Hands-on Workshop - Git Introduction

Timothy Callemein
EAVISE, KU Leuven

1. Introduction

We vertrekken van het idee dat we een nieuw project maken. Hiervoor maken we eerst we een nieuwe folder,

```
$ mkdir -p /home/USER/Desktop/git_course
```

Momenteel is dit een lege folder, zonder project data noch een git repository. We gaan beginnen met eerst een README aan te maken in een IDE. Zet in jou README.md bestand een korte beschrijving met een Header en een korte uitleg over het project.

```
#Git Course Website
Dit project wordt gebruikt om enkele concepten van GIT toe te lichten.
```

Momenteel beschikken we over een projectfolder, met al een projectfile maar nog geen Git reposotory. Om deze aan te maken openen we een terminal en gaan we naar de project root folder.

```
$ cd /home/user/Desktop/git_course
```

Om hier een nieuwe git repository in aan te maken gebruiken we het git commando en het init argument

```
$ git init
```

Wanneer we dit commando uitvoeren zien we dat er een verborgen folder wordt aangemaakt waarin alle git gerelateerde bestanden worden toegevoegd, genaamd *.git*.

```
$ l
```

Een commando die je vaak gaat gebruiken is het status commando. Probeer vaak tijdens het gebruiken van git dit commando tussen verschillende stappen uit te voeren. Zo weet je steeds wat er aan de hand is en kan je jezelf controleren.

```
$ git status
```

1.1. Commits

Aangezien we net een leeg project en repository hebben aangemaakt kent git momenteel nog niets. Ondanks dat we een README.md bestand hebben, heeft git nog geen interactie met dit bestand. We onderscheiden hier dan ook drie domeinen waarin git werkt.

- *untracked*
- *staging*
- *tracked*

Momenteel zien we dat onze README de status *untracked* krijgt. Dit is normaal aangezien we dit bestand nog maar net hebben aangemaakt en nog niet hebben toegevoegd aan onze repository. Om een bestand toe te voegen gaan we eerst de desbetreffende bestanden toevoegen aan de *staging* list.

```
$ git add README.md
```

Als we nu opnieuw een status opvragen zien we dat onze readme inderdaad in de *staging* area zit.

```
$ git status
```

Staging
De *staging* area is een tussenzone waarin je verschillende bestanden kan toevoegen. Wanneer je bestanden zoals hierboven eraan toevoegt, zal git de wijzigingen ten opzichte van de vorige versie zoeken. In dit geval is het bestand nieuw dus zijn er 2 wijzigingen, het bestand werd aangemaakt en er werden X aantal lijnen aan toegevoegd.

Om het bestand nu definitief in onze repo op te nemen moeten we hem comitten naar de branch. Hierbij is het een goede gewoonte om altijd nuttig commentaar over deze commit toe te voegen. Nuttig commentaar kan een korte beschrijving zijn, maar ook een *Issue number* kan nuttig zijn.

```
$ git commit -m "Aanmaken README"
```

Oeps... nu krijgen we een error, want er is by default geen username ingesteld. Tenzij een gebruiker met *–global* werd ingesteld. Indien het er niet is weet git niet welke gebruiker momenteel bezig is met aanpassingen aan te brengen. Dit passen we best aan voordat we ons werk *commiten*.

```
$ git config user.name "Jane Doe"
$ git config user.email "jane.doe@kuleuven.be"
```

Probeer je code opnieuw te *commiten* nadat je de gebruiker hebt aangepast en herbekijke de status van je project. Nu toont de status inderdaad dat alles up-to-date is.
Om een geschiedenis van de vorige commits te bekijken kunnen we het log bestand raadplegen. Je kan hierin navigeren met de pijltjes en stoppen met de **q** toets.

```
$ git log
```

Momenteel staat hier weinig in aangezien we nog maar 1 commit hebben, maar merk wel enkele dingen op. Bij iedere commit staan telkens deze elementen:

- Author van de commit
- Timestamp
- Commit HASH

Het kan voorvallen dat je te vroeg een *add* van bestanden hebt gedaan en je het opnieuw wil doen. Om je de huidige *staging* area weer leeg te maken kan je een reset doen. Dit wijzigt niets aan de bestanden, enkel aan de git *staging* area.

```
$ git reset
```

1.2. Branches

Momenteel hebben we direct in de *master* branch gecommit van de repository. In groepsverband of online is dit echter niet altijd optimaal, zeker tijdens development. Want development code is meestal nog niet zonder fouten en kan dus onstabiel zijn.
Het is dus aangewezen om te werken met branches, vaak zien we dat er gebruik wordt gemaakt van een*devel* branch waarin wordt ontwikkeld. Als we kijken naar alle huidige beschikbare branches zien we momenteel enkel de *master* branch.

```
$ git branch
```

Om een nieuwe branch aan te maken en er onmiddellijk in aan de slag te gaan gebruiken we het checkout commando.

```
$ git checkout -b devel
```

Checkout
Hierboven hebben we gebruik gemaakt van het checkout commando, met een **-b** argument en een naam voor de *branch*. Naast het wisselen van *branch* kan je ook terugkeren in de tijd, gebruikmakende van de HASH die je in het log bestand kan vinden. Je kan kiezen voor de volledige HASH, of de eerste 7 karakters als ID. Elke *branch* heeft een **HEAD**, de recentste commit. Met checkout kan je steeds ook naar de HEAD terugkeren.

```
# Terugkeren naar een vorige commit
$ git checkout HASH_ID

# Als we willen wisselen van branch
$ git checkout BRANCH_NAME

# Als we naar de HEAD terug willen keren
$ git checkout HEAD
```

Opdracht 1
Maak kort en snel een *index.html* pagina waarin we als body een * * en 2 paragrafen *<p>* hebben. De eerste paragraaf heeft drie willekeurige zinnen waarin je jezelf voorstelt op eenzelfde lijn. In de 2de paragraaf dezelfde content, maar na elke nieuwe zin een nieuwe regel of enter.
Commit nadien deze pagina in de devel branch met een nuttige regel als commentaar.

We hebben nu 2 bestanden in onze repo, een README.md en een index.html. Normaal gezien heb je in jou tekst jou naam gebruikt, in mijn geval "Timothy Callemein". Verander nu in beide paragrafen iets subtiel aan jou naam, bv. hoofdletters of de volgorde en sla dit bestand op.
Momenteel is ons bestand die voorheen gekend is gewijzigd ten opzichte van het origineel. Wanneer we opnieuw de status opvragen krijgen we ook de melding welk bestand hier is aangepast.

```
$ git status
```

Omdat we soms vergeten zijn wat we allemaal hebben aangepast kunnen we met ons *version control system* deze wijzigingen vergelijken.

```
$ git diff index.html
```

Denk Oefening
Hier merken we verschillende dingen op?

Stel dat we niet tevreden zijn met deze wijzigingen, en dus liever onze vorige versie behouden dan kunnen we deze wijzigingen ook terugdraaien. **Let wel dit commando verwijderd alle wijzigingen definitief.**

```
$ git reset --hard index.html
```

Als we nu terug kijken naar ons bestand, of naar onze status zien we dat onze branch up-to-date is.
Stel we zijn tevreden met deze index en we willen die graag uit development en in productie stoppen. Dan moeten we deze over kopiëren naar onze master, want normaal staat onze pagina nu in de devel branch.
Om dat te doen kunnen we de devel branch tot op dit punt mergen met de master branch. Dit doe je door eerst naar de master branch te veranderen.

```
$ git checkout master
```

In het log zien we dat onze index pagina niet beschikbaar is, en in onze project folder is de index pagina ook verdwenen na deze checkout
En een merge te starten in de huidige branch

```
$ git merge devel
```

Nu zijn beide de index en de commit te vinden in de project folder en in de log file.

Opdracht 2 Voeg nu eerst in de master branch een H1 toe net voor de eerste paragraaf waarin staat "Master Intro.[en] commit die naar de master branch.
Wijzig dan naar de devel branch, waarin je exact hetzelfde doet (negeer de afwezigheid van de H1), maar dit keer met de inhoud "Devel Intro". Commit deze aanpassing van de intro naar de devel branch.

Hierna mergen we terug de devel branch in de master branch zoals voorheen. Wat gebeurt er hier?

```
$ git checkout master
$ git merge devel
```

In de terminal komt onmiddellijk een auto-merging conflict en dat is logisch ook. We hebben namelijk op 2 plaatsen een h1 toegevoegd aan ons bestand en git kan onmogelijk weten dewelke hij moet kiezen.
In plaats van een keuze te maken gaat hij op plaatsen waar er hij het niet automatisch kan beide opties ter beschikking stellen in de index.html file.
In de status kan je ook zien dat er nog geen merge is gelukt en dat dit dus nog open staat.
Pas in dit geval de code aan zodat de Devel intro blijft staan, deze zou namelijk de recentste moeten zijn"Hierna moet je nog een add en commit doen met een mededeling van de merge resolve

Denk Oefening
Stel dat ik nu voor de *master* branch wijzigingen had gekozen in plaats van de *devel*. Wat zou er gebeuren?

1.3. Online repository

Eerst maken we een public/private repository aan op een online git platform gitlab/github. Normaal is er nadat we dit hebben aangemaakt een url beschikbaar, met op het einde op .git Deze SSH/HTTPS url hebben we nodig om onze repo naar de cloud te pushen.
Om deze link toe te voegen aan onze repo gebruiken we:

```
$ git remote add origin git@github.com:callemein/git_course_test.git
```

Om een goeie sync naar de cloud te hebben zijn moet je ervoor zorgen dat je locale repository geen onbekende wijzigingen in tracked files heeft. Dit kan je controleren met het status commando.
Een manier om deze voorwaarde te overbruggen is gebruik te maken van de *stash*

Stash
Met *git stash push*, stop je alle tracked files hun wijzigen tijdelijk in een lokale opslag. Daarna kan je een sync doen van de cloud. Met *git stash pop* je wijzigingen weer toepassen op de nieuwe versie.
Let op, dit kan ook merge conflicten veroorzaken.
Ik geef zelf de voorkeur aan kleine development branches i.p.v. een *stash*. Zie dit dus eerder als een uitzondering die vooral tijdens development in de *devel* branch handig is. Maar normaal merge je altijd in de *devel* branch en zou dit dus weinig voorvallen.

```
$ git stash push
$ git stash pop
```

Nadat aan de voorwaarde werd voldaan gaan we een git sync doen, in wezen bestaat dit uit 3 stappen.

```
$ git pull origin
$ git push -u origin master
```

De pull bestaat uit 2 commandos fetch en merge, waarbij de fetch de huidige updates download naar de lokale repository. Met erna een merge van deze wijzigingen in jou lokale branches.
De *git push* zorgt in dit geval dat de *master* branch naar *origin* (online) wordt verstuurd. Als we nu online kijken zien we dat dit op de website dingen heeft doen veranderen.

Denk Oefening
We merken ook op dat niet alle branches beschikbaar zijn, bijvoorbeeld de *devel* branch. Hoe komt dit? Wat is hier een voordeel van? Wat is hier een nadeel van?

Ga nu met je terminal naar een nieuwe locatie, niet onder jou huidige project root. We maken hier een clone maken hetzelfde project, maar hernoemen het naar PROJECT_NAAM (kies zelf iets).

```
$ git clone  git@github.com:callemein/git_course_test.git PROJECT_NAAM
```

Met het clone commando ga je een kopie nemen van alles die online staat. Wel enkel de dingen die online staan, dus niet de branches waarin gedevelopt wordt.

Opdracht 3 Probeer nu eens met iemand anders uit de groep je git link uit te wisselen. Daarna probeer je iets te veranderen aan de index van je buur, en die wijziging te pushen. Wat merk je op?
Vraag nu aan jou buur schrijfrechten tot zijn repository en probeer opnieuw. Nadat die wijziging is gebeurt, probeer nu hetzelfde als hiervoor.

Hieronder heb ik nog enkele interessante dingen toegevoegd die buiten de scoop van een introductie vallen.

1.4. Tags

Wordt nog aangevuld.

1.5. Submodules

Wordt nog aangevuld.

1.6. LFS

Wordt nog aangevuld.

1.7. Examples

Wordt nog aangevuld.