Ingegneria Del Software

Università di Roma

Tor Vergata

GIT

Distributed Version Control System

Ivan Bruno

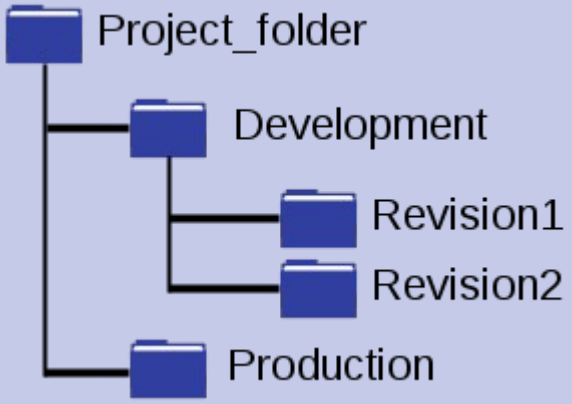
Roadmap

Git basics

Git cooding

Local Version Control

Copiare file in apposite cartelle



Pro:

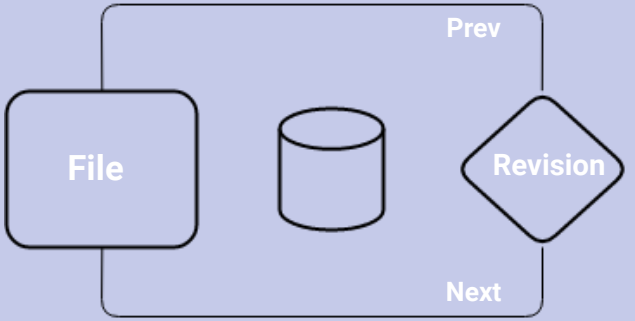
Rapidità e semplicità

Contro:

Alta probabilità di errori nel ripristinare un file

Local Version Control System

Database locale per la memorizzazione delle modifiche effettuate sui file



Pro:

Possibilità di ricreare lo stato di un file in qualsiasi momento dello sviluppo

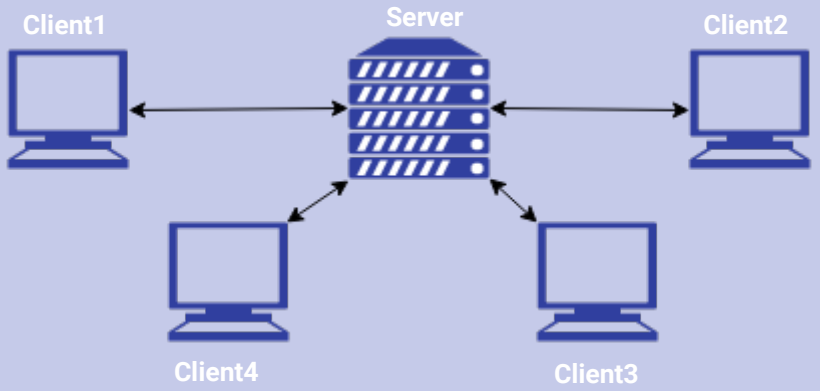
Contro:

Eterogeneità LVCS

Impossibile gestire progetti in team

Centralized **V**ersion **C**ontrol **S**ystem

Server centrale per memorizzare tutte le versioni dei file controllati



Pro:

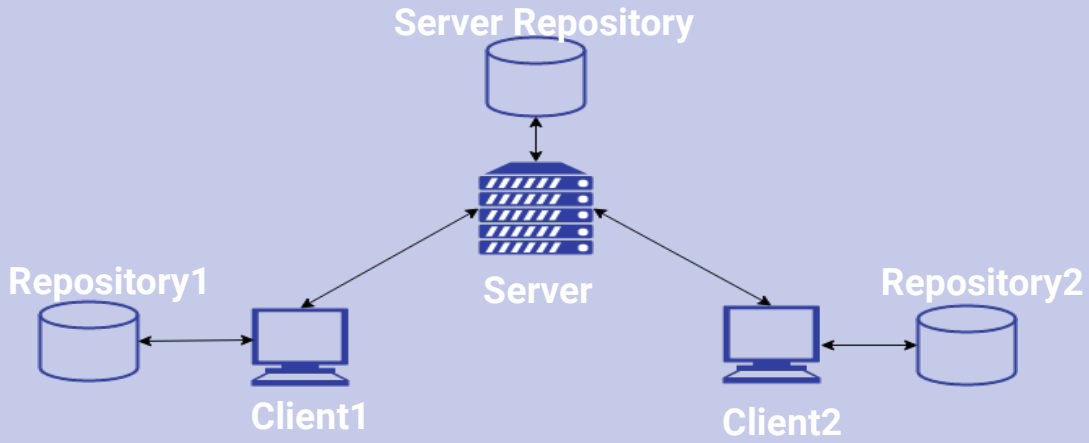
- Amministrazione controllata
- Collaborazione

Contro:

- Server centrale -> **Bottleneck**
- Guasto server centrale -> **Perdita dati**

Distributed **V**ersion **C**ontrol **S**ystem

Ogni membro del team crea una copia locale del contenuto del server



Pro:

- Personale responsabilizzato
- Rete non gerarchizzata

Contro:

- Gestione dei conflitti

Distributed Version Control and Code Management System

Sviluppato da Linus Torvalds dopo la fine del suo raporto clientelare con BitKeeper
Licenza **GNU GPL revision 2**

Obiettivi:

Velocità
Ottimo supporto allo sviluppo non-lineare
Completamente distribuito
Efficenza nella gestione di grandi progetti

Stati fondamentali

Working Directory:

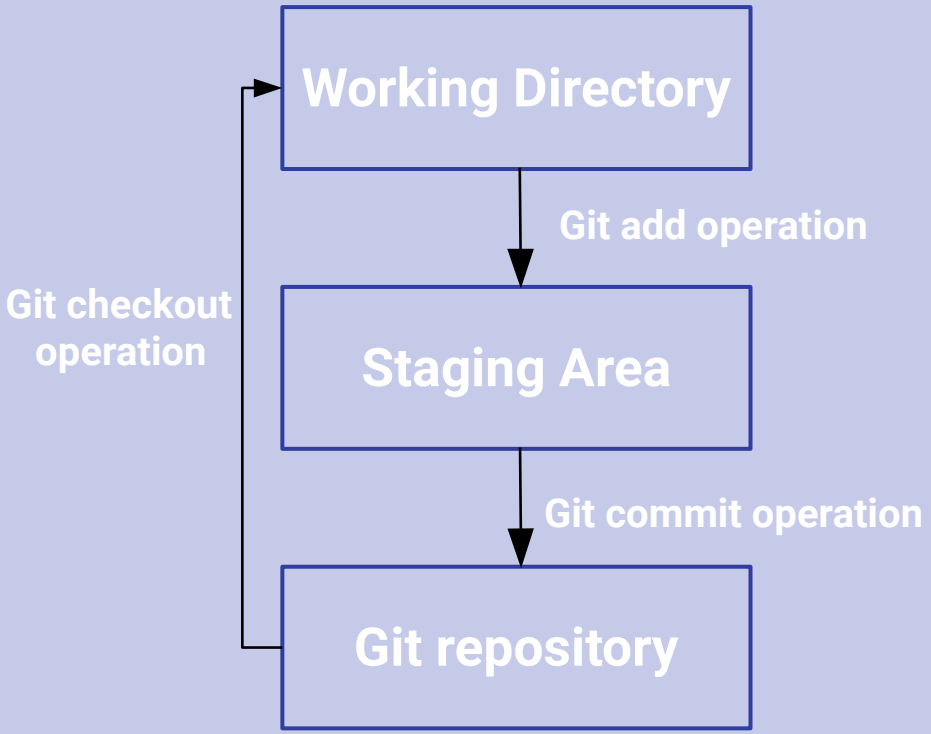
File sui quali si andrà a lavorare
Estratti dal database compresso di git

Staging Area:

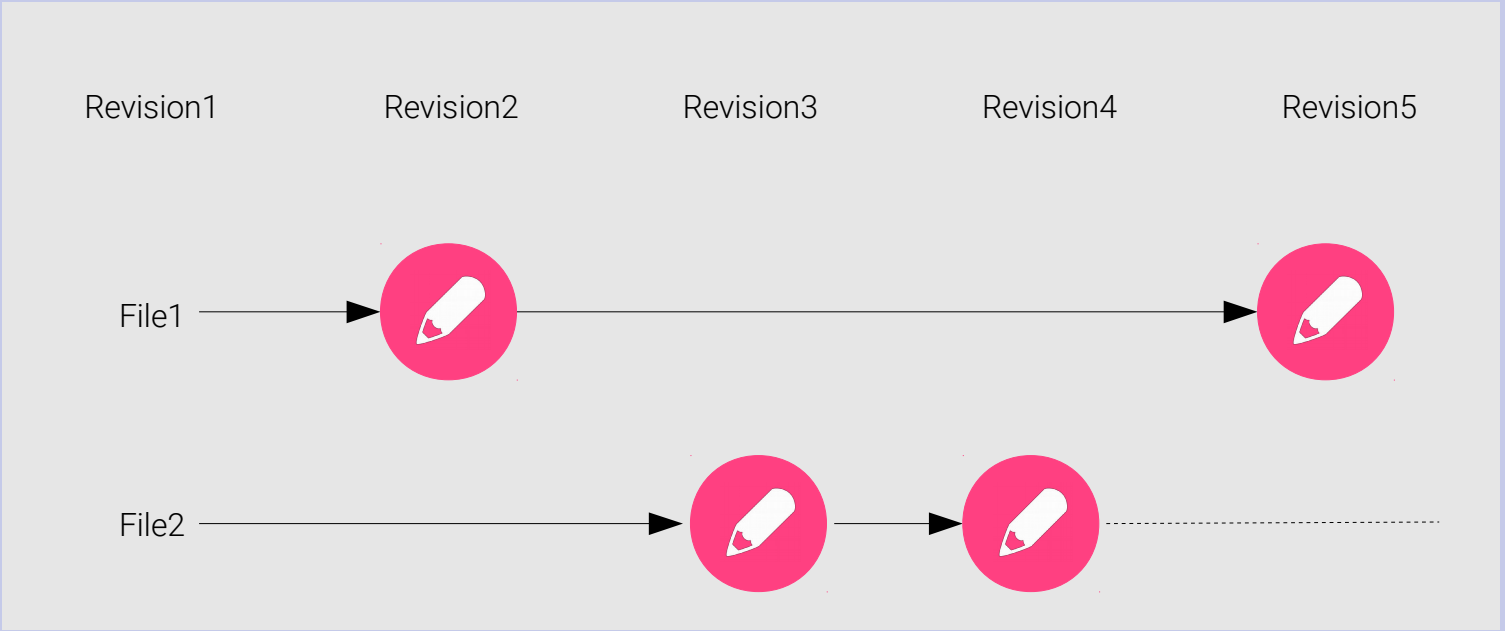
Rappresentata da un file
Memorizza le informazioni per il commit

Git repository (Local):

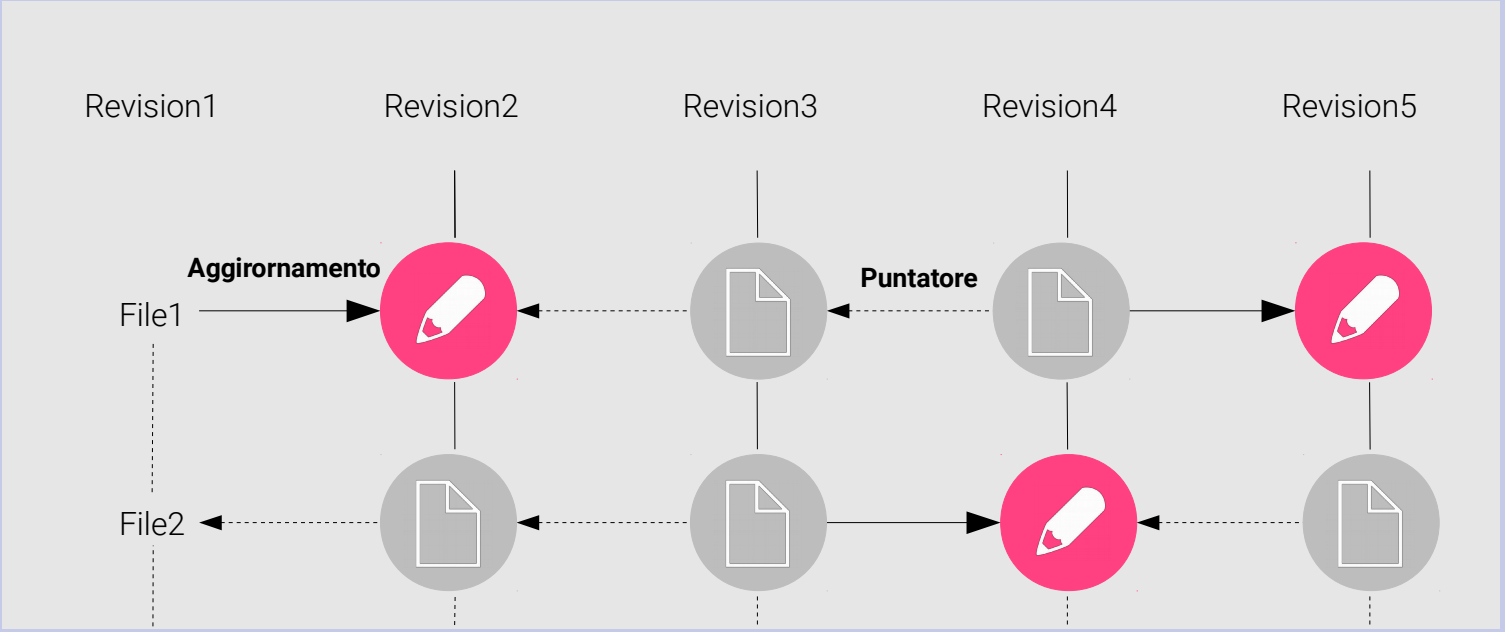
Memorizza metadati e oggetti
Parte più importante di Git



La maggior parte dei VCS tracciano i cambiamenti sui file tramite liste



Git considera i file nel loro complesso -> Ogni revisione è uno **Snapshot** del sistema



Generalmente Git non ha bisogno di acquisire informazioni dalla rete

La maggior parte delle operazioni può essere effettuata in **locale**

Es:

Per esaminare la “storia” di un progetto non c'è l'obbligo di contattare un server remoto. Git recupera le informazioni direttamente dal repository locale.

Pro:

Overhead di rete minimizzato

Ogni cosa in Git, prima di essere memorizzata, è soggetta ad un'analisi di integrità:

- Generazione valore di **checksum** che verrà usato per referenziare i dati.
- Funzione **hash**/generatrice : **SHA-1**

Es:

```
String test = ”IngegneriaDelSoftware”;
String checksum = sha1( test );
System.out.println(checksum) ----> 8cbf1f12caf478d20f0a686b649c70fa4538af26
```

Pro:

Impossibile cambiare i dati senza che Git ne sia a conoscenza

GIT Setup

Scaricare ed installare GIT dal seguente URL -> **https://git-scm.com/downloads**

Dopo aver effettuato l'installazione, bisogna controllare se GIT è stato aggiunto alle **variabili d'ambiente** relative al sistema operativo che si sta utilizzando, in caso non lo sia, bisogna aggiungerlo manualmente.

Per verificare se GIT è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

git ***--version***

```
[ivan18@localhost ~]$ git --version
git version 2.5.5
```

Configurazione

Prima di iniziare ad usare GIT è consigliato settare alcune variabili di configurazione tramite un apposito tool che GIT stesso mette a disposizione: **git config**

Esistono vari livelli di configurazione.

E' possibile accedere alle variabili di uno specifico livello di configurazione passando al comando git config una specifica opzione:

System Level	->	git config --**system** *nome_variabile*
Global Level	->	git config --**global** *nome_variabile*
Project specific Level	->	git config *nome_variabile*

I path di configurazione dipendono sia dal sistema operativo in uso,
sia dal livello di configurazione su cui si intende operare:

1 - **/etc/gitconfig** valori per tutti gli utenti del sistema e tutti i loro repository

2 - **~/.gitconfig** o **~/.config/git/config** valori specifici per l'utente

3 - **.git/config** valori relativi al singolo repository

E' possibile cambiare i path dei file di configurazione tramite: *git config* ***-f*** *new_config_path*

Configurazione dettagli utente

Siamo adesso in grado di settare alcune informazioni che GIT userà per identificare l'utente

git config --global **user.name** "Ivan18"

git config --global **user.email** “ivan.bruno18@gmail.com”

Per visualizzare le variabili di configurazione: *git config* ***--list***

Inizializzazione di un repository

Esistono 2 possibili inizializzazioni per un repository

git init --bare

Inizializza un repository **senza** working directory

Per convenzione il nome per questo tipo di repository termina con **.git**

```
[ivan18@localhost IngegneriaDelSoftware]$ mkdir gitWithoutWorkingDirectory.git
[ivan18@localhost IngegneriaDelSoftware]$ cd gitWithoutWorkingDirectory.git/
[ivan18@localhost gitWithoutWorkingDirectory.git]$ git --bare init
Inizializzato un repository Git in /home/ivan18/development/IngegneriaDelSoftware/gitWithoutWorkingDirectory.git/
```

```
[ivan18@localhost gitWithoutWorkingDirectory.git]$ tree -a .
.
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

9 directories, 13 files
```

Analizzando il risultato del comando precedente, si può notare come il repository venga inizializzato esplicitamente nella directory dalla quale è stato lanciato il comando stesso

Inizializzazione di un repository

git init

Inizializza un repository **con** la relativa working directory

```
[ivan18@localhost IngegneriaDelSoftware]$ mkdir gitWithWorkingDirectory
[ivan18@localhost IngegneriaDelSoftware]$ cd gitWithWorkingDirectory/
[ivan18@localhost gitWithWorkingDirectory]$ git init
Inizializzato un repository Git in /home/ivan18/development/IngegneriaDelSoftware/gitWithWorkingDirectory/.git/
```

```
[ivan18@localhost gitWithWorkingDirectory]$ tree -a .
.
└── .git
    ├── branches
    ├── config
    ├── description
    ├── HEAD
    ├── hooks
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    │   ├── prepare-commit-msg.sample
    │   ├── pre-push.sample
    │   ├── pre-rebase.sample
    │   └── update.sample
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags

10 directories, 13 files
```

Analizzando il risultato del comando precedente, si può notare come, senza l'opzione --**bare**, il repository venga inizializzato in una directory nascosta, generalmente denominata **.git**

Working directory?

Directory corrente

Effettuare cambiamenti in un repository

Dopo aver correttamente inizializzato un repository con la relativa working directory, tramite il comando **git status**, è possibile verificare lo stato dei file gestiti da GIT

Andiamo quindi a creare un primo file nella working directory del respository e successivamente verifichiamo il suo stato con l'apposito comando:

```
[ivan18@localhost gitWithWorkingDirectory]$ touch README.md
[ivan18@localhost gitWithWorkingDirectory]$ git status -s README.md
?? README.md
```

Il primo valore ci inidica lo stato del relativo file.
In questo caso il valore **??** per il file README.md indica che GIT **non** gestisce il file.

Aggiungiamo il file appena creato ai file gestiti da git tramite il comando

git add <nome_file>

Verifichiamone lo stato tramite il comando

git status

```
[ivan18@localhost gitWithWorkingDirectory]$ git add README.md
[ivan18@localhost gitWithWorkingDirectory]$ git status -s
A  README.md
```

In questo caso il valore **A** per il file README.md indica che il file è stato **aggiunto** correttamente nell'**area di staging**.

Creazione snapshot del repository

A questo punto siamo pronti per creare uno snapshot del repository.

La creazione di uno snapshot viene effettuata tramite il comando **git commit**

L'operazione principale del commit consiste nel prelevare tutti i file presenti nell'area di staging e memorizzare permanentemente un “instantanea” del progetto nel database

```
[ivan18@localhost gitWithWorkingDirectory]$ git commit -m 'Aggiunto README.md'
[master (root-commit) 04235f7] Aggiunto README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

E' possibile visualizzare il ristulatato di un commit tramite **git log**

```
[ivan18@localhost gitWithWorkingDirectory]$ git commit -m 'Aggiunto README.md'
[master (root-commit) 04235f7] Aggiunto README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

Più nello specifico il comando **git show *commit_id*** permette di visualizzare nel dettaglio il commit riferito da commit_id

```
[ivan18@localhost gitWithWorkingDirectory]$ git show 04235f70c6fe1ea5ad4b704dc6b9717a369e5cf9
commit 04235f70c6fe1ea5ad4b704dc6b9717a369e5cf9
Author: ivan18 <ivan.bruno18@gmail.com>
Date:   Tue Nov 29 14:57:57 2016 +0100

    Aggiunto README.md

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..e69de29
```

Gestione di un repository remoto

Uno degli aspetti più interessanti di GIT è la possiilità di gestire un repository remoto senza necessità di strumenti aggiuntivi.

GIT infatti mette a disposizione il comado **git remote**

Vorremmo quindi sincronizzare i cambiamenti fatti fin'ora in un repository remoto

Concettualmente assumeremo che:

- Il repository creato con l'opzione --bare sia un **repository remoto**
- Il repository creato senza l'opzione --bare sia un **repository locale**

Procediamo con l'individuazione dei path associati ai repository che andremo ad utilizzare

Repository locale:

```
[ivan18@localhost gitWithWorkingDirectory]$ pwd
/home/ivan18/development/IngegneriaDelSoftware/gitWithWorkingDirectory
```

Repository remoto:

```
[ivan18@localhost gitWithoutWorkingDirectory.git]$ pwd
/home/ivan18/development/IngegneriaDelSoftware/gitWithoutWorkingDirectory.git
```

Associazione di un repository remoto ad un repository locale

Per effettuare l'associazione tra repository locale e remoto utilizziamo il comando:

git **remote add** <shortname> *<url>*

Nel nostro caso lanciamo: git remote add origin path_git_without_working_directory

```
[ivan18@localhost gitWithWorkingDirectory]$ git remote add origin ../gitWithoutWorkingDirectory.git/
```

Aggiornamento repository remoto

Per condividere i cambiamenti precedentemente effettuati sul repository locale utilizziamo il comando:

git **push** <remote_*name*> <*branch_name*>

Tale comando funziona solo se:

- Si hanno i **privilegi** per scrivere sul repository remoto
- Repository locale **sincronizzato** con il repository remoto

```
[ivan18@localhost gitWithWorkingDirectory]$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 221 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /home/ivan18/development/IngegneriaDelSoftware/gitWithoutWorkingDirectory.git
 * [new branch]      master -> master
```

Sincronizzazione di un repository remoto con uno locale

E' possibile recuperare informazioni su file aggiunti in un repository remoto utilizzando il comando:

git fetch <remote_name>

Nota:

Non modifica il contenuto dei file presenti nella working directory

Sincronizzare il contenuto della working directory vuol dire effettuare un operazione di **merge**
Tale operazione avviene automaticamente tramite l'utilizzo del comando

git pull <remote_name> <branch_name>

Inizializzazione ed associazione di un nuovo repository

```
[ivan18@localhost gitWithWorkingDirectory2]$ pwd
/home/ivan18/development/IngegneriaDelSoftware/gitWithWorkingDirectory2
[ivan18@localhost gitWithWorkingDirectory2]$ git init
Inizializzato un repository Git in /home/ivan18/development/IngegneriaDelSoftware/gitWithWorkingDirectory2/.git/
[ivan18@localhost gitWithWorkingDirectory2]$ git remote add origin ../gitWithoutWorkingDirectory.git
```

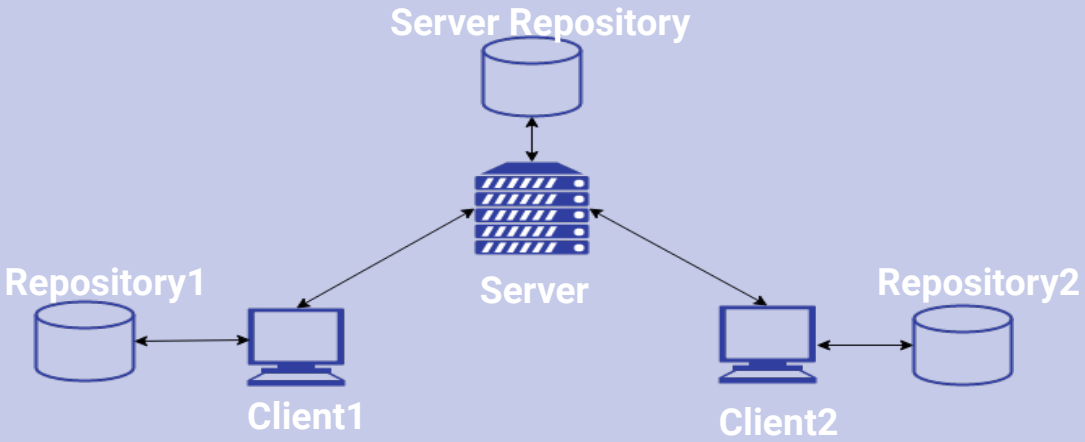
Sincronizzazione repository e working directory

```
[ivan18@localhost gitWithWorkingDirectory2]$ git pull origin master
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Da ../gitWithoutWorkingDirectory
 * branch            master     -> FETCH_HEAD
 * [nuovo branch]    master     -> origin/master
[ivan18@localhost gitWithWorkingDirectory2]$ ls -axn
totale 12
drwxrwxr-x. 3 1000 1000 4096 15 dic 14.21 .
drwxrwxr-x. 5 1000 1000 4096 15 dic 13.21 ..
drwxrwxr-x. 8 1000 1000 4096 15 dic 14.21 .git
-rw-rw-r--. 1 1000 1000    0 15 dic 14.21 README.md
```

Descrizione scenario

Repository remoto contenente unicamente il file vuoto README.md

Team formato da 2 membri che hanno clonato il repository remoto



Generazione conflitto

Suppoiniamo adesso che il membro1 del team modifichi il file README.md

```
[ivan18@localhost gitWithWorkingDirectory]$ echo "Messaggio dal membro1" >> README.md
[ivan18@localhost gitWithWorkingDirectory]$ git status -s README.md
 M README.md
```

E che dopo la modifica sincronizzi il proprio repository con quello remoto

```
[ivan18@localhost gitWithWorkingDirectory]$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 273 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /home/ivan18/development/IngegneriaDelSoftware/gitWithoutWorkingDirectory.git
   1719169..b80f51f  master -> master
```

Generazione conflitto

Il membro2 allo stesso tempo sta modificando il file README.md

```
[ivan18@localhost gitWithWorkingDirectory2]$ echo "Messaggio dal membro2" >> README.md
```

Ma al momento della sincronizzazione git produrrà un errore

```
[ivan18@localhost gitWithWorkingDirectory2]$ git push origin master
To ../gitWithoutWorkingDirectory.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to '../gitWithoutWorkingDirectory.git'
suggerimento: Updates were rejected because the remote contains work that you do
suggerimento: not have locally. This is usually caused by another repository pushing
suggerimento: to the same ref. You may want to first integrate the remote changes
suggerimento: (e.g., 'git pull ...') before pushing again.
suggerimento: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Gestione del conflitto

Il membro2 a questo punto effettua l'operazione di pull

```
[ivan18@localhost gitWithWorkingDirectory2]$ git pull origin master
Da ../gitWithoutWorkingDirectory
 * branch            master     -> FETCH_HEAD
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Merge automatico fallito; risolvi i conflitti ed eseguire il commit
del risultato.
```

Git avvisa l'utente che non è in grado di effettuare l'operazione di merge per cui bisognerà effettuare **manualmente** la risoluzione del conflitto

Gestione del conflitto

Analizziando il contenuto del file README.md il membro2 si accorge di alcune modifiche

```
[ivan18@localhost gitWithWorkingDirectory2]$ cat README.md
<<<<<<< HEAD
Messaggio dal membro2
=======
Messaggio dal membro1
>>>>>>> 9a1c775e8860cc1abbfd0c31858503797cc79772
```

Git inserisce un opportuno tag per indicare la sezione di codice coinvolta nel conflitto

La prima sezione del tag contiene la versione presente sul **branch corrente**
La seconda contiene la versione di codice presente sul **branch master**

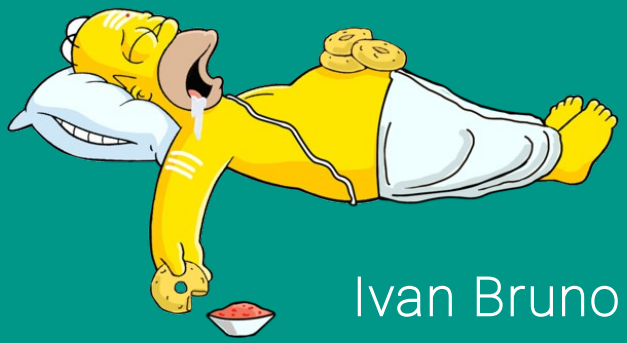
Risoluzione del conflitto

A questo punto il membro2 non deve far altro che seguire dei semplici passi:

1 - mantenere solo la sezione di codice che gli interessa, cancellando il resto
2 - creare un nuovo commit per notificare il fix agli altri membri del team
3 - risincronizzare il tutto con il repository remoto

```
[ivan18@localhost gitWithWorkingDirectory2]$ cat README.md
Messaggio dal membro2
[ivan18@localhost gitWithWorkingDirectory2]$ git add README.md
[ivan18@localhost gitWithWorkingDirectory2]$ git commit -m 'Risoluzione conflitto README.md'
[master 53830cf] Risoluzione conflitto README.md
[ivan18@localhost gitWithWorkingDirectory2]$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 416 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To ../gitWithoutWorkingDirectory.git
   9a1c775..53830cf  master -> master
```

Grazie per l'attenzione



Ivan Bruno