Git

Strumento che registra le differenze fra diverse versioni di uno o più file. NON SALVA l'intero contenuto del file modificato.

Il **COMMIT** è il salvataggio del nuovo stato del **repository** che è l'insieme dei file e cartelle che vengono tracciati.

Prima del commit c'è una fase di **STAGING** nella quale ci sono i file che sono stati modificati/aggiunti/rimossi al repository.

selezione file da tracciare --> modifica/creazione/... files --> staging area (selezione file da salvare)--> commit (salvataggio effettivo dei file selezionati in staging)

CONFIGURAZIONE GLOBALE di Git

Set Default User Name --> `git config --global user.name` "stringa con nome utente"

Set Default User Mail --> `git config --global user.mail` "email di contatto dell'utente"

Set newline character --> (unix-linux)`git config --global core.autocrlf input`

(windows) `git config --global core.autocrlf false`

(per standardizzare il carattere di fine riga nel codice)

INIZIALIZZAZIONE di un repository

Posizionarsi nella cartella che si vuole tenere traccia quindi impartire il comando `git init`
Per **cambiare User Mail** del repository occorre fare `git config user.mail "email"`
Per **ignorare** il tracciamento di **file** che sono generabili (.class, .o, .exe ...) o altri tipi di file e cartelle occorre creare un file `.gitignore` nella quale scrivere la path delle cartelle e/o le estensioni che git deve ignorare nel tracciamento.

STATO del repository
Con il comando `git status` vediamo quali file sono stati modificati o aggiunti alla staging area.

AGGIUNTA file alla Staging Area
Con il comando `git add <file>` mette `<file>` nella staging area, se e solo se il file è stato modificato/è cambiato (cancellazione, creazione sono cambiamenti del file);
Si possono aggiungere anche più file in un unico comando passando i nomi dei file come argomenti.

Cancello file --> cancellazione vista come modifica --> staging area --> Commit (per cancellare effettivamente il file nel repo)

Rinominazione del file = creazione nuovo file con nome nuovo + cancellazione file con nome vecchio

RIMOZIONE FILES dalla Staging Area
Con il comando `git reset <files>` si va/vanno a togliere, se presente/i, dalla staging area `<files>`, in questo modo togliamo gli elementi le cui modifiche non vogliamo che vengano salvate nel prossimo commit. Questo ha senso per esempio se si vuole fare un commit separato per alcuni file che sono finiti dentro la staging area. (NON rimuove il file dal tracking).

CREAZIONE PUNTI DI SALVATAGGIO - COMMIT del repository

`git commit <files> -m "A message"`

(Esegue il commit delle modifiche a `<files>`, se non vengono specificati i files, il commit viene fatto su tutte le modifiche in staging area)

Oltre alle **modifiche applicate** al repository vengono salvate anche altre info:

- l'**autore** della modifica e **la sua mail** (per contattarlo per chiarimenti)
- un **messaggio** che riassuma le modifiche fatte (**OBBLIGATORIO**)
- **data e ora** (acquisite in automatico)
- **id univoco** (hash generato automaticamente)

Ricorda:
É buona norma fare frequenti commit di piccole dimensioni (piccole modifiche) con un messaggio chiaro e poco generico.
Se si omette `-m "<messaggio di commit>"` viene aperto un editor dove scrivere il messaggio, se non scrivo nulla il commit non viene eseguito.

VISUALIZZAZIONE STORIA COMMIT

`git log`

(Visualizza tutti i commit della linea di sviluppo corrente, scorrimento elenco con frecce per uscire tasto Q)

`git log --graph`

(Come sopra, con visualizzazione grafica dell'evoluzione del branch corrente; con `--all` anche gli altri branch)

BRANCH - LINEE DI EVOLUZIONE/LAVORO
Linea di sviluppo (ossia sequenza di commit successivi). Dal momento in cui si può tornare indietro nella storia dei salvataggi e ripartire a sviluppare, si può **creare** una **nuova linea di sviluppo**, che si diparte da quella originale e **procede parallelamente**.

Partendo da uno stesso commit quindi la linea di sviluppo iniziale si può scomporre in più linee di sviluppo permettendo cosi il lavoro in simultanea su linee diverse.

HEAD - Posizione nella linea di sviluppo
Paragonabile ad un cursore/puntatore che indica la attuale posizione all'interno della linea di sviluppo - branch.
Di norma la **head** si trova in fondo al **branch**, per muoversi tra i diversi commit si sposta la head.

MERGE Fusione di due branch (linee di sviluppo) in una sola.

VISUALIZZAZIONE DELLE DIFFERENZE fra due commit

`git diff`

(Mostra le differenze fra il working tree e l'ultimo commit, escludendo il contenuto della staging area)

`git diff FROM`

(Dove FROM è un riferimento ad un commit che voglio confrontare con il working tree)

`git diff FROM TO`

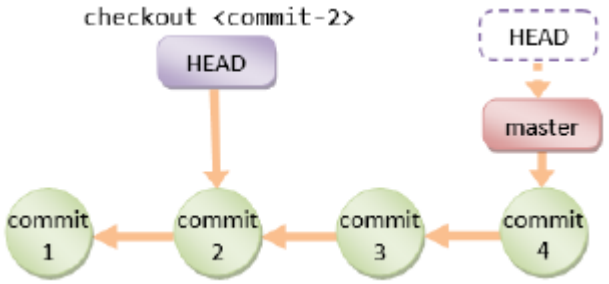
(Dove FROM e TO sono gli hash di due commit che voglio confrontare)

Per semplificare l'accesso agli ultimi commit effettuati, Git mette a disposizione la keyword

HEAD (ultimo commit) --> HEAD~N fa riferimento al N-esimo commit prima dell'ultimo
NAVIGAZIONE fra commit di un branch

`git checkout COMMITREF`

Spostarsi fra i commit e i **branch** tramite (`COMMITREF`) il loro **hash id** o **nome/riferimento**



(nome del branch o usando la keyword HEAD).

Quando si torna indietro nella storia, Git entra in modalità "detached HEAD": la "testa", ossia il commit a cui ci troviamo non sta alla fine del branch. I **commit** effettuati in **questa modalità vengono scartati**.

Per tornare alla **modalità attached**: `git checkout NOMEBRANCH` (spostarsi fra branch)
Il **nome del branch punta sempre all'ultimo commit** su quella linea di sviluppo

Ripristino stato file da commit

`git checkout COMMITREF -- FILENAME`

(Ripristina il file FILENAME prendendolo dal commit COMMITREF)

CREAZIONE BRANCH - Branching

`git checkout -b NOMEBRANCH`

permette di creare un nuovo branch a partire dalla posizione in cui ci si trova nella linea di sviluppo. Se ci si sposta nel branch attuale, la testa viene spostata, e il branch che viene creato si svilupperà a partire da quel commit che punta la testa.

Una volta creato il branch `NOMEBRANCH` i nuovi commit effettuati verranno salvati in questo nuovo branch (dopo la creazione **si è sul nuovo branch**).

Per cambiare branch `git checkout <nome branch>` (modalità attached poco sopra)

MERGE - Unione di più linee di sviluppo

`git merge branchname`

questo comando cerca di unire le modifiche di `branchname` al branch corrente, è necessario **prima spostarsi sul branch di destinazione** (con checkout);

Se non ci sono conflitti, tutti i commit di branchname vengono aggiunti al branch corrente
Viene creato un nuovo commit (merge commit), non cambiare il messaggio di commit predefinito.

In caso di conflitti occorre:

1) **risolverli manualmente** portando il codice allo stato desiderato
2) aggiungere i file in staging area
3) salvataggio modifiche con commit

CANCELLAZIONE BRANCH

`git branch -d <nome branch>`

(Cancella la linea di sviluppo <nome branch>, da usare se dopo un merge non serve più)

Git (GitHub)

CLONE

```
git clone URI localfolder
```

(Scarica l'intera storia del repository conservato in URI all'interno di localfolder, che diventa un repository Git)

Il **sotto-comando** `git remote` consente di gestire repository remoti. Ogni repository remoto ha un nome ed una URL.

- `git remote -v` elenca i repository remoti configurati
- `git remote add <name> <url>` aggiunge un nuovo repository remoto di nome `<name>` che punta ad `<url>`
- `git remote rm <name>` rimuove il repository remoto di nome `<name>`

BRANCH UPSTREAM

É possibile **configurare**, per ciascun branch, un **uri di un repository remoto** dove si andrà a **leggere e scrivere** a meno di diversa specifica.
Si usa l'opzione `-u` oppure `--set-upstream-to`

```
git branch -u remoteName/remoteBranch
```

D'ora in poi, tutte le operazioni accesso remoto lanciate dal branch corrente verranno mappate sul branch `remoteBranch` del repository remoto `remoteName`

Nel caso in cui il repository sia stato clonato e non inizializzato un riferimento al repository di origine viene automaticamente inserito fra i remote e configurato come upstream per tutti i branch col nome di origin.

VISUALIZZAZIONE DEI BRANCH REMOTI

```
git branch -a
```

(Elenca tutti i branch, inclusi quelli remoti)