howest
hogeschool

Git

Continuous Integration Basics

Graduaat in het Programmeren

INHOUD

1 COMMITS

1.1 COMMIT MESSAGES

Wanneer je de logs bekijkt van een willekeurige Git repository zal je zien dat sommige commits duidelijker zijn omschreven dan andere.

Omdat je zéér waarschijnlijk als team zal werken in één repository is het belangrijk om goede afspraken te maken rond de boodschap die een commit moet overbrengen. Een goede Git commit message is de beste manier om **context** te bieden aan je collega's over de verandering die werd aangebracht.

> *A commit message shows whether a developer is a good collaborator*

Wanneer commits gebeuren volgens een aantal afspraken dan is de log *(git log)* een zeer overzichtelijk en bruikbaar instrument. Het doornemen en visualiseren van wat andere developers gewijzigd of toegevoegd hebben aan een project wordt dan een pak makkelijker. Aangezien de meeste projecten een lange levensduur hebben geven goede commit messages een makkelijker beeld waarom iets werd veranderd maanden *(of zelfs jaren)* geleden.

In de beginfase zal je net iets langer moeten nadenken over het maken van een goede commit message en zal dit omslachtig lijken. Maar dit zal al snel herleid worden tot een gewoonte en uiteindelijk een bron van trots en productiviteit voor iedereen die betrokken is bij de repository.

1.2 AFSPRAKEN ROND COMMIT MESSAGES

In dit hoofdstuk zullen we het hebben over de basiselementen die nodig zijn om een gezonde commit history (git log) te bekomen en/of te behouden.

Een goede commit message moet de lezer/developer vertellen **waarom** iets werd aangepast. Als men wil weten **wat** er werd aangepast moet men hiervoor in de code zelf gaan kijken.

Zoals elke programmeertaal zijn afspraken/conventies heeft (naamgeving, formatting, …) zullen we ook omtrent commit messages goeie afspraken maken die gevolgd zullen moeten worden. We onderscheiden hierin drie zaken:

- **Stijl**
  Markup syntax, wrap margins, grammatica, hoofdlettergebruik, leestekengebruik.
  Maar hieromtrent afspraken in het team zodat alle commits consistent zijn opgebouwd.
- **Content**
  Wat moet er in de body van een commit message staan? Wat moet er zeker niet in staan?
- **Metadata**
  Hoe moeten er referenties gelegd worden naar bijvoorbeeld issue tracking ID's, pull request nummers, …

1.3 5 ALGEMENE AFSPRAKEN OMTRENT EEN COMMIT MESSAGE

Gelukkig voor ons zijn er al enkele afspraken gemaakt omtrent verschillende zaken. We gaan zelf niks uitvinden maar door gebruik te maken van onderstaande acht afspraken zullen we zo meer consitentie in onze git commits kunnen brengen.

1.3.1 MAAK JE COMMIT MESSAGE IN HET ENGELS

Aangezien Engels de voertaal is in het programmeren zullen ook wij afspreken om onze commit messages in het Engels te schrijven.

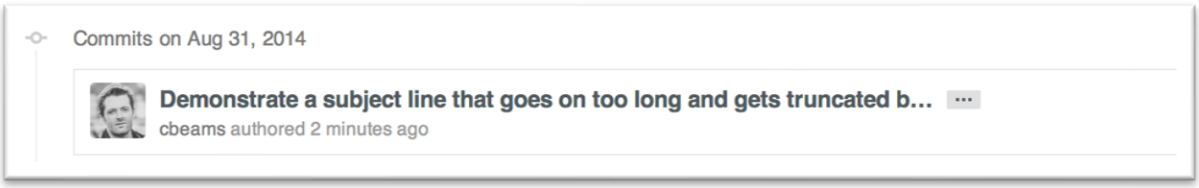
1.3.2 BEPERK DE TITEL TOT 50 KARAKTERS

Dit is geen harde limiet, maar eerder een vuistregel. De harde limiet bedraagt 72 karakters. Hierdoor worden je commits leesbaarder voor je collega's maar dien je wel even na te denken over je titel om duidelijk te maken wat er gebeurt in je commit.

Maar indien je te hard moet nadenken over een goeie titel heb je misschien teveel wijzigingen in je code aangebracht per commit. Het is dan verstandiger om kleine wijzigingen te committen.

Ook de UI van GitHub volgt deze afspraken en geeft je een waarschuwing wanneer je over de 50 karakters gaat:

Als je titel langer is dan 72 karakters zal deze afgekapt worden in de UI. Probeer dus te mikken op 50 karakters voor je titel:



Wanneer we gebruik maken van de commando's `git log`, `git log --oneline` of `git shortlog` geeft ons dit een mooi overzicht.

1.3.3 BEGIN DE TITEL MET EEN HOOFDLETTER

Misschien wel de meest simpele afspraak; begin elke titel met een hoofdletter.

1.3.4 EINDIG DE TITEL NIET MET EEN PUNT OF ANDER LEESTEKEN

Nog een simpele afspraak dus. Zeker wanneer je moet mikken op 50 karakters voor je titel telt elk karakter mee.

1.3.5 GEBRUIK DE GEBIEDENDE WIJS IN JE TITEL

Schrijf of zeg zoals je een commando of een instructie zou geven aan iemand. Bijvoorbeeld:

- Ruim je kamer op
- Sluit de deur
- Zet de vuilniszakken buiten

Alle afspraken die we hier noteren zijn ook in de gebiedende wijs geschreven. Misschien klinkt de gebiedende wijs af en toe wat grof of onbeleefd, want meestal spreken we niet in deze vorm tegen elkaar, maar deze vorm is perfect voor git comment titels. En niemand zal zich beledigd voelen als iedereen weet dat dit een afspraak is.

Ook Git zelf gebruikt de gebiedende wijs, bijvoorbeeld:

```
$ Merge branch “myFeature”
```

Om verwarring te vermijden kan je volgende regel toepassen:

- If applied, this commit will *plaats de titel hier*

Bijvoorbeeld:

- If applied, this commit will *Refactor subsystem X for readability*
- If applied, this commit will *Update getting started documentation*
- If applied, this commit will *Remove deprecated methods*
- If applied, this commit will *Release version 1.0.0*
- If applied, this commit will *Merge pull request #123 from user/branch*

Zie hoe dit niet werkt in de niet-gebiedende wijze:

- If applied, this commit will *Fixed bug with Y*
- If applied, this commit will *Changing behavior of X*
- If applied, this commit will *More fixes for broken stuff*
- If applied, this commit will *Sweet new API methods*

Deze gebiedende wijs gebruiken we enkel in de titel, niet in de body.

1.4 AFSPRAKEN INDIEN JE OOK GEBRUIKT MAAKT VAN DE BODY

1.4.1 ONDERSCHEID TUSSEN TITEL EN BODY

Maak gebruik van een duidelijke titel in je commit. Hoewel dit niet verplicht is, wordt het toch sterk aangeraden om dit te doen. Het onderscheid tussen een titel en een body doen we door middel van een lege tekstregel te plaatsen tussen de titel en de body.

Verschillende Git platformen zullen alle tekst tot aan de lege tekstregel aanzien als een titel. In de body kan je dan wat meer duiding geven door een grondiger beschrijving te maken.

Uitzonderingen zijn er altijd; bijvoorbeeld: “Fix typo in introduction to user guide”. Deze commit zegt al meer dan voldoende; we hebben een typfout opgelost in de gebruikershandleiding. Hierdoor hoeven we geen titel en body te gebruiken om (nog) meer duiding te geven.

Opgepast! Indien je commits doet via Git Bash kun je een titel en body of enkel een titel opgeven. Bijvoorbeeld enkel een titel:

```
$ git commit -m "Fix typo in introduction to user guide"
```

Soms is het beter om toch een body mee te geven om meer duiding te geven. Bijvoorbeeld:

```
Remove portal to the underworld

Apply primal spells, rephrase incantations because the spawn was too
powerfull
Close #666
```

We kunnen dit op twee manieren doen in Bash:

- Git commit zonder flags:

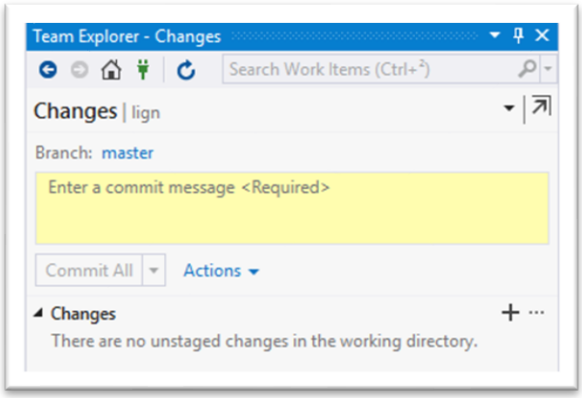
```
$ git commit
```

  Hierdoor zal VS Code (of een andere ingestelde editor openen) waarin je uw commit kan schrijven. Deze commit wordt uitgevoerd nadat je de editor sluit.

- Git commit met titel en body:

```
$ git commit -m "Titel" -m "Body"
```

We kunnen dit ook doen via de Git tools die ingebouwd zijn in Visual Code en/of Visual Studio.



1.4.2 WRAP DE BODY OP 72 KARAKTERS

Git zal nooit de tekst automatisch gaan “wrappen”. Je moet hier dus zelf rekening mee houden en je tekst manueel gaan wrappen.

Het wordt aanbevolen om dit te doen op 72 karakters zodat Git nog ruimte genoeg heeft om tekst te laten inspringen en alles onder de 80 karakters te houden.

Een teksteditor zoals VS Code kan hier zeker bij helpen. Onderaan deze applicatie kan je aflezen op welke regel je zit en op welke kolom (het aantal karakters dus)

1.4.3 GEBRUIK DE BODY OM WAT EN WAAROM UIT TE LEGGEN, NIET HOE

Leg in je body uit **wat** en **waarom** je iets hebt aangepast. Maar leg niet uit **hoe** je iets hebt aangepast. Indien men wil weten hoe je iets hebt aangepast kan men in de code van deze commit gaan kijken.

1-1 xkcd.com

2 .GITIGNORE

2.1 WAT IS HET NUT VAN EEN .GITIGNORE

Omdat we niet altijd alle bestanden uit onze lokale repository wensen op te nemen in de remote repository hebben we uiteraard een manier nodig om dit kenbaar te maken aan git.

In de context van onze opleiding denken we hier bijvoorbeeld aan;

- Configuratie bestanden die gevoelige informatie bevatten.
  (denk hierbij aan wachtwoorden of keys die gebruikt worden door andere applicaties)
- Folders die je niet meteen mee wenst te synchroniseren
  (voor visual studio zijn dit bijvoorbeeld de .vs, bin en obj mappen)
- Documentatie of andere bestanden die voor jezelf van belang zijn, maar niet meteen voor alle andere developers die aan het project werken.
- …

Tal van redenen dus om bepaalde bestanden niet te gaan delen met de remote repository. De .gitignore file zal er dus voor zorgen dat wat je niet wenst op te nemen in je versiebeheer ook daadwerkelijk genegeerd *(ignored)* zal worden. De naam spreekt dus al voor zich!

2.2 WAAR PLAATS JE HET .GITIGNORE BESTAND?

Je .gitignore bestand bevind zich bij conventie **in de root van je repository**. Met andere woorden in de map waar je ook je .git folder ziet staan. *(**Let op!** **Niet in** je .git folder)*

Het is technisch perfect mogelijk om *meerdere .gitignore* bestanden in je repository te hebben. Echter voor de scope van deze module *(alsook de opleiding)* houden we het op **1 .gitignore** bestand per repository.

2.3 HOE MAAK JE EEN .GITIGNORE BESTAND AAN?

Een .gitignore bestand is een gewoon bestand waar je als developer de nodige zaken in noteren kan die niet door git meegenomen moeten worden in het versiebeheer. Hoe je dit precies noteert en wat de mogelijkheden zijn, daar komen we in het volgende puntje op terug.

Het aanmaken van een .gitignore file via je console kan via het touch commando. Uiteraard is het aanmaken rechtstreeks in de folder (via de rechtermuisknop, nieuw bestand aanmaken) ook prima.

Let op! Een **.gitignore** bestand heeft geen bijkomende extentie! Schakel je weergave om bestandsextenties te zien dus zeker aan, zodat je OS hier default geen .txt achter gaat plaatsen.

2.4 HOE WERKT ZO EEN .GITIGNORE NU EIGENLIJK?

Eenmaal je **.gitignore** aangemaakt is kan je deze gaan openen en bewerken met eender welke editor.

Indien je deze vanuit de commandline in je console willen openen dan kan je gebruik maken van het commando `code .gitignore` .

Je **.gitignore** bestand zal vervolgens openen in de editor die je tijdens je installatie als default editor aangevinkt hebt. Normaal zou dit VS Code moeten zijn indien je de stappen voor installatie uit deze cursus gevolgd hebt.

De .gitignore kunnen we vervolgens regel per regel gaan aanvullen, hou er uiteraard rekening mee dat wat je hierin aanvult niet opgenomen zal worden in je git *(versiebeheer)*. De werking van het bestand zelf is heel simpel en kunnen we aan de hand van een kleine opsomming perfect duidelijk maken.

- **Lege regels** worden genegeerd
- Regels die beginnen met een hash *(#)* kan je gebruiken om commentaar of notities toe te voegen.
  Vb: # Onderstaande bestanden zijn niet relevant
- Een regel met de **letterlijke bestandsnaam** *(samen met de extentie)* zorgt ervoor dat dit bestand eender waar het in de repository staat genegeerd zal worden.
  Vb: mijnDocumentatie.pdf
- Een regel met de **naam van een folder** zal een folder met diezelfde foldernaam, eender waar in je repository, gaan negeren. Hou er rekening mee dat je steeds een forward slash aan het einde van je foldernaam toevoegt.
  Vb: deze-folder/
- Een regel waar een **asterisk** in voorkomt, wat gecombineerd kan worden met bovenstaande regels.
  *Vb: *.pdf (alle pdf bestanden negeren)*
  Draft/.xlsx (alle Excel bestanden in de folder Draft negeren)*
- Een regel die met een uitroepteken start zal dan net het omgekeerde gaan doen. Een mogelijkheid dus om een eerdere algemene regel van uitzonderingen te gaan voorzien.
  *Vb: *.pdf (alle pdf bestanden negeren)*
  !docs.pdf (behalve het pdf bestand met exacte naam docs.pdf)

howest.be