
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

# Laboratorio di Programmazione ad Oggetti

Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
https://jdirocco.github.io

# Sommario

- Cosa è un Sistema di controllo di versione (VCS)
- VCS Centralizzato
  - Cosa è
  - Problema e soluzioni di condivisione dei files
  - Apache Subversion
- VCS Distribuito
  - Git
  - Caratteristiche
  - Istantanee e stati di Git
  - Flusso di lavoro in Git
  - Download ed installazione
  - Operazioni di base
  - Lavorare con repository remoti
  - Comandi di Git
- GitHub ed Education

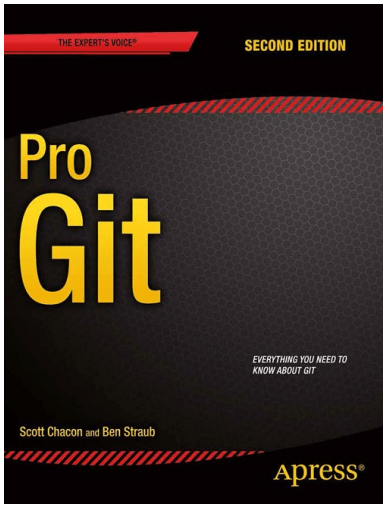
# Risorse

› Git

`https://git-scm.com/book/it/v2`

› Guida in italiano

`https://www.html.it/guide/git-la-guida/`



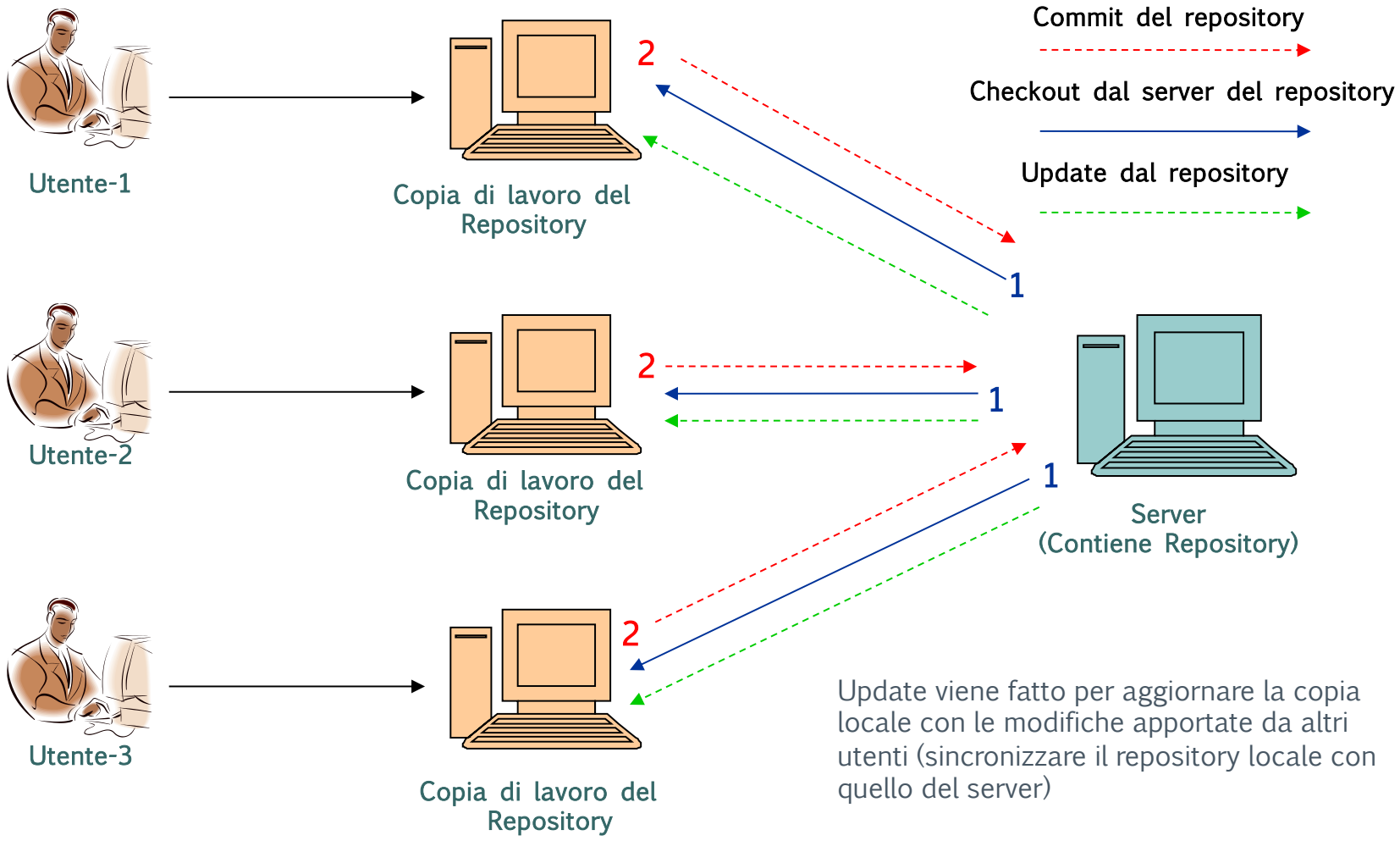
# Cosa è un VCS (1)

- Version control system: Sistema di controllo di versione
- Combinazione di tecnologie e pratiche per il tracciamento e il controllo dei **cambiamenti** dei files di un progetto (codice sorgente, documentazione, pagine web)
- Aiuta la comunicazione, gestione delle release, stabilità del codice, gestione degli errori tra gli sviluppatori all'interno di un progetto,
- Funzionalità **core** è la gestione del cambiamento
  - E' necessario identificare ogni cambiamento fatto sui files del progetto annotando tali cambiamenti con dei metadata (esempio: data e autore del cambiamento)

# Cosa è un VCS (2)

- VCS tiene sotto controllo i files e/o le directory mediante un **repository**
- Lo sviluppatore dovrà prelevare tale repository dal sistema di VCS. Tale operazione è detta di **checkout** o **clone**
- Il sistema VCS gestirà tale repository tenendo traccia di ogni cambiamento
- Tale tracciamento avviene in modo trasparente e avverrà soltanto quando lo sviluppatore deciderà di rendere definitive tali modifiche (**commit**)

# VCS centralizzato (1)




Update viene fatto per aggiornare la copia locale con le modifiche apportate da altri utenti (sincronizzare il repository locale con quello del server)

# VCS centralizzato (2)

Ieri

Commit

Versione-1

```
#include<stdio.h>
Int  main()
{
      int  a;
}
```

File.c

Versione-2

Oggi - Mattina

Commit

```
#include<stdio.h>
Int  main()
{
      int  a;
      int  b;
}
```

File.c

Versione-3

Oggi - Sera

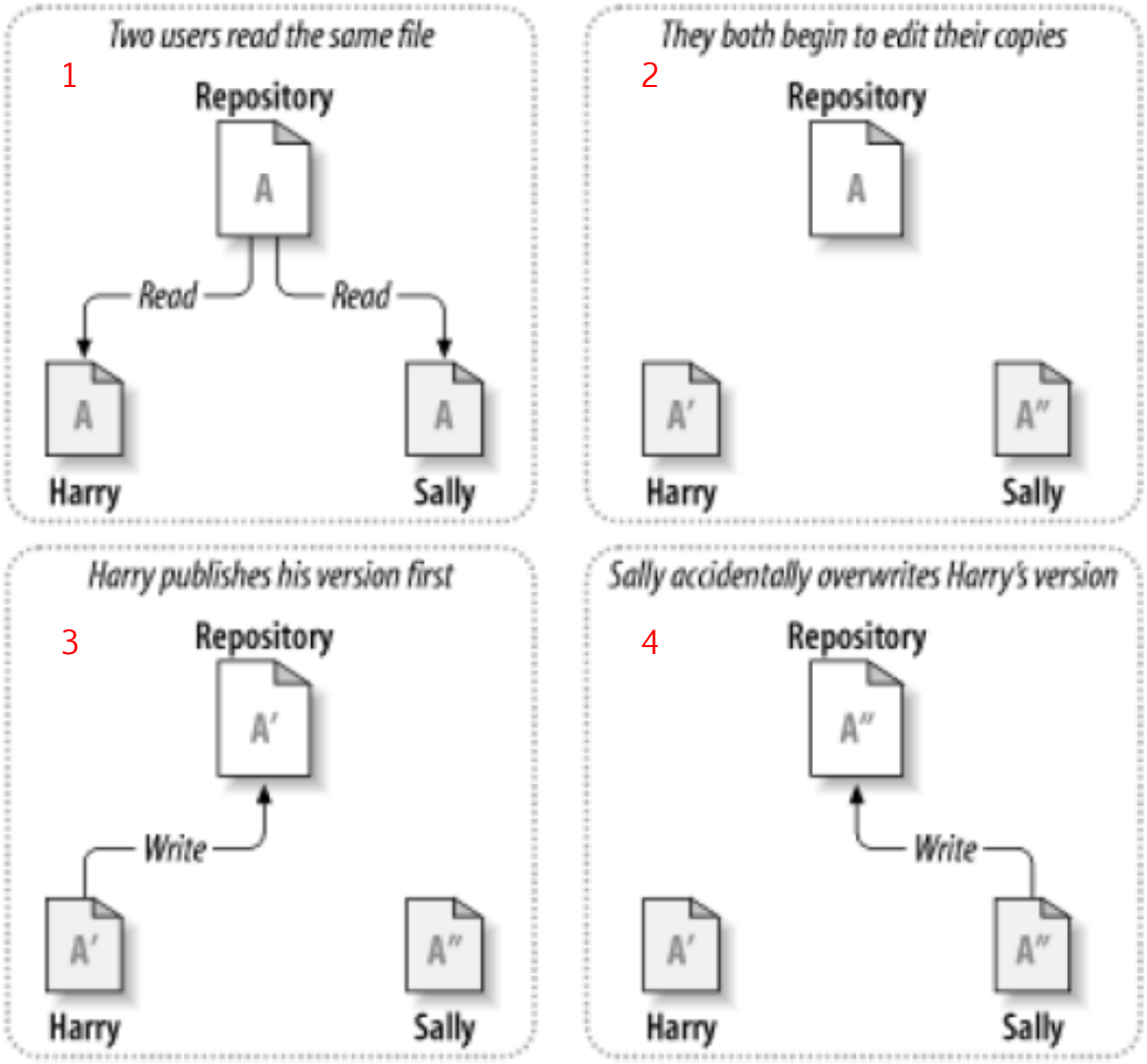
Commit

```
#include<stdio.h>
Int  main()
{
      int  a;
      int  b;
}
```

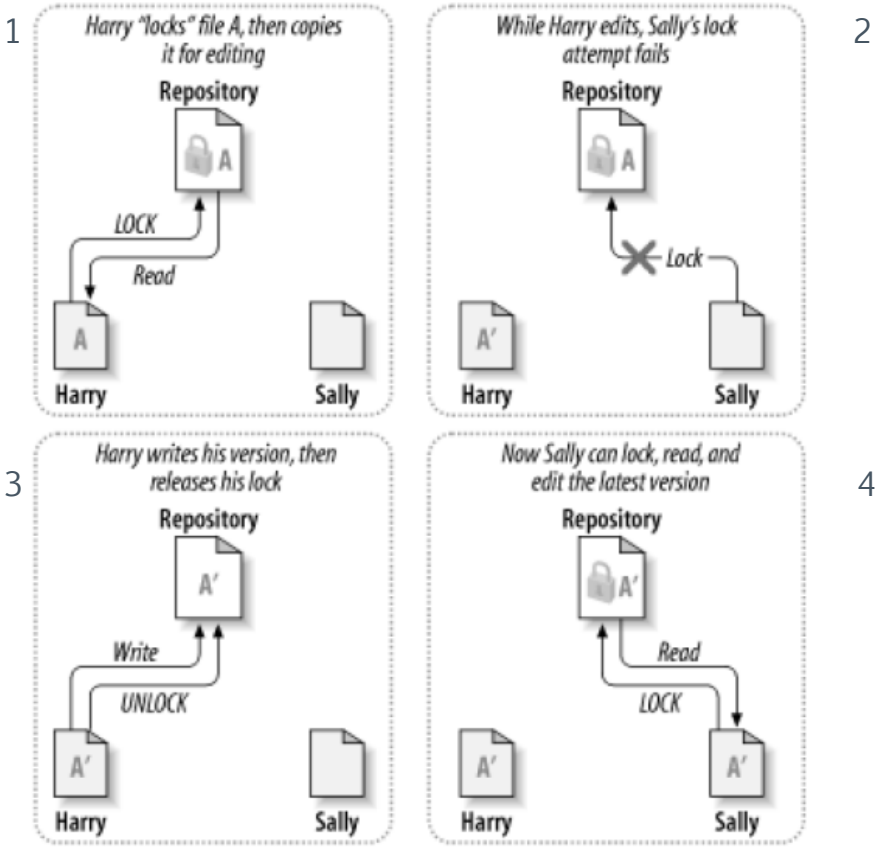
File.c

# Problema di condivisione dei file (1)

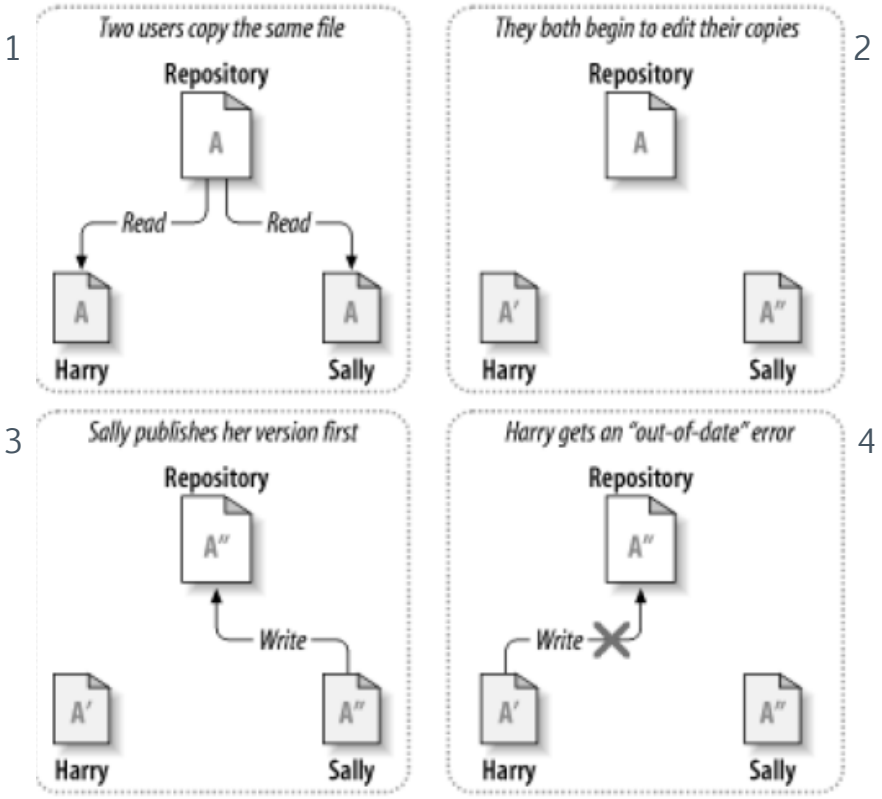
- Una caratteristica di un qualsiasi sistema VCS moderno è quella di consentire la modifica **collaborativa** e la **condivisione** di tali dati
  - In che modo il sistema permetterà agli utenti di condividere le informazioni, ma impedirà loro di pestarsi i piedi a vicenda?
- Sistemi VCS diversi utilizzano strategie diverse per raggiungere questo obiettivo


Two users read the same file
1
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
2
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
3
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
4
Repository
A″
Write
A′
A″
Harry
Sally

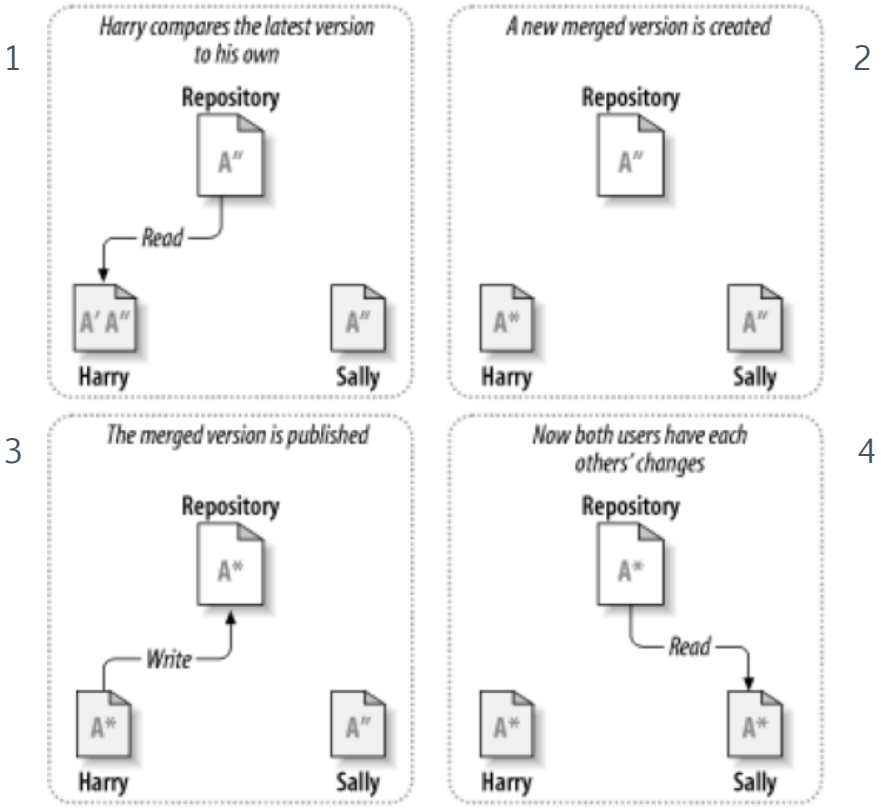
# Soluzione Lock-modify-unlock



# Soluzione Copy-modify-merge (1)



# Soluzione Copy-modify-merge (2)



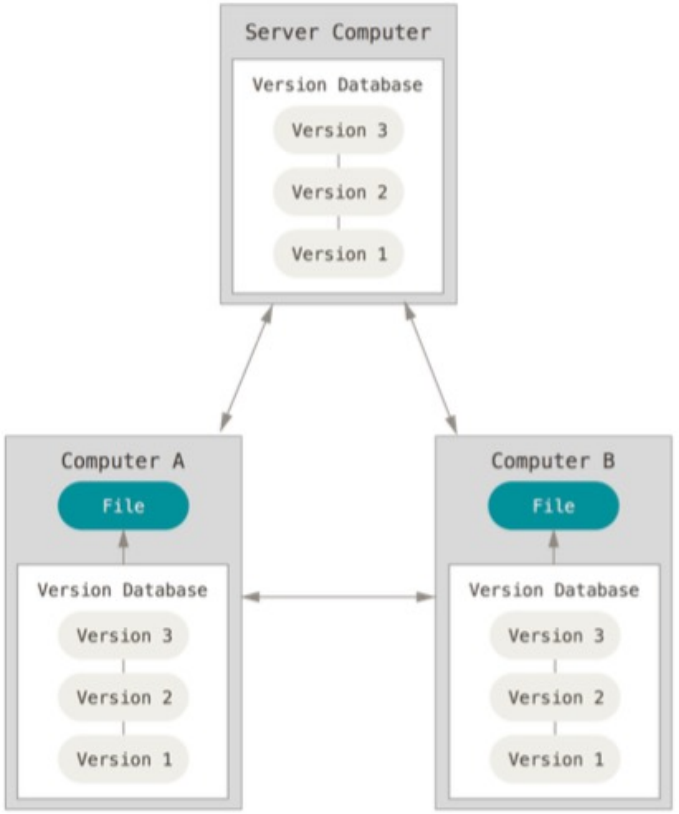
# Apache Subversion

- Apache subversion è un sistema VCS completo, originariamente progettato per essere un CVS migliore
- Da allora subversion si è espanso oltre il suo obiettivo originale di sostituire CVS, ma il suo modello di base, il design e l'interfaccia rimangono fortemente influenzati da tale obiettivo
- Caratteristiche
  - Molte delle caratterstiche di CVS: versionamento di files, gestione di tags, branches, ecc
  - Directory sono versionate
  - Cancellazione, copiare e rinominare sono versionate
  - Commit atomici
  - Lock dei files
  - ecc.

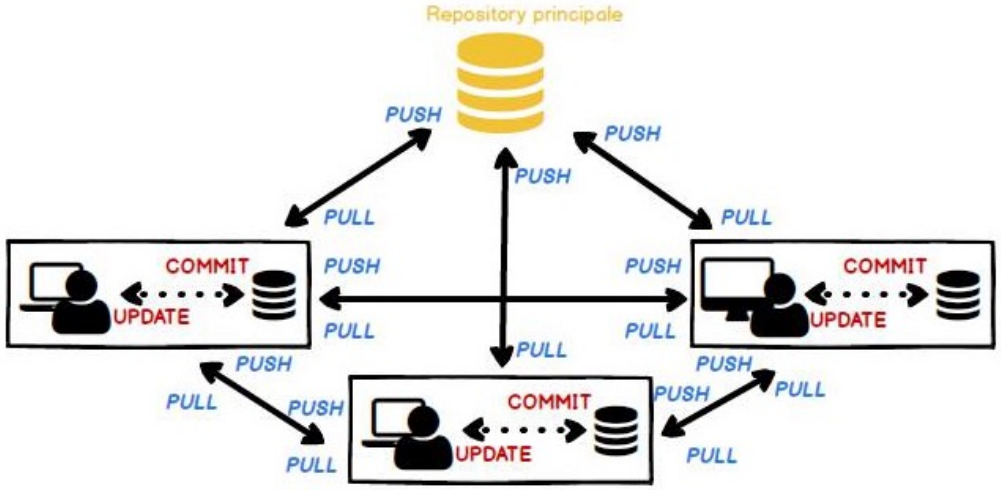
# Download

- Server
  - https://subversion.apache.org/
- Client on windows
  - https://tortoisesvn.net/

# VCS distribuito



# VCS distribuito




- Ogni programmatore può lavorare indipendentemente col proprio repository locale
- Anche se è possibile scambiare set di modifiche con gli altri collaboratori, in pratica si sceglie di far riferimento a un **repository principale**
- Tale scelta non è imposta dall'architettura di Git
- Definito un repository principale, ogni programmatore provvederà a scaricare da quest'ultimo gli aggiornamenti e ad applicarli alla propria copia locale del repository
- Invierà poi le proprie modifiche al repository eseguendo l'operazione **push**

# Storia di Git (1)

- Il kernel di Linux è un progetto software open source di grande portata.  Per buona parte del tempo (1991-2002) della manutenzione del kernel Linux le modifiche al software venivano passate sotto forma di patch e file compressi
- Nel 2002, il progetto del kernel Linux iniziò ad utilizzare un sistema DVCS proprietario chiamato BitKeeper
- Nel 2005 il rapporto tra la comunità che sviluppa il kernel Linux e la società commerciale che aveva sviluppato BitKeeper si ruppe, e fu revocato l'uso gratuito di BitKeeper
- Ciò indusse Linus Torvalds (il creatore di Linux) a sviluppare uno strumento proprio

# Storia di Git (2)

- Sviluppo di Git inizio il 3 Aprile 2005
- Torvalds ha annunciato il progetto il 6 aprile
- Torvalds ha raggiunto i suoi obiettivi di performance; il 29 aprile, il nascente Git è stato sottoposto a benchmarking registrando le patch al kernel di Linux al ritmo di 6,7 al secondo
- Il 16 giugno Git ha gestito il rilascio del kernel 2.6.12
- Il 26 luglio 2005 Torvalds ha affidato la manutenzione a Junio Hamano, uno dei principali contributor del progetto
- Hamano è stato responsabile del rilascio della versione 1.0 il 21 dicembre 2005, e rimane il maintainer del progetto

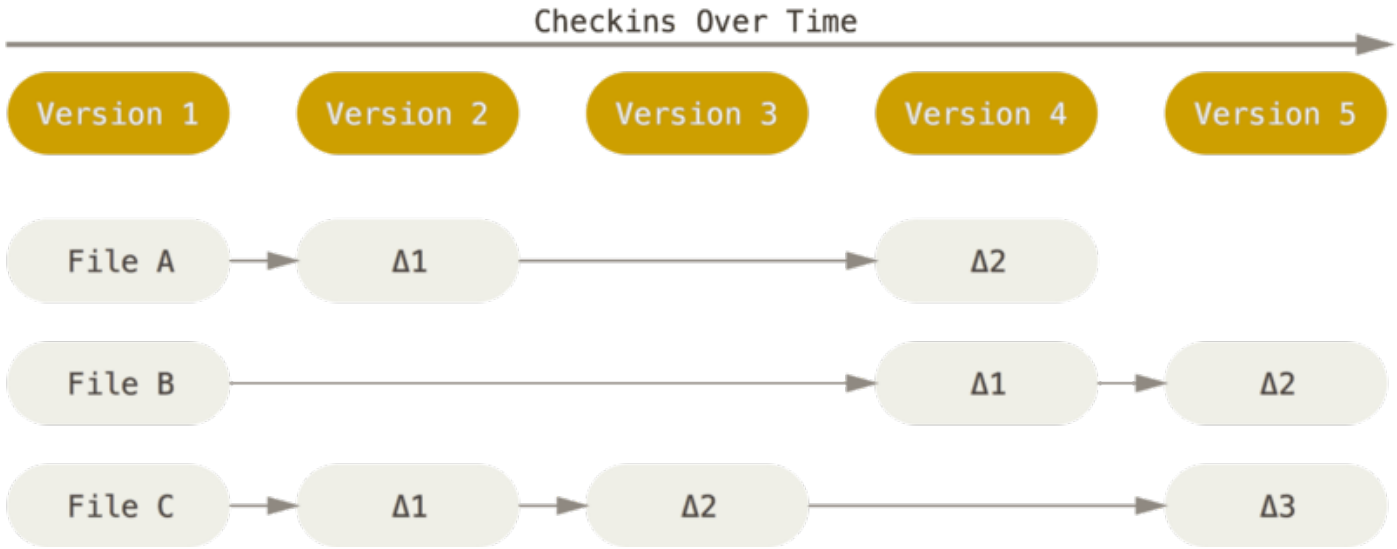
# Caratteristiche di Git (1)

- **Forte sostegno allo sviluppo non lineare**
  - Supporta la ramificazione e la fusione rapida e include strumenti specifici per la visualizzazione e la navigazione di una storia di sviluppo non lineare
- **Sviluppo distribuito**
  - Fornisce ad ogni sviluppatore una copia locale della storia completa dello sviluppo e le modifiche vengono copiate da un archivio all'altro
- **Compatibilità con i sistemi e i protocolli esistenti**
  - Repository possono essere pubblicati tramite Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), rsync (fino a Git 2.8.0), o un protocollo Git su un semplice socket, o Secure Shell (ssh)

# Caratteristiche di Git (2)

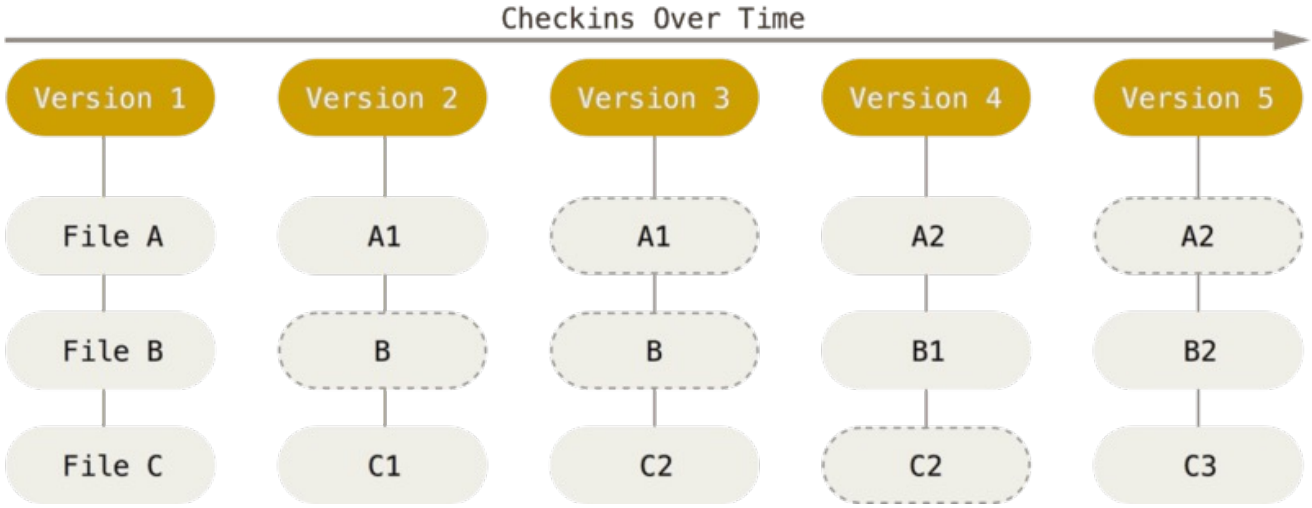
- **Gestione efficiente di grandi progetti**
  - Ordine di grandezza più veloce di alcuni sistemi di controllo di versione, e il recupero della cronologia delle versioni da un repository memorizzato localmente può essere cento volte più veloce del recupero dal server remoto
- **Autenticazione crittografica della storia**
  - La storia di Git è memorizzata in modo tale che l'ID di una particolare versione (un commit in termini di Git) dipende dalla storia completa dello sviluppo che porta a quel commit. Una volta pubblicato, non è possibile cambiare le vecchie versioni senza che venga notato
- **Strategie di fusione collegabili**
  - Ha un modello ben definito di un merge incompleto, e dispone di molteplici algoritmi per completarlo, che culmina nel dire all'utente che non è in grado di completare il merge automaticamente e che è necessaria una modifica manuale

# Istantanee, No Differenze (1)



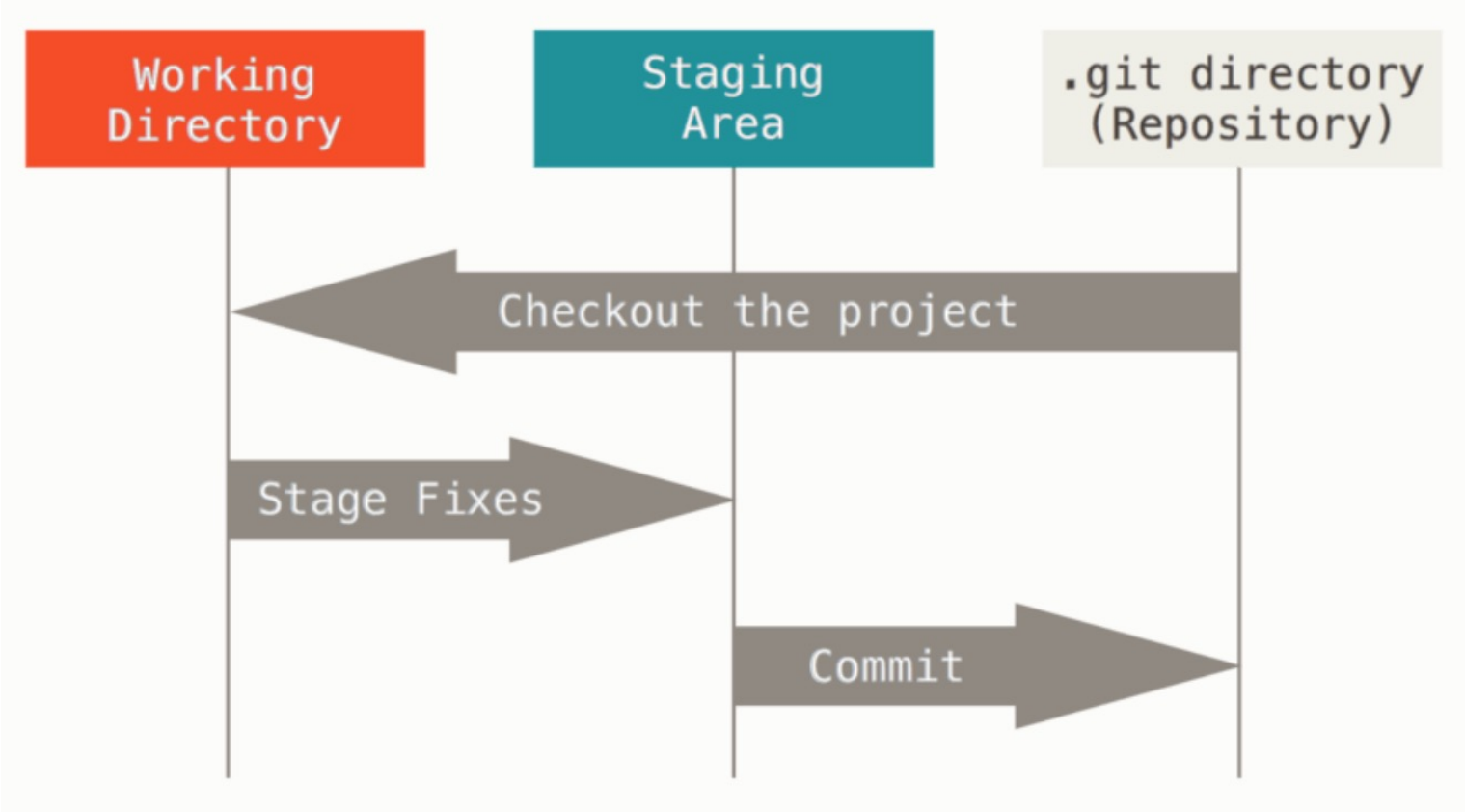

VCS tradizionali considerano informazioni che memorizzano come un insieme di file, con le relative modifiche fatte nel tempo (questo viene normalmente descritto come controllo di versione su base delta)

# Istantanee, No Differenze (2)




- Git considera i dati come una sequenza di istantanee (snapshot) di un mini filesystem
- Git, ogni volta che registri (commit), fa un'immagine di tutti i file in quel momento, salvando un riferimento allo snapshot
- Per essere efficiente, se alcuni file non sono cambiati, Git crea semplicemente un collegamento al file precedente già salvato
- Git considera i propri dati più come un **flusso di istantanee**

# Sezioni di un progetto Git



# Sezioni di un progetto Git

- Cartella di lavoro (**Working directory**) è un checkout di una versione specifica del progetto. Questi file vengono estratti dal database compresso nella directory di Git, e salvati sul disco per essere usati o modificati
- Area di sosta (**Area di stage**) è un **file**, contenuto generalmente nella directory di Git, con tutte le informazioni riguardanti la tua prossima commit
  - Nome tecnico nel gergo di Git è **indice**
- Directory di Git (`.git`) è dove Git salva i metadati e il database degli oggetti del tuo progetto. E' dove viene copiato quando si clona un repository da un altro computer

# Stati dei file in GIT

› File possono essere in tre stati principali
  – **Untracked:** file presenti nella working directory ma non ancora sottoposti a versionamento,
  – **Modified** (modificati): il file è stato modificato, ma non è ancora stato committato nel database (repository locale)
  – **Staged** (in stage): hai contrassegnato un file, modificato nella versione corrente, perché venga inserito nel database al prossimo commit
  – **Committed** (committati): il file è registrato al sicuro nel database locale (repository)

# Flusso di lavoro standard di GIT

- **Modifica** i files nella tua working directory
- **Seleziona** per lo stage solo quei cambiamenti che vuoi facciano parte del tuo prossimo commit, che aggiunge solo queste modifiche all'area di stage
- **Committa**, ovvero salva i file nell'area di stage in un'istantanea (snapshot) permanente nella tua directory di Git

# Creazione di un account

- In Git i commit fanno riferimento a chi li ha effettuati
- Ogni contributor deve essere dotato di una **username** e di un indirizzo di posta elettronica che ne definiscano l'identità
- Informazioni entreranno a far parte delle variabili globali utilizzate nel corso delle sessioni di lavoro
- Sintassi

```
$ git config --global user.name 'quattromori'
```

- Stessa procedura per quanto riguarda l'email
- Sintassi

```
$ git config --global user.email quattromori@latuaemail.com
```

N.B.: Per un singolo progetto si può omettere opzione `--global`

# Operazioni di base di Git (1)

Due approcci per lavorare su un progetto Git

› Creare un repository in una cartella esistente

```
$ git init: crea una nuova sottodirectory chiamata .git che
contiene tutti i file del repository necessari
```

› **Clonare** un repository esistente

```
$ git clone https://github.com/libgit2/libgit2
```

In questo caso viene creata una directory `libgit2` contenente il progetto Git

OPPURE

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

# Operazioni di base di Git (2)

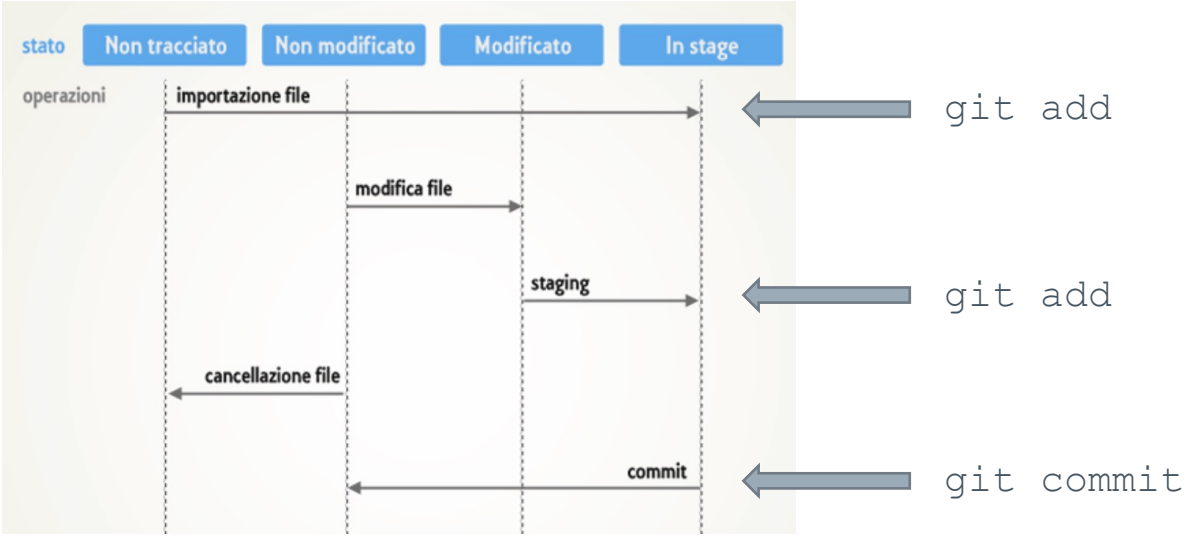
- Creare o modificare i files e/o directory dentro il progetto
- Una volta creati/modificati eseguire i seguenti comandi dalla root del progetto

```
$ git add *.java
$ git commit -m 'initial project version'
```

- Compito dell'opzione `add` **non** è quello di creare un file da zero
- Comando `git add` svolge la funzione di aggiornare l'indice (ovvero staging area) di un repository con le informazioni relative ai contenuti correnti

- Il commit tramite `git commit` è seguito da un messaggio che descrive lo stato del progetto al momento corrente
- Messaggio viene specificato tramite l'opzione `-m`
- `git commit` non fa altro che caricare i contenuti dell'indice precedentemente aggiornato tramite `git add` in un commit per la conferma di una modifica a carico del progetto gestito

# Operazioni di base di GIT (3)




- Non tracciato (**Untracked**): il file non è stato preso in considerazione da Git
- Modificato (**modified**): rispetto l'ultimo salvataggio/commit il file è cambiato
- **Staged**: il file è marcato per essere salvato sul prossimo commit
  **Solo i file in staging area verranno salvati sul commit!**
- **Committed**: il file, e una sua copia, sono stati congelati in quel preciso istante sul database di Git

# Lavorare con i repository remoti (1)

› Repository remoti sono versioni di un progetto che sono ospitati su Internet (es. GitHub) oppure nella rete locale

```
git clone https://github.com/amletodisalle/empty-repository.git
```

```
cd empty-repository
```

```
git remote
```

› Consente di ottenere una lista dei server remoti associati ad uno specifico progetto

› `origin` indica il server remoto di default del progetto

› Parametro `-v` permette di elencare gli URL associati ad esso

```
$ git remote
origin

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ git remote -v
origin  https://github.com/amletodisalle/empty-repository.git (fetch)
origin  https://github.com/amletodisalle/empty-repository.git (push)
```

# Lavorare con i repository remoti (2)

- Dopo che si sono creati files/directories aggiunti nell'area di stage (`git add`) e committati (`git commit`) è necessario inviare il repository nel server dove viene ospitato
- Comando

`git push`

```
MINGW64:/c/Users/amleto/Desktop/helloworld-git/empty-repository/empty-repository
$ ls -la
total 26
drwxr-xr-x 1 amleto 197121     0 apr 23 16:43 ./
drwxr-xr-x 1 amleto 197121     0 apr 23 15:49 ../
drwxr-xr-x 1 amleto 197121     0 apr 23 16:43 .git/
-rw-r--r-- 1 amleto 197121   301 apr 23 15:49 .gitignore
-rw-r--r-- 1 amleto 197121 11558 apr 23 15:49 LICENSE
-rw-r--r-- 1 amleto 197121     7 apr 23 16:38 pippo

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ echo 'file pluto' > pluto

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ git add .
warning: LF will be replaced by CRLF in pluto.
The file will have its original line endings in your working directory

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   pluto


amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ git commit -m "aggiunto file pluto"
[master 74dd27a] aggiunto file pluto
 1 file changed, 1 insertion(+)
 create mode 100644 pluto

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/amletodisalle/empty-repository.git
   0980264..74dd27a  master -> master

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository/empty-repository (master)
$
```

# Lavorare con i repository remoti (3)

`git fetch`

- Va nel progetto remoto e tira giù tutti i dati di quel progetto remoto che non si hanno ancora nella copia locale
- Dovreste avere i riferimenti a tutti i branches (rami) di quel progetto, che potete unire o ispezionare in qualsiasi momento
- I dati non vengono mergiati (fusi) automaticamente con nessuno dei vostri lavori e non modifica ciò su cui state lavorando
- Bisogna effettuare un merge **manuale**

# Lavorare con i repository remoti (4)

**`git pull`**

› Recupera automaticamente e poi **unisce** quel ramo (branch) remoto nel ramo corrente

› Per impostazione predefinita, il comando `git clone` imposta automaticamente il vostro ramo **`master`** locale per tracciare il ramo master remoto sul server da cui avete clonato

# Verifiche di stato dei file

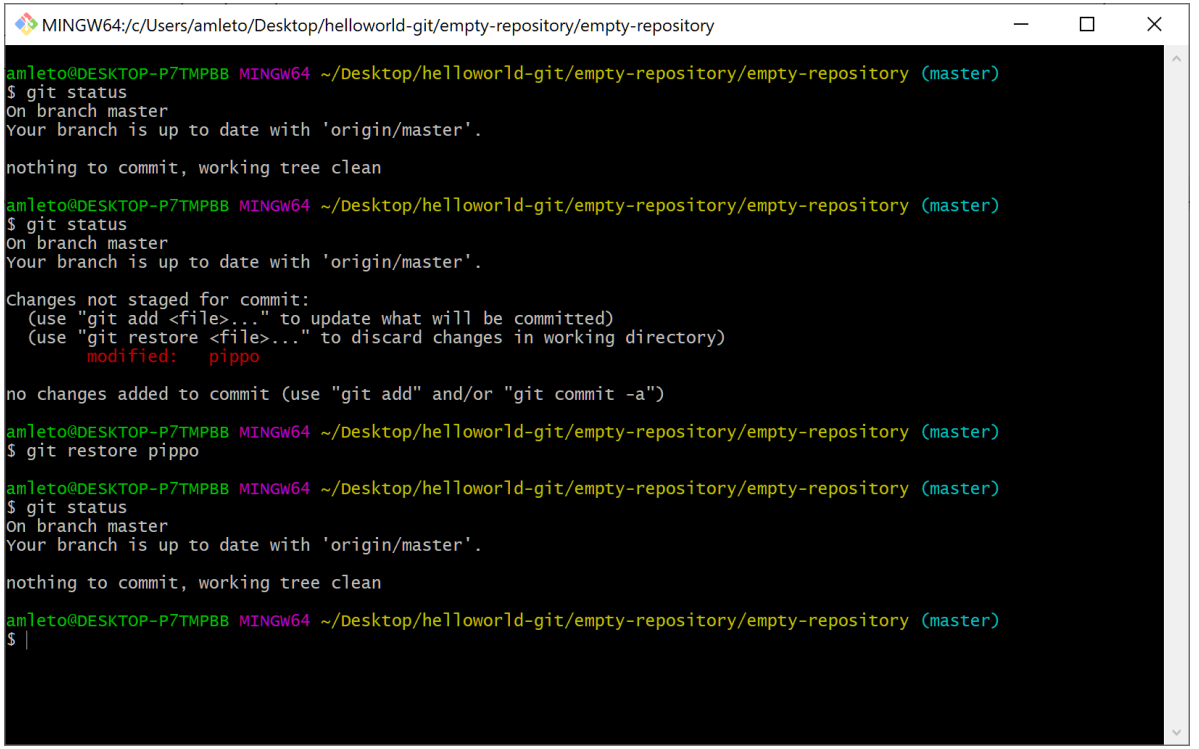
- `git status` comando di base per operare verifiche di stato a carico dei files di un progetto
- Deve essere lanciato all'interno della directory di lavoro

```
MINGW64:/c/Users/amleto/Desktop/helloworld-git/empty-repository

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git
$ git clone https://github.com/amletodisalle/empty-repository.git
Cloning into 'empty-repository'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), 4.68 KiB | 4.68 MiB/s, done.

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git
$ cd empty-repository/

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository (master
)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

amleto@DESKTOP-P7TMPBB MINGW64 ~/Desktop/helloworld-git/empty-repository (master
)
$
```

# Restore di un file (1)

- Modificare un file `pippo`
- Per riportarlo alla versione non modificata se **non** è nella area di staging (ovvero **non** si è eseguito il comando `git add`)
- Eseguire comando

`git restore pippo`



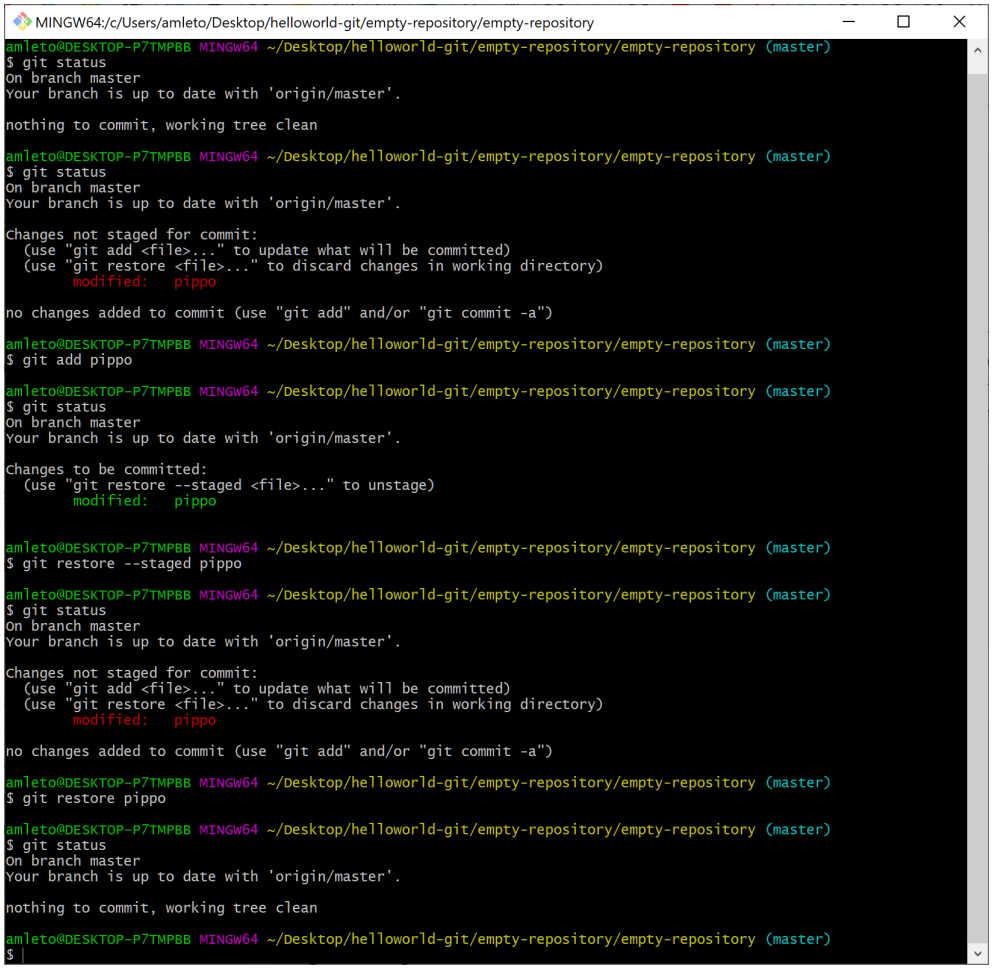
# Restore di un file (2)

- Modificare un file `pippo`
- Se si è eseguito il comando `git add`
- Eseguire comando (tolto dall'area di staging)

```
git restore –staged pippo
```

- Eseguire (di nuovo) comando

```
git restore pippo
```



# Cancellazione di un file

› Comando `git rm <nome file>`

› Un file da cancellare verrà rimosso dall'area di stage in modo da escluderne il tracciamento

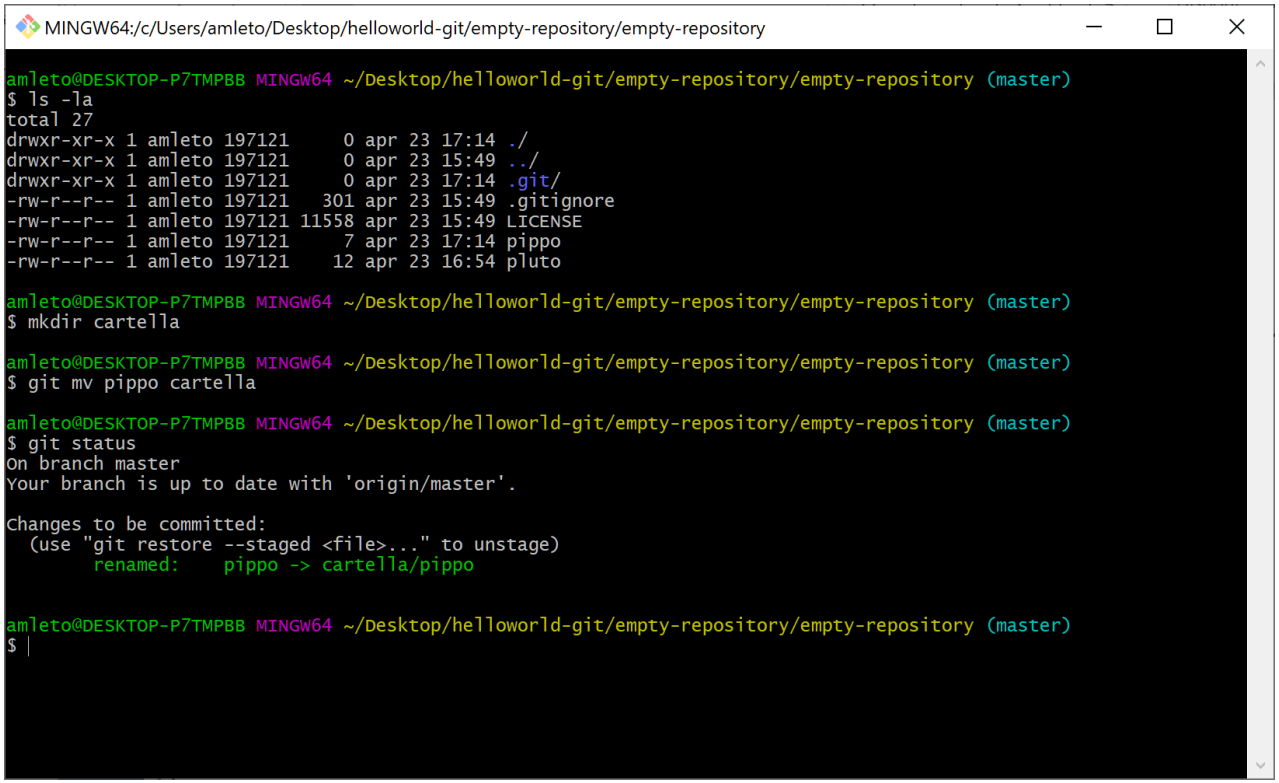
› Successivamente dovrà essere committato con il comando

`git commit –m ""`

# Muovere e rinominare un file

› Per spostare e/o rinominare un file in Git

```
$ git mv file_from file_to
```



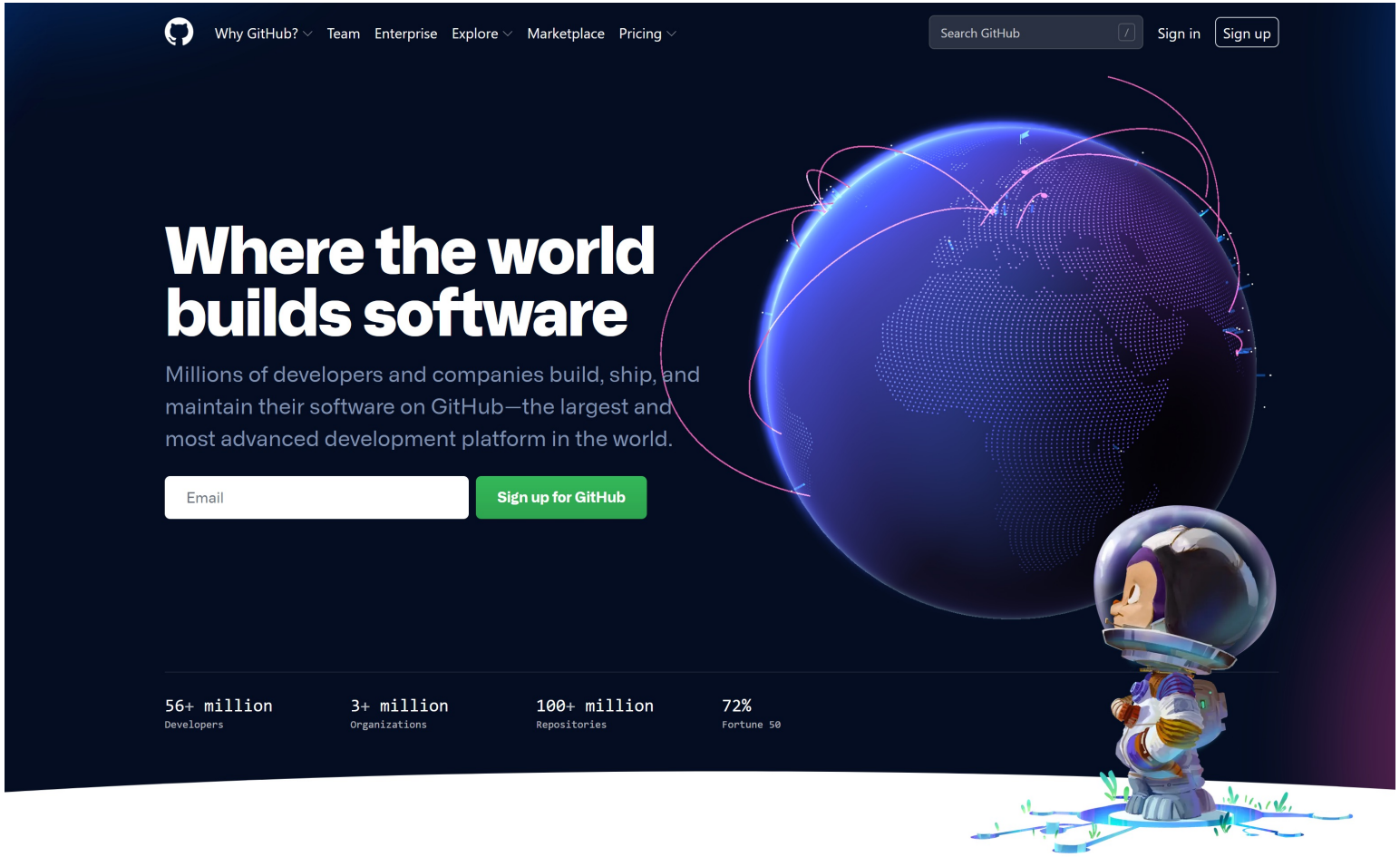
# Git != GitHub


Git
SOFTWARE PER IL
CONTROLLO DI
VERSIONE DISTRIBUITO
GitHub
PIATTAFORMA ONLINE
PER L'HOSTING
DEI REPOSITORY GIT

# GitHub (1)

- Un sito per l'archiviazione online dei repository Git
- Funzionalità
  - Numero illimitato di repository public e privati
  - Numero illimitato di collaboratori
  - 2,000 azioni al mese per repository pubblici
  - 500MB di spazio per I package GitHub per repository pubblici
  - 50MB massima dimensione di un file.
  - Interfaccia grafica web
  - documentazione
  - Tracciamento dei bug (problemi)
  - Richieste di funzionalità, richieste di pull
  - Interazioni sociali tra gli sviluppatori
  - seguire, controllare le attività, scoprire nuovi repo
- Sito: https://github.com/

# GitHub (2)


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
/
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email
Sign up for GitHub
56+ million
Developers
3+ million
Organizations
100+ million
Repositories
72%
Fortune 50

# GitHub Education: https://education.github.com/

https://education.github.com

90%

Back to GitHub.com GitHub Support Contact GitHub

Education

Students Teachers Schools Events Get benefits

## Real-world tools, engaged students

GitHub Education helps students, teachers, and schools access the tools and events they need to shape the next generation of software development.

### GitHub Student Developer Pack

The best developer tools, free for students

### GitHub Campus Experts

Training to enrich the technology community at your school

### GitHub Teacher Toolbox

The best developer tools for teaching, free for academic use

### GitHub Campus Advisors

Teacher training to master Git and GitHub

### GitHub Classroom

The GitHub workflow, scaled for the needs of students

### GitHub Campus Program

GitHub for your whole school, with everything you need to make it great