

# ***Unità di apprendimento 1***

**Architettura di rete e  
metodologia di sviluppo**

# ***Unità di apprendimento 1***

## **Laboratorio 1**

**Il controllo delle versioni  
con Git**

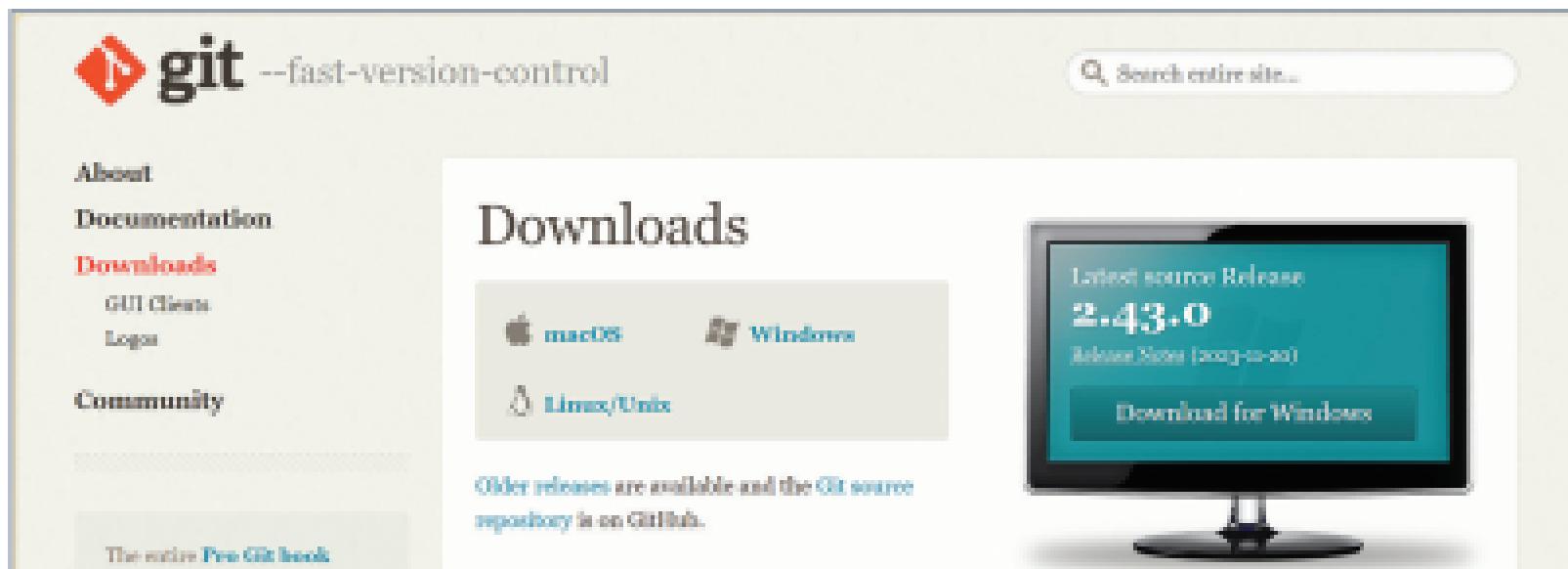
# Il controllo delle versioni con Git

---

- Le quattro “caratteristiche vincenti” di Git sono:
  - il **cambio di contesto senza conflitti**: si crea un ramo e dopo aver eseguito più commit si può **tornare** al punto da cui si è fatta la ramificazione e, se desiderato, si possono **unire** i due percorsi
  - le **linee di codice basate sui ruoli**: si può avere un ramo che contiene sempre solo ciò che va in produzione e altri rami per altri scopi
  - il **flusso di lavoro basato su funzionalità**: si possono creare nuovi rami per ogni nuova funzionalità su cui si sta lavorando eliminando ogni ramo quando la funzione in esame viene unita alla linea principale
  - la **sperimentazione usa e getta**: se si crea un ramo in cui sperimentare e ci si rende conto che non funzionerà, questo potrà essere cancellato senza che nessun altro lo veda mai

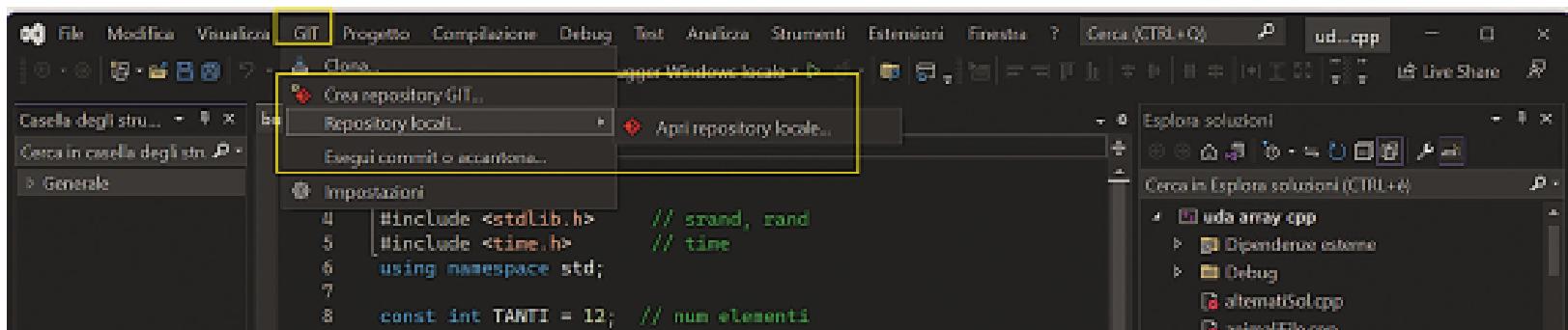
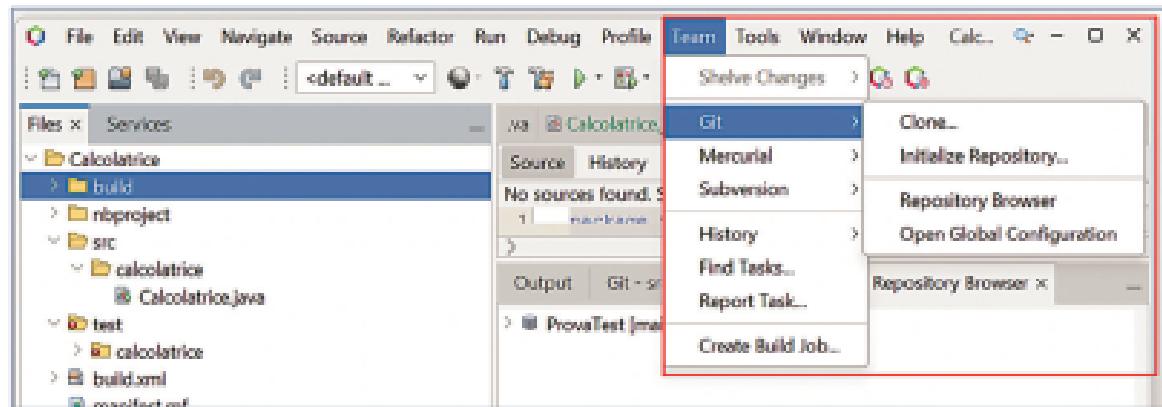
# Il controllo delle versioni con Git

- La versione ufficiale scaricabile gratuitamente è disponibile all'indirizzo <https://git-scm.com/downloads>



# Il controllo delle versioni con Git

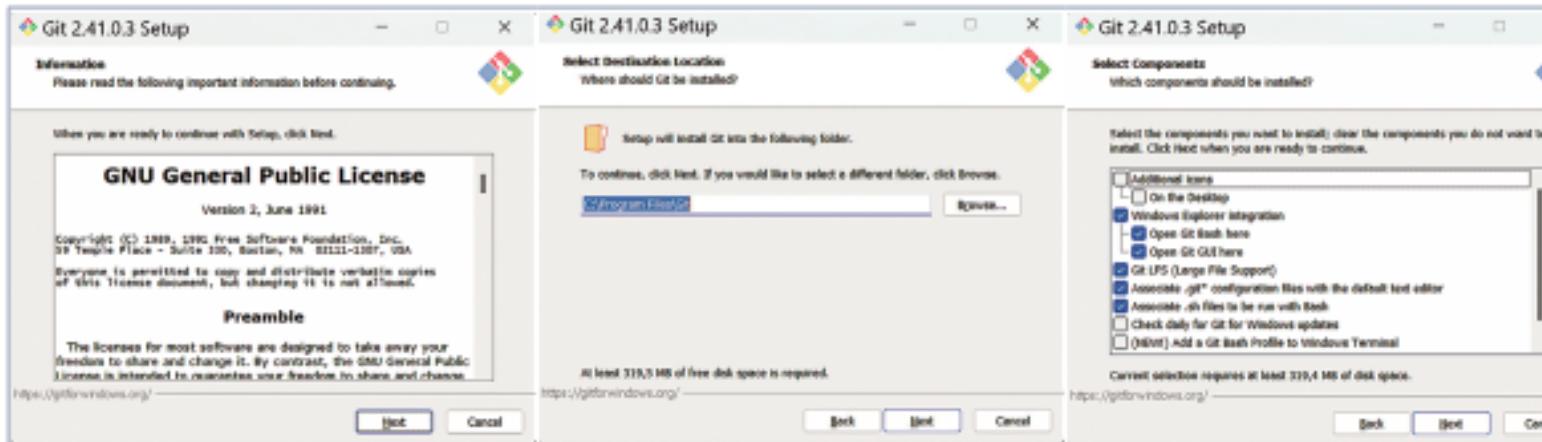
- È possibile utilizzare Git direttamente all'interno degli ambienti di sviluppo più utilizzati, come **NetBeans** oppure **VisualStudio**:



# Il controllo delle versioni con Git

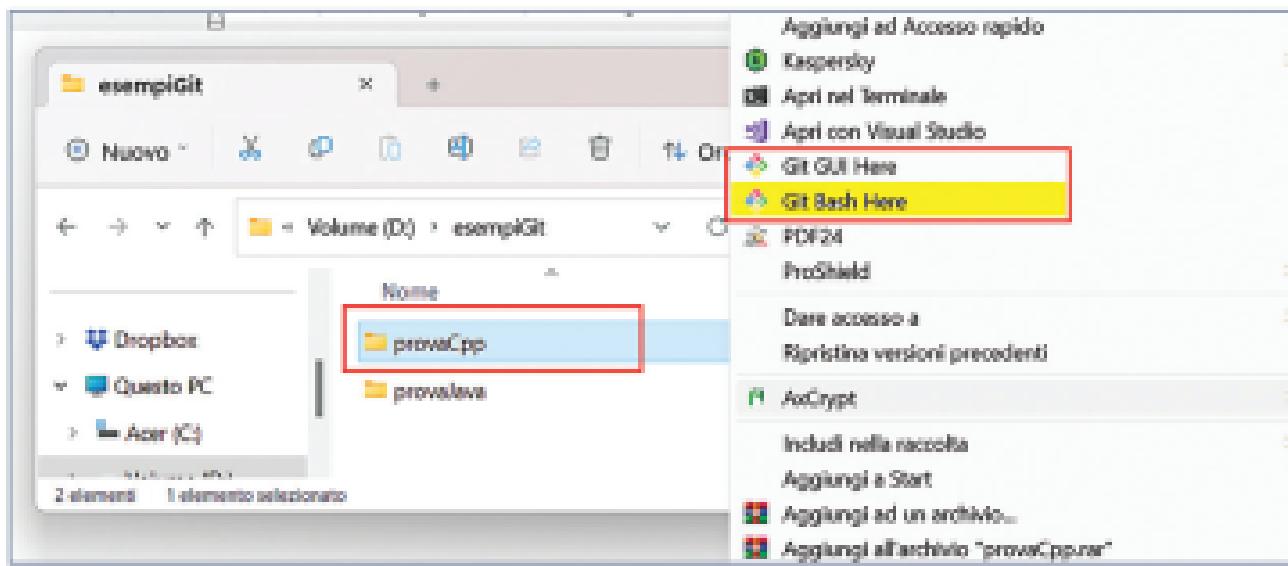
## Installiamo Git in locale

- Su una macchina **Linux** l'installazione di Git viene effettuata con i seguenti comandi:  
`sudo apt-get update`  
`sudo apt-get install git`
- Su una macchina **Windows**, al termine del download del file eseguibile si avvia automaticamente



# Salvare lo stato corrente

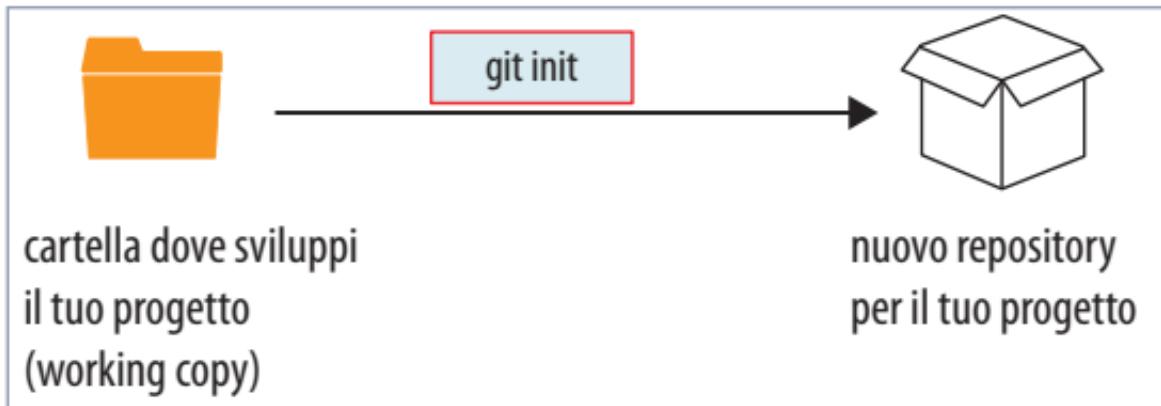
- Come prima operazione posizioniamoci nella directory di lavoro dove sono presenti i nostri file sorgente
- Il menù contestuale che ci appare cliccando con il tasto destro sulla cartella ci mostra due opzioni relative a Git che ci permettono di creare un **repository**



# Il controllo delle versioni con Git

- Selezionando l'opzione **Git Bash Here** ci apparirà la CLI (interfaccia a linea di comando) in cui potremo inizializzare l'ambiente con il comando **init**

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp  
$ git init
```



# Il controllo delle versioni con Git

---

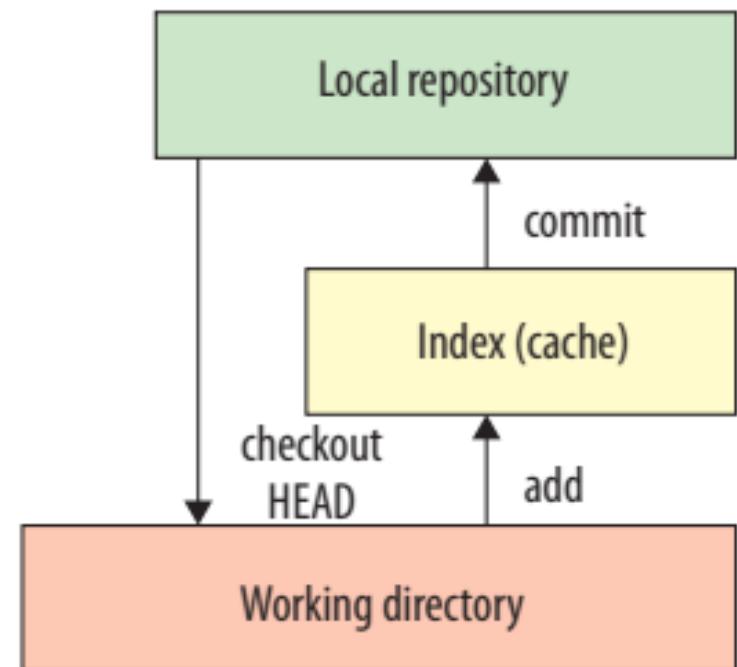
- La sua esecuzione crea una directory `.git` che sarà il repository gestito automaticamente da Git

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp
$ git init
Initialized empty Git repository in D:/esempiGit/provaCpp/.git/
```

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$
```

# Il controllo delle versioni con Git

- Il repository locale che abbiamo creato contiene i link ai “tre alberi” che verranno mantenuti da Git:
  - il primo è il **working directory** che contiene i file di progetto
  - il secondo è l'**index** che fa da spazio di transito per i file aggiunti con *add*
  - il terzo è l'**HEAD** che punta all’ultimo commit fatto

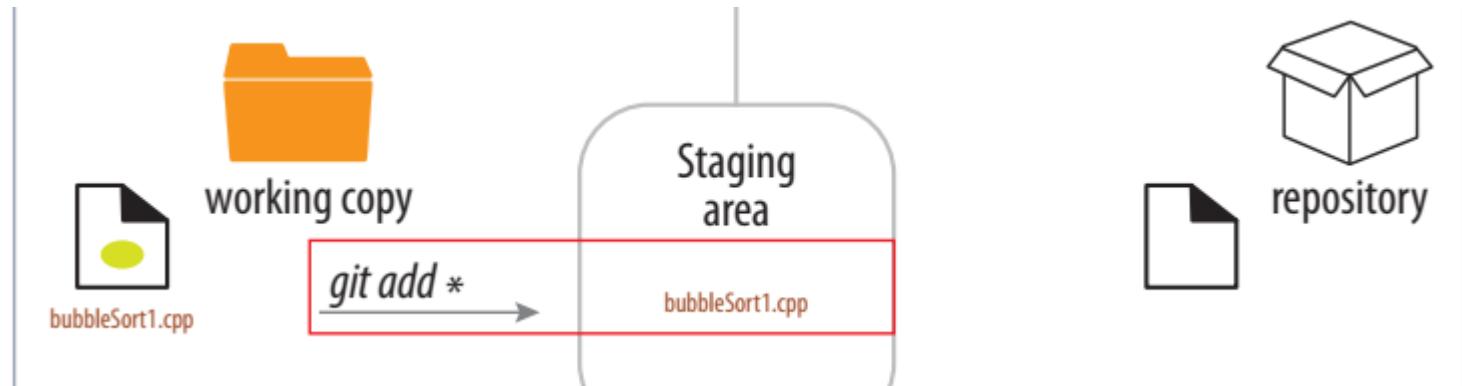


# Il controllo delle versioni con Git

- Aggiungiamo i file creando una copia del repository nella cache col comando **add**

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git add *
```

- Ora il nostro file è presente nella **staging area**



# Il controllo delle versioni con Git

---

- Per poter procedere con il salvataggio sul repository è necessario che Git conosca chi è ogni autore delle varie modifiche
- Per aggiungere le nostre va dunque fornito l'indirizzo email

```
utente@portable-paolo MINGW64 /c/esempiGit/provaCpp (master)
$ git config --global user.email "paolo.camagni@magis.edu.it"
```

# Il controllo delle versioni con Git

---

- Validiamo questa situazione con il comando `commit -m "<nome_commit>"`
- all'esecuzione del comando vengono visualizzati i file che sono inseriti in questo “salvataggio” nel repository locale

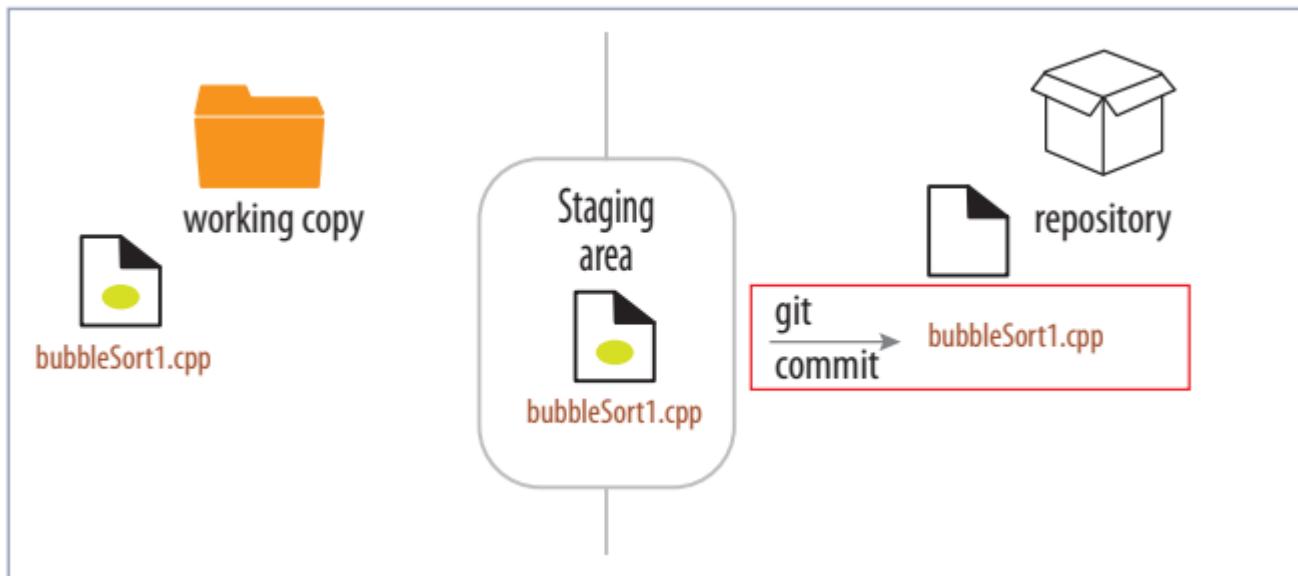
```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git commit -m "primo backup"
[master (root-commit) fb99374] primo backup
 1 file changed, 45 insertions(+)
 create mode 100644 bubbleSort1.cpp
```

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ |
```

# Il controllo delle versioni con Git

---

- Il nostro file viene ora duplicato nel repository:



# Il controllo delle versioni con Git

---

- Ogni volta che si esegue un **commit**, all'operazione viene assegnato un **identificatore** composto da quaranta caratteri esadecimali (che specificano un hash SHA-1 a 160 bit), dei quali sono visualizzati solo i primi 7:

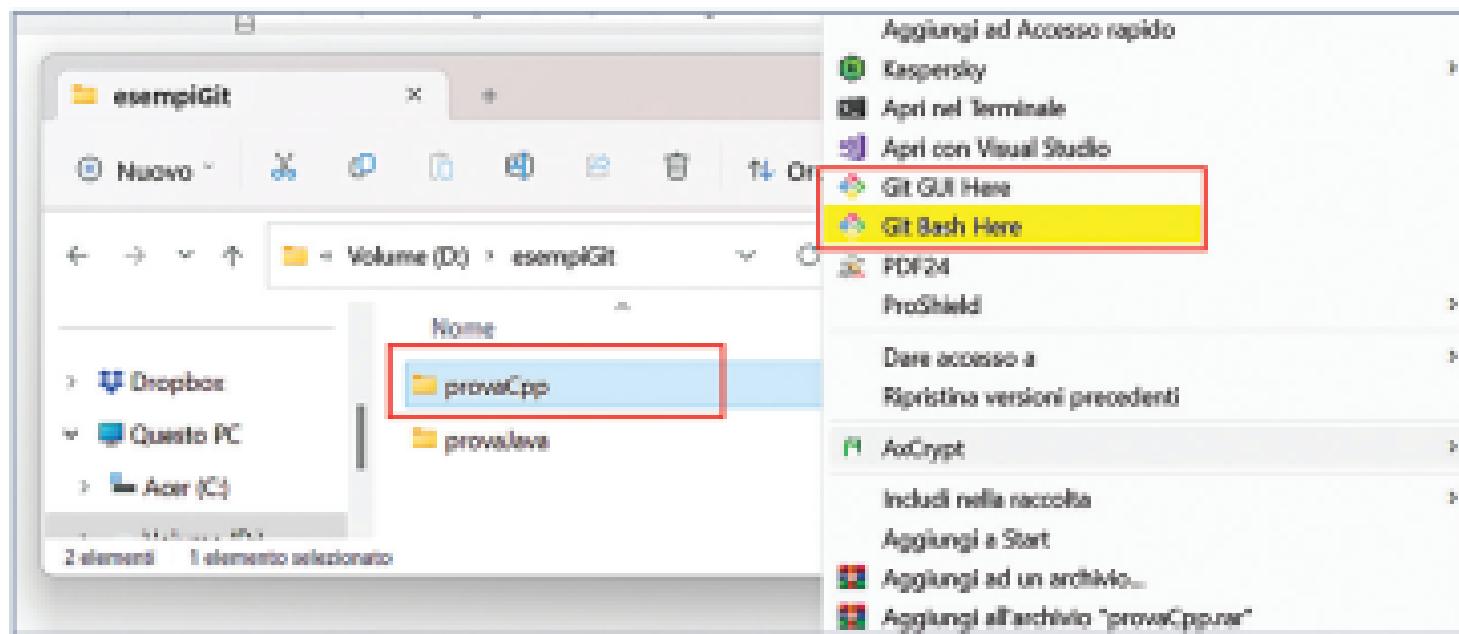
```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git commit -m "primo backup"
```

```
[master (root-commit) fb99374] primo backup
 1 file changed, 45 insertions(+)
 create mode 100644 bubbleSort1.cpp
```

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ |
```

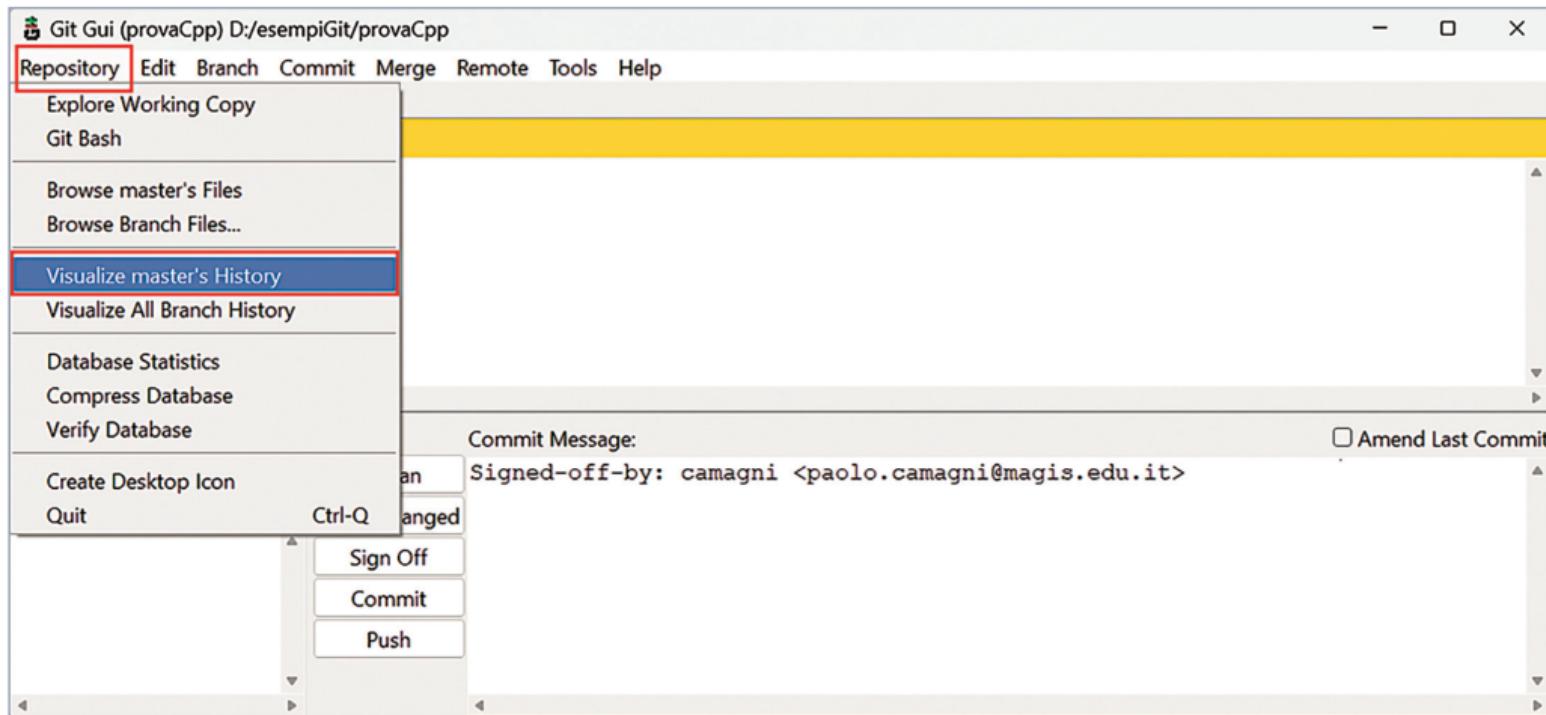
# Il controllo delle versioni con Git

- Proviamo ora a utilizzare l'**interfaccia GUI**, selezionando dalla tendina delle opzioni il corrispondente comando



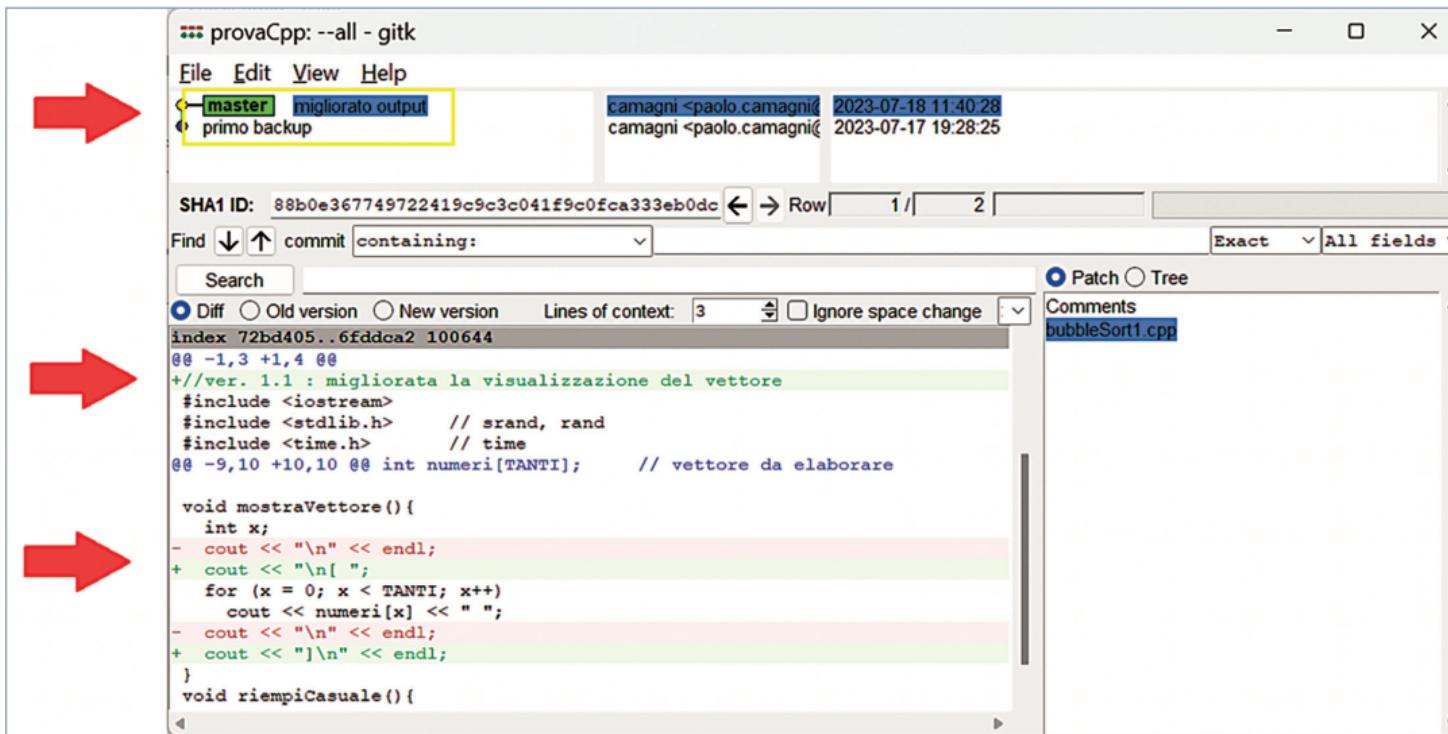
# Il controllo delle versioni con Git

- otteniamo la seguente schermata
- selezioniamo l'opzione evidenziata che ci mostra la storia del nostro progetto



# Il controllo delle versioni con Git

- Nella parte superiore possiamo visualizzare l'albero
- Nell'immagine seguente l'albero ha due commit



The screenshot shows the gitk graphical interface for a repository named 'provaCpp'. A red arrow points to the top-left where the 'master' branch is selected. Another red arrow points to the commit list on the right, which displays two commits:

- camagni <paolo.camagni@...> 2023-07-18 11:40:28  
camagni <paolo.camagni@...> 2023-07-17 19:28:25

A third red arrow points to the bottom-left, highlighting the diff view. The diff shows changes made to a file named 'bubbleSort1.cpp':

```
index 72bd405..6fddeca 100644
@@ -1,3 +1,4 @@
+//ver. 1.1 : migliorata la visualizzazione del vettore
 #include <iostream>
 #include <stdlib.h>      // srand, rand
 #include <time.h>         // time
@@ -9,10 +10,10 @@
 int numeri[TANTI];      // vettore da elaborare

 void mostraVettore() {
    int x;
-   cout << "\n" << endl;
+   cout << "\n[ ";
    for (x = 0; x < TANTI; x++)
        cout << numeri[x] << " ";
-   cout << "\n" << endl;
+   cout << "]\n" << endl;
}
```

# Il controllo delle versioni con Git

- Nella parte inferiore vengono invece visualizzate le modifiche che sono state apportate tra un commit e un altro
- in rosso sono evidenziate le righe del commit più vecchio tra i due e in verde le righe del commit più recente tra i due



```
Diff Old version New version Lines of context: |3| Ignore space change | Comments bubbleSort1.cpp
index 72bd405..6fddeca 100644
@@ -1,3 +1,4 @@
+//ver. 1.1 : migliorata la visualizzazione del vettore
#include <iostream>
#include <stdlib.h>      // srand, rand
#include <time.h>         // time
@@ -9,10 +10,10 @@
int numeri[TANTI];      // vettore da elaborare

void mostraVettore(){
    int x;
-   cout << "\n" << endl;
+   cout << "\n[ ";
    for (x = 0; x < TANTI; x++)
        cout << numeri[x] << " ";
-   cout << "\n" << endl;
+   cout << "]\n" << endl;
}
void riempieCasuale(){
```

# Annnullare/Ripristinare

---

- Se si vuole ritornare alla versione precedente si utilizza il comando **reset**

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git reset --hard
HEAD is now at 75a83ff ridotto indice ciclo interno

utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ |
```

- Ci viene indicato quale diviene la nuova **HEAD** del progetto e viene **riscritta** la cronologia dei commit cancellando i successivi

# Il controllo delle versioni con Git

---

- Per visualizzare tutti i commit che abbiamo eseguito si utilizza il comando `log` che mostra anche i rispettivi ID

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git log
commit 75a83ff64752d14e3dca12dd97c650794f9ff6c2 (HEAD -> master)
Author: camagni <paolo.camagni@magis.edu.it>
Date:   Tue Jul 18 11:58:24 2023 +0200

    ridotto indice ciclo interno

commit 88b0e367749722419c9c3c041f9c0fca333eb0dc
Author: camagni <paolo.camagni@magis.edu.it>
Date:   Tue Jul 18 11:40:28 2023 +0200

    migliorato output

commit fb993745b29a0243da821d9c8891c1a4dfa41960
Author: camagni <paolo.camagni@magis.edu.it>
Date:   Mon Jul 17 19:28:25 2023 +0200

    primo backup
```

# Il controllo delle versioni con Git

---

- Al comando `reset` possiamo aggiungere le prime 4 cifre dell'ID di un commit al quale vogliamo ritornare

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git reset --hard 88b0
HEAD is now at 88b0e36 migliorato output

utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$
```

- tutti i commit **successivi** a quello indicato verranno cancellati definitivamente e verrà ripristinato lo stato corrispondente al commit specificato

# Il controllo delle versioni con Git

---

- revert crea un **nuovo** commit che annulla le modifiche di un commit precedente, lasciando **intatta** la cronologia
- utile quando nel frattempo altri collaboratori hanno rilasciato modifiche e un reset causerebbe la cancellazione anche di questi
- utile se le modifiche sono state rilasciate in un repository remoto non volendo (ad es. non sono state prima testate)

# Il controllo delle versioni con Git

---

- È anche possibile tornare a un commit precedente
- Si utilizza il comando **checkout** specificando l'ID del commit a cui tornare

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git checkout 88b0
Note: switching to '88b0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 88b0e36 migliorato output

utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp ((88b0e36...))$
```

# Il controllo delle versioni con Git

- Con questa operazione preserviamo i commit successivi a quello nel quale ora siamo posizionati, facendo un “ritorno al passato”
- se osserviamo il diagramma delle versioni troviamo la seguente situazione:

The screenshot shows a commit history for a repository named 'provaCpp'. The interface has a menu bar with 'File', 'Edit', 'View', and 'Help'. Below the menu, there's a list of commits:

- master (highlighted in green)
- migliorato output
- primo backup

On the right, a detailed view of the 'migliorato output' commit is shown:

Author	Date
camagni <paolo.camagni@...	2023-07-19 13:09:50
camagni <paolo.camagni@...	2023-07-18 11:40:28
camagni <paolo.camagni@...	2023-07-17 19:28:25

- cioè il **commit** attivo ora è quello evidenziato

# Il controllo delle versioni con Git

---

- Cosa succede se poi si effettua un nuovo commit?

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp ((88b0e36...))  
$ git commit -am "aggiunta virgola tra i numeri in output"  
[detached HEAD 656a006] aggiunta virgola tra i numeri in output  
 1 file changed, 2 insertions(+), 2 deletions(-)  
  
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp ((656a006...))  
$ |
```

- Effettuando il commit si crea una situazione “**alternativa**”, ovvero si realizza una diramazione che devia dalla linea di sviluppo corrente

# Il controllo delle versioni con Git

- Questa diramazione viene detta ramificazione o branch

The screenshot shows a software interface for viewing Git commit history and differences. The title bar says "provaCpp: --all - gitk". The menu bar includes File, Edit, View, and Help. The main area displays a commit log and a diff viewer.

**Commit Log:**

- aggiunta virgola tra i numeri in output (commit by camagni on 2023-07-19 13:23:23)
- ridotto indice ciclo interno (commit by camagni on 2023-07-19 13:09:50)
- migliorato output (commit by camagni on 2023-07-18 11:40:28)
- primo backup (commit by camagni on 2023-07-17 19:28:25)

SHA1 ID: 656a0066983e6c02e0abe6a453d1ae5ce317229a

**Diff Viewer:**

Find commit containing: bubbleSort1.cpp

Diff Options: Diff (radio button selected), Old version, New version, Lines of context: 3, Ignore space change.

Diff Content:

```
//ver. 1.1 : migliorata la visualizzazione del vettore
+//ver. 1.2 : migliorata la visualizzazione del vettore
#include <iostream>
#include <stdlib.h>      // srand, rand
#include <time.h>         // time
@@ -12,7 +12,7 @@ void mostraVettore(){
    int x;
    cout << "\n[ ";
    for (x = 0; x < TANTI; x++)
-     cout << numeri[x] << " ";
+     cout << numeri[x] << ", "; // aggiunto separatore
    cout << "]\n" << endl;
}
```

**Right Panel:**

Patch (radio button selected), Tree.

Comments: bubbleSort1.cpp

# Il controllo delle versioni con Git

---

- La possibilità di creare dei branch è di fondamentale importanza e di grande utilità per gli sviluppatori
- Nella realizzazione di un'applicazione potrebbe essere necessario realizzare degli sviluppi **paralleli** su un progetto
- Possiamo anche avere molti rami e, quindi, molti percorsi che si evolvono in parallelo

# Il controllo delle versioni con Git

---

- Possiamo visualizzare tutti i branch del progetto con l'istruzione `git branch -v`

```
utente@portable-paolo MINGW64 /c/esempiGit/provaCpp (master)
$ git branch -v
gestione_output ce2d88d corretta virgola tra i numeri in output
* master           54e23a1 migliorato output
```

```
utente@portable-paolo MINGW64 /c/esempiGit/provaCpp (master)
$
```

- Per creare un nuovo branch usiamo l'istruzione `git branch <nome_nuovo_branch>` e successivamente per lavorare su quest'ultimo usiamo il comando `checkout` già visto in precedenza
- O combinando i due comandi `git checkout -b <nome_nuovo_branch>`

# Il controllo delle versioni con Git

- Quando si sono completate le modifiche si possono far confluire nel ramo principale facendo una fusione ([merge](#)) o abbandonarle
- Scriviamo in modo esplicito il nome del ramo che vogliamo fondere al **branch main**:

```
utente@PC-PAOLO MINGW64 /d/esempiGit/provaCpp (master)
$ git merge "gestione output"
```

- Visualizziamo ora graficamente cosa abbiamo ottenuto e osserviamo che i due rami si sono fusi:

The screenshot shows a graphical interface for Git called Gitk. It displays a commit history with two branches: 'master' and 'gestione\_output'. The 'master' branch has several commits, and the 'gestione\_output' branch has one commit, which is a merge commit. The merge commit is labeled 'Merge branch 'gestione\_output''. The commit message for this merge is 'corretta virgola tra i numeri in output'. The commit history also includes other commits from the 'gestione\_output' branch, such as 'aggiunta virgola tra i numeri in output', 'bubble sort con sentinella', 'aggiunta sentinella', 'ridotto indice ciclo interno', 'migliorato output', and 'primo backup'. On the right side of the interface, there is a list of commits with their authors and dates.

Author	Date
camagni <paolo.camagni@magis	2023-07-19 19:06:30
camagni <paolo.camagni@magis	2023-07-19 13:56:55
camagni <paolo.camagni@magis	2023-07-19 13:23:23
camagni <paolo.camagni@magis	2023-07-19 18:52:48
camagni <paolo.camagni@magis	2023-07-19 17:57:24
camagni <paolo.camagni@magis	2023-07-19 13:09:50
camagni <paolo.camagni@magis	2023-07-18 11:40:28
camagni <paolo.camagni@magis	2023-07-17 19:28:25

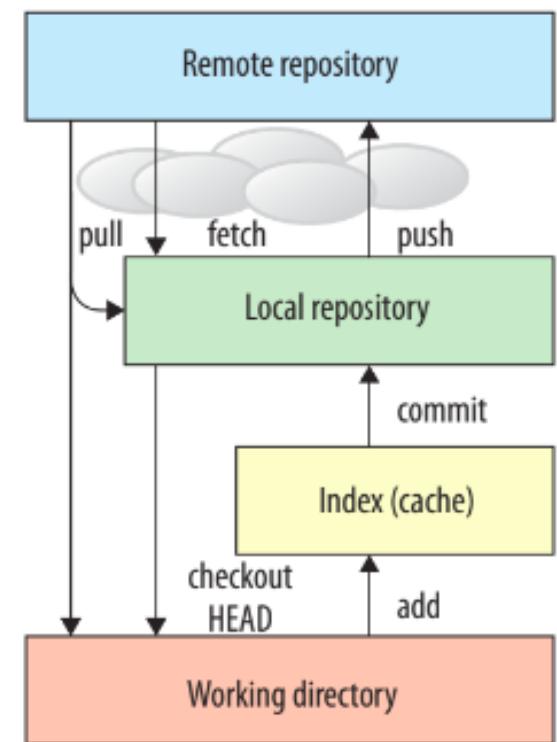
# Il controllo delle versioni con Git

---

- Se abbiamo bisogno di eliminare un file **presente** nel repository, ovvero versionato con un precedente commit, possiamo usare il comando **rm**
- Se invece volessimo rimuovere un file dall'area di stage (tramite un precedente comando add) per spostarlo nella working directory possiamo utilizzare il comando rm in combinazione con l'opzione **--cached**

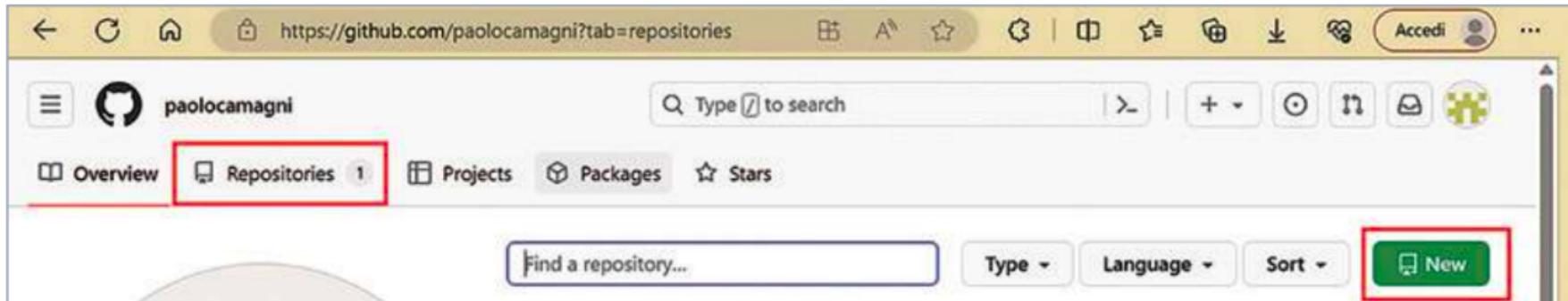
# Collaborare in gruppo: Github

- La memorizzazione di un progetto in rete:
  - consente all'utente di lavorare su macchine diverse
  - facilità il lavoro di gruppo, in particolare la condivisione del codice
- Solitamente i file vengono condivisi tramite **web cloud**, dunque oltre al repository locale avremo una **copia** di esso ospitata in un server remoto
- Con il repository remoto aggiungiamo un ulteriore livello alla nostra architettura



# Collaborare in gruppo: Github

- La piattaforma più conosciuta per l'hosting di repository Git è [Github](https://github.com) (<https://github.com>), ma non offre archivi privati nella versione gratuita
  - L'alternativa è [Bitbucket](https://bitbucket.org) (<https://bitbucket.org>) che invece offre archivi privati illimitati
- Per prima cosa bisogna avere un account su Github, quindi iscriversi
- Successivamente possiamo creare un nuovo repository



- Dargli un nome e ottenere l'url di quel repository in cloud che avrà il formato [https://github.com/<account>/<nome\\_repository\\_remoto>.git](https://github.com/<account>/<nome_repository_remoto>.git)

# Collaborare in gruppo: Github

- Il collegamento tra il repository remoto e quello locale si effettua posizionandoci nella futura working directory (ovvero una directory vuota, non versionata!)
- Accediamo alla cartella da riga di comando
- Lanciamo il comando  
`git clone <percorso_al_repository_remoto>`  
che trasferisce tutto il contenuto del repository remoto in locale

```
utente@PC-PAOLO MINGW64 /d/esempiGit
$ git clone https://github.com/paolocamagni/provaJava
Cloning into 'provaJava'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
```

```
utente@PC-PAOLO MINGW64 /d/esempiGit
$ |
```

# Collaborare in gruppo: Github

---

- Come facciamo a sincronizzare un ramo del nostro repository locale con un ramo del repository remoto?
  - Innanzitutto dobbiamo specificare dove si trova il repository remoto (è una configurazione, basta farlo una volta sola)  
`git remote add origin <url_del_server_remoto>`
  - Per visualizzare dove si trova il repository remoto  
`git remote -v`
  - Si utilizza l'opzione del comando branch (anche in questo caso è una configurazione e basta farlo una volta sola)  
`--set-upstream-to=origin/<branch_remoto> <branch_locale>`  
es. `git branch -- set-upstream-to=origin/main master`

# Collaborare in gruppo: Github

---

- **Upload:** per condividere in un repository remoto delle modifiche pubblicate nel repository locale (in modo da condividerlo con il gruppo di lavoro) si utilizza il comando **push**
- **Download:** analogamente per copiare delle modifiche presenti nel repository remoto, ma non ancora presenti nel repository locale si utilizza il comando **pull**
- **Attenzione ai conflitti!** Se un collaboratore:
  - a) ha modificato dei file che noi abbiamo in modifica nella working directory o nell'area di stage
  - b) piuttosto che ha modificato dei file che noi abbiamo versionato sul repository locale (e solo lì al momento)

avremo bisogno di effettuare un **merge** che genererà un nuovo commit e dunque richiederà un nuovo push per allineare il repository remoto!

# Collaborare in gruppo: Github

---

- Come facciamo a sapere se il repository locale è aggiornato ovvero è allineato al repository remoto?
- Si utilizza il comando `fetch` dal ramo che vogliamo verificare.
- **Attenzione!** Questo comando andrebbe eseguito sempre almeno all'inizio di ogni nuova sessione di lavoro su un progetto in collaborazione!

# Il controllo delle versioni con Git

---

- Esistono altri comandi utili, ad es.:
  - **git diff**, per ritrovare le modifiche fatte dall'ultimo commit; possibili opzioni per il comando:
    - `git diff "@{yesterday}"`, per le modifiche fatte da ieri
    - `git diff 1b6d "master~2"`, per confrontare una versione specifica con due versioni precedenti
- L'elenco completo dei comandi è disponibile su <https://git-scm.com/docs>