# HomeWork Day 2-Risposte Angelo Giordano

Come modello di workflow utilizzeremo due master branch, **“dev”** (o “develop”) utilizzato per lo sviluppo, ed il **“master”** per i rilasci ufficiali.

Questi due branch assolvono il compito di “historical branch”, cioè essi devono mantenere la storia pulita del progetto, a prescindere dagli sviluppi di nuove feature e hotfix, che altrimenti confluirebbero senza filtri sul ramo principale, rendendo meno comprensibile la storia del progetto e quindi più difficoltose operazioni di reversione ([revert](https://en.wikipedia.org/wiki/Reversion_%28software_development%29)) a stati precedenti.

Dopodiché verranno usati uno o più branch di funzionalità, ovvero un **“feature branch”**, dove aggiungere nuove funzionalità ([feature](https://en.wikipedia.org/wiki/Software_feature)) al software. Questi branch **devono sempre essere derivati dal branch “develop”** e rappresentano una **feature**, ovvero una nuova funzionalità che si intende aggiungere al software.

Questi branch conterranno tutte le commit relative alle funzionalità implementate.

Durante lo sviluppo di questi rami non è strettamente necessario mantenere una storia delle versioni pulite o usare particolari nomenclature per le varie commit in quanto prima di riportare il branch su develop, utilizzeremo una sola commit che conterrà solo l’informazione relativa all’aggiunta della nuova funzionalità, in modo da avere la storia del branch di sviluppo più pulita e facilmente consultabile.

Dopo averlo riportato su develop, è possibile rimuovere il feature branch dal repository, tramite comando:

$> git branch -D feat-nomedelfeaturebranch

Sarà possibile mantenere **più feature branch contemporaneamente**, a patto di riportarli correttamente e in tempi relativamente brevi sul branch di sviluppo, ovviamente in base anche alla difficoltà di implementazione. Ogni singolo sviluppatore si concentrerà sul rilascio di una feature per volta, salvo imprevisti quali risoluzione di un bug grave.

Quando il ramo di sviluppo (develop) conterrà abbastanza feature per un rilascio, da questi verrà derivato un nuovo branch detto di rilascio, o [**release branch**](http://producingoss.com/en/release-branches.html).

Questi rami dovranno avere un nome significativo e consistente per distinguerli da altri rami, ad esempio possono tutti iniziare con il prefisso “release-”.

Durante questa fase di sviluppo non possono essere più aggiunte feature e le uniche aggiunte possibili possono essere quelle di bug fixing e documentazione.

Quando il ramo di rilascio sarà maturo e testato, a questo punto verrà eseguita l’operazione di merge sul branch “master”, contrassegnandola con un’opportuna [tag](https://git-scm.com/book/en/v2/Git-Basics-Tagging), che ne indichi l’avanzamento di versione.

L’operazione di merge verrà eseguita anche sul branch “develop”, tralasciando l’aggiunta dei tag di versione.

Una volta mergiato con i branch storici, il branch di release non ha più motivo di esistere e quindi può essere rimosso.

Nell’eventualità di problemi da risolvere con estrema urgenza in questo caso, e solo in questo caso, deriviamo direttamente da master un branch di manutenzione, contenente la fix necessaria a risolvere il problema insorto. Sarà necessario la collaborazione di gran parte del team per risolvere in tempi brevi il problema insorto.

Questi branch di manutenzione useranno come prefisso la voce “hotfix”.

Tale branch sarà ovviamente riportato su “master” il più presto possibile, ed successivamente anche su “develop” per mantenere la situazione dei due branch storici allineata e non incorrere nuovamente nello stesso bug.

Come nel caso dei branch di rilascio, quando un branch di manutenzione viene riportato su master, il tag di master verrà contrassegnato appropriatamente.

Per quel che riguarda l’utilizzo dei vari tag da utilizzare nei rami principali per la denominazione dei rilasci la guida da seguire è la seguente:

* Se abbiamo rilasciato una hotfix, che sostanzialmente non va ad intaccare le modalità di utilizzo del codice, dovremmo utilizzare il tag **patch,** seguito da un numero sequenza alfabetica e un numero che verrà incrementato a ogni patch rilasciata (Es patch v1.1)
* Se abbiamo implementato una modifica minore, per esempio l’aggiunta di una feature che non pregiudica la retrocompatibilità del codice, utilizzeremo il tag **minor**. Anche questo ultimo verrà incrementato a ogni minor rilasciato.
* Rilasci più consistenti che cambiano in maniera apprezzabile il funzionamento del codice andranno a incrementare il numero di **major**, utilizzando questo ultimo tag.

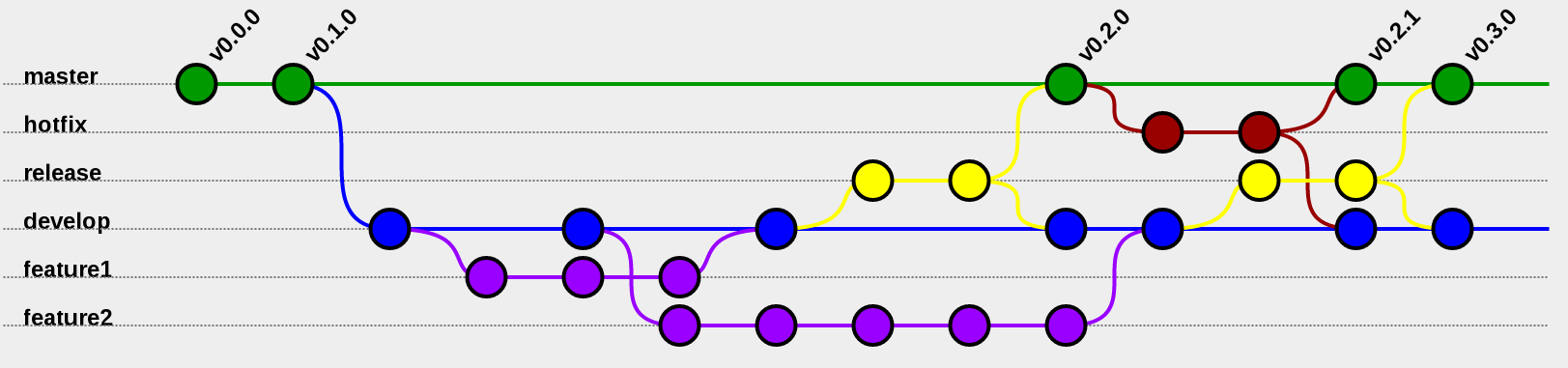


Figura Schema git-flow

**Cosa è la pull request**

La pull request consiste in una richiesta da parte dello sviluppatore per poter inserire il proprio lavoro, sia esso una feature o altro, nel ramo develop in remoto.