

Git e GitHub

Indice

- I. Introduzione al controllo del codice sorgente
 - A. Storia e concetti dietro il controllo del codice sorgente
 - B. Controllo versioni centralizzato distribuito
- II. Software Git
 - A. Cos'è git? concetti base ed architettura
 - B. flusso di lavoro di git: creazione di un repository (adding, committing)
 - C. Comandi git
 - D. Master vs branch
 - E. creazione di una branch e spostamenti tra branch
 - F. merging branches e risoluzione conflitti
 - G. Tagging

Indice

III. Piattaforma GitHub

- A. Cos'è github?
- B. Github in pratica: controllo delle versioni distribuito
- C. Clonare un repository remoto
- D. Fetching / Pushing to a remote repository

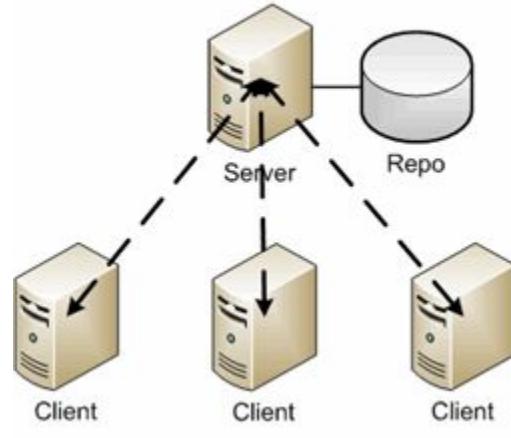
Cos'è un “sistema di controllo delle versioni”?

- Una strategia per gestire file e cartelle
- Tenere traccia delle modifiche nel tempo
- Recuperare versioni precedenti
- controllo delle versioni è un sottoinsieme di VCS (Version Control System)

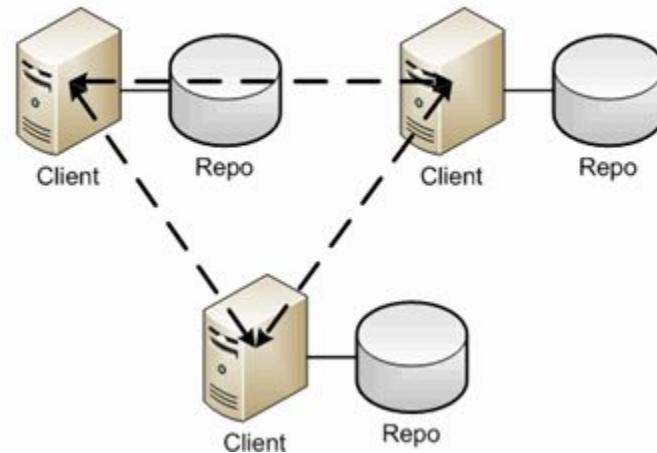
Sistema di controllo delle versioni distribuito e centralizzato

I files e cartelle possono risiedere centralizzate su un unico server, come nel caso del utilizzo di github, oppure possono essere memorizzati su più' computer che agiscono sia da client sia da server (actor). nei sistemi centralizzati le modifiche rimangono sul computer dello sviluppatore finchè quest'ultimo non propaga le modifiche su tutti gli altri computer.

Traditional



Distributed





git

Che cos'è git?

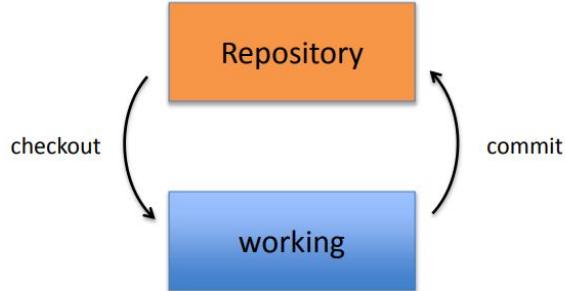
- Creato da Linus Torvalds nel aprile 2005
- Creato per rimpiazzare BitKeeper nella gestione dei cambiamenti del kernle di linux
- Un programma per il controllo delle versioni a riga di comando
- Utilizza somme di controllo per assicurare l'integrità dei dati
- Un sistema di controllo distribuito (come BitKeeper)
- Cross-platform
- Open source e gratuito

Git - Caratteristiche

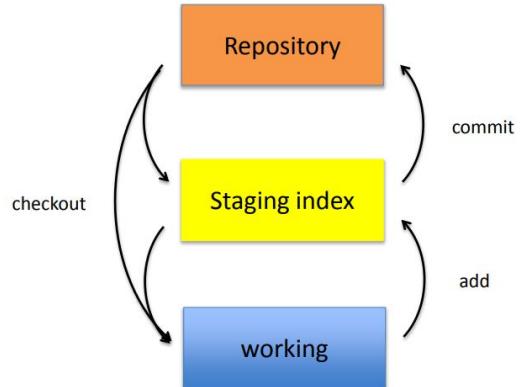
- Non ha bisogno di un server centrale
- Può funzionare senza connessione a internet
- Non presenta un single-point of failure
- Gli sviluppatori possono lavorare indipendentemente e fare il merge dei loro lavori successivamente
- Ogni copia di un repository git può essere usato sia come server sia come client
- Git tiene traccia dei cambiamenti, non delle versioni

Git

Altri VCS



Le modifiche vanno ad aggiornare immediatamente il repository centrale, solitamente centralizzato.



Git dispone di un area di stage. Questo significa che se vengono effettuati 100 cambiamenti, vi è la possibilità di spezzare questi 100 cambiamenti in 10, 20 o più commits ognuno con i loro commenti e la loro dettagliata spiegazione.

Flusso di lavoro di Git

1. inizializzare un nuovo repository
 - a. Attraverso il comando: `git init`

```
deuterio211@PCandrea:~/gitExample$ git init
Initialized empty Git repository in /home/deuterio211/gitExample/.git/
```

2. Creare i files attraverso un editor nella cartella
3. Aggiungere ogni cambiamento allo staging index
 - a. attraverso il comando `git add .`

```
deuterio211@PCandrea:~/gitExample$ touch ilMioFilePreferito.txt
deuterio211@PCandrea:~/gitExample$ ls
ilMioFilePreferito.txt
deuterio211@PCandrea:~/gitExample$ git add .
```

Flusso di lavoro di Git

4. Visualizzare i files in stage

- attraverso il comando: `git status`

```
deuterio211@PCandrea:~/gitExample$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

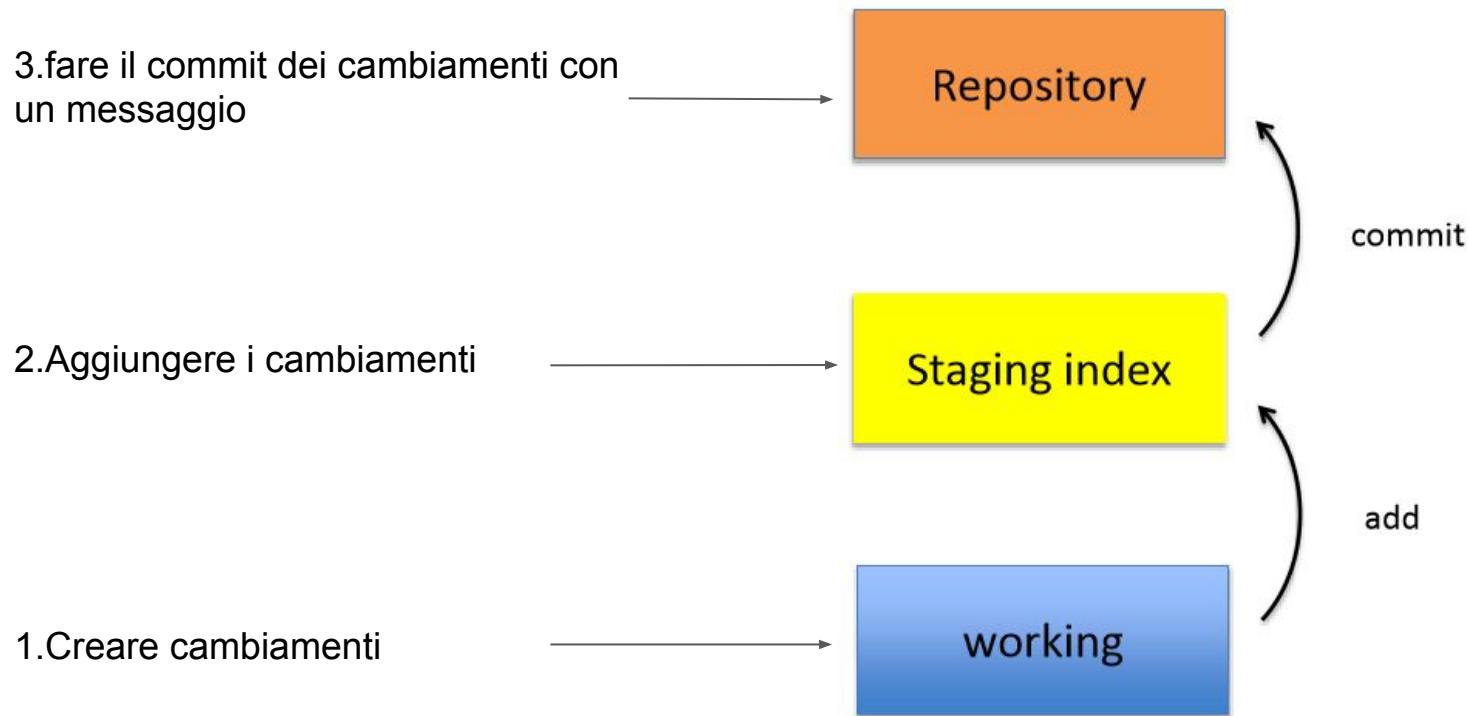
    new file:   ilMioFilePreferito.txt
```

5. Fare il commit dei cambiamenti al repository

- attraverso il comando: `git commit -m "messaggio importante qui"`

```
deuterio211@PCandrea:~/gitExample$ git commit -m "il mio primo commit"
[master (root-commit) a735f8a] il mio primo commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ilMioFilePreferito.txt
```

Dopo l'inizializzazione di un nuovo repository



Comandi Git

- **git status** : consente di vedere quali files sono in fase di stage
- **git diff** : compara i files nel repository e quelli nella cartella di lavoro
 - git diff --staged : compara i file nello stage e quelli nel repository
 - git diff [filename] : compara solo quel file invece che l'intera cartella di lavoro

```
deuterio211@PCandrea:~/gitExample$ echo Questo è un messaggio > ilMioFilePreferito.txt
deuterio211@PCandrea:~/gitExample$ git diff
diff --git a/ilMioFilePreferito.txt b/ilMioFilePreferito.txt
index e69de29..acc3b44 100644
--- a/ilMioFilePreferito.txt
+++ b/ilMioFilePreferito.txt
@@ -0,0 +1 @@
+Questo è un messaggio
```

- **git rm -f [filename]** : cancella un file dall'area di stage

```
deuterio211@PCandrea:~/gitExample$ git rm -f ilMioFilePreferito.txt
rm 'ilMioFilePreferito.txt'
```

Comandi Git

- **git mv [filename1] [filename2]** : sostituisce il filename2 nell'area di stage

```
deuterio211@PCandrea:~/gitExample$ git mv ilMioVecchioFile.txt ilMioNuovoFile.txt
deuterio211@PCandrea:~/gitExample$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    ilMioFilePreferito.txt -> ilMioNuovoFile.txt
```

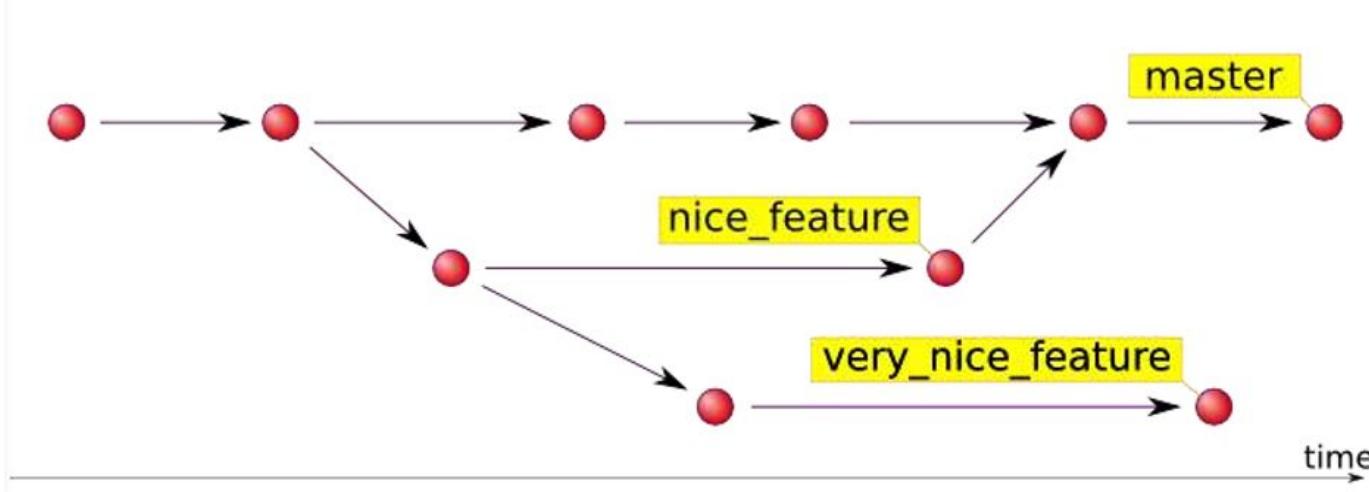
- **git reset HEAD [filename]** : cancella i cambiamenti nell'area di stage

```
deuterio211@PCandrea:~/gitExample$ git reset HEAD ilMioFileNuovissimo.txt
deuterio211@PCandrea:~/gitExample$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ilMioFileNuovissimo.txt
```

Branches

- Permettono di sperimentare nuove idee
- Se un'idea non funziona, è necessaria cancellare solo il branch, senza intaccare i cambiamenti nel branch master
- Se funziona, permette di unire (merge) il branch al master branch
- C'è solo una cartella di lavoro



Comandi applicabili ai branch

- **git branch**

- visualizza i branch

```
deuterio211@PCandrea:~/gitExample$ git branch
* master
```

- **git branch [nome_branch]**

```
deuterio211@PCandrea:~/gitExample$ git branch nuovoBranch
deuterio211@PCandrea:~/gitExample$ git branch
* master
  nuovoBranch
```

- **git checkout [nome_branch]**

- cambia il branch attuale

```
deuterio211@PCandrea:~/gitExample$ git checkout nuovoBranch
Switched to branch 'nuovoBranch'
```

Comandi applicabili ai branch

- **git diff [branch1]...[branch2]**

- confronta i due branch

```
deuterio211@PCandrea:~/gitExample$ git diff master nuovoBranch
diff --git a/ilMioFileNuovissimo.txt b/ilMioFileNuovissimo.txt
new file mode 100644
index 000000..ad2f398
--- /dev/null
+++ b/ilMioFileNuovissimo.txt
@@ -0,0 +1 @@
+Questo file è nel nuovo branch
```

- **git merge [branch_da_unire]**

- unisce il branch indicato al branch master

```
deuterio211@PCandrea:~/gitExample$ git merge nuovoBranch
Updating 9eb2e83..e9be435
Fast-forward
  nuovoFileBranch | 0
   1 file changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 nuovoFileBranch
```

Comandi applicabili ai branch

- **git branch -m/--move [vecchio_nome] [nuovo_nome]**

- rinomina un branch

```
deuterio211@PCandrea:~/gitExample$ git branch vecchioBranch
deuterio211@PCandrea:~/gitExample$ git branch --move vecchioBranch nuovoBranch
deuterio211@PCandrea:~/gitExample$ git branch
* master
  nuovoBranch
```

- **git branch -D [nome_branch]**

- cancella il branch con tutti i commit associati

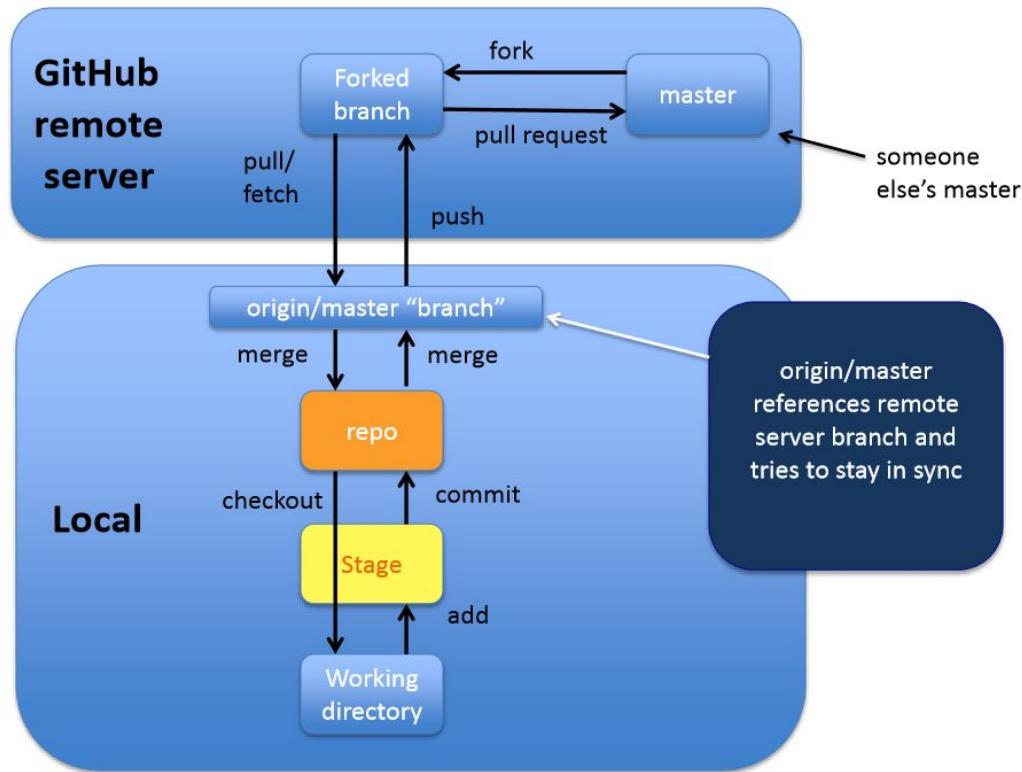
```
deuterio211@PCandrea:~/gitExample$ git branch -D nuovoBranch
Deleted branch nuovoBranch (was 9eb2e83).
deuterio211@PCandrea:~/gitExample$ git branch
* master
```

GitHub



GitHub

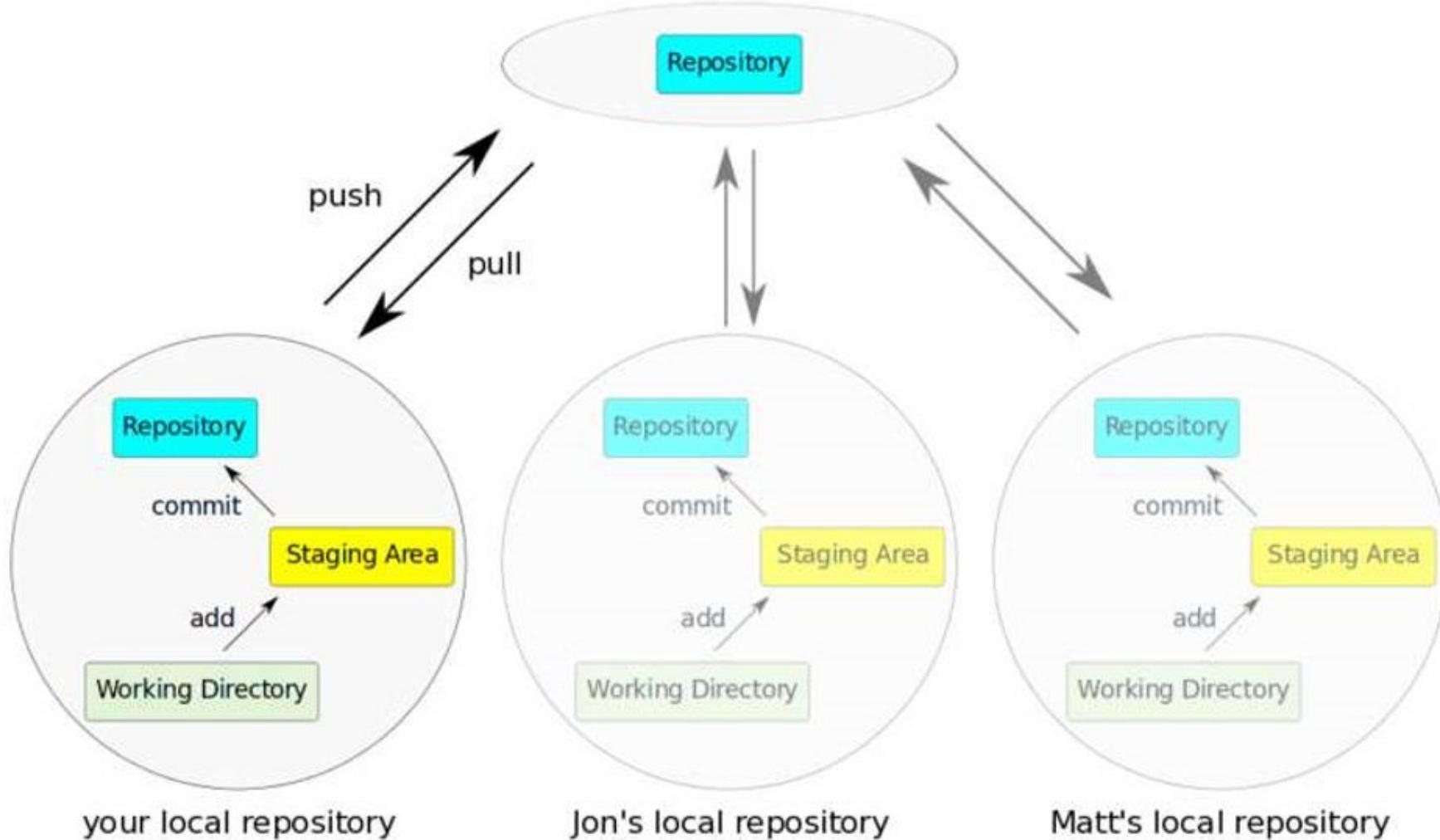
- Una piattaforma per ospitare codice da repository git
- Nato nel 2008
- Git Host più popolare
- permette agli utenti di collaborare su progetti da qualsiasi parte
- Free to start



Cosa importante da ricordare

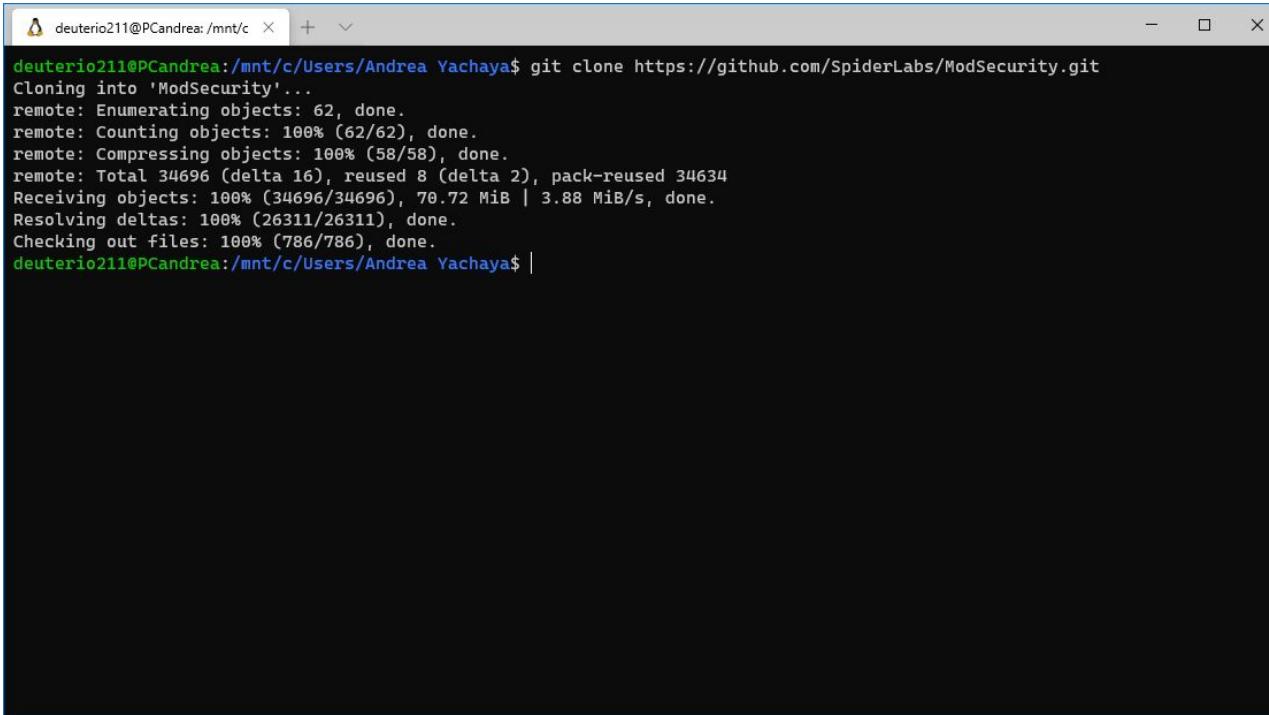
A volte gli sviluppatori scelgono di utilizzare GitHub come repository centralizzato dove ognuno può mandare (commits) i cambiamenti, ma non deve essere per forza GitHub. Esistono altri repository centralizzati come GitLab, BitBucket, SorceForce, AWS CodeCommit, ecc...

Central Server



Clonare un repository remoto

- **git clone [url]** :
 - Permette di scaricare un repository remoto sul proprio sistema locale.



A screenshot of a terminal window titled "deuterio211@PCandrea: /mnt/c". The window displays the output of a "git clone" command:

```
deuterio211@PCandrea:/mnt/c/Users/Andrea Yachaya$ git clone https://github.com/SpiderLabs/ModSecurity.git
Cloning into 'ModSecurity'...
remote: Enumerating objects: 62, done.
remote: Counting objects: 100% (62/62), done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 34696 (delta 16), reused 8 (delta 2), pack-reused 34634
Receiving objects: 100% (34696/34696), 70.72 MiB | 3.88 MiB/s, done.
Resolving deltas: 100% (26311/26311), done.
Checking out files: 100% (786/786), done.
deuterio211@PCandrea:/mnt/c/Users/Andrea Yachaya$ |
```

Come aggiungere un repository remoto?

- **git remote add [alias] [url]**
 - Questo comando stabilisce una connessione... nessun file è copiato o spostato
 - è possibile avere più di un repository remoto collegato alla tua cartella locale
- **git remote**
 - visualizzata i repository remoti collegati

```
deuterio211@PCandrea:~/gitExample$ git remote add FirstPush https://github.com/deuterius/FirstPush.git
deuterio211@PCandrea:~/gitExample$ git remote
FirstPush
```

Come aggiungere un repository remoto?

- `git push [local_branch_alias] [branch_name]`
 - aggiorna il repository remoto

```
deuterio211@PCandrea:~/gitExample$ git push https://github.com/deuterius/FirstPush.git master
Username for 'https://github.com': deuterius
Password for 'https://deuterius@github.com':
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 271 bytes | 271.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/deuterius/FirstPush.git
  6ea2cd6..5db3175  master -> master
```

Come aggiungere un repository remoto?

- `git fetch [remote_repo_name]`
 - aggiorna i nuovi files dal repository remoto

```
deuterio211@PCandrea:~/gitExample$ git fetch https://github.com/deuterius/FirstPush.git
Username for 'https://github.com': deuterius
Password for 'https://deuterius@github.com':
From https://github.com/deuterius/FirstPush
 * branch           master      -> FETCH_HEAD
```

Tagging

Git ha la l'abilità di segnare particolari importanti nella storia del sorgente, come ad esempio una versione di rilascio (v1.0, 2.0)

- **git tag**
 - visualizza i tag esistenti

```
deuterio211@PCandrea:~/gitExample$ git tag
1.0
2.0
2.1
2.1.2
2.2
2.4
```

Tagging

Esistono due tipologie di tag:

- **lightweight**: puntatore ad uno specifico commento.
 - Corrisponde a un SHA memorizzato in un file
 - `git tag [nome_tag]`

```
deuterio211@PCandrea:~/gitExample$ git tag nuovoTag
deuterio211@PCandrea:~/gitExample$ git tag
nuovoTag
```

- **annotated**: un oggetto memorizzato nel database di git.
 - composto da un file SHA, tagger name, email, date, message e può essere firmato e verificato con GNU Privacy Guard (GPG)
 - `git tag -a [nome_tag] -m [messaggio]`

```
deuterio211@PCandrea:~/gitExample$ git tag -a nuovissimoTag -m "nuovissima versione"
deuterio211@PCandrea:~/gitExample$ git tag
nuovissimoTag
nuovoTag
```

Tagging

- `git show [nome_tag]`
 - visualizza una versione prolissa dei tag esistenti

```
deuterio211@PCandrea:~/gitExample$ git show nuovissimoTag
tag nuovissimoTag
Tagger: Andrea Yachaya <deuterio211@gmail.com>
Date:   Tue May  5 13:00:45 2020 +0200

nuovissima versione

commit 9eb2e8343fc211c52390f1539bd37c392758da66 (HEAD -> master, tag: nuovoTag, tag: nuovissimoTag)
Author: Andrea Yachaya <deuterio211@gmail.com>
Date:   Tue May  5 11:51:29 2020 +0200

    aggiornamento

diff --git a/ilMioFilePreferito.txt b/ilMioNuovoFile.txt
similarity index 100%
rename from ilMioFilePreferito.txt
rename to ilMioNuovoFile.txt
```