Linee Guida

Jawa Druids

Versione	x.x.x
Data approvazione	xx-xx-xxxx
Responsabile	Nome Cognome
Redattori	Andrea Dorigo Margherita Mitillo
Verificatori	Nome Cognome Nome Cognome
Stato	stato
Lista distribuzione	Jawa Druids Tullio NomeAziendaProponente
Uso	Interno

Sommario

Linee guide da seguire per una sessione lavorativa standard con git flow, Trello e google sheet

Registro delle modifiche

Modifica	Autore	Ruolo	Data	Versione
Revisione prima stesura	Margherita Mitillo	*Analista*	26-11-2020	v0.0.2
Prima stesura bozza	Andrea Dorigo	*Responsabile*	26-11-2020	v0.0.1

Indice

Norme sull'utilizzo di git flow

Esempio di un'aggiunta della **stesura di un verbale** a scopo esemplificativo.
In questo caso, si tratta di un'aggiunta molto semplice che non richiede particolari revisioni e il cui merge è poco rischioso (non c'è il rischio di immettere bug); pertanto si può semplicemente committare e pushare sul `develop`.
Esempio di un'aggiunta di una **funzionalita' al software** a scopo esemplificativo.
Poniamo che, nel capitolato 3, noi vogliamo aggiungere una funzione al nostro software, che consideri anche i dati provenienti da Uber per calcolare il flusso di persone.
Il primo passo e' creare una nuovo feature branch con il comando
`git flow feature start analisi-dati-uber`
Questo comando crea un branch a partire dal develop chiamato `feature/analisi-dati-uber`.
Per rendere pubblico il branch e rendere possibile la collaborazione, e' possibile pubblicarlo con il comando
`git flow feature publish analisi-dati-uber`
A questo punto chiunque voglia lavorare sullo stesso branch puo' farlo con il comando
`git flow feature track analisi-dati-uber`.
Qui ora e' possibile fare i vari `commit` e `push` su questo nuovo branch, senza il rischio di aggiungere bug o errori nel develop. Una volta terminata e testata la nuova funzionalita', si puo' richiudere il branch nel develop ed eliminarlo tramite il comando
`git flow feature finish analisi-dati-uber`.
Si consiglia, nei casi di feature/hotfix/bugfix importanti di richiedere prima l'approvazione del responsabile di progetto.
Il responsabile di progetto si riserva la pubblicazione delle `release`.

Potete trovare un ottimo cheatsheet dei comandi di git flow e delle loro funzioni qui: `https://danielkummer.github.io/git-flow-cheatsheet/`

Un buon riassunto sull'utilizzo di git flow:
`https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`

Un altro ottimo riassunto sulle best practice dei commit qui:
`https://www.theserverside.com/video/Follow-these-git-commit-message-guidelines`

Fra le tante cose importanti nel titolo del commit si evidenzia:

- descrivi cosa hai fatto e perche', ma non come.

- l'utilizzo dell'imperativo;

Per quest'ultimo punto vi riporto un paragrafo del link qui sopra che aiuta la comprensione (c'e' un errore nel testo dell'articolo, che ho corretto qui sotto):

```
Fixed the fencepost error //bad

Fixing the fencepost error //bad

Fix the fencepost error //good
```

The imperative mood is the one git commit message guideline that developers tend to violate most often. A good rule of thumb is that a git commit message can be appended to the statement "If applied, this commit will ...." The resulting sentence should make grammatical sense. As you can see from the following three examples, gerunds and past tense commits fail the test, while the imperative tense does not:

```
If applied, this commit will Fixed the fencepost //error

If applied, this commit will Fixing the fencepost //error

If applied, this commit will Fix the fencepost //good
```