

Software Development Lifecycle

Hints & Insights



Presentation as a code

Questa presentazione è scritta in codice markdown e *buildata* con [marp](#).

Il repository del sorgente è su [github](#).

L'hosting e il *continuous deployment* sono forniti da [netlify](#).

👉 <https://gest-progetti-iis.netlify.com/>

Aureliano Bergese (fullstack@mondora)



Explorer
Functional Programming enthusiast
Father of 2
Professional Scrum Product Owner

Sommario - Modulo 1 VCS

1. Intro
2. Versionamento
3. Git
4. Esercizi
5. Risorse

Introduzione

Entità nello sviluppo software nel 2109.

SUBJECTS (cosa)

- progetto: ciò che dobbiamo realizzare
- prodotto: uno o più progetti, volti a fornire un servizio unificato
- valore: benefit fornito dal prodotto
- revenue: guadagno fornito dal prodotto al proprietario

STAKEHOLDERS (chi)

- team: chi lavora al progetto
- utenti finali: chi usa il prodotto
- clienti: chi commissiona il progetto
- funder: chi mette i soldi per il progetto

ENVIRONMENTS (dove)

- dev: per i programmatori del team
- test: per i tester del team
- integration: per le altre entità che concorrono al progetto
- demo: per mostrare le funzionalità al cliente / funder
- preproduction: per i tester del prodotto
- production: per gli utenti finali

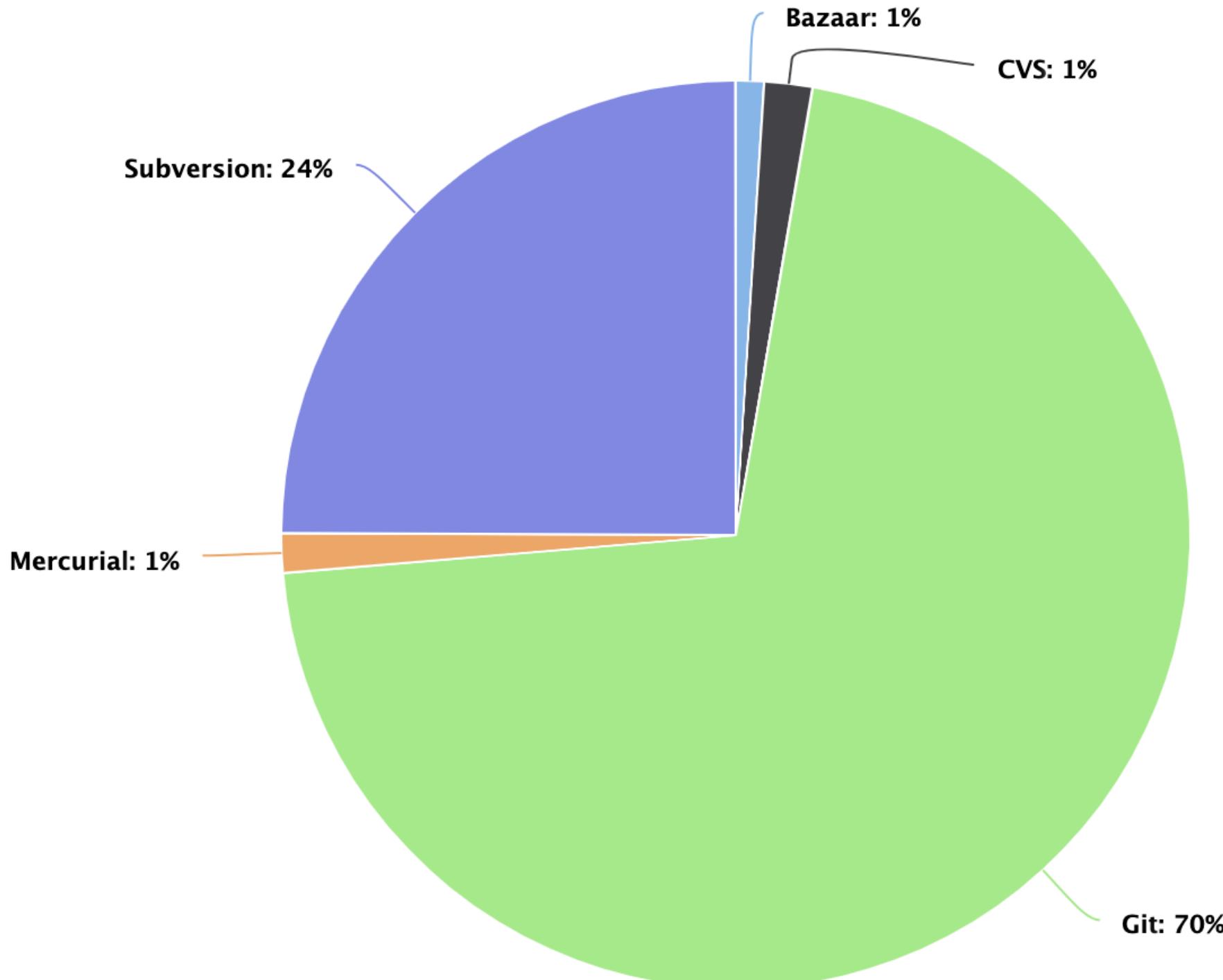
Versionamento

**Un VCS traccia la storia dei
cambiamenti di persone e team che
lavorano insieme ad un progetto.**

Un DVCS lo fa in maniera distribuita.

Perchè? per rispondere alle domande:

- quali modifiche sono state fatte?
- chi ha fatto le modifiche?
- quando sono state fatte le modifiche?
- perchè sono state richieste le modifiche?





GIT

(Distributed) Version Control System
(git = idiota)

Linus Torvalds 2005

Junio Hamano v1 (attualmente mantainer)

Basato su **checksum** di file e folder (per la velocità)

Repository

Un repository (repo) o git-project è l'insieme di file e folder associati ad un progetto. Comprende anche lo storico delle modifiche.

Crea nuovo (locale)

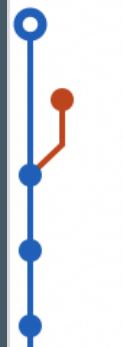
```
git init
```

Clona esistente (remoto)

```
git clone https://github.com/auridevil/iis_classes_2019_src.git  
cd iis_classes_2019
```

Branching

un branch è un ramo dell'alberatura della storia del codice
il branch principale è **master**

Graph	Description	Commit	Author
	<p>dummy master commit</p> <p>this is a branch demo</p> <p>rock'n'rolling</p> <p>chore: intro and first section</p> <p>[chore] initial commit</p>	653e27b 38e9abb db9c55b f65d605 47ee630	auridevil
	<p>origin/master</p> <p>origin/HEAD</p>	653e27b	auridevil
	<p>origin/branchdemo</p> <p>branchdemo</p>	38e9abb	auridevil
	<p>rock'n'rolling</p>	db9c55b	auridevil
	<p>chore: intro and first section</p>	f65d605	auridevil
	<p>[chore] initial commit</p>	47ee630	auridevil

Tutti i branch del repo

```
mox@urania$ git branch  
  branchdemo  
* master  
(END)
```

Cambia branch attivo

```
git checkout branchdemo
```

Crea nuovo branch

```
git branch funzionalita1425
```

Crea nuovo branch e utilizza

```
git checkout -b funzionalita1425
```

Aggiungi

L'aggiunta di un set di modifiche è in due fasi: staging e commit.
git add aggiunge alcune modifiche allo stage

Aggiungi modifica di un file

```
git add PITCHME.md
```

Aggiungi tutte le modifiche

```
git add .
```

Aggiungi modifiche di un folder

```
git add assets/*
```

NB la cancellazione è una modifica. Solitamente il renaming / move è considerata cancellazione + aggiunta.

git commit salva lo snapshot delle modifiche (sul branch corrente) e completa il tracciamento

commit con editor

```
git commit
```

commit inline

```
git commit -m "this is a inline commit message"
```

Commit message (best practices)

- usare l'imperativo
- max 50 chars
- no punti finali
- dichiarare le modifiche fatte ad alto livello
- includere eventuali riferimenti a documentazione (e.g. jira code, tiketing...)

```
git commit -m "[type][branch] What is done "
```

type è il tipo di modifica:

- feat: funzionalità nuova
- fix: correzione errore
- chore: piccola sistemazione
- test: copertura del codice
- doc: documentazione

Se il branch è master, si esclude dal commit message.

Status

```
mox@urania$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   game.txt
```

Diff

Mostra le differenze non ancora nell'area di staging (pre *git add*)

```
git diff
```

Mostra le differenze tra staging e l'ultima modifica

```
git diff --staged
```

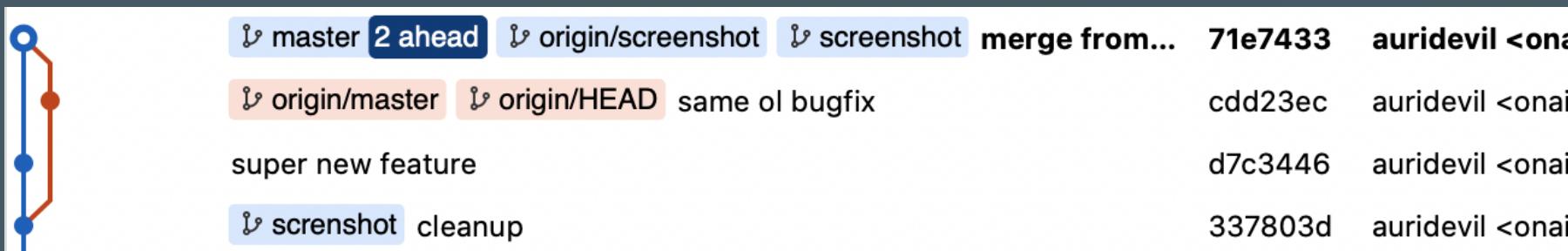
Unstage

Rimuovi un file dall'area di staging, ma mantieni le modifiche:

```
git reset game.txt
```

Merge (locale)

Combina le modifiche fatte su due branch differenti. E' direzionale:
merge **of** un branch **into** un'altro branch.



Porta le modifiche di **branch1** in **branch2**

```
git checkout branch2  
git merge branch1
```

Porta le modifiche di **branch2** in **master**

```
git checkout branch2  
git merge master  
git checkout master  
git merge branch2
```

Conflitti

```
mox@urania$ git merge master
CONFLICT (add/add): Merge conflict in sample2.txt
Auto-merging sample2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

file in conflitto:

```
><<<<<< HEAD
code from the current branch
=====
code from master branch
<>>>>>> master
```

modificare a mano il file in conflitto per ottenerе

```
code mix from both branches
```

aggiungere la modifica al branch per completare il merge

```
git add sample2.txt  
git commit -m "merge from master"
```

Remote

E' possibile aggiungere un server remoto (se non si è partiti da un clone) per usare git distribuito:

```
git remote add origin https://github.com/auridevil/iis_classes_2019_src.git
```

il nome di default del remote è *origin*. Per verificare l'aggiunta:

```
mox@urania$ git remote -v
origin  https://github.com/auridevil/iis_classes_2019_src.git (fetch)
origin  https://github.com/auridevil/iis_classes_2019_src.git (push)
```

Pull

Per ottenere le modifiche dal server su master:

```
git pull origin master
```

o (unsafe):

```
git pull
```

per *pullare* un branch:

```
git pull origin branch3
```

git pull prova a mergiare in locale quello che c'è sul server, se serve più controllo si può usare *git fetch* (avanzato)

```
git fetch
```

Push

Per inviare le modifiche al server da locale, sul branch remoto di master:

```
git push origin master
```

per *pushare* su un branch particolare, remoto:

```
git push origin branch3
```

Merge (remote)

Porta le modifiche di **branch2** in **master**, con fast-forward

```
git checkout master
git pull origin master
git checkout branch2
git merge master
git push origin branch2
git checkout master
git merge branch2
git push origin master
```

Porta le modifiche di **branch2** in **master**, con conflitto:

```
git checkout master
git pull origin master
git checkout branch2
git merge master
<fix conflict>
git add .
git commit -m "merge from master"
git push origin branch2
git checkout master
git merge branch2
git push origin master
```

Rebase

Prendi tutti i cambiamenti fatti su un branch e portali su un altro

```
git rebase branch3
```

Warning: il rebase è pericoloso da usare. E' facile alterare la salute del repository, l'utilizzo è sconsigliato salvo rari casi di emergenza (seguire le guide ufficiali online).

Incompleti

Capita di voler mettere da parte delle modifiche incomplete, per farlo si usa lo *stash* (una specie di cut'n'paste gigante).

Aggiungi tutte le modifiche (non staged) allo stash:

```
git stash
```

Riapplica le modifiche nello stash

```
git stash apply
```

Storico

E' possibile vedere lo storico dei commit da cli:

```
git log
```

File ignorati

E' quasi sempre necessario NON versionare alcuni file e folder, ad esempio dove sono contenute password o stringhe di connessione ai database, i file compilati, i file degli editor e le dipendenze.

Per questo si mette nella root del progetto un file `.gitignore` dove si indicano i file (regex allowed) da ignorare. Git non vedrà modifiche a questo tipo di file, né traccierà la loro esistenza.

[Git Hub .gitignore list](#)

Tag

E' possibile aggiungere dei tag ad un determinato commit su un determinato branch (solitamente master), ad esempio per tracciare un rilascio, o una breaking change:

```
git tag -a v1.5.2 -m "Version 1.5.2 on westeurope + northeurope servers PROD"  
git push --tags origin master
```

HEAD

La HEAD del repository locale è un puntatore al branch attuale

```
mox@urania$ cat .git/HEAD  
ref: refs/heads/master
```

E' possibile spostare la HEAD a un determinato commit passato o fuori branch e si definisce **DETACHED HEAD**. E' da considerarsi una procedura di emergenza.

Per le operazioni più chirurgiche si consiglia [learn git branching](#).

Pull Request

E' un processo di *code review* delle modifiche create in un *feature branch* (branch a cui è associata una funzionalità di progetto), prima di farlo convergere (*merge*) su un branch principale.

Le pull request sono il cuore della **collaborazione**.

```
.... (from `git push origin branch-to-PR` output)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'branch-to-PR' on GitHub by visiting:
remote:     https://github.com/auridevil/iis\_classes\_2019\_src/pull/new/branch-to-PR
remote:
To https://github.com/auridevil/iis\_classes\_2019\_src.git
 * [new branch]      branch-to-PR -> branch-to-PR
```

This is my PR title #1

 Open

auridevil wants to merge 1 commit into [master](#) from [branch-to-PR](#) 

 Conversation 0

 Commits 1

 Checks 0

 Files changed 1



auridevil commented 34 seconds ago • edited 

Owner

+  ...

Here I have to make my changes understandable to other teammates

 this will be on pr

 d095ac3

Add more commits by pushing to the **branch-to-PR** branch on [auridevil/iis_classes_2019_src](#).



Continuous integration has not been set up

Several apps are available to automatically catch bugs and enforce style.



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Changes from all commits ▾ File filter... ▾ Jump to... ▾ ⚙ ▾ 0 / 1 files viewed ⓘ Review changes ▾

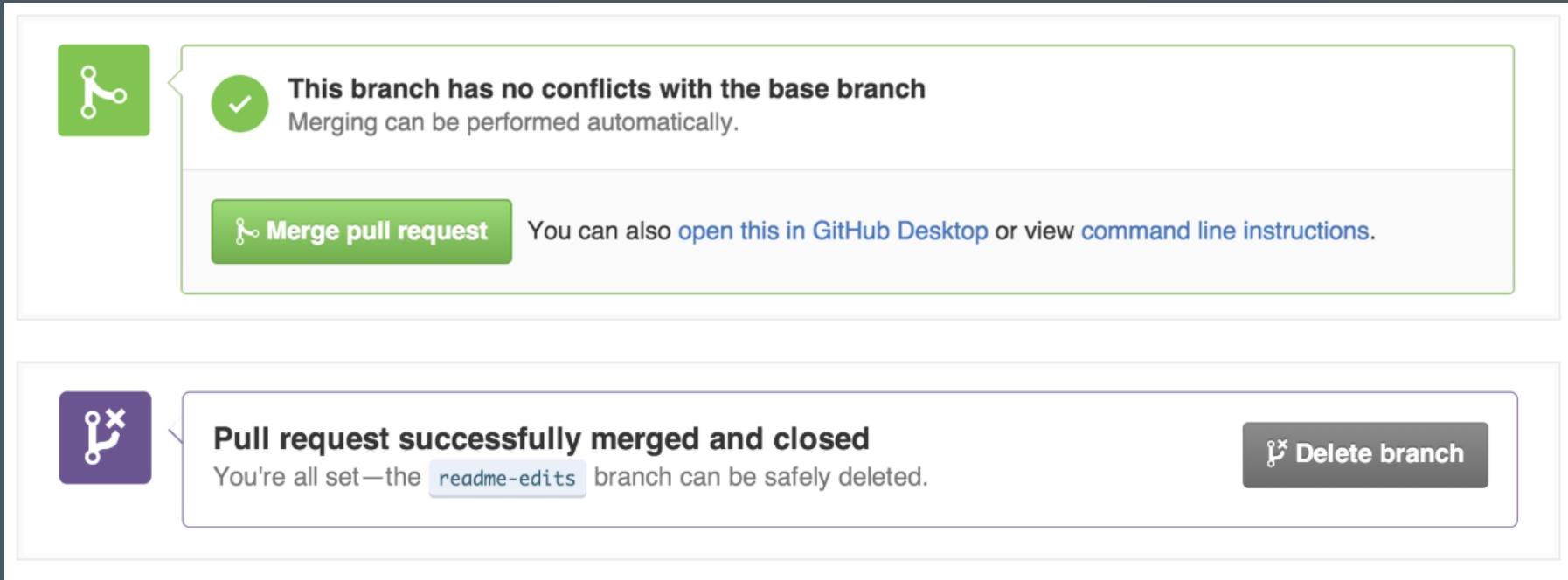
15 PITCHME.md

@@ -374,14 +374,19 @@ ref: refs/heads/master

	374	375	376	377	378	379	380	381	382	383	384	385	386	387										
374	---			374	375	376	377	378	379	380	381	382	383	384	385	386	387							
375	E' possibile spostare la HEAD a un determinato commit passato o fuori branch e si definisce **DETACHED HEAD** . E' da considerarsi una procedura di emergenza.			375	E' possibile spostare la HEAD a un determinato commit passato o fuori branch e si definisce **DETACHED HEAD** . E' da considerarsi una procedura di emergenza.		+ Per le operazioni più chirurgiche si consiglia [learn git branching] (https://learngitbranching.js.org/).	377	---	380	# Pull Request	381	+ E' un processo di *code review* delle modifiche create in un *feature branch* (branch a cui è associata una funzionalità di progetto), prima di farlo convergere (*merge*) su un branch principale.	382	+ Le pull request sono il cuore della **collaborazione** .	383	+	384	---	385	---	386	# Git Flow	387
376				376	377	378	379	380	381	# Pull Request	381	382	383	384	385	386	387							
377	---			377	378	379	380	381	382		381	382	383	384	385	386	387							
378	# Pull Request			378	379	380	381	382	383		381	382	383	384	385	386	387							
379	-			379	380	381	382	383	384		381	382	383	384	385	386	387							
380	---			380	381	382	383	384	385		381	382	383	384	385	386	387							
381	# Git Flow			381	382	383	384	385	386		381	382	383	384	385	386	387							
382	---			382	383	384	385	386	387		382	383	384	385	386	387								
383	# Git Hub Flow			383	384	385	386	387			383	384	385	386	387									
384				384	385	386	387				384	385	386	387										

E' possibile fare il merge se

- non ci sono conflitti
- c'è un numero sufficiente di *approve* (da stabilire con il team)



Git Hub Flow

Workflow semplice in 6 punti per la collaborazione con git:

1. **create branch**: feature branch, dal branch di deploy (solitamente master), orientato alla feature
2. **commit**: aggiungere in piccoli step le modifiche necessarie alla feature
3. **pull-request**: per massimizzare la trasparenza, tutto ciò che va su master deve passare da pull-request

4. **code-review**: si fanno test e code review collettiva di team, per valutare la bontà della soluzione e del codice, eventuali modifiche necessarie riportano al punto 2
5. **merge**: si porta master nel feature branch, si controlla che tutto sia coerente e poi si porta il feature branch in master; il feature branch viene chiuso (opzionalmente si fa un tag)
6. **deploy**: si pubblica il codice modificato, manualmente o tramite un processo di continuous deployment. Se non possibile, bisogna comunque considerare il codice come deployabile (production ready)

Git Flow (accenni)

In questo workflow ci sono alcuni branch fissi per ogni stage degli environment (**dev,test,demo,master**,etc).

Le modifiche vengono fatte su un feature branch, creato a partire dal branch **dev**.

"Quando il team è soddisfatto di una feature" la porta nel branch di **dev** dove si prova la convivenza con le altre feature sviluppate.

Il codice in **dev*** viene promosso in **test / integration** e viene sottoposto a controllo manuale / automatico delle funzionalità.

Quindi il codice viene promosso in **demo / preprod** e viene validato con il cliente / product owner.

Insieme al cliente si stabilisce una data di rilascio in cui si promuove il codice in **master** e quindi di deploya in produzione.

Eventuali bug di produzione vengono fixati creando un branch da **master** e, a fix completato, riportando le modifiche su tutti gli ambienti, compreso quindi master.

Approfondimento

Fork and pull

Workflow legato all'open source, figlio del github flow. Necessario quando non si hanno i permessi per lavorare su un repository che si vuole modificare.

1. **fork**: fare un fork nel proprio account di un repository remoto
2. **commit**: aggiungere nel proprio repo le modifiche fatte
3. **pull-request**: richiedere un merge dal proprio repository a quello originale
4. **code-review**: i mantainer e la community faranno una code review e valuteranno se mergiare la modifica nel repository originale

esercizi

1. Da soli:

- clonare questo repo
- cambiare le immagini riferite in [PITCHME.md](#) (vedi /assets/*.jpg)
- salvare le modifiche in un branch

2. In team (~3 persone):

- creare un repository github con un file testo.txt (insieme)
- scrivere una parte di testo (ognuno) a scelta, e.g. inventare una poesia o un mini racconto
- salvare le proprie modifiche in un branch
- creare una pull-request verso master dei propri branch
- seguire la review fino a ottenere tutte le modifiche su master

3. Da soli

- prendere un repository open source su github
- fare una fork e modificare qualcosa (e.g. documentazione)
- creare una pull-request verso il repository originale
- (rimuovere tutto, se non utile)

risorse

Git as-a-service

- [Git Hub](#)
- [Git Lab](#)
- [Bitbucket](#)

Tools

- [Atlassian Sourcetree](#)
- [Visual Studio Code](#)
- [Github classroom](#)

Approfondimenti

- [Git Cheat Sheet](#)
- [Learn Git Branching](#)
- [Git Hub Learning Lab](#)
- [Visualizing Git](#)
- [Git Hub Flow](#)
- [Git Flow](#)



Created by Aureliano Bergese

<https://github.com/auridevil/>

<https://twitter.com/elmozzo>

https://www.instagram.com/elmozzo_buendia/

<https://medium.com/@elmozzo>

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

