Git e Github

Githb è un software che si utilizza per condividere codice e fare versionamento (VCS).

Esistono diversi software di versionamento, quello più utilizzato è Git.

# Bash Git

Tasto dx ovunque -> Git bash here: ti crea una bash di Git che come percorso ha il percorso di dove hai fatto tasto dx: per spiegarsi se sul Desktop fai questo procedimento otterrai:

putrinog@SDGNB681 MINGW64 ~/Desktop

Creare una cartella:

putrinog@SDGNB681 MINGW64 ~/Desktop

$ mkdir nome\_cartella

## Navigare fra directory

Per navigare fra le directory basterà utilizzare il commando **cd** (sta infatti per change directory)

## File contenuti all’interno di una cartella

Con il commando **ls -a**

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ ls -a

./ .git/ LICENSE debug.log file01.txt temp/

../ .gitignore README.md file01 file02.txt

## Contenuto di un file

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ cat file01.txt

hello

new line from dev

## Inizializzazione

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git init

Questo comando crea una cartella temporanea di nome .git all’interno della cartella repo (potrebbe essere file nascosto quindi per vederlo attivare visualizzazione file nascosti da view sulla cartella). Questa conterrà tutto lo storico dei salvataggi effettuati durante lo sviluppo

## Status

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Mostra lo status attuale del nostro Progetto

## Aggiungere allo staging

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git add nome\_file

In questo modo aggiungi allo staging ( si chiama anche index ) di Git un file presente nella cartella.

Per aggiungere tutti i file presenti nella cartella:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git add .

Oppure si può decidere di aggiungere tutti I file di un certo tipo, ad esempio tutti I file di tipo testo

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git add \*.txt

## Commit

### Commit senza specifiche

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git commit

Se non passiamo altri parametri verrà aperto l’editor impostato di default all’interno della shell.

### Digitare all’interno della shell

**I + *testo che si vuole digitare***

Una volta terminato esc e poi per quittare digitare :wq

### Commit con specifiche

Altrimenti per semplificarsi la vita

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git commit -m "titolo commit" -m "eventuale descrizione aggiuntiva"

### Flusso

Per capire quindi quello che abbiamo fatto con commit e add:

1. siamo nella working directory: si fanno modifiche
2. con git add si aggiungono all’area di staging di Git
3. con git commit andiamo poi a crerae una fotografia dello stato attuale dei nostri file e quindi una storia

Ovviamente si è può percorrere questo flusso in entrambi i versi, quindi sia con prima add e poi commit che rimuovere dalla staging e annullare dei commit

## Storico dei commit

Il comando che si può utilizzare per vedere lo storico dei vari commit è **git log:**

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git log

commit 8df1b9abc56e17ba8deea9a61e2d0682a864332f (HEAD -> master, origin/master)

Author: L025434 <putrino\_giovanni@network.lilly.com>

Date: Mon Jan 17 11:54:59 2022 +0100

Created License

commit 35c6ee46c279c1e62c6dea7aa6707e923dd4039f

Author: L025434 <putrino\_giovanni@network.lilly.com>

Date: Mon Jan 17 11:46:58 2022 +0100

Created README.md

commit f788f405dc16af621513e0b9ae041c8c8e33bc67

Merge: a8918a4 f66060d

Author: Giovanni Putrino <putrinog@sdg.local>

Date: Mon Jan 17 10:22:22 2022 +0100

Merge branch 'dev'

commit f66060d0cb0df6b8c92500734a0994cdf54cde32

Author: Giovanni Putrino <putrinog@sdg.local>

Date: Fri Jan 14 09:49:55 2022 +0100

Come possiamo vedere Git salava all’interno del suo DB ogni commit che eseguiamo in un formato chiave-valore, dove la chiave sarà una stringa di 40 caratteri. Questa servirà ad evitare che ci siano dei conflitti fra le varie commit

### Head

HEAD è un puntatore che punta sempre all’ultimo commit effettuato nel branch in cuisiamo in questo momento.

Possiamo vederlo all’interno dello stesso file HEAD all’interno della cartella .git :

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

cat .git/HEAD

ref: refs/heads/master

Vediamo che c’è il riferimento all’ultimo branch.

Per avere l’output del log in versione più compatta:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git log --oneline

8df1b9a (HEAD -> master, origin/master) Created License

35c6ee4 Created README.md

f788f40 Merge branch 'dev'

f66060d new line from dev

a8918a4 (fix) addedd new line

44b0a73 updated gitignore

96e9acb gitignore

a8317e6 Second commit

6234a9d First commit

Se invece vogliamo l’ultimo commit in fondo e il primo in cima **git commit - -oneline - -reverse**

## Commit più rapidi

Per evitare di dover fare prima git add e poi git commit si fa direttamente:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git commit -am "Titolo commit"

## Annullare un git add

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git restore --staged nome\_file

## Annulare una commit

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git reset

Cancella interamente lo storico. Da utilizzare solamente se stiamo lavorando in locale e non abbiamo ancora pushato in remoto.

Questa accetta varie opzioni. Le più conosciute sono:

1. **git reset - -soft**: Si elimina la commit e si riporta lo stato dei file alla staging area. Se si va a vedere lo status i nuovi file saranno verdi
2. **git reset [**- -**mixed]:** Mixed l’ho messo fra quadre perché quando si scrive git –reset questa è l’opzione standard: Con questa si vanno a rimuovere anche i file che precedentemente erano nella staging. Ti riporta quindi nella working directory. Se fai status i file te li darà rossi: si dovrà fare git add . per aggiungerli nella staging
3. **git reset - -hard:** Si cancella tutti i file che non sono presenti nei commit precedenti. Ovvero si elimina e si torna al commit precedente, si ripristina lo stato alla lcommit precedente, prima che facessimo modifiche.

In pratica se non indichi l’ID della commit da resettare di default fa l’ultima. In generale quindi sarebbe git reset –opzioni ID\_commit. Essendo l’ID complesso quello che conviene fare è sfruttare il puntatore HEAD: questo starà sempre sull’ultima commit, quindi si specificherà il contatore e di quante commit siamo interessati andare indietro:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git rest --soft HEAD^

Se si vuole andare alla commit precedente. Se si vuole andare a due commit prima:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git rest --soft HEAD^^

In generale senno:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git reset --soft HEAD~N

Dove con N si indicherà il numero di quante commit si vuole tornare indietro.

### Revert

Se non si vuole ricorrere a reset altrimenti si può utilizzare:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git revert

In questo modo si cancella la commit e se ne crea una nuova andando a ripristinare lo stato precedente

## Modificare una commit

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git commit --amend

Aprirà l’editor e ci permetterà di modificare l’ultima commit eseguita

## Evitare che alcuni file siano tracciati da Git (.gitignore)

Creando un file .gitignore all’interno della nostra repository:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git .gitignore

Ora si dovrà aprire questo file e scriversi all’interno tutti i nomi dei file che si desidera siano ignorati da Git. In generale varrà la stessa logica di git add, ovvero se nel file .gitignore andassi a scrivere \*.log non verrebbero considerati tutti i file di tipo log presenti all’interno della cartella.

### Non escludere alcuni file

Mettiamo che vogliamo nascondere tutti i file di tipo .log tranne due: per escluderli dal file .gitignore e quindi fare in modo che Git gli consideri andrà scritto all’interno del file .gitignore il nome del file preceduto da punto esclamativo:

\*.log

!debug.log

In questo modo si ignora tutti i file di log tranne il file debug.log

Oltre a file si possono ignorare anche intere cartelle, sempre semplicemente inserendo il loro nome all’interno del file .gitignore

## Branch

La potenza di Git non è solo nel creare istantanee in diversi punti temporali ma da la possibilità di lavorare in versioni parallele dei nostri file in modo da tenere separato lo sviluppo.

I branch sono molto utili per separare differenti versioni di sviluppo ed evitare di sporcare la versione stabile che normalmente abbiamo in master (aka main)

Master è il branch di default che viene aggiunto da git appena facciamo git init

Un branch si può quindi definire come un’etichetta cge funziona come puntatore e punta l’ultimo commit che abbiamo eseguito, mentre siamo sotto quel branch.

Ogni volta che andremo a creare un nuovo commit il branch in cui saremo in quel momento seguirà lo sviluppo di quel ramo e punterà sempre all’ultimo commit eseguito

### Graph

Un modo per vedere l’alberatura del nostro progetto con le varie commit e i vari branch è il seguente

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git log --all --decorate --oneline --graph

Invece di riscriverlo tutte le volte possiamo utilizzare un alias

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ alias graph='git log --all --decorate --oneline --graph'

(NB per utilizzare l’alias devi eseguire almeno una volta il comando così com’è prima)

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ graph

\* 8df1b9a (HEAD -> master, origin/master) Created License

\* 35c6ee4 Created README.md

\* f788f40 Merge branch 'dev'

|\

| \* f66060d new line from dev

\* | a8918a4 (fix) addedd new line

|/

\* 44b0a73 updated gitignore

\* 96e9acb gitignore

\* a8317e6 Second commit

\* 6234a9d First commit

Se invece digiti

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git branch

fix

\* master

Mostra la lista di tutti i branch presenti e con l’asterisco viene indicato il branch in cui siamo in questo momento

### Creare nuovi Branch

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git branch nome\_branch

### Muoversi fra I vari branch

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git checkout 'nome\_branch'

## Editor vi

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ vi nome\_file

In questo modo si possono modificare i file con editor della bash. Quindi come precedentemente per scrivere all’interno **i+***testo*.

Per salvare e uscire esc+:wq

## Differenza fra due branch

Per vedere le differenze fra due branch:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

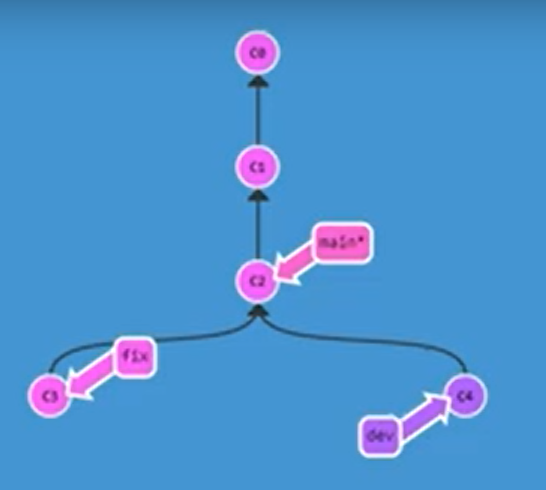
$ git diff nomebranch1..nomebranch2

## Unire due branch

Nel caso si voglia unire due branch ci sono sostanzialmente due casistiche:

1. Ultimo commit del branch1 è parente diretto del commit nel branch2
2. Non sono parenti diretti

Per parente diretto si intende che non ci sono diramazioni ma sono collegati direttamente



Per esempio fix e dev sono parenti diretti di master (main) , fix e dev non sono fra loro parenti diretti

#### Caso parenti diretti:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git merge fix

Stiamo quindi facendo un merge fra il branch master dove ci troviamo in questo momento (in azzurro in parentesi) e il branch fix. Dopo questo comando il branch master e il branch fix saranno sullo stesso commit

### Vedere branch che sono stati uniti

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git branch --merged

fix

\* master

#### Caso parenti non diretti

Quando si va a fare un merge fra due branch che non risultano essere parenti Git andrà a verificare la commit comune ai due (nel caso del disegno la C2) e farà poi un paragone fra le commit successive che sono state fatte in modo separato nei due branch ed alla fine andrà ad eseguire il merge

Il merge te lo farà quindi fare ma ti segnalerà un conflitto fra i due branch. Infatti andando poi a vedere lo status vedremo che prima di committare occorrerà risolvere i conflitti ( per esempio possiamo annullare il merge con **git merge - -abort** )

## Eliminare un branch

Per esempio dopo questa merge il branch fix non risulta essere più di nessun aiuto e andrà allora eliminato.

Per eliminare un branch:

* Se abbiamo fatto un merge precedentemente con questo branch che vogliamo eliminare

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git branch -d nome\_branch

* Se non abbiamo fatto un merge precedente con questo branch

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git branch -D nome\_branch

## Risolvere conflitti durante git merge

Andando ad utilizzare editor vi git ci indica quali sono i conflitti:

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ vi file01.txt

hello

<<<<<<<<<<HEAD

New line from fix

=================

New line from dev

>>>>>>>>>>dev

Ci separa quindi ambienti tramite gli uguali distinguendo quello contenuto nel master e quello contenuto in dev

Per eliminare righe o lo faccio entrando all’interno dell’editor con i e eliminandole manualmente o altrimenti muovendomi con il cursore e facendo d+d

Una volta effettuate le modifiche git add . e poi committare.

# GitHub

1. Crei cartella all’interno di GitHub
2. Una volta creata verremmo reindirizzati su pagina in cui GitHub da tutte info per andare ad aggiungere progetto creato in locale sul nostro pc
3. putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git remote add origin URL

Con URL si intende lo URL della cartella su GitHub che puoi ottenere dal tasto code su GitHub

Possiamo vedere info sul server remoto

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git remote show

Origin

Oppure più nel dettaglio visto che si chiamo origin

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git remote show origin

\* remote origin

Fetch URL: https://github.com/giovanniputrino-sdg/corso-git.git

Push URL: https://github.com/giovanniputrino-sdg/corso-git.git

HEAD branch: master

Remote branch:

master tracked

Local branch configured for 'git pull':

master merges with remote master

Local ref configured for 'git push':

master pushes to master (up to date)

Abbiamo quindi tutte le info su dove avviene push fetch etc.

Si è creato quindi un collegamento fra il locale e il server di GitHub.

## Rinominare Server remoto

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git remote rename nome\_server\_old nome\_server\_new

Come si può vedere dall’inizio del punto 3 inizialmente l’avevo chiamato origin

## Cancellare server remoto

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git remote remove nome\_server

## Sincronizzare il local con la repository remota

Se per esempio sono stati aggiunti dei file sul repository e vogliamo avere una sincronizzazione con quello che abbiamo in locale possiamo utilizzare due metodi

### Fetch

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git fetch

Facendo poi status ci indica di quante commit siamo indietro rispetto al server remoto ed occorrerà allora fare una merge

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git merge origin/master

Con il nuovo branch creato che prende il nome di origin/master

### Pull

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git pull

Con pull si sincronizza e allo stesso tempo si fa anche la merge.

## Clonare repository da GitHub

Nel caso in cui volessimo lavorare in locale sul progetto di qualcun altro si potrà clonare la cartella

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git clone URL

Dove con URL si ha l’URL della cartella su GitHub. Si può anche specificare un nuovo nome con

putrinog@SDGNB681 MINGW64 ~/Desktop/repo (master)

$ git clone URL new\_name

## Fork

Un fork è un clone della repository che ci interessa sul nostro account: è come se si effettuasse il comando clone ma a livello di server questa volta. Avremo in questo modo la possibilità di effettuare modifiche e fare poi quello che si chiama pull request, ovvero notifiche al manteiner di modifiche da parte nostra riguardo il progetto. Se queste pull request sono accettate diventiamo contributor del progetto.

1. Vai sul progetto che ti interessa
2. Lo forki
3. Lo cloni in locale in un nuova cartella
4. Fai le modifiche che vuoi
5. Le aggiungi e le committi

Una volta fatto tutto questo devi fare pull request (menu a tendina della repository forkato che si trova ora sul tuo account e clicchi “contribute” e open pull request

## Sincronizzare Fork con repo originaria

Apri la bash di Git della repo forkata:

putrinog@SDGNB681 MINGW64 ~/Desktop/new\_project (master)

$ git remote -v

Altro modo più conciso per farti vedere dove push e pull. Poi fai:

putrinog@SDGNB681 MINGW64 ~/Desktop/newproject (master)

$ git remote add upstream URL\_repository

Dove upstream è un nome qualsiasi. In questo modo se rifai git remote -v vedrai che è stato aggiunto l’url della repository originale e questo ci permetterà di fare fetch e/pull per sincronizzare le repository.

Dopo diche occorrerà fare **git push** per fare passare questa sincronizzazione dal locale a remoto