



GIT

Repository

Un repository (repo) o git-project è l'insieme di file e folder associati ad un progetto. Comprende anche lo storico delle modifiche.

Crea nuovo (locale)


```
git init
```

Clona esistente (remoto)

```
git clone https://github.com/auridevil/iis_classes_2019_src.git  
cd iis_classes_2019
```

Branching

Un branch è un ramo dell'albertatura della storia del codice
il branch principale è **master**

Graph	Description	Commit	Author
	master origin/master origin/HEAD dummy master commit	653e27b	auridevil
	origin/branchdemo branchdemo this is a branch demo	38e9abb	auridevil
	rock'n'rolling	db9c55b	auridevil
	chore: intro and first section	f65d605	auridevil
	[chore] initial commit	47ee630	auridevil

Branch del repo

Tutti i branch del repo

```
git branch
```

Cambia branch attivo

```
git checkout branchdemo
```

Branch del repo

Crea nuovo branch

```
git branch funzionalita1425
```

Crea nuovo branch e utilizza

```
git checkout -b funzionalita1425
```

Aggiungi

L'aggiunta di un set di modifiche è in due fasi: staging e commit.
git add aggiunge alcune modifiche allo stage

Aggiungi modifica di un file

```
git add PITCHME.md
```


Aggiungi

Aggiungi tutte le modifiche

```
git add .
```

Aggiungi modifiche di un folder

```
git add assets/*
```

NB la cancellazione è una modifica. Solitamente il renaming / move è considerata cancellazione + aggiunta.

Commit

git commit salva lo snapshot delle modifiche (sul branch corrente) e completa il tracciamento. Commit con editor

```
git add .
```

Commit inline

```
git commit -m "this is a inline commit message"
```

Commit message Best practice

Siamo arrivati a considerare importanti:

- usare l'imperativo
- max 50 chars
- no punti finali
- dichiarare le modifiche fatte ad alto livello
- includere eventuali riferimenti a documentazione (e.g. jira code, tiketing...)

Commit message

Best practice

```
git commit -m "[type][branch] What is done"
```

type è il tipo di modifica:

- **feat**: funzionalità nuova
- **fix**: correzione errore
- **chore**: piccola sistemazione
- **test**: copertura del codice
- **doc**: documentazione

Se il branch è master, si esclude dal commit message.

Status

```
git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file: game.txt
```

Diff

Mostra le differenze non ancora nell'area di staging (pre *git add*)

```
git diff
```

Mostra le differenze tra staging e l'ultima modifica

```
git diff --staged
```

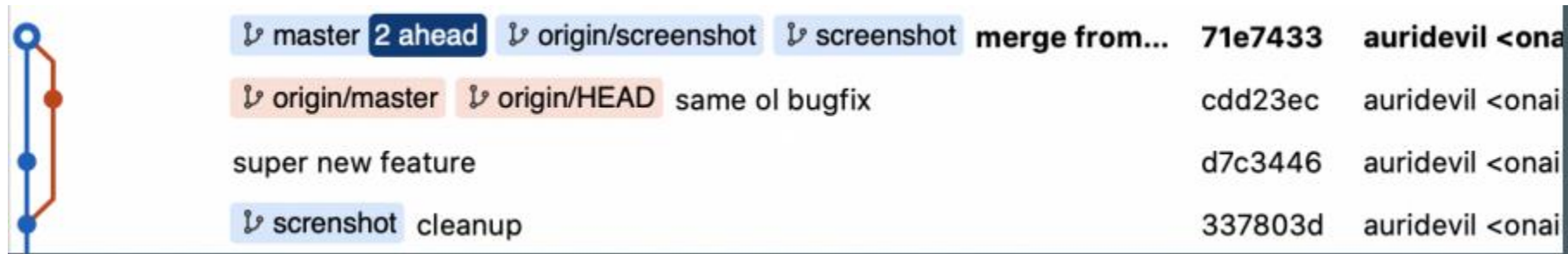
Unstage

Rimuovi un file dall'area di staging, ma mantieni le modifiche:

```
git reset game.txt
```

Merge (locale)

Combina le modifiche fatte su due branch differenti. E' direzionale: merge **of** un branch **into** un'altro branch.



Porta le modifiche di branch1 in branch2

```
git checkout branch2  
git merge branch1
```

Porta le modifiche di **branch2** in master

```
git checkout branch2  
git merge master  
git checkout master  
git merge branch2
```

Porta le modifiche di branch1 in branch2

```
git merge master
CONFLICT (add/add): Merge conflict in sample2.txt
Auto-merging sample2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

file in conflitto:

```
><<<<<<<< HEAD
code from the current branch
=====
code from master branch
<>>>>>>>> master
```

Conflitti

modificare a mano il file in conflitto per ottenere

```
code mix from both branches
```

aggiungere la modifica al branch per completare il merge

```
git add sample2.txt  
git commit -m "merge from master"
```

Remote

E' possibile aggiungere un server remoto (se non si è partiti da un clone) per usare git distribuito:

```
git remote add origin https://github.com/...
```

il nome di default del remote è *origin*. Per verificare l'aggiunta:

```
git remote -v  
origin https://github.com/...
```

Pull

Per ottenere le modifiche dal server su master:

```
git pull origin master
```

o (unsafe):

```
git pull
```

Pull

per pullare un branch:

```
git pull origin branch3
```

git pull prova a mergiare in locale quello che c'è sul server, se serve più controllo si può usare git fetch (avanzato)

```
git fetch
```

Push

Per inviare le modifiche al server da locale, sul branch remoto di master:

```
git push origin master
```

per pushare su un branch particolare, remoto:

```
git push origin branch3
```

Merge (remote)

Porta le modifiche di branch2 in master, con fast-forward

```
git checkout master
git pull origin master
git checkout branch2
git merge master
git push origin branch2
git checkout master
git merge branch2
git push origin master
```


Merge (remote)

Porta le modifiche di **branch2** in **master**, con fast-forward

```
git checkout master
git pull origin master
git checkout branch2
git merge master
git push origin branch2
git checkout master
git merge branch2
git push origin master
```

Merge (remote)

Porta le modifiche di **branch2** in **master**, con conflitto:

```
git checkout master
git pull origin master
git checkout branch2
git merge master
<fix conflict>
git add .
git commit -m "merge from master"
git push origin branch2
git checkout master
git merge branch2
git push origin master
```

Rebase

Prendi tutti i cambiamenti fatti su un branch e portali su un altro

```
git rebase branch3
```

Warning: il rebase è pericoloso da usare. E' facile alterare la salute del repository, l'utilizzo è sconsigliato salvo rari casi di emergenza (seguire le guide ufficiali online).

Incompleti

Capita di voler mettere da parte delle modifiche incomplete, per farlo si usa lo *stash* (una specie di cut'n'paste gigante).
Aggiungi tutte le modifiche (non staged) allo stash:

```
git stash
```

Riapplica le modifiche nello stash

```
git stash apply
```

Storico

E' possibile vedere lo storico dei commit da cli:

```
Git log
```

File ignorati

E' quasi sempre necessario NON versionare alcuni file e folder, ad esempio dove sono contenute password o stringhe di connessione ai database, i file compilati, i file degli editor e le dipendenze.

Per questo si mette nella root del progetto un file *.gitignore* dove si indicano i file (regex allowed) da ignorare. Git non vedrà modifiche a questo tipo di file, né traccierà la loro esistenza.

[.gitignore list](#)

Tag

E' possibile aggiungere dei tag ad un determinato commit su un determinato branch (solitamente master), ad esempio per tracciare un rilascio, o una breaking change:

```
git tag -a v1.5.2 -m "Version 1.5.2 on westeurope + northeurope servers  
PROD"  
git push --tags origin master
```

