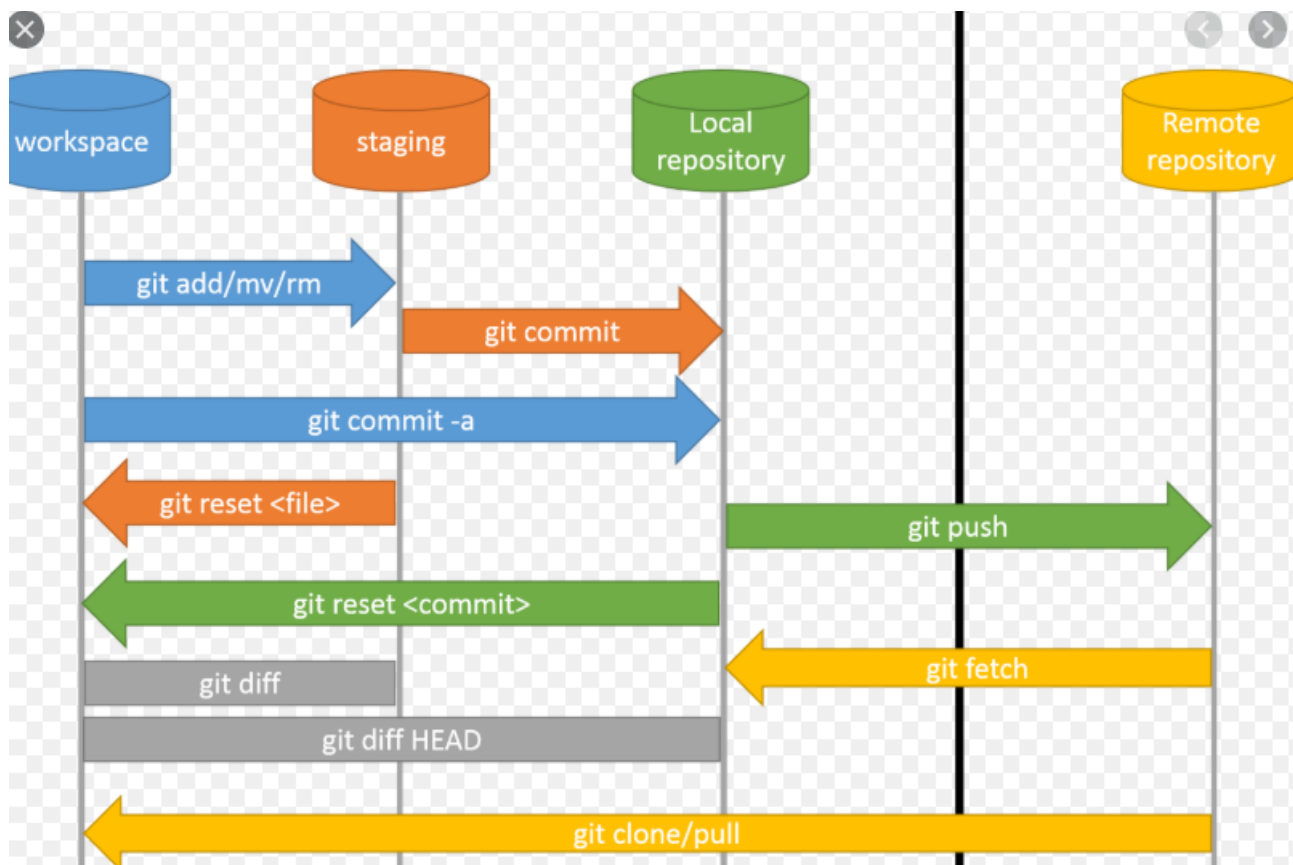
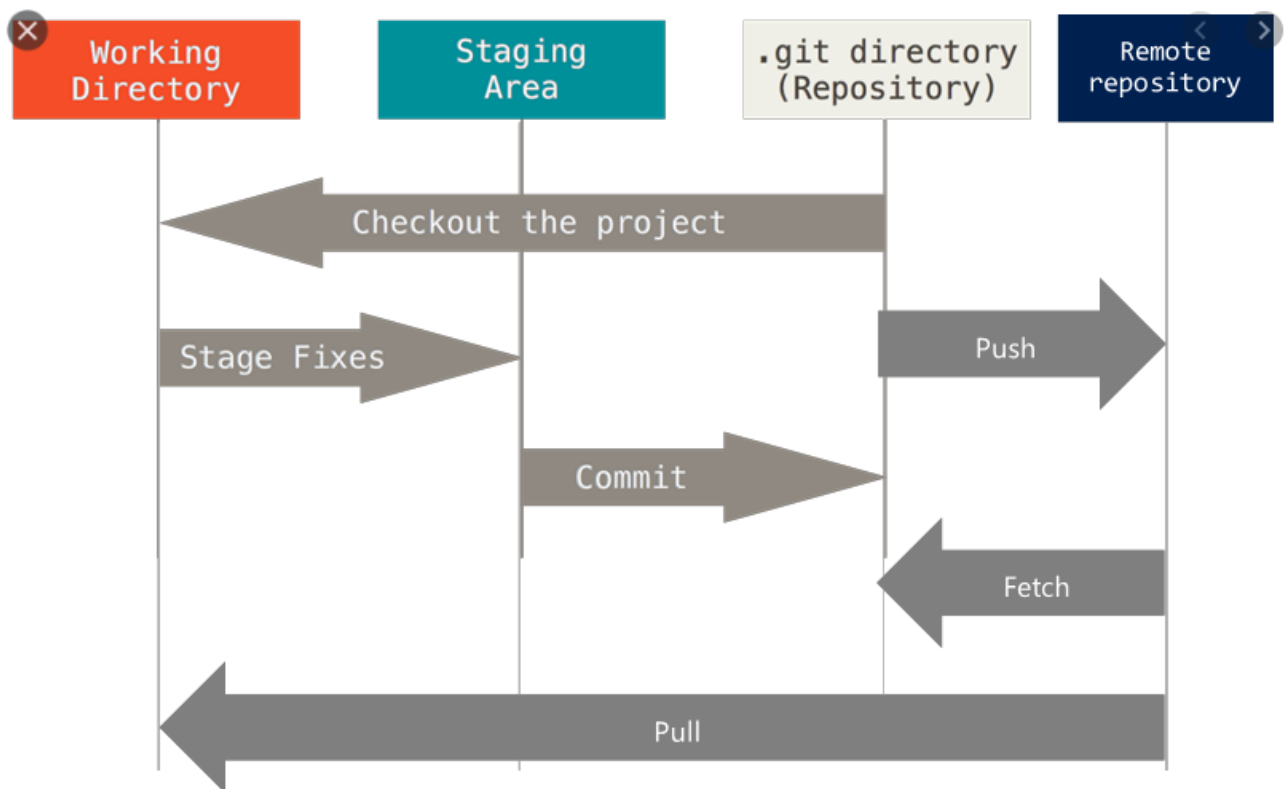
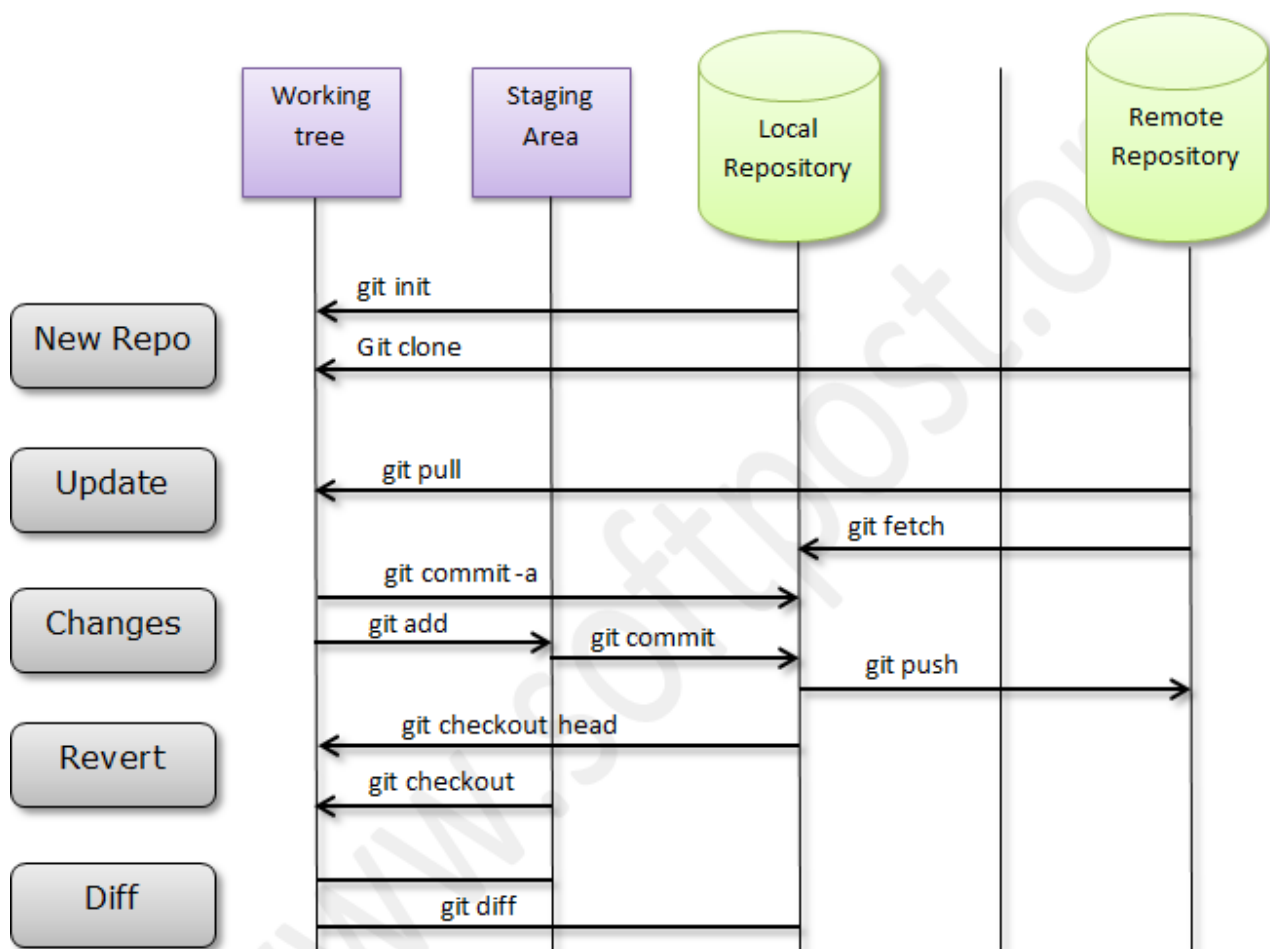
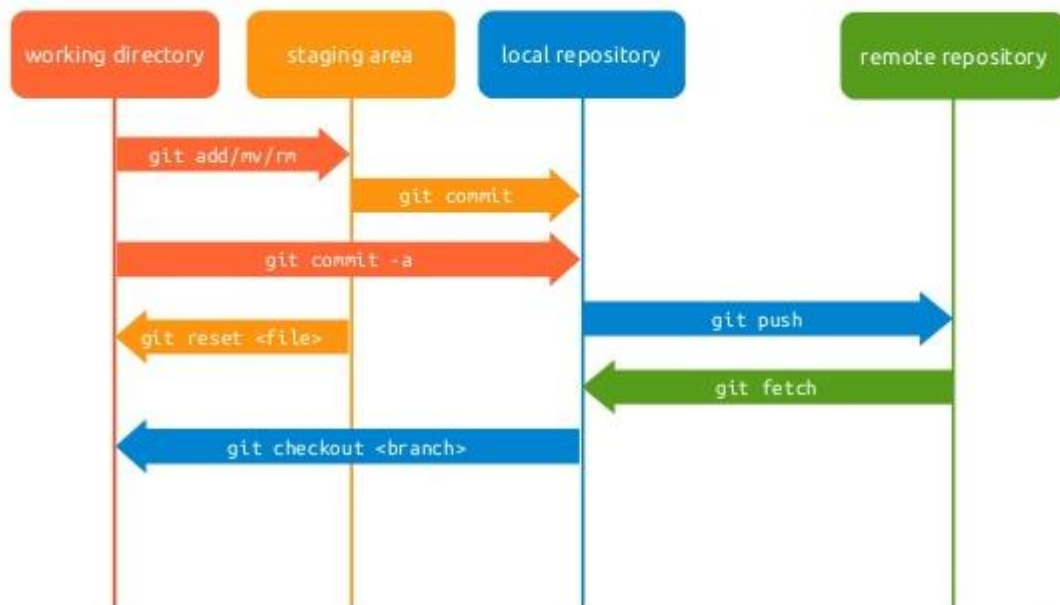


## SCHEMA FUNZIONAMENTO GIT

Una serie di schemini che riassumono i comandi ed i flussi principali di git

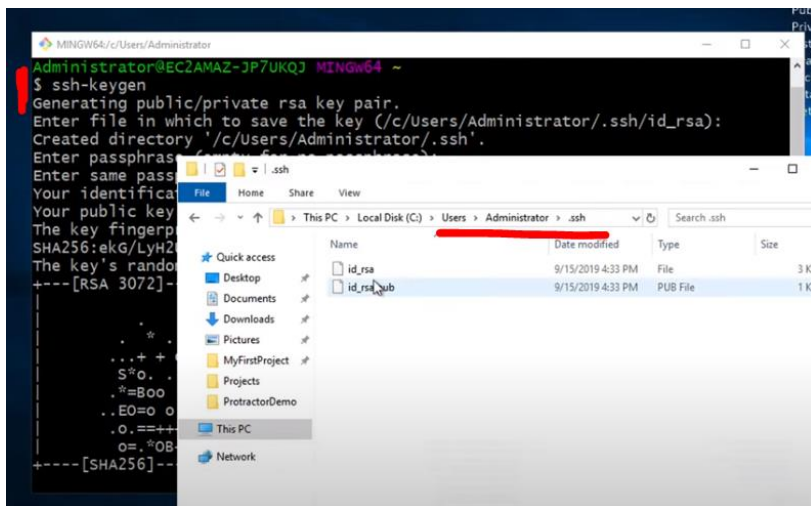




## COME CREARE CHIAVI SSH

È possibile clonare un progetto utilizzando https e fornendo username e password o in alternativa SSH (secured shell) così possiamo connetterci senza fornire ogni volta username e password ma in quest'ultimo caso è necessario definire una chiave nella repository remota al fine di supportare una comunicazione sicura.

### Ssh-keygen



Viene creata una chiave pubblica e privata

Poi dentro gitlab in setting ssh keys incollare la chiave pubblica copiata da rsa\_pub

## per risolvere un conflitto:

caso in cui ci siamo dimenticati di fare il pull e nel frattempo qualcun altro ha modificato i file...

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master)
$ git push origin master
To https://github.com/lucaSchiavon/testgit.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/lucaSchiavon/testgit.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Faccio il pull da remoto, questo causa il merge del file modificato da remoto in quello locale modificato localmente:

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master)
$ git pull origin master
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/lucaSchiavon/testgit
* branch                master      -> FETCH_HEAD
d44beb8..9023975 master      -> origin/master
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.

Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master|MERG
ING)
$
```

Infatti guardiamo nel file:

```
File1.txt - Blocco note di Windows
File  Modifica  Formato  Visualizza  ?
<<<<<<< HEAD
modificato1 modifica locale
=====
modificato1 modifica remota

questa la modifica remota....
>>>>>>> 9023975c8bfe6f16b2f9304755ebd9fb211b6118
```

Notiamo che il commit remoto è stato mergiato con il commit locale (HEAD)

A questo punto sta a noi risolvere i conflitti cancellando ciò che non vogliamo tenere e quindi, una volta risolta la cosa, aggiungiamo allo stage con add poi committiamo e quindi pushiamo.

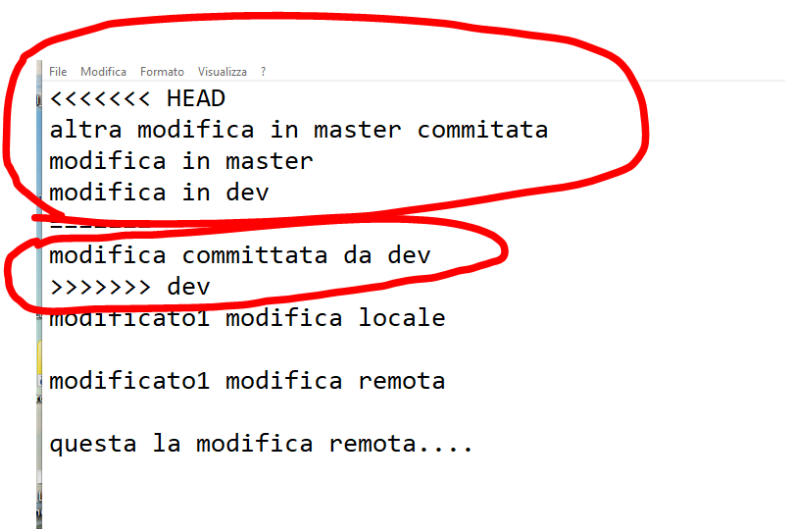
## Altro caso di conflitto

Il file1 del master viene modificato e committato, viene anche modificato il file 1 del dev e committato senza prima fare il merge del master nel dev, faccio il merge del dev dentro il master ed ottengo questo errore:

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master)
$ git merge dev
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.

Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master|MERG
ING)
$
```

Entriamo nel file1 e:



```
File  Modifica  Formato  Visualizza  ?
<<<<<<< HEAD
altra modifica in master committata
modifica in master
modifica in dev
-----
modifica committata da dev
>>>>> dev
modificato1 modifica locale

modificato1 modifica remota

questa la modifica remota....
```

Anche qui dobbiamo risolvere i conflitti noi decidendo cosa tenere e poi lanciando add commit, a questo punto potremo spostarci in dev (se non risolviamo il conflitto non possiamo)

Il file2 del dev viene modificato e committato, viene anche modificato il file 2 del master e committato senza prima fare il merge del dev nel master, faccio il merge del dev dentro il master ed ottengo questo errore:

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master)
$ git merge dev
Auto-merging file2.txt
CONFLICT (content): Merge conflict in file2.txt
Automatic merge failed; fix conflicts and then commit the result.

Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master|MERG
ING)
$
```

Apro il file incriminato e risolvo manualmente il conflitto tenendo ad esempio entrambe le modifiche:

File Modifica Formato Visualizza ?

```
|xxx
<<<<<<< HEAD
modifica committata da master branch
=====
modifica e commit da dev branch
>>>>>>> dev
```

**LA FINESTRA DI EDIT**[illegible]

Per inserire il messaggio nella finestra di edit:

```
premere"i" (per andare in modalità inserimento)
```

write your merge message

press "esc" (per uscire dalla modalità di inserimento)

```
write ":wq"
```

then press enter (per uscire dall'editor)

## COMANDI UTILI per affrontare situazioni particolari

**Clear** pulisce la console

Nel caso ci fosse l'esigenza di vedere l'alberatura dei branch per un progetto da riga di comando:

```
git log --all --decorate --oneline --graph
```

nel caso ci trovassimo in situazione di conflitto per far vincere sovrascrivendo con il current branch il branch di destinazione ossia nel caso non si riuscisse più a passare da un branch ad un altro per problemi di files non allineati:

```
git branch -f otherbranch currentbranch
```

otherbranch (branch di destinazione)

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (dev)
$ git branch -f master dev
```

Si noti che il comando non funziona se lanciato dal branch di destinazione:

```
Luca@WORKSTATION MINGW64 /d/L_SCHIAVON_DOCUMENTS/LAVORO/GIT/testgit (master)
$ git branch -f master dev
fatal: Cannot force update the current branch.
```

Se vogliamo vedere le differenze tra ciò che è prima e ciò che è ora lanciamo:

**git diff**

se volessimo vedere le differenze tra due branch

questo comando mostra le differenze tra branch in caso di disallineamenti e non riuscendo più a fare il merge di un branch sull'altro:

**git diff <branch\_sorgente> <branch\_target>**

## CHECKOUT

Se vogliamo tornare alla versione subito dopo l'ultimo commit cancellando le modifiche fatte occorre lanciare il comando (**how to undo uncommitted changes**)

**Git checkout -- nomefile**

Se vogliamo cancellare le modifiche per tutti i file

**Git checkout -- .** (ricordarsi il punto!)

Come cancellare delle modifiche committate? (**how to undo committed changes**)

## REVERT

**Git revert commitid** (dove commit id è il codice lungo alfanumerico che identifica il commit)

## RESET

per tornare alla versione di un file prima dell'ultimo commit in caso si fossero fatti pasticci:

questo comando rimpiazza il file corrente riportandolo all'ultimo commit:

**git reset head <nomefile>** (a differenza di revert qui viene cancellato tutto lo storico dei commit)

per fare il rollback all'ultimo commit:

**git reset --hard HEAD**

se si vuole fare il rollback ad un certo commit basterà lanciare il comando:

**git reset --hard commitid**



## STASH

Lo stash si usa per accantonare del codice scritto prima di committarlo per spostarsi su un altro branch, fare delle cose e poi ritornare dove si era fatto lo stash e riapplicare le modifiche accantonate.

Per fare lo stash:

```
$ git stash save "Worked on add function"
```

**Git stash list** ci da la lista degli stash:

```
$ git stash list
stash@{0}: On add: Worked on add function
```

Per riapplicare lo stash: **git stash apply** (meglio **git stash pop** che oltre ad applicare lo stash lo cancella dalla lista degli stash, infatti l'apply lo applica ma non lo cancella dalla lista)

```
$ git stash apply stash@{0}
On branch add
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working director
y)

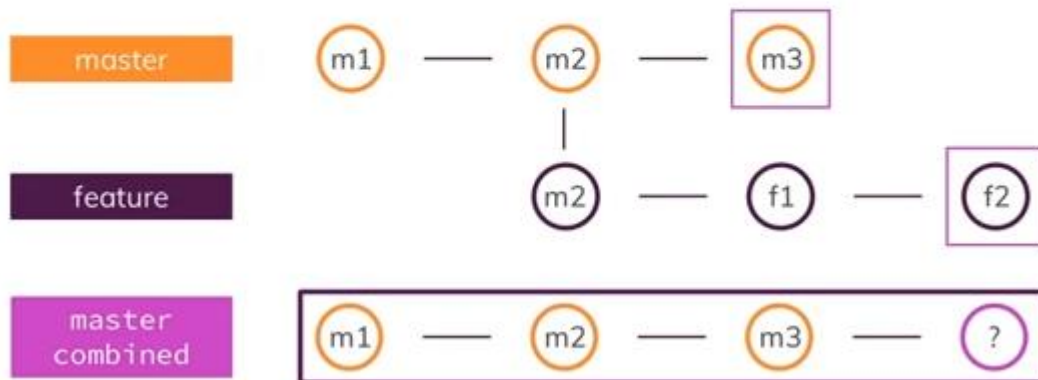
    modified:   calc.py

no changes added to commit (use "git add" and/or "git commit -a")
```

**git stash clear** (cancella tutti gli accantonamenti dalla lista riportando allo stato iniziale dell'ultimo commit quindi attenzione a lanciare il comando perché ci perdiamo tutti gli stash)

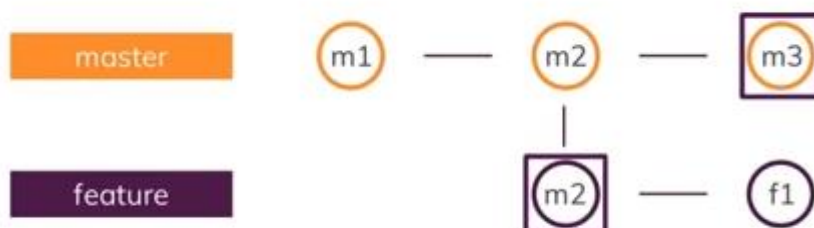
## REBASE

## Our Project



Ora se dal branch feature lanciamo

**Git rebase master** m3 ossia l'ultimo commit di master diventa la base per il branch feature a cui verranno aggiunti poi anche gli specifici commit f1 e f2, in sostanza i commit mancanti del master vengono interpolati nel feature tra l'ultimo commit comune a master e feature ed i nuovi commit del feature (la figura successiva evidenzia che la base di feature non è più m2 ma m3:



Ora per riallineare anche il master per riversare le modifiche del feature sul master lanciare dal master il comando

**Git rebase feature**

In questo modo entrambi i branch avranno m1 m2 m3 f1

Usare rebase nella macchina locale e non in produzione