Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Metodika commitovania a verziovania

Tím sixPack

Bc. Jozef Blažíček

Bc. Ján Ďurica

Bc. Jakub Chalachán

Bc. Matúš Ivanoc

Bc. Maryna Kovalenko

Bc. Miloš Štefčák

Vedúci projektu: Ing. Ivan Kapustík

Predmet: Tímový projekt I

Ročník: 2016/2017

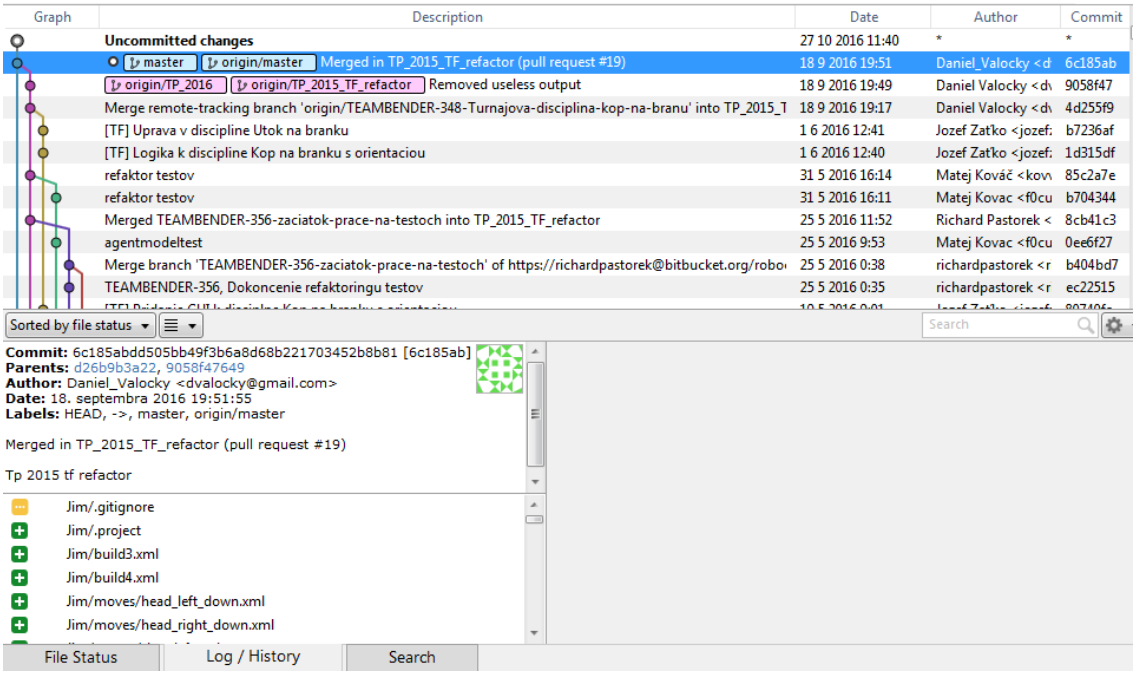
Obsah

Metodika je určená všetkým ktorí pracujú na vývojovej časti tímového projektu. Je dôležité ovládať pritom funkcionalitu gitu, aby nevznikali problémy pri zdieľaní jednotlivých zmien v prostredí Bitbucket.
Táto metodika sa venuje:

- spôsobu commitovania
- vytváraniu vetiev k jednotlivým úlohám
- prácou v prostredí SourceTree

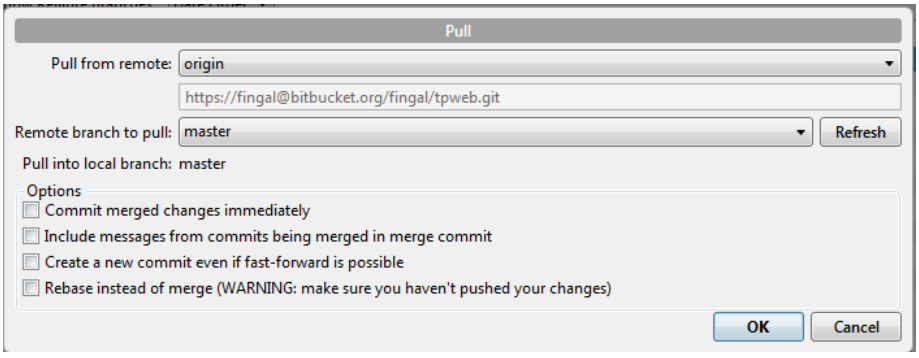
1. Commitovanie

Každý commit musí mať commit správu! Formát commit správ dodržujte v tvare
FUTBAL3D_<id_task> „krátky opis činností“
Príklad: FUTBAL3D-37 oprava mensich chyb + doplnenie komentarov
ID_task je označenie úlohy z Jiry a slúži na prehľad, ku ktorej úlohe daná zmena v kóde patrí.



Obrázok 1:Hlavné okno SourcTree prostredia

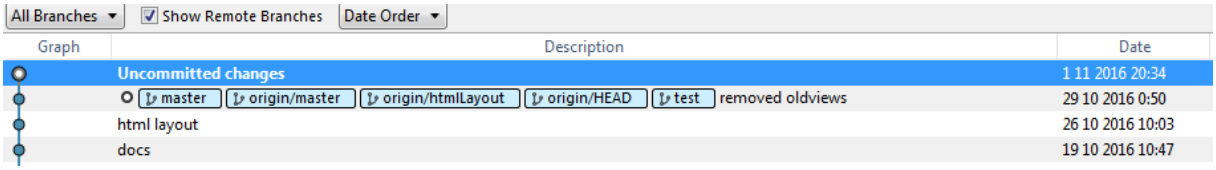
Na úlohách pracuje zvyčajne viacero ľudí, takže pred každým commitom je potrebné vždy urobiť **Pull**. Okno Pull príkazu obsahuje viacero informácií, medzi nimi je dobré vždy overiť že pull vykonávam z rovnakej vzdialenej vetvy do lokálnej vetvy.



Obrázok 2:Okno menu pre operáciu pull

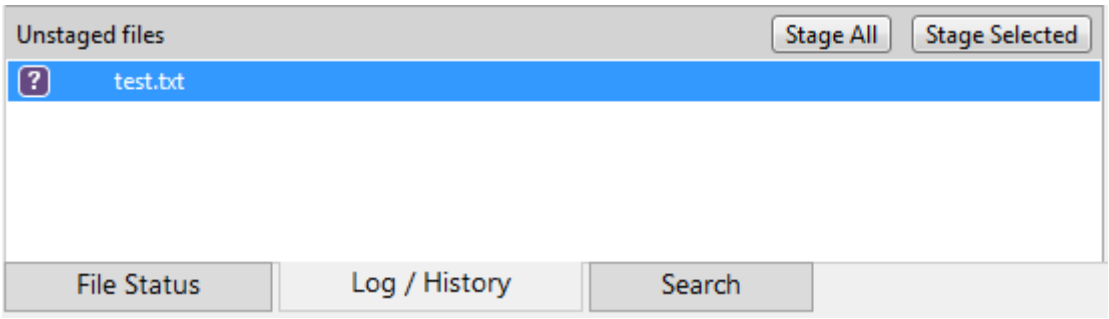
1.1. Postup commitovania v SourceTree

V SourceTree sú necommitnuté zmeny označené v okne commitov ako **Uncommited changes**



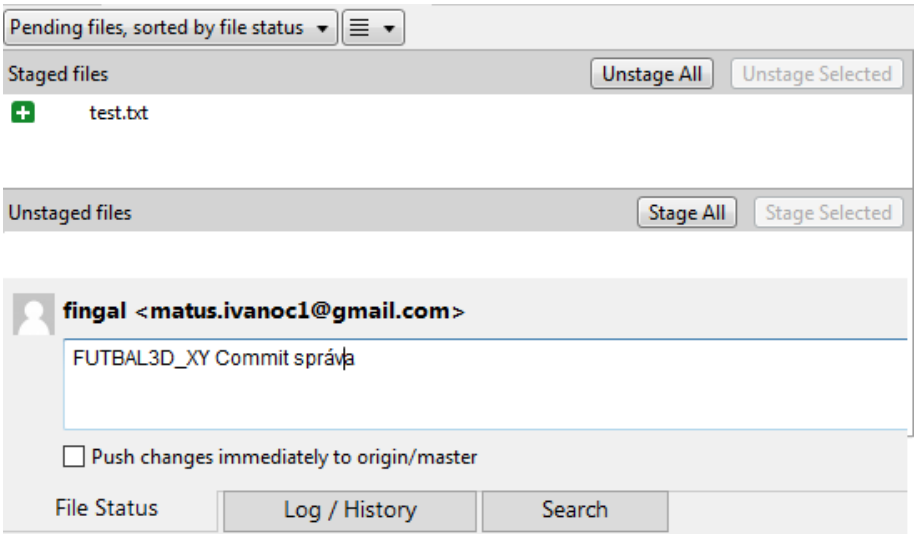
Obrázok 3:Okno uncommited changes

V zozname Unstaged Files sa buď vyberú jednotlivé súbory alebo všetky ktoré chceme commitovať.



Obrázok 4: Zoznam unstaged súborov

Nakoniec obrazovka commitu kde je potrebné vypĺniť commit message a skontrolovať vybrané súbory, prípadne doplniť ďalšie.

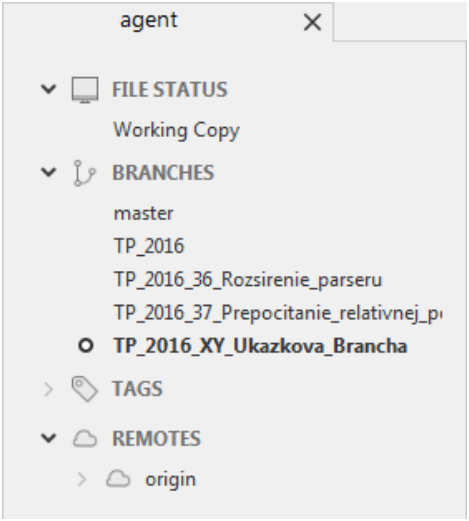


Obrázok 5: Okno commitu

Po commite stačí vykonať git push a zmeny sa odošlú na vzdialený repozitár.

1.2. Prepínanie medzi vetvami

Pred akýmkoľvek commitom sa uistite, že pracujete na správnej vetve. Prepnúť medzi vetvou sa dá jednoducho v SourceTree kliknutím na jednu z vetví v zozname.



Obrázok 6: Prehľad vetiev lokálneho repozitára

Po prepnutí sa vám aktualizuje repozitár a je možné pracovať.

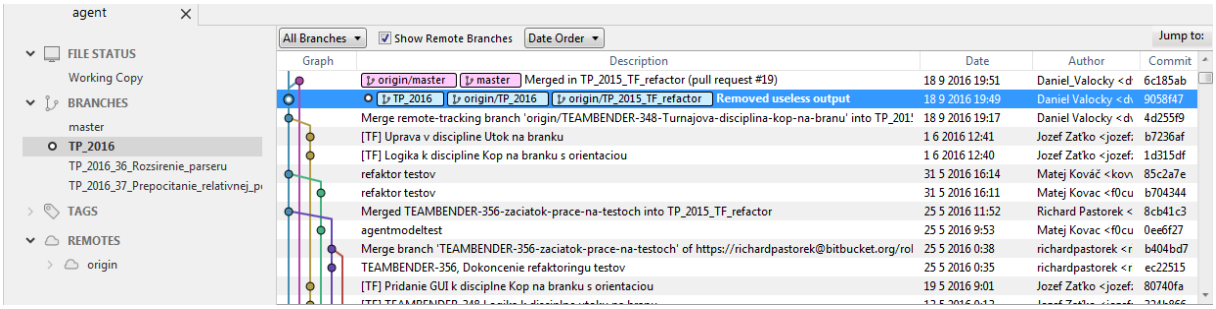
1.3. Git Stash

Ak máte už rozpracovanú úlohu a potrebujete sa prepnúť medzi vetvami, tak je možné buď vykonať commit z časti hotovej práce, alebo nedokončenú prácu odložiť pomocou **Stash**. Je to veľmi šikovná funkcionalita ktorá umožňuje presúvať zmeny ktoré ešte nie sú úplne pripravené na commit.

Obrázok 7: Menu Git operácií

2. Vytváranie vetiev

Projekt má vytvorenú hlavú vetvu **TP_2016** ktorá je nastavená ako ***main.*** Do tejto vetvy sa mergujú postupne všetky dokončené vetvy, alebo inak súvislé celky kódu. **TP_2016** sa považuje za **stable** vetvu, to znamená že sa tam nesmú mergovat žiadne nedokončené časti kódu, alebo netestované vetvy.

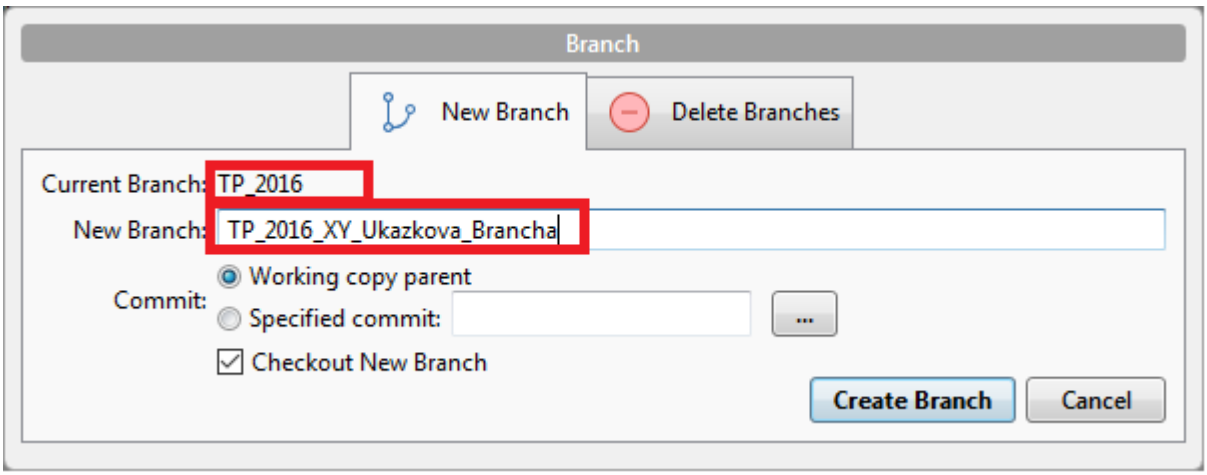


Obrázok 8:Prehľad vetiev, a commitov v repozitári

K samotnému postupu vytvárania vetiev je potebné dodržiavať niekoľko bodov:

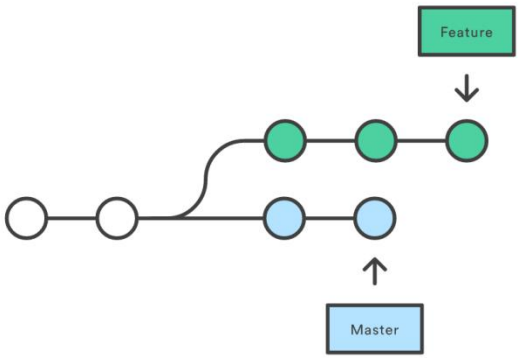
- Vetvy sa vytvárajú pre každú úlohu ktorá sa týka vývoja. Na úlohách pracuje zvyčajne viacero ľudí, takže pred každým commitom je potrebné vždy urobiť **Pull**.
- Názov vetvy vytvárať v tvare: **TP_2016_<id_úlohy>_<názov_úlohy>**, príklad na predstavu: TP_2016_37_Prepocitavanie_relativnej_pozicie_z_parsera
- Po dokončení tasku sa commitnuté zmeny sa pošlú do nástroja Bitbucket. Dokončená úloha ide na Codereview o ktorom pojednáva samostatná metodika.
- Po zrevidovaní a prípadnom otestovaní sa vytvorí nový pull request a vetva sa mergne do hlavnej TP_2016 vetvy. Merge do master vetvy vykonáva autor kódu, pretože on najlepšie vie aké zmeny sa v kóde vykonali.

Príklad vytvorenia vetvy v SourceTree. Vetvu možno vytvoriť na základe originálnej vetvy, alebo priamo z určitého commitu.

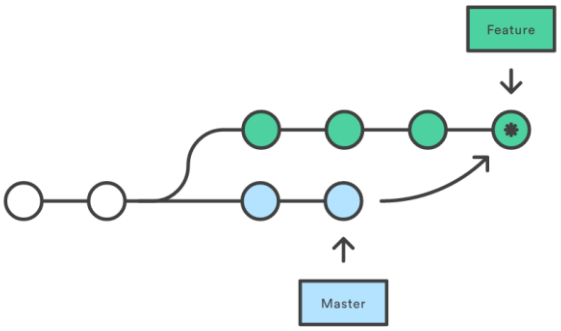


Obrázok 9:Okno vytvorenia novej vetvy

3. Mergovanie vetiev

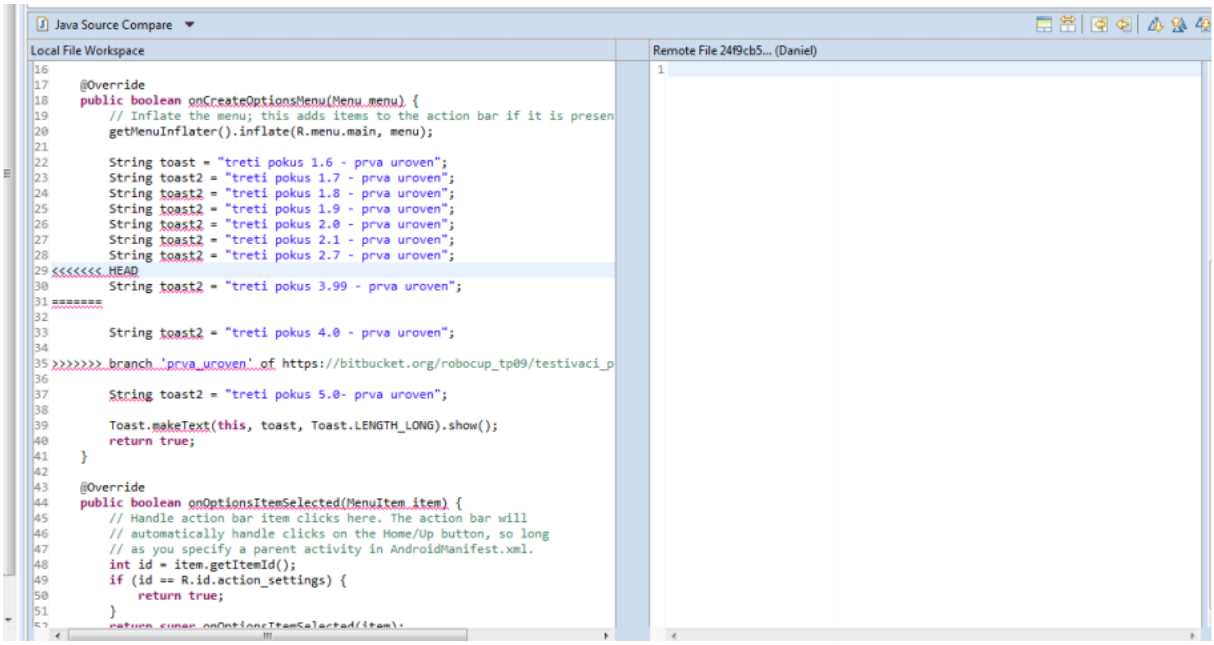


Obrázok 10:Prehľad situácie pred merge



Obrázok 11: Situácia pred a po mergovaní

- Mergovanie v zásade prebieha dvoma spôsobmi. Vykonáva sa buď automaticky fast-forward kde nie sú žiadne konflikty v kóde.
- Väčšinou ale fast-forward metóda nestačí, pretože viacero ľudí robí nad rovnakými zdrojovými súbormi. Vtedy nastávajú konflikty ktoré treba ručne vyriešiť.



Obrázok 12: Príklad konfliktu v kóde

Operácia merge nám spojila konfliktné riadky nasledovne :
<<<<< HEAD
V tejto časti je kód vetvy z lokálneho repozitára, a ten čo sme tu mali doteraz.
====== označuje oddeľovač medzi lokálnou (remote) a vzdialenou verziou.
V tejto časti je kód zo vzdialeného repozitára. Vyberieme si, ktorý z kódov, chceme ponechať a ostatné riadky odstránime. Takisto je potrebné odstrániť ohraničujúce značky
>>>> branch 'prva uroven' – koncová značka remote vetvy.

4. Prehľad príkazov

- Git clone – naklonovanie repozitára
- Git pull – stiahnutie zmien z repozitára

- Git add – vloženi súborov do stage area. Pred commitom je potrebné pridať zvolené súbory do stage aby ich bolo možné v ďalšom kroku commitnúť. Takisto touto cestou sa volia presne súbory ktoré sú relevantné pre daný commit.
- Git commit – commitnutie zmien ktoré sú v stage area. Je potrebné písať commit správy aby sa vedelo o čom daný commit je.
- Git push – uloží zmeny na vzdialený repozitár. Príkazy commit a push je možné chápať ako ukladanie zmien na lokálnom repozitári – commit a vzdialenom repozitári – push
- Git checkout – prepínanie medzi vetvami alebo obnovenie súborov na pracovnej vetve
- Git stash – slúži na odkladanie zmien ktoré nie sú commitnuté, ale v budúcnosti sa k nim programátor chce vrátiť.
- Git merge – príkaz na spájanie dvoch vetiev. Nedoporučuje sa používať rebase, toto je bezpečnejší prístup ku riešeniu problémov.

Jednotlivé príkazy samozrejme nie je potrebné písať cez konzolu, na to slúži buď prostredie Eclipse s Git pluginom, alebo preferovaný SourceTree od Atlassianu. Uvázdané sú len ako prehľad pre rýchlu orientáciu.

Vypracoval: Matúš Ivanoc
Upravoval: Ján Ďurica