Git workflow Uni4Justice

Borsa di ricerca

9 Gitflow - trunk based develop

In questo modello gli sviluppatori creano una ramificazione del branch develop per ogni feature che si sviluppa, dopo aver terminato lo sviluppo si compie un rebased rispetto al branch develop aggiornato per poi compiere il merge con esso.

10 Best practice

Ora vedremo la sequenza di istruzioni da seguire per lo sviluppo di una nuova feature. Dobbiamo considerare che si potrebbero avere situazioni differenti tra repository locale e remoto.

1. `git checkout develop`: ci spostiamo sul branch develop;
2. `git pull`: aggiornare il branch develop;
3. `git checkout -b feature01`: iniziamo a sviluppare una nuova feature creando un branch che parte dal develop,
4. `git add <updated_file>`: aggiungiamo i file aggiornati alla fase di checkout per prepararli al commit;
5. `git commit -m "[issue_number] Text of commit"`: commit inerente ad un issue preciso;
6. `git checkout develop`: ci si sposta sul branch di develop per aggiornarlo con le modifiche che potrebbero essere state fatte da remoto;
7. `git pull`: aggiorniamo il branch di develop con le modifiche da remoto;
8. `git checkout feature01`: ci si sposta sul branch della feature che si è finito di sviluppare;
9. `git rebase develop`: eseguiamo il rebase ovvero facciamo un forward del branch develop, ovvero aggiorniamo il branch della nostra feature con le modifiche presenti sul branch di develop. Questa operazione potrebbe creare dei conflitti nei file all'interno del branch feature01;
10. `meld .`: con il programma meld possiamo risolvere i conflitti che si sono generati con un sistema di visualizzazione semplice e funzionale, a sinistra e a destra ci sono le due versioni che entrano in conflitto del file, mentre al centro c'è la versione che vogliamo generare dopo la risoluzione dei conflitti, si consiglia di scaricare meld per gestire i conflitti, dopo aver gestito i conflitti si preme sull'icona con la freccia verso in basso per applicare le modifiche ed infine si esce dall'applicazione per marcare come risolto il conflitti;
11. `git rebase --continue`: per concludere il rebase digitiamo questo comando;

12. `git push`: cercando di pushare il codice git da terminale consiglia il comando corretto per pushare le modifiche, perché a questo punto stiamo proponendo su github il codice per compiere una merge request. Infatti dopo aver eseguito il comando consigliato da git viene prodotto un URL, se clicchiamo su questo veniamo portati su browser nel repository GitHub;

13. `GitHub`: a questo punto dobbiamo svolgere una merge request su GitHub, verso il branch develop, RICORDIAMOCI DI FARE LA MERGE REQUEST VERSO IL BRANCH DI DEVELOP E NON VERSO ALTRI BRANCH, IN PARTICOLARE VERSO IL MASTER CHE È IL DEFAULT DI GITHUB. Dopo aver fatto la merge request se GitHub non trova ulteriori conflitti è possibile mergiare il codice su GitHub.

14. `git checkout develop`: ora possiamo spostarci sul branch di develop per aggiornare in locale il nuovo branch di develop con tutte le nuove feature;

15. `git pull`: aggiorniamo il branch di develop in locale con le nuove feature;

16. `git branch -D feature01`: dopo aver finito di sviluppare la feature è buona norma cancella sia in locale che in remoto il branch inerente ad essa a meno che non sia utile compiere dei reverse commit. Per la cancellazione del branch in locale si usa il comando appena scritto mentre per cancellarlo in remoto di deve spostarsi su browser su GitHub nella sezione branches;

Nel caso in cui si voglia compiere il backup del codice su GitHub senza compiere un rebase di questo perché non si ha voglia in quel momento è possibile farlo con un semplice push, in questo caso si genera il branch della feature01 su GitHub e quando si effettuerà il rebased in locale (procedura vista prima), poi si deve fare il push con il comando `git push --force-with-lease` che forza il push della history che è avvenuta sul repo locale anche nel branch remoto. Questo viene fatto perché il remoto non è stato aggiornato con il rebased effettuato in locale.