Metodika Git

Tabuľka verzií

DÁTUM	VERZIA	POPIS ÚPRAV	ZODPOVEDNÁ OSOBA
16.11	1.0	Vytvorenie prvej verzie dokumentu	Jaroslav Loebl

Obsah

1 Metadata

Vymedzenie obsahu

Toto je metodika pre prácu s verziovacím systémom Git, špecificky vo verzií 1.9.4.

Dedikácia metodiky

Metodika je určená pre každého člena tímu, ktorý bude pracovať so zdrojovým kódom aplikácie.

Poznámka k prekladom

Nie všetko sa dá rozumne preložiť a kvôli ďalšiemu hľadaniu informácií som sa rozhodol neprekladať niektoré bežne používané výrazy označujúce príkazy v Gite:

Push – uloženie lokálny zmien na server (vzdialený repozitár)
Commit – uloženie lokálnych zmien do lokálneho repozitára
Merge – zjednotenie dvoch vetiev

Jediný bežne používaný výraz, ktorý sa v dokumente používa preložený:

Branch – vetva

2 Štruktúra hlavného repozitára

Existovať jeden viac menej klasický pseudocentrálny repozitár nazvaný **origin**. Ten obsahuje dve vetvy, jednu s názvom **master**, druhu s názvom **develop**.

Develop vetva na **origin** repozitári je integračná vetva, sem sa pushujú všetky nové featury a iné záležitosti, ktoré potrebujeme integrovať. V momente keď je táto vetva dostatočne stabilná a obsahuje požadované featury, mergujeme ju do master vetvy.

Master vetva je produkčná verzia aplikácie. Vždy obsahuje stabilný kód a každý commit do nej je vlastne vydanie novej verzie.

Developeri majú u seba klon tohto repozitára:

```
$ git clone https://jarino@bitbucket.org/user/projekt.git
```

U seba následne vytvárajú podporné vetvy ktoré naspäť mergujú do centrálneho repozitára.

3 Podporné vetvy

Pre každý vyvíjanú vec sa vytvára osobitný branch, či už ide o novú vlastnosť alebo hotfix. Udržuje sa tak lepší prehľad vo vývoji a v histórií repozitára.

Feature vetvy

- vetvy, v ktorých sa vyvíja nová funkcionalita
- vytvárajú sa z develop vetvy a merguju sa naspäť do develop vetvy.
- konvencia pomenovania:
  - ***feature/cislo_tasku***
  - Každá featura by mala po správnosti spadá pod nejaký task, ak k nej z nejakých dôvodov task neexistuje, tak treba task vytvoriť, a ak ani to nie je možnosť, tak vetvu pomenovať ***feature/opis***, kde opis je čo najkratší a najvýstižnejší opis o čom vetva je. V takomto neželanom prípade potom pri mergovaní treba do commit správy okrem iného napísať aj prečo nemohol byť k danej vetve priradený task.
- Príkaz na vetvenie:

```
$ git checkout -b feat-456 develop
```

- Príkazy na mergovanie:

```
$ git checkout develop
$ git merge --no-ff feat-456
$ git branch -d feat-456
$ git push origin develop
```

Release vetvy

- vetvy, ktoré slúžia na pomoc a prípravu pred vydaním novej produkčnej verzie (teda commitu do master vetvy). Môžu sa v nich opravovať posledné chyby, pripadne nastavovať rôzne metadáta, typicky pridaním tagu s číslom verzie. vytvaraju sa z develop vetvy a merguju sa zaroven do develop a master vetvy
- konvencia pomenovania:
  - ***release/verzia***
  - kde verzia je číslo verzie, ktoré bude odpovedať aktuálnemu číslu verzie produkcie po commitnuti do master vetvy
- príkaz na vetvenie:

```
$ git checkout -b release-1.2.1 develop
```

- príkazy na mergovanie (mergujeme do dvoch vetiev):

```
$ git checkout master
$ git merge --no-ff release-1.2.1
$ git tag -a 1.2.1
$ git checkout develop
$ git merge --no-ff release-1.2.1
$ git branch -d release-1.2.1
```

Hotfix vetvy

- vetvy ktoré sa používajú na opravy chýb zistených v produkcií.
- vytvárajú sa z master vetvy a mergujú sa naspät do master vetvy a zároveň aj do develop vetvy. **Stále platí, že sa pri ich mergovaní vytvára nová produkčná verzia.** Do týchto vetiev treba rovno pridávať aj tag.
- konvencia pomenovania:
  - ***hotfix/verzia[-*]***,
  - kde za pomlčkou je verzia master vetvy, v ktorej sa nachádza chyba vyžadujúca tento hotfix a za druhou pomlčkou dobrovoľný mini-opis
- príkaz na vetvenie

```
$ git checkout -b hotfix-1.2.1-fatal master
```

- mergovanie je identické ako v prípade release vetiev.

4 Archivovanie hotových vetiev

Po každom uvoľnení novej veľkej verzie do produkcie nastáva archivovanie vetiev, aby sa uzavreté vetvy nemiešali s aktuálnymi. Vetvu archivujeme jej otagovaním:

```
$ git tag -a archive/nazov_vetvy_v_archive nazov_archivovanej_vetvy
$ git branch –d nazov_archivovanej_vetvy
```

Pridanie tagu nám zaručí uchovanie vetvy aj s commit správami aj po jej vymazaní. V tomto prípade nemusíme vytvárať annotated tagy (viď kapitola Taggovanie).

Na obnovenie archivovanej vetvy do novej:

```
$ git checkout –b nazov_vetvy_arch archive/nazov_vetvy
```

Pri takomto vytváraní vetvy z archívu treba za názov novej vetvy pridať *_arch*, aby bolo jasné, že ide o vetvu ktorá je z nejakého dôvodu vytiahnutá z archívu.

5 Fast-forward

Všetky merge sa vykonávajú s **flagom** ***--no-ff***, to zabezpečí, že sa nevykoná fast-forward aj v prípade, že je to možné a tým pádom sa zachová informácia o vetvení, ktorá by inak bola zabudnutá.

6 Rebase

Nepoužívať ako náhradu mergovania, keďže sa snažíme vyhnúť fast-forward. Ideálne iba v prípade, že do vetvy, z ktorej som vytváral moju vetvu, medzičasom prišlo veľa commitov a moja práca ešte nie je hotová – priebežne si takto uchovávať aktuálny stav vetvy.

```
$ git rebase nazov_vetvy
```

Kde názov vetvy je vetva, na ktorej aktuálny stav sa chceme dostať.

7 Commitovanie

Commitovať treba ucelené, ideálne aj funkčné celky práce. Príkaz:

```
$ git commit –a
```

Prepínač –a automaticky pridá do commitu zmenené a vymazané súbory. *Untracked* súbory stále treba pridať manuálne cez:

```
$ git add nazov_suboru
```

Nepoužívať inline zadávanie commit správ cez prepínač –m.

Revertovanie zmien

Na odstránenie súboru z commitu (zmeny ponechá, len súbor sa necommitne):

```
$ git reset HEAD nazov_suboru
```

Na úplne zabudnutie lokálnych zmien:

```
$ git checkout -- nazov_suboru
```

Keď sa náhodou vyskytne chyba v commite (omylom pridaný aj zlý súbor, chyba v commit správe...), nevyrábať ďalší commit na opravu, ale namiesto toho použiť:

```
$ git commit –-amend
```

Commit s prepínačom --amend prepíše posledný commit.

Na vrátenie projektu do stavu určitého commitu:

```
$ git checkout checksum_commitu
```

Commit messages

Správy, ktoré sa vypĺňajú pri **každom** commite by mali stručne a jasne vystihovať obsah daného commitu.

Prvý riadok commit správy je viac menej názov, alebo nadpis daného commitu – nesmie obsahovať viac ako 50 znakov a končiť bodkou. Z nadpisu musí byť jasné, aké zmeny sa v commite udiali.

Pri mergovaní, nech je názov commitu vždy v tvare:

```
merge source_branch->target_branch
```

Kde source branch je vetva v ktorej sme vyvíjali novú funkcionalitu a target branch je vetva do ktorej ideme mergovať (teda väčšinou develop alebo master).

Ďalšia časť commit správy je podrobnejší opis. Od nadpisu ho oddeľujeme prázdnym riadkom. Riadky v popise nesmú mať viac ako 72 znakov a odsek treba oddeliť prázdnym riadkom.

Na záver, pod podrobnejší popis, ešte uviesť aj číslo tasku a priradenej user story (tento odsek opäť oddeliť prázdnym riadkom). Sem možno potom hodiť ešte aj nejaké iné externé zdroje, ako napríklad linky na riešenie, alebo zdroj inšpirácie s podrobnejším popisom.

Príklad commit message:

Refactoring triedy BadOne do novej triedy NiceOne
Kod v triede BadOne sa zvrhol v nenormalny bordel v ktorom sa nedalo
vyznat. Refactoringom sme zabranili zvacsovaniu technologickeho dlhu.
Nova trieda bola nakodena tak pekne, ze sme jej dali rovno krajsie meno.
API kupodivu zostalo nezmenene.

Suvisiaci task: 789
User story: 456

8 Taggovanie

Pri commite do master vetvy treba vždy vytvoriť aj tag, ktorý určuje daný release, aby bola história prehľadnejšia. V Gite sú dva druhy tagov:

- lightweight - pointer na určitý commit,
- annotated - celý objekt, majú vlastný tagging message, meno developera ktorý tag pridal, jeho email, dátum atď. **budeme používať tento druh tagov.**

Príklad pridania annotated tagu:

```
git tag -a v1.4 -m ‘dolezity milnik’
```

Tag pripájame ku commitu, takže ho treba vytvoriť ešte pred commitovaním. Ak sa na to náhodou zabudne, je možné pridať tag aj do už existujúceho commitu uvedením checksumu commitu za verziou:

```
git tag -a v1.4 fc3b96f -m ‘dolezity milnik’
```

Keď tagy vytvárame lokálne a potom ich pushujeme na server, `git push` **automaticky tagy nezahŕňa.** Na ich pridanie je potrebný flag `–tags`.

9 Stashing

Príkaz git stash zobere všetky lokálne zmeny a uloží ich mimo working area, čím „vyčistí“ stav –zmeny vráti a git status nám vypíše, že pracovný adresár je čistý. Toto sa hodí, keď chceme zmeniť vetvu, pričom nemáme prácu v takom stave aby sme ju mohli commitnúť.

Použitie:

Uloženie zmien:

```
$ git stash -u
```

po spustení príkazu sa lokálne uložia všetky zmeny (s prepínačom –u aj súbory ktoré sú *untracked* – silno odporúčam) a vyčistí sa pracovná oblasť. Zmeny môžeme uložiť aj viackrát.

Na výpis uložených stashov použijeme:

```
$ stash list
```

Ukážkový výpis:

```
stash@{0}: WIP on master: 049d078 uprava controllera
stash@{1}: WIP on master: c264051 uprava modelu
```

Obnovenie zmien uložených v stashi (a rovno ho aj vymaže):

```
$ git stash pop stash@{0} --index
```

Ak nešpecifikujeme ktorý stash sa má vytiahnuť, teda napíšeme iba `git stash pop`, git automaticky zvolí posledný pridaný stash. Prepínač `--index` obnoví aj stav súborov, teda či sú zahrnuté v commite alebo nie. Treba vždy používať verziu s `--index` prepínačom.

Mazanie uložených stashov:

```
$ git stash drop stash@{0}
```

10 Presúvanie zmien do iného branchu

Vytiahnutie stavu z git stash nemusí byť nutne na vetve, v ktorej bol stav uložený. Tým pádom sa dá stash použiť v prípade, že sme nejakým omylom začali vyvíjať v inej vetve.

```
$ git stash
$ git checkout nazov_spravnej_vetvy
$ git stash pop
```

Následne ešte vymazať urobené zmeny.

11 Zlučovanie commitov

Na zlúčenie niekoľkých posledných commitov do jedného sa dá použiť príkaz rebase:

```
$ git rebase –i HEAD~n
```

Kde n je počet posledných commitov ktoré chceme zlúčiť. Po spustení príkazu sa nám otvorí textový editor, kde pred každý commit (okrem prvého), ktorý ideme zlučovať prepíšeme „pick“ na „squash“. Následne sa otvorí ďalší textový editor kde napíšeme commit message, rovnako ako je opísané v kapitole o commit správach. Informáciu o squash pridávať netreba, keďže ide upratovanie commitov pred zverejnením (mergom do hlavnej vetvy) pred kolegami.