# HomeWork Day 2

Siete il release manager di un progetto, descrivete il git flow che tutto il team dovrà seguire.

Spiegate le naming convention che volete adottare, le modalità di commit e integrazione dei vari branch.

In caso di hotfix, descrivete la modalità con cui volete procedere per fixare eventuali bug nel miglior modo possibile e come integrare tali fix.

Descrivete inoltre tutte le operazioni che non devono essere eseguite per preservare l’integrità del progetto.

Come esercitazione:

Staccare un branch dal develop del vostro repository (CognomeNome)

Nelle cartella HomeWork pushare il vostro homeWork2.

Spiegare cosa è una Pull Request e fare la richiesta da GitHub indicando “uiip32” come reviewer.

Il flusso di lavoro che un team dovrà seguire parte sempre dai due branches (rami) per registrare la storia del progetto:

Il branch (ramo) master che memorizza la storia della release ufficiale ed è un’immagine di quanto rilasciato in produzione e un branche (ramo) di develop che serve come ramo di integrazione per le features. E’ conveniente contrassegnare i commit nel ramo principale con un numero di versione.

Tutto il resto del flusso di lavoro ruota attorno a questi due branches. Quindi dal branches “padre/master” viene staccato il branch di develop da cui a sua volta si staccano vari branches detti di features attraverso i quai ogni membro del team può sviluppare una funzionalità del progetto. Grazie al meccanismo di **pull e push**, la sincronizzazione tra le repository GIT di ciascun membro è alquanto efficiente. Il meccanismo distribuito permette di creare tante repository private (locali) e sincronizzarle con quelle pubbliche. Un **integration manage**r può pullare tutte le copie pubbliche su un unico “gold repository”, dove ci sarà la versione stabile da deployare in produzione. Quindi il master è il ramo centrale che contiene il codice che verrà messo in produzione e l’altro ramo, il develop è quello dove vengono pushate tutte le modifiche di sviluppo di ciascun membro del team di sviluppo e che viene continuamente aggiornato fino a quando non si è prodotta una versione stabile da mettere in produzione. Quando si reputa stabile, il codice su develop viene mergiato su master. Oltre ai branch “principali”, esistono altre tipologie di branch di vita limitata (detti *branch di supporto*) che si classificano come segue:

* Feature branches
* Release branches
* Hotfix branches

Il feature branch si stacca dal develop e generalmente per sviluppare(o modificare) una funzionalità e la si assegna ad un membro del team e sempre sul develop si mergia al termine dello sviluppo della funzionalità.

Una volta certi di essere nel branch corretto, sincronizziamo il progetto locale prelevando il codice aggiornato dal repository upstream (quindi il progetto d’origine) con un operazione di pull, e sincronizziamo il repository origin (comando di push origin master) con il progetto locale.

DopoiIl git flow da seguire è il seguente:

* Si crea un branch che chiamo myfeature a partire dal develop ed effettuo switch su du esso con il comando $ git checkout -b myfeature develop
* Per integrare il ramo di feature sul develop si effettua un operazione di merge, una volta sviluppata la funzionalità assegnata al membro del team si eseguono i comandi:

$ git checkout develop che switcha sulla develop;

$ git merge --no-ff myfeature effettua il merge di myfeature sulla develop;

$ git branch -d myfeature elimina il branch myfeature;

$ git push origin develop push il develop sul master.

Il caso della creazione di hotfix branch si verifica nel caso in cui avviene un malfunzionamento grave del progetto o bug critico da risolvere e quindi nel master. Questo è l’unico caso in cui si stacca un ramo dal master( e si mergia nel master una volta risolto i bug grave) detto appunto di hotfix . Il processo è detto di branch off e la naming convention è *hotfix* . Il branch off si fadal corrispondente tag del master che marca la versione in produzione, supponendo di avere in produzione la versione 1.2 su cui è avvenuto un  bug, un hotfix branch si crea così:

* $ git checkout -b hotfix-1.2.1 master creo il branch hotfix-1.2.1 e switcho su esso;
* $ ./bump-version.sh 1.2.1 passo alla versione 1.2.1
* $ git commit -a -m "Bumped version number to 1.2.1"

$ git commit –m effettuo una commit delle modifiche per la correzione del bug;

* effettuo merge sul branch master( per integrare la hotfix sul master e sul develop) con i comandi

$ git checkout master switcho sul ramo master;

git merge --no-ff hotfix-1.2.1 effettuo il merge della versione hotfix 1.2.1 sul amster

$ git tag -a 1.2.1 metto il tag alla versione rilasciata;

infine integro la hotfix anche con il develop con i comandi :

* $ git checkout develop swicho sul branch di develop
* $ git merge --no-ff hotfix-1.2.1 effettuo il merge della versione hotfix 1.2.1

Unica eccezione si ha quando la fix riguarda un release branch esistente. In tal caso, l’hotfix va mergiato su tale release, invece che su develop.

Per rimuovere l’ hotfix branch:

|  |  |
| --- | --- |
|  | $ git branch -d hotfix-1.2.1.  Per quanto riguarda le operazioni che i membri del team non possono eseguire e staccare direttamente un ramo dal master ma l’operazione avviene sempre dal develop. |

**Pull-request**

Una volta realizzata una funzionalità e/o modifica ed aver effettuato l’operazione di push per mandare il lavoro fatto ai manteiners del progetto originale si fauna una Pull-request che si effettua con un comando dal nostro repository sul sito github oppure con il comando git pull request sulla git bash e inviare manualmente l’output del comando ai manteiners, tale comando riceve il branch sul quale si vogliono far apparire le modifiche, l’url del repository da cui queste modifiche vengono prese.