Git Conflictbeheersing

Dit document bevat een aantal tips die ervoor zorgen dat je minder merge conflicten krijgt in Git. Ze gaan niet alleen over het merge commando (`git merge`) zelf, maar ook over werkwijzes die de kans op een merge conflict verkleinen.

Heel in het algemeen geldt: ZORG DAT JE EEN SCHONE WORKING DIRECTORY HEBT!!!

Begin niet aan een merge als je geen schone working directory hebt; begin niet aan een pull als je geen schone working directory hebt; ga niet naar een andere branch als je geen schone working directory hebt.

Ontsnappen aan een merge conflict

Scenario
Er is een merge conflict ontstaan en je wilt je eigen branch “gewoon terug” in de staat zoals die was.

Oplossing
```
git reset --hard
git clean -df
```
NB hier mee worden ook alle ongecommitte changes ongedaan gemaakt en alle nieuwe files verwijderd. Daarom moet je nooit aan een merge beginnen als je working directory vervuild is (als er nog nog untracked files of ongecommite changes zijn).

Beste manier om een merge uit te voeren

Scenario
Je wilt de laatste wijzigingen in de master/development branch naar je eigen topic branch trekken.

Oplossing
➤ Zorg dat je op je eigen branch zit:
```
git checkout my-topic-branch
```
➤ Doe een diff met de master/development branch, om al te grote verrassingen te voorkomen (met name aangaande files waarin jijzelf hebt zitten werken)
```
git diff development
```
➤ (Optie A) Zorg dat je een schone working directory hebt (alles committen):
```
git add --all
git commit -m “Changed stuff”
```
➤ (Optie B) Zorg dat je een schone working directory hebt (alles terugzetten):
```
git reset --hard
git clean -df
```
➤ Merge:
```
git merge -Xtheirs [Xignore-all-space] development
```

De `ignore-all-space` optie kun je gebruiken als je zeker weet dat er alleen maar code is

veranderd (PHP files, Java files, etc). Dit voorkomt conflicten als gevolg van het feit dat iemand de code formatter op een file heeft los gelaten.

Met de `theirs` strategie hoef je niet bang te zijn dat changes die jij hebt gemaakt sinds jouw branch afsplitste van de development branch verloren gaan. Alleen als de development branch sindsdien **ook** wijzigingen heeft ondergaan in dezelfde files krijg je conflicten, maar daar is niets aan te doen.

Als je zeker weet dat jouw versie van de file(s) de betere is kun je -Xours meegeven. Het belangrijkste is dat als je jouw branch weer terug-merget naar de master er geen conflicten meer zijn.

Terugkeren naar de master/development versie van een file

```
git checkout my-own-topic-branch
git checkout development index.php
git add index.php
git commit -m
```

(Dit kun je uiteraard gebruiken om uit elke willekeurige branch om een file naar je eigen branch te halen.)

Ook hier geldt dat je working directory, of in ieder geval index.php, schoon moet zijn. index.php mag niet *modified* zijn.

Terugkeren naar een eerdere versie van een file

Ook dit doe je middels een checkout

```
git checkout <old-commit> index.php
git add index.php
git commit -m
```

Complexe merges

Als je een `git diff` doet en je constateert dat de branches ver uit elkaar zijn gaan lopen, weet je misschien niet of je beter de `ours` of de `theirs` strategie moet gebruiken. In dat geval kun je “cherry picking” met behulp van de vorige twee tips. Per conflicterende file kies je welke versie je wilt hebben.Dus bijvoorbeeld:

```
git checkout my-own-topic-branch
// Voor aaa.php en bbb.php moet de development versie gebruikt worden:
git checkout development aaa.php
git checkout development bbb.php
// Voor de rest, bij conflicten mijn versie gebruiken:
git merge -Xours development
```

Wat kun je wel en niet verwachten van een merge

Om niet voor onaangename verrassingen komen te staan is het belangrijk dat je je realiseert dat het moment dat twee branches van elkaar afsplitsten cruciaal is voor het merge proces. Dit heeft de volgende gevolgen:

Scenario: file verwijderd (1)
Jij hebt een file verwijderd uit jouw topic branch (en gecommit). In de master branch is file gewoon blijven bestaan. Merge: **master ➜ topic**.

Gevolg
Er gebeurt niets. Je hoeft niet bang te zijn dat de file weer in je eigen branch terecht komt. Omgekeerd hoef je er ook niet op te rekenen dat dit een manier is om de file weer terug te krijgen in jouw eigen branch!

Scenario: file verwijderd (2)
Jij hebt een file verwijderd uit jouw topic branch. In de master branch is file gewoon blijven bestaan. Merge: **topic ➜ master**.

Gevolg
De file wordt uit de master branch verwijderd.

Scenario: file verwijderd (3)
Een andere developer heeft een file uit de master branch verwijderd. Merge: **topic ➜ master**.

Gevolg
De file wordt uit *jouw* topic branch verwijderd.

Terugkeren naar een vorige commit

Als de commit nog niet gepushed is (undo local changes)
```
git reset --hard <old-commit>
git clean -df
```
Het reset commando verwijdert nog altijd *niet* untracked files in je working directory. Vandaar het clean commando. Nogmaals: nooit doen als de commit waarnaar je terugkeert (of één van de daarop volgende) al gedeeld is met andere developers.

Als de commit wel al gepushed is
Als je een bescheiden aantal commits hebt gedaan sinds de commit waarnaar je terug wilt keren:
```
git revert C10 C9 C8 C7
```
Hiermee keer je terug naar C6. (In werkelijkheid maak je een nieuwe commit C11 aan waarvan de code base identiek is aan C6, maar die wel volgt op C10.) Met `git log` bekijk je de commit history in omgekeerd-chronologische volgorde.

Terugkeren naar een zeer oude commit
Als je ver in de tijd (oftewel veel commits) terug wilt gaan:
```
git reset --hard <old-commit>
git clean -df
git checkout -b <new-branch>
git push origin <new-branch>
```

Maar hiermee heb je <old-branch> in feite obsolete gemaakt, wat pijnlijk is als het om de master branch gaat of andere belangrijke gedeelde branches. In feite zou je <old-branch> moeten verwijderen zodat niemand in de verleiding kan komen er op te committen.

Checkout nieuwe branch terwijl je nog ongecommitte changes hebt

Scenario
Je werkt in branch A en je wilt switchen naar branch B terwijl er nog ongecommite changes in branch A zijn (veranderde of nieuwe files).

Oplossing
Niet doen! In tegenstelling tot wat je zou verwachten wordt je working directory daarmee niet schoon geveegd (zodat je een maagdelijke versie van branch B hebt). De nieuwe files staan nog steeds in je working directory (als *untracked*) en de veranderde files zijn niet in hun oude staat herstelt. Als je vervolgens een commit doet, commit je dus de changes die je in branch A had gemaakt naar branch B. Dus: voordat je van branch verandert, altijd eerst
ofwel:

```
git commit -am “Changed stuff”
git clean -df
```

ofwel:

```
git reset --hard
git clean -df
```

In tegenstelling tot de angsten en geruchten: als je maar zorgt dat je een schone working directory hebt, zullen files die wel in branch B staan, maar niet in branch A gewoon “verdwijnen” als je naar branch B switcht, en ze zullen ook gewoon weer terugkomen als je terugkeert naar Branch A.