Auteur: Michiel Borkent, michiel.borkent@hu.nl

1 Git

Dit gedeelte is bedoeld om je bekend te maken met het distributed version control system `git`. We maken daarbij gebruik van bestaande online materialen.

Stap 1 Lees http://progit.org/book/ch1-3.html. Deze pagina vertelt de beginselen van `git`.

Stap 2 We gaan nu gebruik maken van Github. Ga naar https://github.com/plans en maak een gratis - dat wil zeggen een open source - account aan. Als je dit practicum met zijn tweeën doet, maak dan allebei een eigen account aan, bij voorkeur op gescheiden systemen. Als je alleen werkt, maak dan twee accounts aan. Je hebt dan wel twee verschillende e-mailadressen nodig[1]. Dit is belangrijk verderop in het practicum. Je moet dan namelijk iemand toe kunnen voegen als collaborator aan jouw repository. Het is handig als je jouw practicumpartner dan kan toevoegen, dus moet hij of zij een ook een Github-account hebben.

Stap 3 Installeer daarna `git`. Volg daarvoor het volgende stappenplan: http://help.github.com/win-set-up-git/. Via deze link zijn ook manuals voor linux of Mac OS X beschikbaar. Het is belangrijk dat je ook het gedeelte over ssh-keys voltooit! Er wordt van een ssh-key gebruik gemaakt bij het uploaden, of beter in `git`-terminologie: pushen naar Github. Gebruik bij het aanmaken van de ssh-key hetzelfde e-mailadres als waarmee je je hebt aangemeld bij Github.

Stap 4 In deze stap wordt uitgegaan van Windows. Als je al een linux- of Mac OS X gebruiker bent, dan kan je deze stap waarschijnlijk overslaan. Open "Git Bash" (hierna kortweg bash genoemd) via Start Menu. Om met deze console te kunnen werken moet je wel enkele Unix-zaken c.q. Bash-commando's kennen. Probeer er eens wat uit.

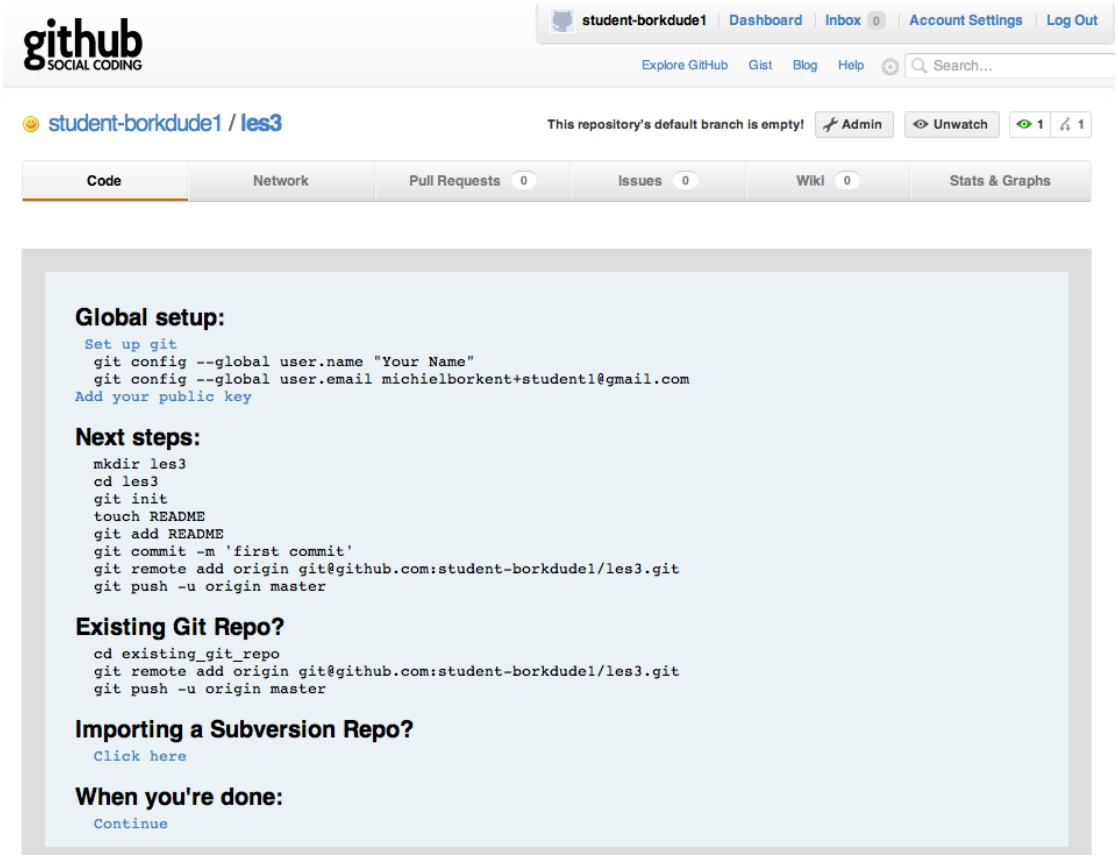
- Directories worden genoteerd met een / ipv een \
- Het station (in Windows bijvoorbeeld `c:` of `d:`) wordt aangegeven met `/<station>`. Dus de directory `c:\projects\les3` wordt in bash weergegeven als `/c/projects/les3`.
- Het commando `mkdir`: een directory aanmaken. Bijvoorbeeld `mkdir /c/projects`.
- Het commando `cd`, dat is "change directory". Bijvoorbeeld `cd /c/projects/les3`
- `ls`: opvragen inhoud directory (`dir` in cmd.exe)
- `rm`: een bestand verwijderen (`del` in cmd.exe)

Zie verder deze lijst met bash-commando's: http://ss64.com/bash/. Daarnaast is het handig om te weten dat als je op het icoontje in de linkerbovenhoek van "Git Bash" klikt, je kunt knippen, plakken, etc. Gebruik de tab-toets voor auto-completion.

Stap 5 Kies op de Github-website (als je bent ingelogd) "New Repository". Vul bij "Project Name" in: "les3" en bij "Description": "test voor les 3". Kies vervolgens "Create Repository". Als het goed is krijg je nu een soortgelijk scherm te zien als in figuur 1.

[1] Met Gmail kun je dit faken door `+...` achter je e-mailnaam te zetten. Bijvoorbeeld: `michielborkent+student1@gmail.com`

Figure 1: Github



Stap 6 Voer nu de stappen uit die in het scherm staan genoemd, in bash. Ga alvorens dat te doen even naar de directory waar je je programmeerprojecten bewaard met `cd`. In die directory ga je dan zometeen een directory genaamd `les3` maken tijdens het uitvoeren van de stappen (`mkdir les3`).

Uitleg bij de stappen.

- `git init` - hiermee maak je van de huidige directory een git repository. Tevens wordt er een `.git`-directory aangemaakt in `les3` waarin allerlei informatie staat. Er staat nu nog niks in de repository. Als er al wel files in de directory stonden, zijn deze niet automatisch toegevoegd aan de git repository. Git kent, zoals je in de introductie hebt kunnen lezen drie niveaus. Zie ook figuur 2.
- `touch README` - hiermee maak je een (lege) file aan met de naam `README`. Dat zou je kunnen controleren door de inhoud van de directory op te vragen met `ls`.
- `git add README` - hiermee voeg je de file `README` toe aan de staging area (zie weer figuur 2). Alles wat is toegevoegd aan de staging area zal met een commit worden toegevoegd aan de git repository.
- `git commit -m 'first commit'` - hier wordt er een daadwerkelijke commit gedaan met als commit message "first commit".
- `git remote add origin git@github.com:.../les3.git` - als remote repository, genaamd `origin` (je had ook een andere naam kunnen kiezen), wordt de repository toegevoegd die je net op github.com hebt aangemaakt.
- `push -u origin master` - verstuur de inhoud van onze `master` branch (dit is de branch waarin we nu werken) naar de remote repository `origin`. `-u` zorgt ervoor dat we de volgende keer simpelweg: `push` kunnen typen om de inhoud van de `master` branch naar `origin` te sturen.

Stap 7 Je kunt bij git-commando's documentatie raadplegen. Als je bijvoorbeeld meer wil weten over `git push` in bovenstaande lijst kun je dat opzoeken in de documentatie van `git push`. Typ om die documentatie op te vragen `git help push` in in de console, gevolgd door een enter. Er opent zich nu een webbrowser met de gevraagde documentatie. Als je de algemene documentatie van `git` wil raadplegen, typ je simpelweg `git help`

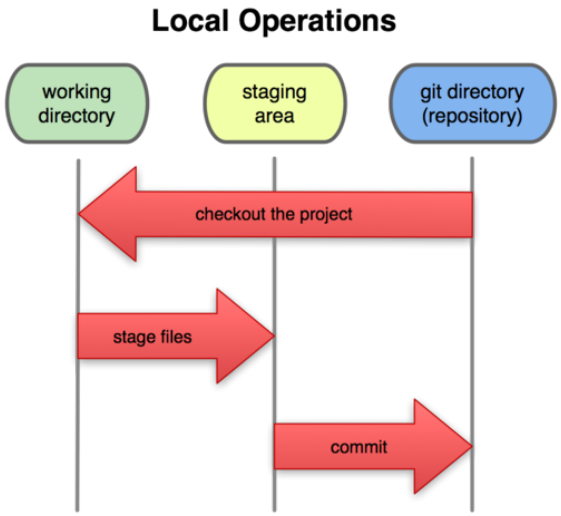
Stap 8 Bekijk nu op Github het resultaat van je eerste push! Dit zou zichtbaar moeten zijn onder de volgende url: `https://github.com/<username>/les3` met voor `<username>` jouw username ingevuld.

Stap 9 Klik op het tabje "Commits". Daar vind je een SHA-key van jouw commit. Er zijn vier soorten objecten in `git`: blobs (files), trees (directories), commits en tags. Al deze vier soorten objecten worden geïdentificeerd aan de hand van een unieke SHA1-hash[2]. Lees nu het gedeelte over het object-model van `git`: `http://book.git-scm.com/1_the_git_object_model.html`.

Stap 10 Kopieer de SHA1-hash (vanaf nu kortweg sha genoemd) van je eerste commit en typ in bash: `git show <sha>`, waarbij je je eigen sha invult, bijvoorbeeld: `git show d3d01ef8fbaac87f9a3cedfc2839a132`

[2]Voor meer info over SHA1 zie `http://en.wikipedia.org/wiki/SHA-1`

Figure 2: Git: drie niveaus. (Bron)



`Git` laat nu informatie zien over je eerste commit.

Stap 11 Lees nu het gedeelte over de git-index. `Git` maakt onderscheid tussen de "working directory", de "index" of "staging area" en de "repository" (zie nogmaals 2!). Wijzigingen in de working directory worden niet automatisch meegenomen in een commit. Je moet ze eerst toevoegen aan de index. Met behulp van het commando `git status` kan je zien wat er in de index staat en dus klaar staat om gecommit te worden.

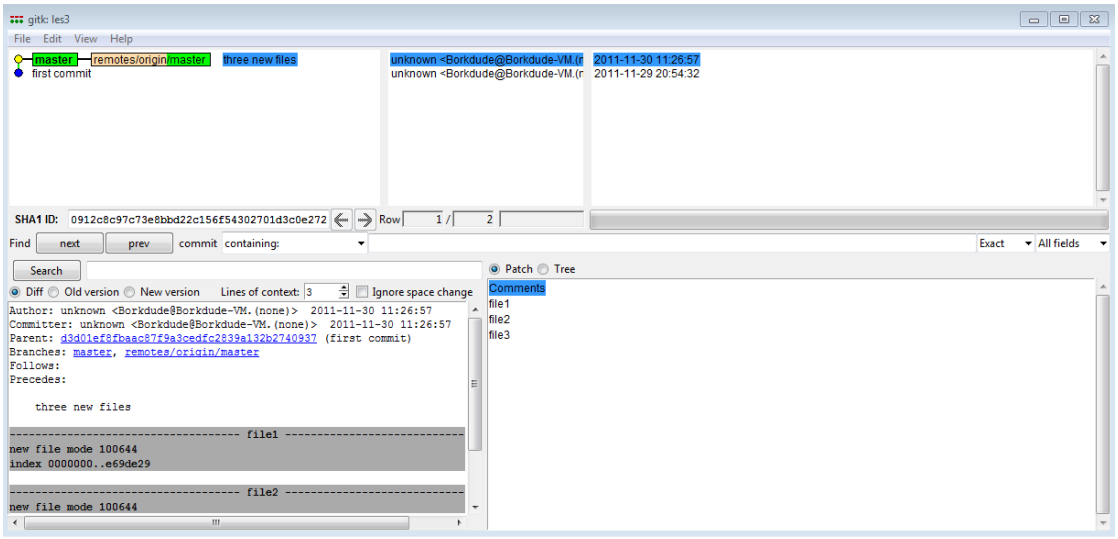
Stap 12 Maak drie nieuwe files aan door `touch file1 file2 file3` te typen. Maak wat wijzigingen in deze files met bijvoorbeeld Notepad. Typ vervolgens `git status`. Git komt nu met de melding: `nothing added to commit but untracked files present`. Oftewel: bij een volgende commit zal er niets worden ingecheckt in de repository, maar er staan wel wijzigingen in de working directory. Je zal de wijzigingen (nieuwe files zijn ook wijzigingen) eerst moeten toevoegen aan de index.

Stap 13 Typ `git add file1 file2 file3`. Probeer nu nogmaals `git status`. Git komt nu met een beschrijving van wat er toegevoegd is aan de index: drie nieuwe files. Ook vertelt git ons hoe we de index kunnen terugzetten naar `HEAD`. `HEAD` is een afkorting voor de sha die correspondeert met onze laatste commit. Dit kan je ook zien aan de hand van het volgende commando wat de sha laat zien van `HEAD`: `git rev-parse HEAD`. Dit levert de sha op van onze laatste commit.

Stap 14 Typ `git reset HEAD`. We draaien de index terug naar het moment van onze laatste commit. Bestaan file1, file2 en file3 nog en is hun inhoud nog intact? Leg uit hoe dit komt.

Stap 15 We gaan nu weer opnieuw de files toevoegen aan de index, om ze vervolgens te committen. Dit zal dan onze tweede commit worden. Je kan alle nieuwe files en wijzigingen in die files toevoegen door `git add .` te typen. Typ daarna `git status` om te controleren wat er zich in de staging area bevindt.
Stap 16 Typ nu `git commit -m 'drie nieuwe files toevoegd'`. Met `-m` geef je een commit

Figure 3: Gitk geeft een grafische weergave van de history



message mee. Als je dit niet doet, opent er een editor waarin je de commit message moet intypen. Standaard is dat de editor VIM. Als je een andere editor wilt gebruiken moet je dit instellen. Bijvoorbeeld voor Notepad: `git config --global core.editor notepad`. Typ nogmaals `git rev-parse HEAD`. Zie je dat de sha van HEAD nu gewijzigd is? Deze sha is de identifier van onze laatste commit. Je zou nu een push naar Github kunnen doen. Weet je nog hoe dit moest?

Stap 17 Grafische weergave. Typ nu in bash `gitk`. Je ziet nu een grafische weergave van de history. Ook kun je de sha's van de beide commits terugvinden. Als je een push hebt gedaan naar Github zie je dat `master` gelijk is aan `remotes/origin/master`. Bekijk wat er verder allemaal te zien is in `gitk` en sluit de applicatie.

Stap 18 Ga nu verder vanaf de volgende webpagina. Het vormt een herhaling van wat we hierboven al gedaan hebben, maar het kan geen kwaad om het nog eens te doen en het nog eens op je in te laten werken. http://book.git-scm.com/3_normal_workflow.html

Stap 19 Branching. Een van de bijzonderheden van `git` is dat je kosteloos lokaal een branch kan maken van je repository, waarin je even kan experimenteren. Mocht het experiment op niks uitlopen, dan verwijderen je gewoon de branch. Mocht er iets succesvols uitkomen, dan kun je jouw experiment met de master-branch mergen. De volgende pagina gaat daar over. Lees de pagina en volg de instructies: http://book.git-scm.com/3_basic_branching_and_merging.html. Je vraag je misschien af waar `git reset --hard` voor staat. Lees dan deze pagina nog door: http://progit.org/2011/07/11/reset.html.

Stap 20 We gaan nu met meerdere mensen aan hetzelfde project werken, zoals je ook gewend bent in een themaopdracht. Je moet eerst iemand toegang geven in Github als collaborator. Ga naar je repository, selecteer "Admin" en vervolgens "Collaborators". Kies dan de naam van je Github-partner en klik op Add. Jouw Github-partner krijgt nu een bericht in zijn inbox dat hij

push-rechten heeft tot jouw repository.
Stap 21 We werken nu verder op het systeem van je Github-partner. Als je alleen werkt kun je dit faken door het project zometeen in een andere directory te clonen. Wel moet je inloggen met je tweede account bij Github om te simuleren alsof je met zijn tweeën werkt. De voorkeur heeft het zeker om een ander systeem dan waarmee we zojuist hebben gewerkt te gebruiken, om het samenwerken aan een themaopdracht beter te kunnen simuleren. Als je nog geen `git` hebt geïnstalleerd op het andere systeem moet je dat eerst doen.
Log in met het account van je Github-partner bij Github en bezoek dan de repository die zojuist is aangemaakt. Je bezoekt dus een repository van een ander account dan waarmee je nu bent ingelogd, maar die jou wel push-rechten heeft gegeven. Kopieer de git-link. Deze ziet er ongeveer zo uit: `git@github.com:<naam>/les3.git`. Open nu bash en `cd` naar een directory waar je de repo kan neerzetten. Vervolgens gaan we in deze directory de repository clonen: `git clone` gevolgd door de gekopieerde git-link.

Stap 22 Nu gaan we wat wijzigen verrichten aan onze lokale repository. Open file1, maak de file leeg en typ op de eerste regel: `tekst systeem partner`. Sla de file op en typ vervolgens `git add file1` gevolgd door `git commit -m 'een regel tekst toegevoegd'`.

Stap 23 Typ nu: `git log`. Je ziet nu informatie over de drie commits. Onze lokale repo loopt één commit voor op origin/master, de repo op Github.

Stap 24 Haal de sha van de commit uit de logs van voor de wijziging aan file1. En typ dan: `git reset <sha>`. Je krijgt nu van `git` de melding dat er unstaged changes zijn na de reset. Dit betekent dat file1 nog steeds de wijzigingen bevat van zojuist, maar dat deze wijziging niet niet gestaged is, d.w.z. nog niet is toevoegd aan de index. Je kunt dus een reset doen, zonder de files in je working directory te wijzigen.

Stap 25 Om de wijziging écht terug te draaien zullen we de optie `--hard` moeten meegeven aan `git reset <sha>`. Probeer het dus nogmaals, maar nu met `git reset <sha> --hard`.

Stap 26 Bekijk de inhoud van file1. Je kunt nu zien dat de wijziging is teruggedraaid. Wees dus voorzichtig wanneer je `--hard` gebruikt, je files in de working directory kunnen hierdoor wijzigen. Zorg er wederom voor dat file1 leegt wordt en voeg daarna op de eerste regel weer `tekst systeem partner` toe. Doe vervolgens `git commit -a -m 'wijziging in file1'`. De optie `-a` geeft aan dat je gewoon alle wijzigingen in bestaande files (niet van nieuwe files) wil stagen en committen.

Stap 27 Doe een `git status`. Je krijgt nu de melding dat onze lokale repository één commit voorloopt op origin/master. Nu gaan we onze wijziging doorsluizen naar Github. Dit kan met `git push origin master`. Kijk op Github of de wijziging is doorgekomen.

Stap 28 Stap nu weer over naar het andere systeem waarmee we begonnen. Maak ook hier file1 leeg en voeg op de eerste regel de tekst `tekst mijn eigen systeem` toe, en add+commit deze wijziging. Probeer dan een `git push`. Als het goed is weigert `git`deze wijziging. We moeten namelijk van `git`eerst een pull doen (volgende stap). Dit is het omgekeerde van push: we gaan wijzigingen van de Github-repo (origin/master) naar onze eigen repo halen.

Stap 29 Doe `git pull`. `Git` haalt nu de wijzigingen vanaf Github naar onze lokale repo. Omdat er nu een conflict is op de eerste regel van file1, zullen we handmatig moeten bepalen wat we

hiermee doen. Toevoegingen of verwijderingen kan `Git` automatisch oplossen, maar wijzigingen op dezelfde regel niet. De inhoud van file1 zal er nu ongeveer zo uit zien:

```
<<<<<<< HEAD
tekst mijn eigen systeem
=======
tekst systeem partner
>>>>>>> e70c12d8fc8e7f0eae9cf43405ea32741f7bbd15
```

De tekst tussen `<<<<<<< HEAD` en `=======` is de tekst van revisie HEAD, de huidige revisie dus. De tekst tussen `=======` en `>>>>>>> e70c12d8fc8e7f0eae9cf43405ea32741f7bbd15` (waarbij de sha heel waarschijnlijk zal verschillen van de jouwe!) is afkomstig van jouw Github-partner. We moeten nu kiezen wat we gaan doen. Als we besluiten om beide regels te houden, dan moeten we de rest van de tekst die het conflict aangeeft verwijderen en opnieuw een add+commit doen. De tekst in file1 wordt nu dus:

```
tekst mijn eigen systeem
tekst systeem partner
```

Doe vervolgens weer een add+commit.

Stap 30 Push de wijziging weer naar Github en kijk of het goed is aangekomen. Bekijk ook de history van de file. Je kunt de wijzigingen per commit terugkijken.

Stap 31 Ga vervolgens weer naar het systeem van de Github-partner. Doe daar een `git pull`. Vervolgens zijn beide collaborators up to date. Doe een `gitk` om een grafische representatie te bekijken van de commits. Maak daarvan een screenshot en plak deze in je portfolio met een toelichting erbij.

Stap 32 Op een bepaald moment gedurende de ontwikkeling van een project wil je soms een (herkenbare) versie vastleggen. Niet alleen voor ontwikkelaars zelf is het handig om over versie 1.2 of 0.5-beta te praten, maar ook voor de buitenwereld is het handig om te weten welke versie van een project bruikbaar is in een bepaalde context. Daarvoor kun je in `git` tags gebruiken. Tags corresponderen met een bepaalde commit-sha van het project. Stel dat we nu bepaald hebben dat ons project les3 klaar is om als versie 0.1 de wereld in te gaan, dan kunnen we dat aan git vertellen als volgt: `git tag -a v0.1 -m 'version 0.1'`. Om de tag zichtbaar te krijgen in Github moet je bij push nog een extra optie meegeven: `git push --tags`. Probeer dit uit. Kijk nu onder het tabje tags van het project op Github. Daar zie je als het goed is de tag. Ook is er bij elke tag de optie om die versie van het project te downloaden als zip: handig voor mensen die wel jouw code willen hebben, maar even geen `git` (willen) gebruiken. Voor meer info over tagging zie http://learn.github.com/p/tagging.html.
We weten nu (net) genoeg om via `git` en Github samen te kunnen werken aan een project Nog wat laatste tips:

- Met `git add .` voeg je alle wijzigingen, inclusief nieuwe files in één keer toe aan de index
- Met `git commit -a` voeg je alle wijzigingen toe aan de index én doe je een commit. Pas op: nieuwe files worden hierbij niet meegenomen. In dat geval moet je eerst `git add .` doen.

- Het is zeer aan te bevelen de documentatie over tags, .gitignore, rebasing, etc. een andere keer eens rustig door te nemen. Ga in dat geval verder vanaf de webpagina http://book.git-scm.com/3_git_tag.html.
- Als je geen contributor bent of mag worden van een bepaald project op Github, kun je altijd het project forken. Er ontstaat dan een kopie van het andere project onder jouw Github-account en daar kun je dan je eigen ding in doen. Mocht je iets heel cools hebben gemaakt, of bepaalde bugs hebben opgelosts, dan kan je een pull request indienen bij het oorspronkelijke project. Zij ontvangen dan een bericht waarin ze de keuze hebben om wel of niet jouw wijzigingen te accepteren.
- Voor integratie met `git` en Eclipse is er een Eclipse-plugin beschikbaar, genaamd `EGit`. Zie http://eclipse.org/egit/.

1.0.1 Links

http://www.github.com
http://book.git-scm.com/
http://progit.org/book/
http://gitref.org/
http://eclipse.org/egit/