utili per gestire i branch git:

- ❖ `git branch`: visualizza la lista dei branch disponibili;
- ❖ `git branch <nome_branch>`: crea un nuovo branch;
- ❖ `git checkout <nome_branch>`: per puntare al branch specificato;
- ❖ `git push -set-upstream origin <nome_branch>`: mette a disposizione di tutti i membri del repository il nuovo branch.

I due comandi precedenti possono essere uniti con:

```
git checkout -b <nome_branch>
```

che crea il nuovo branch ed effettua automaticamente lo switch.

# Installazione e comandi base

Il comando si basa su uno stack, una struttura dati che segue il paradigma LIFO (Last In – First Out).

È utile per **archiviare momentaneamente** le modifiche di cui ancora non vogliamo fare commit:

❖ `git stash`

Per **visualizzare le modifiche** archiviate:

❖ `git stash list`

Se proviamo a fare `git status` non visualizzeremo modifiche da committare:

Per **ripristinare una modifica** archiviata:

❖ `git stash pop` oppure `git stash pop <index>`

Per **eliminare una modifica** archiviata:

❖ `git stash drop`



# GIT

## Installazione e comandi base

**Annullare** un commit o una push e **conservare le modifiche** apportate ai file:

- ❖ `git reset --soft <hash_da_ripristinare>`

Esempio:

- ❖ Modificare file, aggiungerlo all'area di staging e committarlo;
- ❖ Recuperare l'hash della commit da ripristinare;
- ❖ Lanciare il comando
  - `git reset --soft <hash_da_ripristinare>`
- ❖ Il contenuto del file **non perde** la modifica effettuata

**Annullare** un commit o una push e **NON conservare le modifiche** apportate ai file:

- ❖ `git reset --hard <hash_da_ripristinare>`

# Esercitazione: git bash

1. Effettuare il **checkout** del progetto dal seguente URL:

`https://us-south.git.cloud.ibm.com/Valerio.Cammarota-CIC/ibm_cic_java_academy_q4.git;`

2. Creare un nuovo **branch** “<nome-cognome” ed effettuare lo switch sul nuovo branch creato;
3. Copiare l'intero progetto **JavaAcademyFirstProject** all'interno della directory creata;
4. Eseguire l'**add** dei file del progetto al repository locale;
5. Eseguire la **commit** e la **push** delle modifiche;
6. Verificare che il repository locale e remoto siano allineati;
7. Verificare nei **log** il messaggio della commit effettuata.

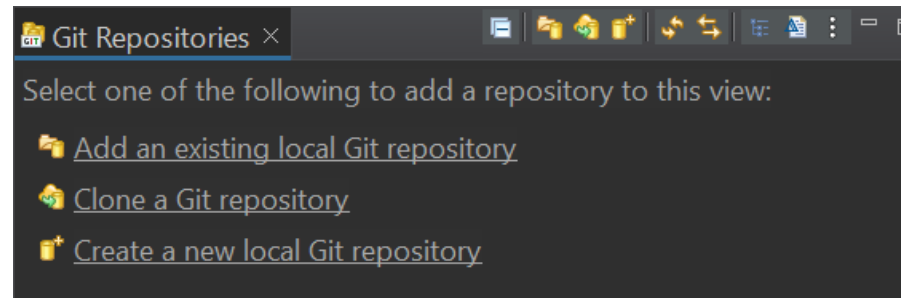
GIT

# GIT Repository in Eclipse

Aprire la perspective GIT Repositories:

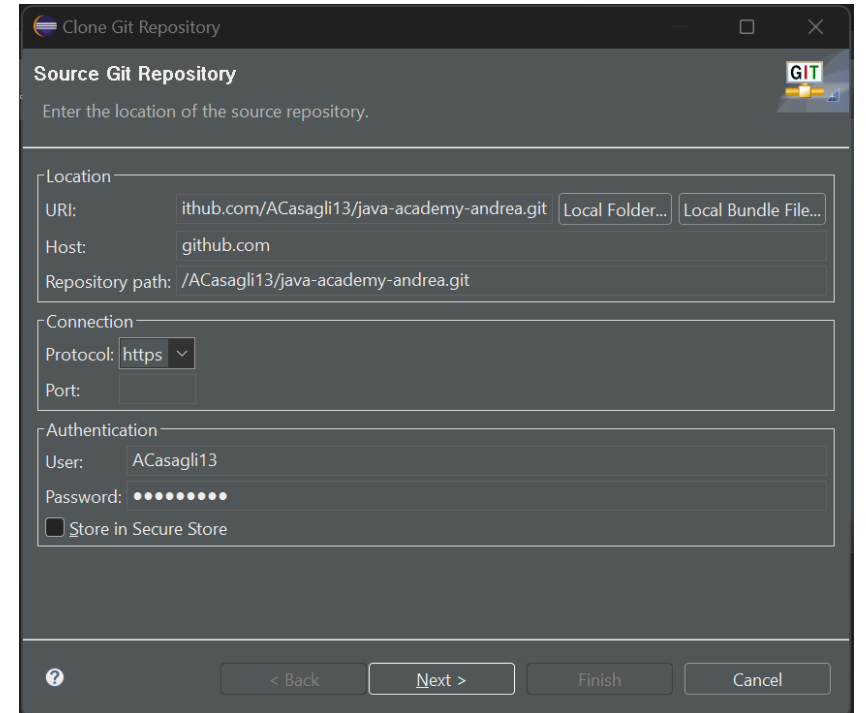
❖ Window > Perspective > Open Perspective > GIT

Selezionare la clonazione di un repository GIT:



# GIT Repository in Eclipse

- ❖ **URI:** riportare l'URL del repository GIT;
- ❖ **Host** e **repository path** saranno compilati in automatico;
- ❖ **User:** User dell'account Git-Hub;
- ❖ **Password:** Inserire il token generato da Git-Hub;
- ❖ Click su **Next** senza modifiche, fino a quando non si abilita il tasto **Finish**.

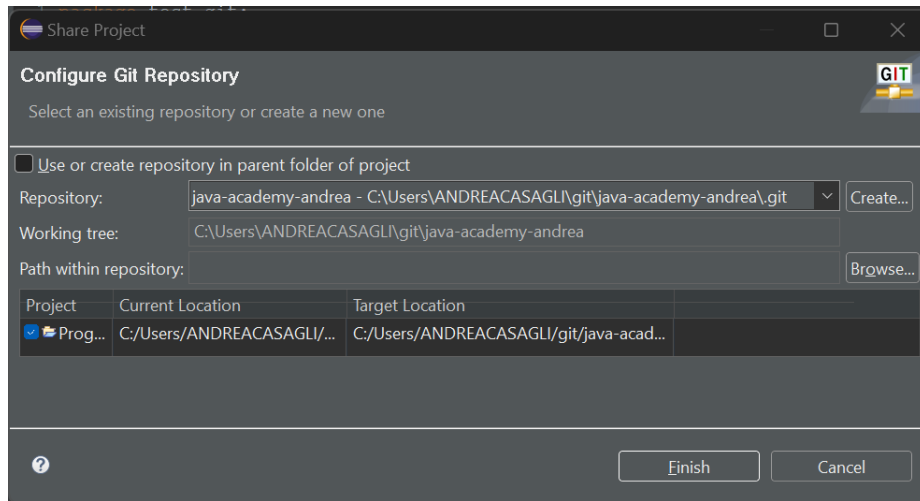


GIT

# GIT Repository in Eclipse

Una volta clonato il Repository creare un nuovo Progetto Java (se il workspace non lo contiene già) e condividerlo sul repository remoto di GIT.

❖ Tasto dx sul Progetto > Team > Share Project



Selezionare il repository appena clonato e cliccare Finish

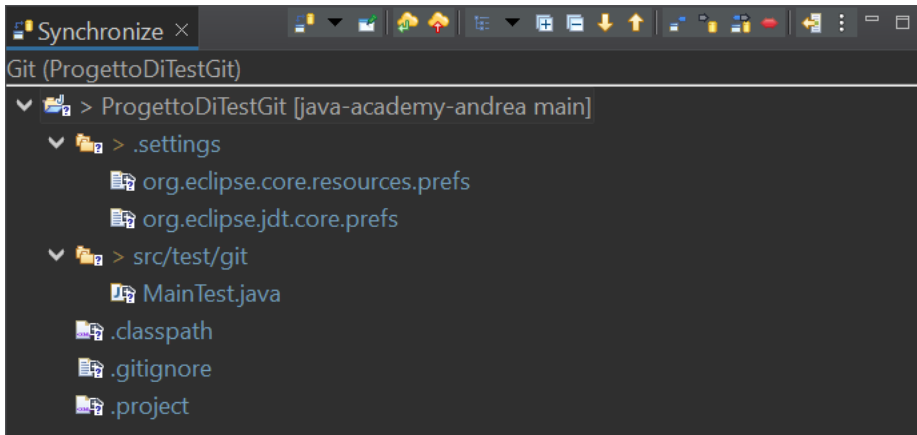
GIT

# GIT Repository in Eclipse

Di seguito i passi per committare il progetto sul repository locale e pushare sul repository remoto:

❖ Tast dx sul Progetto > Team > Synchronize Workspace

Da questa view è possibile vedere le modifiche in ingresso, in uscita e gli eventuali conflitti.



❖ Tasto dx sul Progetto > Commit...