Versiebeheer met Git

Dieter Depuydt

Augustus 2021

Woord vooraf

Deze cursus over versiebeheer met Git heb ik in de eerste plaats geschreven voor mijn leerlingen in de Burgerschool te Roeselare. Iedereen mag de cursus gratis gebruiken. Ik apprecieer het als je mij laat weten dat je met mijn cursus werkt.

Versiebeheer met Git is technisch gezien (de juiste commando's intikken) eenvoudig maar voor wie nooit eerder met gedistribueerd versiebeheer heeft gewerkt zijn de concepten wel complex. Het is dan ook een cursus die minder geschikt is voor zelfstudie.

Het hoofddoel van deze cursus is het concept van gedistribueerd versiebeheer ten volle te begrijpen. Daarom is ruim 90% van deze cursus opgebouwd rond de GIT-commando's. Hoewel je er alles wat GIT te bieden heeft mee kan doen, zal je in praktijk voor een GUI omgeving kiezen. Het is makkelijker, sneller en intuïtiever maar net dat maakt dat je de concepten minder onder de knie krijgt.

In deze cursus werk ik rond twee totaal fictieve projecten. In de voorbeelden werk ik rond software die beveiligde toegang tot een gebouw voorziet. Voor iedereen die toegang heeft tot het gebouw voorzien we een tekstbestand. De bestandsnaam is de naam van de persoon (bv. dieter depuydt.txt), de inhoud bevat de naam van de persoon, zijn code in de vorm van een md5-hash … In de oefeningen maken we een woordenboek waarbij we een aantal tekstbestanden hebben, eentje per letter van het alfabet.

Vragen en opmerkingen kan je mij stellen via mail dieter@depuydt.eu. Van deze en andere cursussen worden regelmatig updates gepubliceerd op mijn website https://meneer.depuydt.eu

Dieter Depuydt

Inhoudsopgave

1 Git? ... 3
1.1 Versiebeheer ... 3
1.2 Gedistribueerd ... 3

2 Git installeren ... 4
2.1 Git downloaden ... 4
2.2 Git installeren ... 4
2.3 Git configureren ... 8
2.4 Oefeningen ... 8

3 Starten met Git-Cmd ... 9
3.1 Command-line vs GUI ... 9
3.2 Git terminologie ... 9
3.3 Een Git project starten ... 10
3.4 Git life-cycle ... 11
3.5 Meer commando's ... 15
3.6 Een stapje terug ... 17

4 Branches ... 20
4.1 Wat zijn branches? ... 20
4.2 Een nieuwe branch aanmaken ... 20
4.3 Switchen tussen branches ... 20
4.4 Een branch stashen ... 21
4.5 Twee branches samenvoegen ... 23
4.6 Een branch verwijderen ... 26
4.7 Een branch maken van een (oude) commit ... 26
4.8 De tak verlaten (HEAD detached) ... 27
4.9 De tak een stukje afzagen (reset) ... 28
4.10 Oefeningen ... 28

5 Remotes ... 30
5.1 GitHub ... 30
5.2 Een Git project aanmaken bij GitHub ... 31
5.3 Remote toevoegen aan een project ... 32
5.4 Versies naar de remote kopiëren ... 32
5.5 Oefeningen ... 33
5.6 Versies naar je cliënt kopiëren ... 33
5.7 Omgaan met conflicten ... 34

6 Alfabetisch overzicht commando's ... 36

1 Git?

Git is een open source gedistribueerd versiebeheersysteem (of in het Engels “Distributed Version Control System” of kortweg DVCS). Het kenmerkt zich onder andere door het feit dat een centrale server geen belangrijke rol krijgt en eigenlijk zelfs totaal niet nodig is. Linus Torvalds (die van de Linux kernel) kwam er mee op de proppen in 2005. Sindsdien won het snel aan populariteit waardoor het vandaag met ruime voorsprong het meest gebruikte versiebeheersysteem op de markt is.

1.1 Versiebeheer

Het woord is duidelijk, we gaan verschillende VERSIES van iets BEHEREN. Versiebeheer wordt met name veel toegepast in de wereld van programmeren, ontwikkelen … omdat het in zo’n omgeving belangrijk is om verschillende versies van een project te kunnen bijhouden en inkijken. Maar het kan ook perfect voor andere doelen gebruikt worden. Wanneer je een eindwerk schrijft kan het ook interessant zijn om vorige versies te kunnen inkijken en dan kan je ook perfect gebruik maken van versiebeheer.

Je kan aan versiebeheer doen om verschillende versies te bekijken die je zelf hebt gemaakt maar je kan ook aan versiebeheer doen om met meerdere mensen samen aan een project te werken. Elke deelnemer aan het project kan dan aanpassingen doen aan het project en die publiceren in “versies”. Wil je later weten wie wat gedaan heeft in het project dan kan je dat terugvinden in de versiegeschiedenis.

1.2 Gedistribueerd

Bij gedistribueerd versiebeheer beschikt elke deelnemer aan het project over alle versies van alle bestanden in het project. Wanneer één deelnemer of zelfs de centrale server verdwijnt dan is er niets verloren. Zolang er minstens één deelnemer over het project beschikt is de volledige versiegeschiedenis beschikbaar.

Van zodra er meer dan één deelnemer is in een project is er nood aan een centrale server. De centrale server zorgt ervoor dat alle versies bij alle deelnemers terecht komen maar speelt verder geen cruciale rol. Het is niet zo dat de server op de een of andere manier over meer informatie beschikt dan de deelnemers.

2 Git installeren

2.1 Git downloaden

Om met Git te kunnen werken moet je het programma eerst installeren. Je kan het gratis downloaden op de website https://git-scm.com/downloads. Er bestaat zowel een 32-bits als een 64-bits versie van. Wie over een 64-bits besturingssysteem beschikt mag zeker de 64-bits versie downloaden en installeren, voor de andere mensen is er ook goed nieuws, de 32-bits versie werkt even goed. De meest recente versie van het programma is versie 2.32.0 (juli 2021).

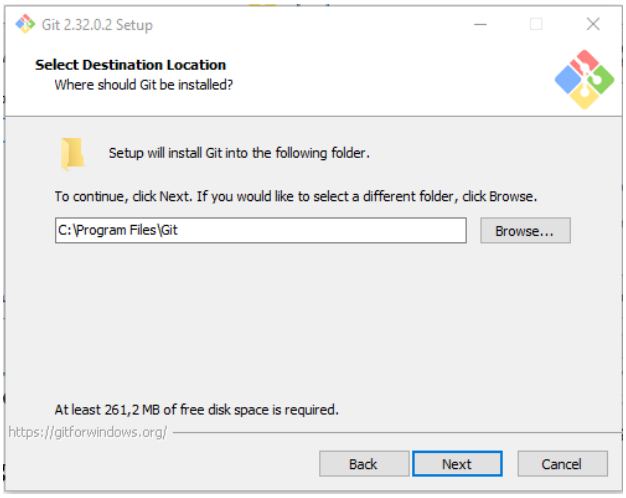
Het installatieprogramma bevat naast Git zelf ook een aantal grafische tools. Naast deze aangeboden grafische tools bestaan er nog tal van andere grafische tools die vaak wat gebruiksvriendelijker zijn al is dat natuurlijk persoonlijk. Je vindt een overzicht van heel wat GUI software op https://git-scm.com/downloads/guis.

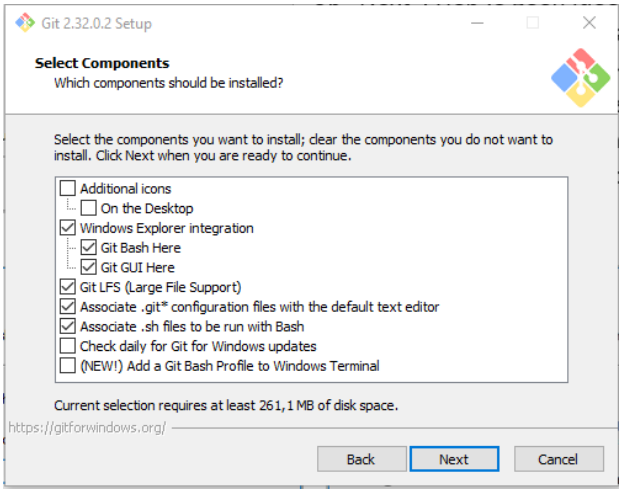
Merk ook op dat er heel wat programmeeromgevingen zijn die standaard ondersteuning bieden voor Git, denken we onder andere aan Visual Studio.

2.2 Git installeren

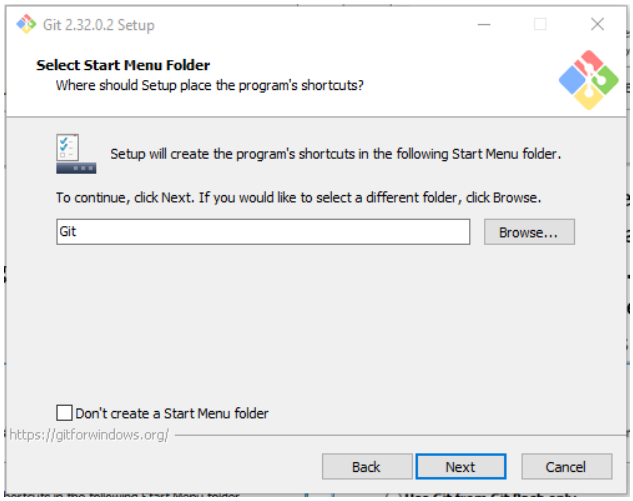
Dubbelklik op de installer (git-2.32.0.2-64-bit.exe) om de installatie te starten, een wizard wijst je de weg doorheen de installatie.

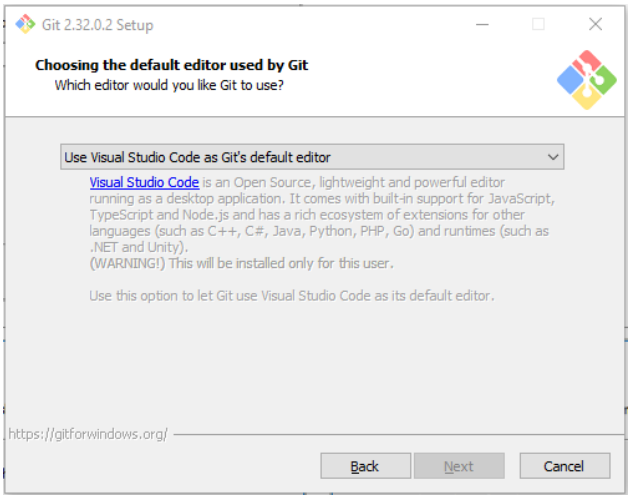
(links) Afhankelijk van je besturingssysteem en de daarbij horende instellingen krijg je een waarschuwing. Ben je wel zeker dat je dit programma wil uitvoeren? Je mag gerust zijn, het is perfect veilig om Git op je computer te installeren. Klik op “Ja”. (rechts) Je krijgt nu de licentietekst te zien waarmee je akkoord moet gaan als je Git wil installeren. Git maakt gebruik van de GNU General Public License en dat wil (ongeveer) zeggen dat je het programma gratis mag gebruiken. Klik op “Next”.





(links) Vervolgens krijg je de vraag waar je Git wil installeren. Klik op “Browse” en kies de plek op je computer waar je Git wil installeren en klik vervolgens op “Next”. Heb je geen idee waar je Git wil installeren dan is de standaard voorgestelde locatie (c:\Program Files\Git) prima, je mag gewoon op “Next” klikken. (rechts) Welke onderdelen van Git wil je installeren in je Windows omgeving. Wil je pictogrammen op je bureaublad? Wil je “.git” bestanden koppelen aan het programma … ? Neem weloverwogen beslissingen en klik op “Next”.



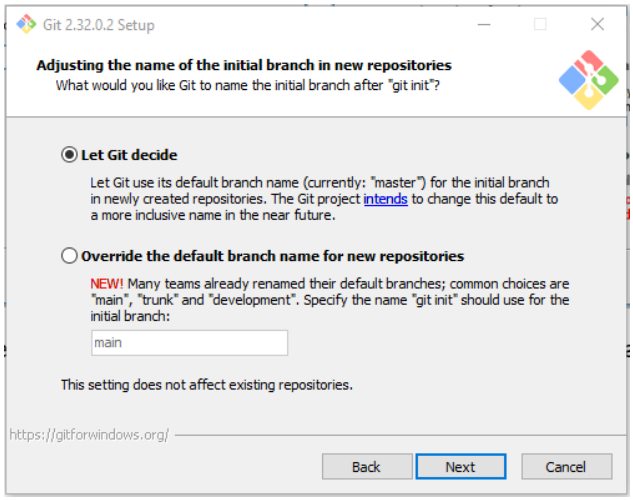


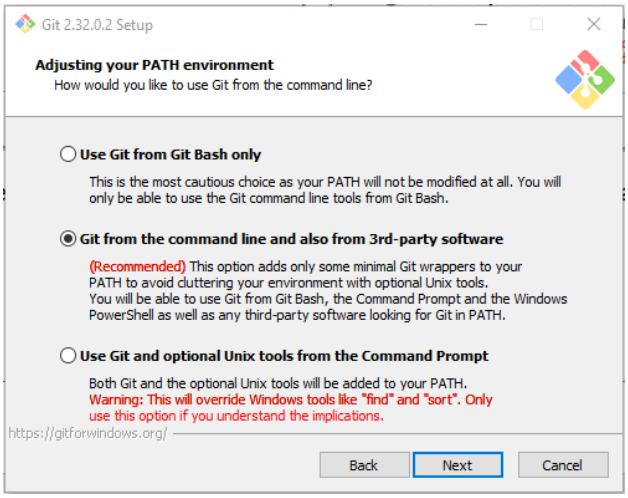
(links) Wil je een map, met de naam “Git” (kan je wijzigen), toevoegen aan het startmenu van Windows? Indien niet dan zet je het aankruisvakje bij “Don’t create a Start Menu folder” aan. Klik op “Next”. (rechts) Elke keer je wijzigingen doorvoert of commit krijg je de kans om daar wat toelichting bij te geven. Zeker bij grotere projecten is het belangrijk om dit voldoende ter harte te nemen. (bv. wel “vanaf deze versie gebruiken we een andere manier om de databank te bevragen” en niet “versie 34”). Deze toelichting kan je intikken in een teksteditor. Hier kan je aangeven welke teksteditor je wil gebruiken, je kan dat later nog makkelijk wijzigen. Kies in eerste instantie best voor “Use the nano editor by default” en klik op “Next”.

Om later de teksteditor te wijzigen naar bv. het klassieke notepad, kan je onderstaand commando gebruiken.

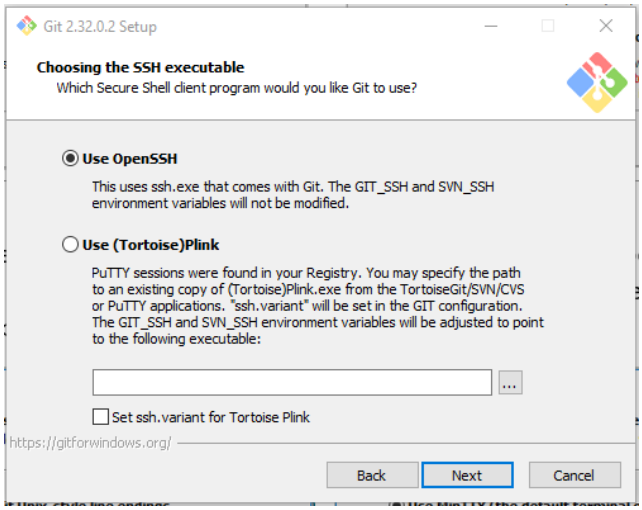
```
git config --global core.editor "c:/windows/notepad.exe"
```

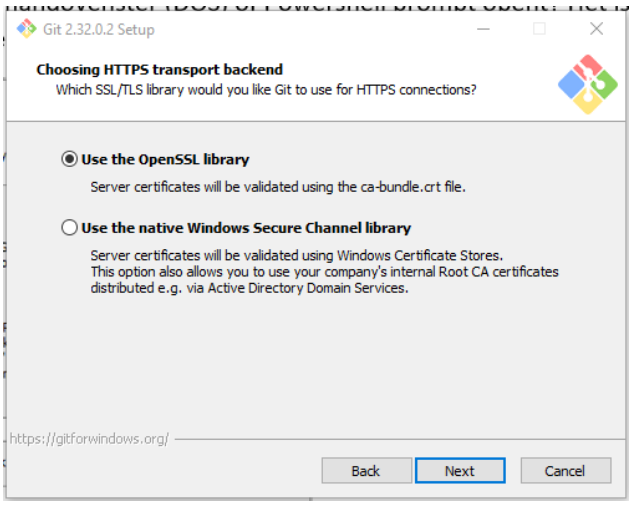
Let op de omgekeerde /, dit is geen tikfout!



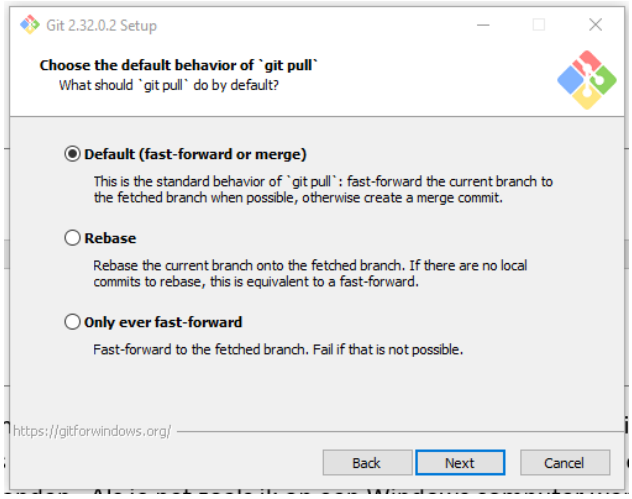


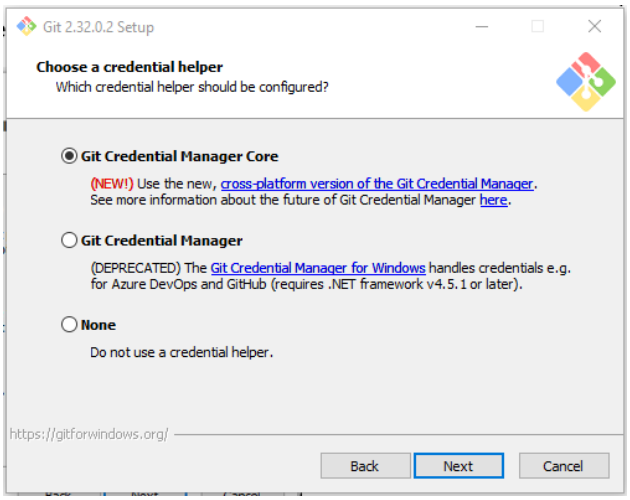
(links) Later in deze cursus zal je zien dat je op een bepaald moment met een GIT-project moet starten. Je moet het project aanmaken. Dat wordt ook wel eens initialiseren genoemd. Bij dat initialiseren moet je de hoofdtak (branch) een naam geven maar als je dat niet doet, wordt er een standaard naam gebruikt. Gewoonlijk is die standaard naam “master” maar als je wil kan je die naam in dit scherm al wijzigen. Maak je keuze en klik op “Next”. (rechts) Wil je Git enkel kunnen gebruiken als je Git Cmd of Git Bash opstart of wil je het ook kunnen gebruiken als je een commandovenster (DOS) of Powershell prompt opent? Het is aan te bevelen het tweede te kiezen. Maak een keuze en klik op ‘Next’.



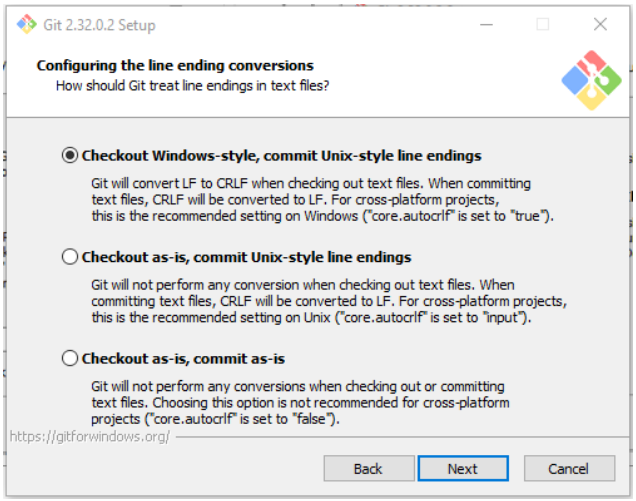


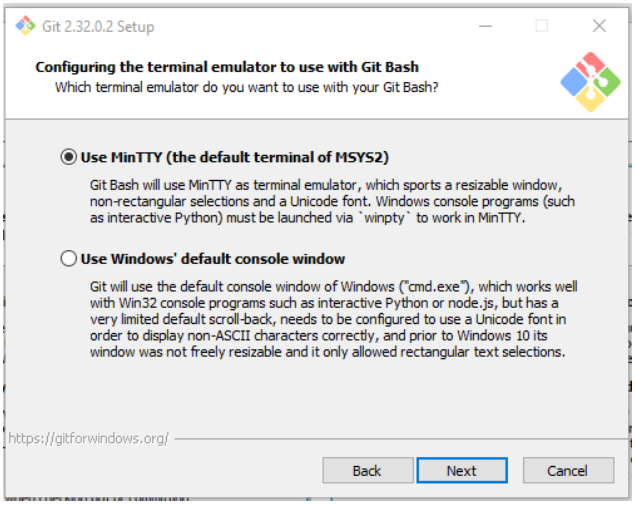
(links) Als je een centrale server wil gebruik om je projecten op te bewaren (bv. GitHub) dan moet je computer op een veilig manier communiceren met die centrale server. De twee meest gebruikte protocollen daarvoor zijn https en ssh. In dit venster kan je bepalen welke tool er gebruikt wordt als ssh-client. Het gratis (en meegeleverde) OpenSSH is een prima keuze. Klik op “Next”.
(rechts) De communicatie tussen je computer en de server wordt geëncrypteerd op basis van certificaten. Er bestaan meerdere tools om met certificaten te werken waaronder het opensource programma OpenSSL én een oplossing van Microsoft genaamd de Certificate Store. Kies bij voorkeur voor OpenSSL en klik op “Next”.



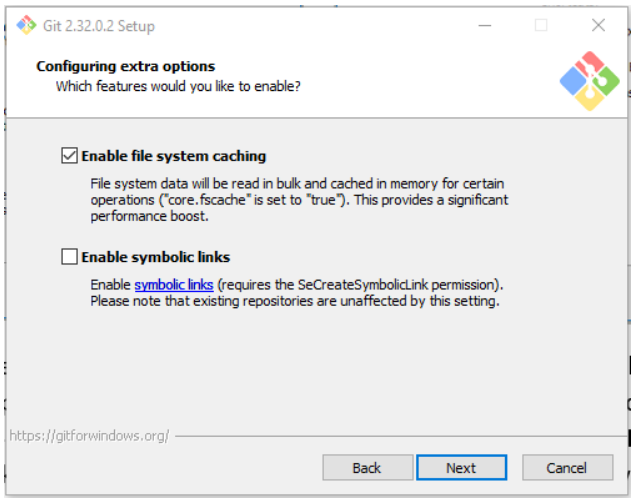


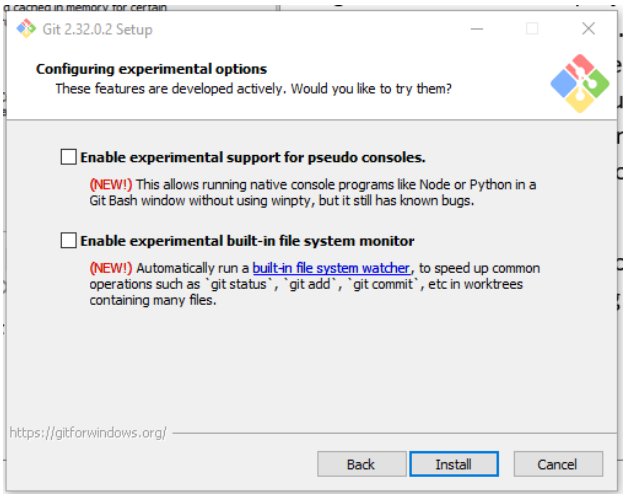
(links) In dit scherm laat je weten hoe er moet omgegaan worden met het commando “git pull”. Dit commando haalt updates van een centrale server. Het verschil tussen fast-forward, merge en rebasen lees je in hoofdstuk 4. For now … “Default” is just fine! Klik op “Next”. (rechts) Als je code online wil publiceren of wil downloaden van een gedeelde server dan moet je natuurlijk aanmelden op die server. Dat kan met een klassieke gebruikersnaam en wachtwoord maar dat is zeer onveilig, daarom is het gebruik van een credential manager sterk aan te bevelen. “Git Credential Manager Core” is de nieuwste tool op dat vlak in Git, de tool kan o.a. overweg met oAuth authenticatie. (zie o.a. hoofdstuk 5). Klik op “Next”.





(links) Wellicht is het je al duidelijk geworden dat Git zijn oorsprong kent in de Linux wereld. Linux en Windows gebruiken verschillende regeleindes en dat is belangrijk zowel bij het intikken van commit meldingen als bij het doorvoeren/committen van tekstbestanden. Als je net zoals ik op een Windows computer werkt kies je best de eerste optie. Klik op “Next”. (rechts) Eén van de meegeleverde tools, Git Bash, simuleert een Linux omgeving. Hier kan je instellen in wat voor venster dit zal draaien. Het is aan te bevelen om de eerste optie te kiezen. In deze cursus maken we geen gebruik van Git Bash. Klik op “Next”.





(links) Tot slot kunnen we "file system caching" aanzetten, dit zou voor aanzienlijke snelheidswinsten zorgen maar in de kleine projectjes die wij gebruiken heeft dit geen enkele invloed. Klik op "Next". (rechts) Tenslotte kan je nog enkele nieuwe, experimentele zaken activeren maar die liggen buiten de scope van deze cursus. Klik op "Install".

Git wordt nu effectief op je computer geïnstalleerd. Enkele ogenblikken later krijg je de melding dat Git geïnstalleerd is. Klik op "Finish".

2.3 Git configureren

Na de installatie is Git klaar voor gebruik. De configuratie is niet verplicht maar zal het je een pak makkelijker maken in de toekomst. We stellen de naam en het e-mailadres van de gebruiker in. Deze zullen vermeld worden bij elke commit. Open een gewone opdrachtprompt of een GIT CMD venster (beschikbaar sinds de installatie van GIT) en gebruik volgende commando's.

```
git config --global user.name "JE NAAM"
```

```
git config --global user.email "JE E-MAIL ADRES"
```

```
git config --global core.editor "c:/windows/notepad.exe"
```

Deze en alle andere configuratie instellingen worden bewaard in een tekstbestandje dat de naam ".gitconfig" krijgt. Dit bestand wordt bewaard in de map "C:\Users\[gebruikersnaam]". Ook in de map "c:\program files\git" kan zo'n configuratiebestand te vinden zijn. Wanneer er op beide locaties een bestand beschikbaar is worden de bestanden gecombineerd. Daar waar ze met elkaar in conflict komen zal het configuratiebestand in "c:\users\[gebruikersnaam]" zwaarder doorwegen.

2.4 Oefeningen

- ✓ Installeer Git zoals werd beschreven in 2.2
- ✓ Configureer Git zoals werd beschreven in 2.3

3 Starten met Git-Cmd

3.1 Command-line vs GUI

Zonet heb je Git geïnstalleerd; je hebt zowel de command-line omgeving als een GUI geïnstalleerd. We beginnen met het aanleren van de command-line. Eens je alle concepten goed snapt is de stap naar de GUI omgeving eenvoudig en vanzelfsprekend. Je hebt zowel het programma "Git Bash" op je computer als "Git Cmd". In deze cursus werk ik ALTIJD met Git Cmd. Alle commando's kunnen ook gewoon aan de opdrachtprompt ingetikt worden.

3.2 Git terminologie

Tenzij je al ervaring hebt met (gedistribueerd) versiebeheer zijn er weinig zaken die je kan vergelijken met versiebeheer. Je kan moeilijk zeggen "het lijkt een beetje op …". De meeste concepten en termen zijn dus nieuw, het vergt wat tijd en inspanning om dit te snappen. Soms is één stukje van deze cursus lezen meer verwarrend dan verhelderend, het is belangrijk om ALLES te lezen zodat alle puzzelstukjes op hun plaats vallen.

3.2.1 Werkmap of working directory

Versiebeheer is steeds van toepassing op één map, eventueel met heel wat submappen. Het is niet mogelijk om versiebeheer toe te passen op vb. "c:\project 1" en "c:\project 2". Je kan uiteraard wel twee verschillende Git projecten opstarten voor deze twee mappen. Wil je deze twee mappen samen in één Git project dan zal je die eerst samen in een map moeten plaatsen bv. "c:\project\project 1" en "c:\project\project2" om vervolgens een Git project te definiëren op niveau van de map "c:\project".

De map waarin alle bestanden komen en waarop een Git project geïnitialiseerd zal worden noemen we de "**working directory**" of "**werkmap**". In bovenstaand voorbeeld is "c:\project" onze werkmap.

3.2.2 Staging area of index

In Git gaan we niet zomaar versies opslaan van bestanden, we gaan een verzameling van bestanden groeperen en die samen toevoegen aan een versie van het versiebeheer.

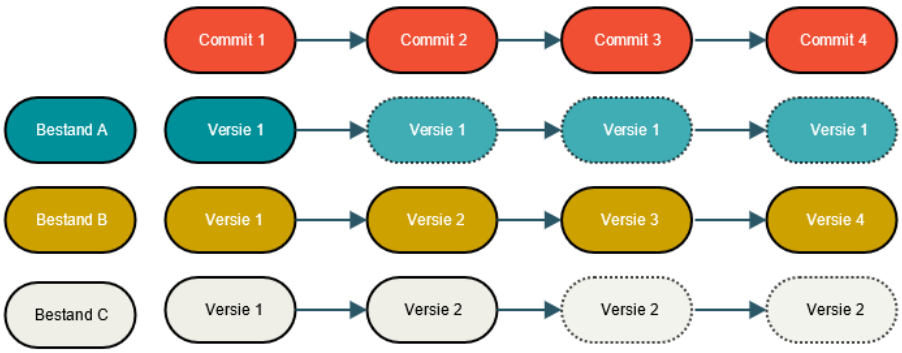
De **staging area**, die ook vaak index wordt genoemd, is de verzamelplaats van alle bestanden die in een volgende commit (versie) zullen toegevoegd worden aan de repository. Je moet die staging area niet zien als iets fysiek. Het is niet dat bestanden in een mapje afzonderlijk worden geplaatst ofzo … het is meer een concept.

Het is de programmeur die beslist wat er in een volgende versie zal zitten. Bestanden die niet gewijzigd zijn EN bestanden die gewijzigd zijn maar niet toegevoegd worden aan de staging area zullen NIET in de volgende commit (of versie) zitten.

3.2.3 Commit (versie)

Op een bepaald moment is de gebruiker tevreden over zijn staging area en wil hij deze verzameling bestanden toevoegen aan de repository. Hij gaat de staging area dan "committen" of "doorvoeren". Er wordt nu een NIEUWE versie van het project toegevoegd aan de repository. Later kunnen we geheel of gedeeltelijk terugkeren naar deze commit / versie.

Opgelet: het is niet omdat een bepaald bestand NIET opgenomen wordt in een versie / commit dat het bestand verdwijnt uit de versiegeschiedenis. Dat vraagt om wat verdere uitleg. Bekijk onderstaande afbeelding alvast …



In bovenstaand voorbeeld hebben we een project met drie bestanden. Bij de eerst "commit" hebben we alle drie de bestanden "gestaged" en daarna onze staging area gecommit.

Bij de tweede commit hebben we enkel bestand B en C gestaged en daarna onze staging area gecommit. Bestand A zit dus niet in de tweede versie van het project. Dit kan verschillende redenen hebben nl. misschien werd bestand A niet gewijzigd en was het dus zinloos om bestand A toe te voegen aan de nieuwe versie OF misschien werd bestand A wel gewijzigd maar wilden we deze wijzigingen nu nog niet in een versie opnemen.

Als we later het hele project willen terugdraaien naar hoe het was op het ogenblik van "commit 2" dan zullen we wel degelijk 3 bestanden krijgen nl. versie 2 van de bestanden B en C maar ook versie 1 van de bestand A.

3.2.4 Repository

Elk Git project heeft één repository. Dit is de verzameling van alle opgeslagen versies (commits) van het project. De volledige repository wordt opgeslagen in een map ".git" die je terugvindt in de map waarin je je project initialiseert.

3.3 Een Git project starten

Er bestaan twee manieren om een nieuwe Git project op je computer te krijgen nl. een "working directory" aanmaken en daar een git project op initialiseren of een Git project downloaden (clonen) van een Git server.

3.3.1 Zelf een Git project aanmaken

- ✓ Maak een map aan die zal dienen als "working directory".
- ✓ Start "Git cmd" en navigeer naar deze map.
- ✓ Tik het commando "**`git status`**" in en kijk welke feedback je krijgt.

Git CMD

```
D:\GitProjectA>git status
fatal: not a git repository (or any of the parent directories): .git
```

- ✓ Tik het commando "**`git init`**" in.

Git CMD

```
D:\GitProjectA>git init
Initialized empty Git repository in D:/GitProjectA/.git/
```

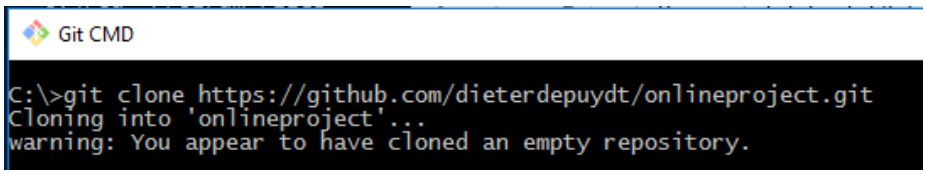
Bemerk dat er een submap ".git" in de werkmap verschijnt. De ".git" map bevatten ALLE mogelijke

informatie van het versiebeheer. Het volstaat deze map te verwijderen om ALLE versiegeschiedenis van een project te verwijderen.

3.3.2 Een Git project downloaden van een server

In een volgend hoofdstuk leg ik je uit hoe je je project kan "uploaden" naar een online server om zo samen met anderen aan het project te werken. Van zodra je met meerdere mensen werkt heb je zo'n centrale plaats nodig. Er moet maar één iemand van het team het Git project aanmaken en online plaatsen. Alle andere deelnemers gaan het project "clonen". Dat wil zeggen het gehele Git project (werkmap en volledige repository) downloaden op hun computer. Dat is wat we hieronder gaan doen.

✓ Tik het commando "**`git clone [git url]`**" in.



Het project wordt nu gedownload en geplaatst in een map met als naam de projectnaam. Als het project bv. "test" noemt en je stond in "e:\project" dan wordt het Git project in "e:\project\test" gedownload. Het project bevat niet alleen de werkmap maar ook de volledige geschiedenis (repository) van het project.

Wil je het project opslaan in een map met een andere naam, dan gebruik je het volgende commando.

✓ Tik het commando "**`git clone [git url] [afwijkende naam]`**" in.

Als je als afwijkende naam "test2" intikt terwijl je in de map "e:\project" staat, dan zal het project gedownload worden in de map "e:\project\test2" ongeacht de naam van het project.

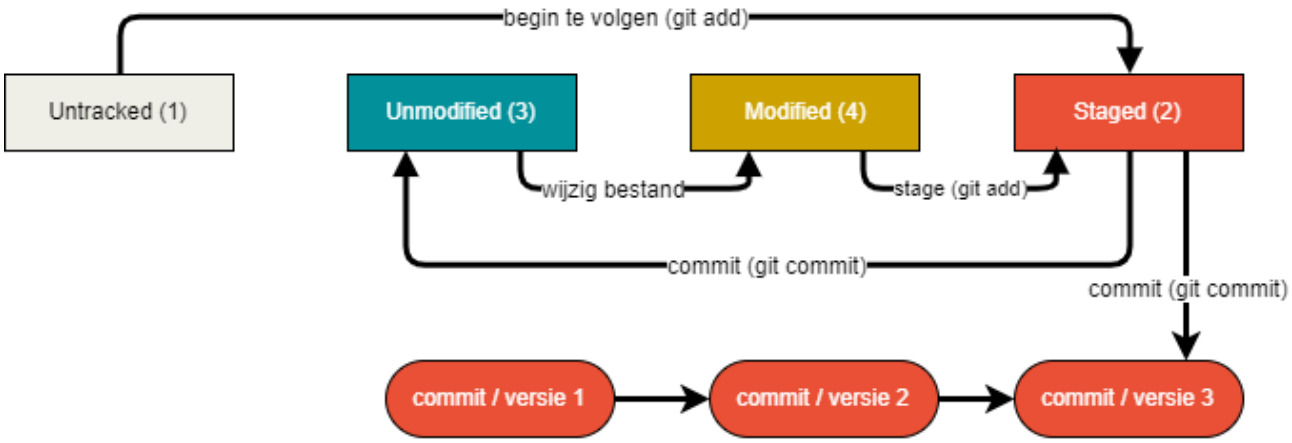
3.3.3 Oefeningen

✓ Start "Git cmd" en navigeer naar de plaats waar het project moet komen.
✓ Maak een map "woordenboek" aan.
✓ Maak van deze map een Git project.
✓ Bekijk de status van het project.

✓ Start "Git cmd" en navigeer naar de plaats waar het project moet komen.
✓ Download mijn Git project vanaf https://github.com/dieterdepuydt/onlineproject.git.

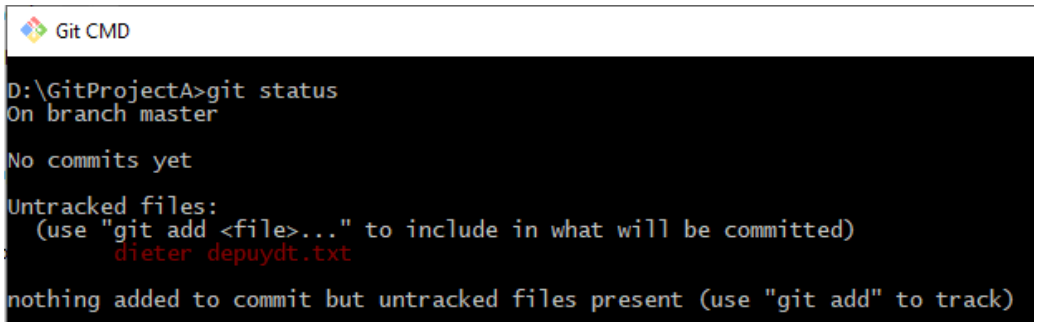
3.4 Git life-cycle

Elk bestand kan binnen Git vier verschillende toestanden hebben nl. untracked, unmodified, modified en staged. Onderstaande afbeelding geeft een mooi overzicht van de life-cycle binnen Git.



3.4.1 Untracked

Als je een nieuw bestand aanmaakt in een Git werkmap dan is dat bestand standaard "untracked". Dat wil zeggen dat het wel in een Git werkmap staat maar Git trekt er zich absoluut niets van aan: er worden geen versies van bijgehouden, er wordt zelfs niet bijgehouden of dit bestand gewijzigd wordt. Bij het bekijken van de status van een project staan deze bestanden in de rubriek "Untracked files".



Als een bestand niet "Untracked" is, dan is het "Tracked" en dan kan het één van de drie andere toestanden (unmodified, modified, staged) aannemen. Om een bestand "te volgen" (te tracken) ga je als volgt te werk:

✓ Tik het commando "**`git add [bestandsnaam]`**" in.

Het bestand komt nu onmiddellijk in de toestand "Staged" waarover zo dadelijk meer. Bemerk trouwens dat je niet noodzakelijk bestand per bestand hoeft te volgen (tracken), je kan perfect gebruik maken van wildcards zoals:

✓ **`git add *.txt`**
✓ **`git add adres*.*`**

Verder is het zo dat wanneer je een folder/map volgt, je automatisch ALLES wat in die map zit mee volgt. *Om het moeilijk te maken … voeg je later bestanden toe tot die map dan worden die nieuwe bestanden standaard niet gevolgd. Je moet die nieuwe bestanden dus ook opnieuw gaan toevoegen.*

3.4.2 Staged

Wanneer je een bestand begint te volgen (tracken) dan komt het in de toestand "Staged" terecht. De toestand "staged" is voor bestanden die gewijzigd zijn sinds ze laatst werden toegevoegd aan een versie (zie commit) van het project EN die je wil opslaan in de volgende versie van het project. (Het is perfect normaal dat je deze zin drie keer moet lezen voor je die snapt ☺).

Bij het bekijken van de status van een project staan deze bestanden in de rubriek “Changes to be committed”.

```
Git CMD

D:\GitProjectA>git add "dieter depuydt.txt"

D:\GitProjectA>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   dieter depuydt.txt
```

De toestand “staged” kan je dus vergelijken met een soort van “wachtkamer” voor bestanden die binnenkort zullen opgeslagen worden in een versie van het project. De wachtkamer blijft al bij al maar de wachtkamer, het is belangrijk om die regelmatig om te zetten in een versie. Dit kan als volgt:

✓ Tik het commando **`git commit -m “[boodschap]”`** in.

Een versie bevat minstens één bestand maar kan ook zeer veel bestanden bevatten. Wanneer de wachtkamer (staging area) leeg is zal een commit niet doorgevoerd worden, dat heeft immers geen zin.

Je kan het commando ook beperken tot “git commit”, er zal een teksteditor geopend worden waarin reeds een commit boodschap wordt voorgesteld. Je kan die boodschap nog aanpassen voor je effectief de commit doorvoert. Het intikken van duidelijke commit boodschappen is even belangrijk als je programmacode goed documenteren.

3.4.3 Unmodified

Als je een commit uitvoert dan worden alle bestanden die de toestand “staged” hebben opgeslagen in een versie van het project én ze krijgen de toestand “unmodified”. (Ze zijn nl. ongewijzigd sinds de laatste commit/versie).

Bij het bekijken van de status van een project worden deze bestanden standaard niet weergegeven.

```
Git CMD

D:\GitProjectA>git commit -m "Eerste versie van het project"
[master (root-commit) caa866c] Eerste versie van het project
 1 file changed, 1 insertion(+)
 create mode 100644 dieter depuydt.txt

D:\GitProjectA>git status
On branch master
nothing to commit, working tree clean
```

Deze bestanden zijn NIET gewijzigd sinds we voor het laatst een “versie” van het project hebben opgeslagen. Je kan bestanden die “Unmodified” zijn niet toevoegen aan een volgende versie van het project, dat zou ook volledig zinloos zijn.

3.4.4 Modified

Van zodra je een bestand gaat wijzigen komt het in de toestand “Modified”. Bij het bekijken van de status van een project staan deze bestanden in de rubriek “Changes not staged for commit”.

Een bestand dat de toestand “Modified” heeft wordt niet zomaar opgenomen in de volgende versie van het project (commit) daarvoor moet je het verplaatsen naar de toestand “Staged”. Dat kan als volgt:

✓ Tik het commando "**`git add [bestandsnaam]`**" in.

Inderdaad met hetzelfde commando als waarmee we een untracked bestand beginnen te volgen. Uiteraard kan je ook hier gebruik maken van wildcards.

3.4.5 Modified & Staged

Een bestand kan tegelijk de toestand "Staged" en de toestand "Modified" hebben. Op het moment dat je een bestand naar de toestand "Staged" brengt wordt er een "kopie" gemaakt van het bestand op dat moment. Het is die kopie die zal toegevoegd worden aan de volgende versie. Als je na het stagen (maar voor het committen) het bestand wijzigt dan heb je opnieuw een gewijzigd bestand (tov de staging area).

Git CMD

```
D:\GitProjectA>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   bestanden/bart devos.txt
        new file:   bestanden/dieter depuydt.txt
        new file:   bestanden/herlinde peeters.txt
        new file:   bestanden/lieselot verschuere.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   bestanden/dieter depuydt.txt
```

Je kan nu twee zaken doen: Ten eerste kan je kiezen om te committen. Dan zal het bestand aan een nieuwe versie worden toegevoegd in de toestand zoals op het ogenblik dat je het hebt gestaged. Hierna zal het bestand enkel nog de toestand "Modified" hebben (en niet meer Staged) want het is gewijzigd ten opzichte van de laatste keer dat het in een versie werd opgenomen. Je kan het bestand nu onmiddellijk opnieuw stagen en een nieuwe commit uitvoeren om ook deze versie op te slaan.

Je hebt ook nog een tweede mogelijkheid nl. het bestand EERST nog eens stagen en dan pas een commit uitvoeren. Als je dat doet dan ga je een nieuwe kopie van het bestand toevoegen aan de staging area. Daarbij gaat de vorige versie van het bestand in de staging area verloren, je kan die nooit meer terughalen.

3.4.6 Genegeerde bestanden

Sommige bestanden wil je NIET in je project opnemen. Je zal ze nooit gaan volgen, laat staan stagen of committen en je wil daar liefst ook niet constant aan herinnerd worden (bv. in een git status). Dit soort bestanden kan je laten negeren door gebruik te maken van een .gitignore bestand.

✓ Maak een bestand aan in de werkmap met als naam ".gitignore"
✓ Plaats er de namen van bestanden in die je wil negeren (één bestand per regel)
✓ Maak eventueel gebruik van wildcards bv.
  - *.exe (alle bestanden met de extensie EXE negeren)
  - dossier* (alle bestanden die beginnen met "dossier" negeren)
  - …

Opgelet: bestanden die reeds door Git worden gevolgd ondervinden hier geen hinder van. De bestanden blijven opgevolgd worden door Git.

3.4.7 Oefeningen

✓ Start "Git cmd" en navigeer naar het woordenboek project uit de vorige oefening.

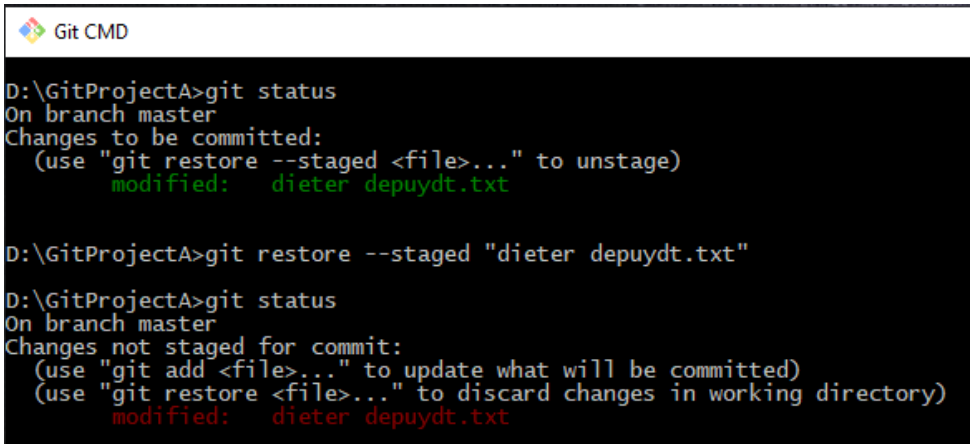
- ✓ Maak een bestand “a.txt” aan.
- ✓ Begin het bestand te volgen.
- ✓ Bewaar een eerste versie (commit) met de mededeling “Letter A toegevoegd”.
- ✓ Maak de bestanden “b.txt”, “c.txt” en “d.txt” aan en volg ze allemaal (gebruik wildcards).
- ✓ Bekijk de status van het project.
- ✓ Bewaar een tweede versie (commit) met de mededeling “Letters B, C en D toegevoegd”.
- ✓ Plaats in het bestand “a.txt” de woorden “aanbakken”, “aanbelanden”, “aanbellen”, “aanbevelen”, “aanbieden”, “aanbinden”, “aanblikken”, “aandrijven” en “aanduwwen” onder elkaar.
- ✓ Bekijk de status van het project.
- ✓ Stage het gewijzigde bestand.
- ✓ Bewaar een nieuwe versie (3) van het project met de mededeling “Letter A aangevuld”.
- ✓ Corrigeer in het bestand “aanduwwen” => “aanduwen”.
- ✓ Stage het gewijzigde bestand.
- ✓ Bewaar een nieuwe versie (4) van het project met de mededeling “Letter A gecorrigeerd”.

3.5 Meer commando’s

3.5.1 Een bestand uit de staging area halen

Als je een bestand hebt verplaatst naar de toestand “staged” dan zal het bij de volgende commit opgenomen worden in een versie van het project. Als je je om één of andere reden bedenkt kan je een bestand terug uit de staging area halen zodat je een commit kan uitvoeren ZONDER dit bestand. Je kan dit als volgt doen:

- ✓ Tik het commando “**`git restore --staged [bestandsnaam]`**” in.



Het bestand zal niet langer de toestand “staged” maar “modified” hebben. Dit heeft verder geen enkele invloed op het bestand. De wijzigingen worden niet ongedaan gemaakt bijvoorbeeld en als het bestand al in vorige versies van het project is opgenomen dan zal daar niet aan geraakt worden.

3.5.2 Een bestand niet langer volgen

Wanneer een bestand in de toestand “Staged”, “Modified” of “Unmodified” is dan wordt het door Git gevolgd. Je kan stoppen met een bestand te volgen door het terug in de toestand “Untracked” te brengen. Dit gaat als volgt:

- ✓ Tik het commando “**`git rm --cached [bestandsnaam]`**” in.

```
Git CMD

D:\GitProjectA>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   dieter depuydt.txt

no changes added to commit (use "git add" and/or "git commit -a")

D:\GitProjectA>git rm --cached "dieter depuydt.txt"
rm 'dieter depuydt.txt'

D:\GitProjectA>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    dieter depuydt.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        dieter depuydt.txt
```

Git zal nu stoppen met dit bestand te volgen maar verder heeft dit geen gevolgen. Het bestand zal niet gewijzigd worden en als het al werd opgenomen in vorige versies van het project dan zal daar niet aan geraakt worden. In de versie geschiedenis zal je zien dat het bestand op een bepaald moment uit het project werd verwijderd.

3.5.3 De versiegeschiedenis raadplegen

Naarmate je langer aan een project werkt zal je steeds meer versies hebben, steeds meer commits uitvoeren. Een overzicht van alle versies kan je opvragen als volgt:

✓ Tik het commando "**`git log`**" in.

```
Git CMD

D:\GitProjectA>git log
commit 18c5bc8a30e5b1b1998edea6cc26463947abe9dc (HEAD -> master)
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Mon Jul 26 12:48:00 2021 +0200

    Versie 1
```

Van elke uitgevoerde commit krijg je nu de commit omschrijving, datum en tijdstip van de commit, naam en e-mailadres van de persoon die de commit heeft uitgevoerd en een uniek ID van de commit. Het unieke ID is belangrijk omdat je het voor tal van zaken (zie verder) zal kunnen gebruiken.

3.5.4 Een commit omschrijving wijzigen.

In het heetst van de strijd laat je je wel een verleiden om weinig zeggende commit omschrijvingen te gebruiken, en daar heb je later spijt van. Gelukkig kan je die nog makkelijk wijzigen.

✓ Tik het commando "**`git commit --amend`**" in.

Een teksteditor wordt geopend met daarin de laatste commit boodschap, je kan de boodschap naar wens aanpassen. Als je de teksteditor sluit wordt de commit boodschap aangepast.

3.5.5 Oefeningen

✓ Start "Git cmd" en navigeer naar het woordenboek project uit de vorige oefening.
✓ Maak een bestand "e.txt" aan.
✓ Plaats in het bestand "e.txt" de woorden "echt", "echtpaar", "eclair", "ecocheque", "ecouteur" onder elkaar.
✓ Begin het bestand te volgen.

- ✓ Bewaar een nieuwe versie (5) van het project met de mededeling “Letter E”.
- ✓ Voeg de volgende woorden toe aan het bestand “e.txt”: “egaal”, “eruit”, “erpel”, “estaminet”.
- ✓ Voeg de volgende woorden toe aan het bestand “d.txt”: “dekenij”, “degusteren”, “dispatcher”.
- ✓ Stage beide bestanden. *Verifieer met git status dat dit gelukt is.*
- ✓ Verwijder “e.txt” uit de staging area. *Verifieer met git status dat dit gelukt is.gi*
- ✓ Stop met het bestand “e.txt” te volgen.
- ✓ Bewaar een nieuwe versie (6) van het project met de mededeling “E verwijderd, D aangevuld”.
- ✓ Bekijk de geschiedenis van het project.
- ✓ Wijzig de laatste commit mededeling in “E gewist, D toegevoegd”.

3.6 Een stapje terug …

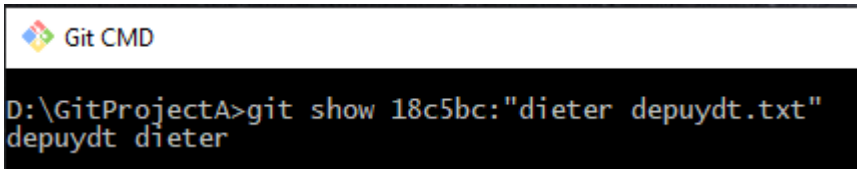
In dit deeltje van de cursus zien we eindelijk waar het hem nu allemaal om te doen is. Waarom houden we verschillende versies bij van al onze bestanden? Om te kunnen zien wat er is gewijzigd en eventueel om de wijzigingen te kunnen terugdraaien.

Het bekijken van vorige versies van bestanden, of het bekijken van verschillen tussen bestanden kan enkel met tekstbestanden. (3.6.1 en 3.6.2). Weet daarbij dat pakweg C# code zich ook in een klassiek tekstbestand bevindt. Git kan uiteraard perfect om met andere bestanden (afbeeldingen, filmpjes, word- en excelbestanden) maar kan ze zelf niet tonen of ons de verschillen tonen.

3.6.1 Hoe was het toen? (git show)

Als het volstaat om een oude versie van een bestand te zien (<> de verschillen tov de huidige versie) dan kan je als volgt te werk gaan.

- ✓ Tik het commando “**`git show [commit-code]:[bestandsnaam]`**” in.



De “commit code” kan je opvragen met het commando “git log”. Je krijgt dan een overzicht van alle commits met per commit een lange hexadecimale code. Het volstaat de eerste 6 tekens over te nemen. Zie ook 3.5.3. Het tonen van een bestand met ‘git show’ lukt enkel voor tekstbestanden. Wil je vorige versies van bv. afbeeldingen bekijken dan zal je die moeten terugzetten en met het juiste programma bekijken.

3.6.2 Wat is er gewijzigd? (git diff)

Voor we een oudere versie van een bestand willen terugzetten gaan we eerst kijken wat het verschil is tussen de huidige versie van het bestand en de oudere versie die we misschien willen terugzetten. Het commando “git diff” is hierbij ons hulpmiddel.

Wil je van een bepaald bestand de verschillen zien tussen de versie in de working directory en de staging area dan kan dat als volgt:

- ✓ Tik het commando “**`git diff [bestandsnaam]`**” in.

Opgelet, als het bestand nog niet werd gestaged dan wordt er automatisch vergeleken met de laatste commit.

Wil je van een bestand de verschillen zien tussen de versie in de staging area en de laatste commit, dan kan dat als volgt:

✓ Tik het commando "**`git diff --cached [bestandsnaam]`**" in.

Wil je van een bestand de verschillen zien tussen de versie in de working directory en de laatste commit (combinatie van voorgaande twee dus), dan kan dat als volgt:

✓ Tik het commando "**`git diff HEAD [bestandsnaam]`**" in.

Wil je van een bestand de verschillen zien tussen de versie in de working directory en een commit, dan kan dat als volgt:

✓ Tik het commando "**`git diff [commit-code] [bestandsnaam]`**" in.

De "commit code" kan je opvragen met het commando "git log". Je krijgt dan een overzicht van alle commits met per commit een lange hexadecimale code. Het volstaat de eerste 6 tekens over te nemen. Zie ook 3.5.3.

Wil je van een bepaald bestand de verschillen zien tussen twee bepaalde commits, dan kan dat als volgt:

✓ Tik het commando "**`git diff [commit-code] [commit-code] [bestandsnaam]`**" in.

Ongeachte het gebruikte commando … je gaat steeds twee versies van een bestand met elkaar vergelijken. De output ziet eruit als volgt:

Git CMD

```
D:\GitProjectA>git diff 18c5bc d4e1ec "dieter depuydt.txt"
diff --git a/dieter depuydt.txt b/dieter depuydt.txt
index 20340a3..d7bb4ee 100644
--- a/dieter depuydt.txt
+++ b/dieter depuydt.txt
@@ -1 +1,2 @@
-depuydt dieter
\ No newline at end of file
+depuydt dieter
+aanpassing 1
\ No newline at end of file
```

Hier werden de verschillen bekeken tussen twee versies (commits) van "dieter depuydt.txt". Er werd de regel "aanpassing 1" aan toegevoegd.

De output van het commando "git diff" is vooral in het begin moeilijk om te interpreteren en om die reden is het sterk aan te raden om een grafische tool te gebruiken. Uiteraard kan het tonen van verschillen slechts voor een beperkt aantal soorten bestanden zoals tekstbestanden, source-code …

3.6.3 Een bestand terugbrengen naar de laatste commit

Op voorwaarde dat je een bestand al minstens één keer gecommit hebt kan je steeds terugkeren naar de laatste commit. Je zal dus al je wijzigingen die je hebt uitgevoerd sinds die laatste commit verwijderen. Je kan dit doen als volgt:

✓ Tik het commando "**`git checkout [bestandsnaam]`**" in.

Opgelet, dit lukt niet als het bestand reeds is gestaged. Als het bestand reeds is gestaged moet het eerst terug uit de staging area gehaald worden. (zie 3.5.1)

3.6.4 Een bestand terugbrengen naar een bepaalde commit

Als een bestand al meerdere keren werd gecommit dan kan je een bestand steeds terugdraaien naar hoe het in een bepaalde commit zat. Ga als volgt te werk:

- ✓ Tik het commando "**`git checkout [commit-code] [bestandsnaam]`**" in.

Voor het achterhalen van de juiste commit-code gebruik je het commando git log. (zie ook 3.5.3)

Na het uitvoeren van dit commando zal je bestand de toestand "modified" hebben. Je hebt het bestand GEWIJZIGD (door een oude versie over de bestaande te schrijven). Om deze oude versie op te nemen in de volgende versie van het project moet je deze dus toevoegen aan de staging area en het project committen.

3.6.5 Het hele project terugbrengen naar een bepaalde commit/versie

De concepten en commando's zijn nauw verweven met branching en daarom heb ik ze daar genoteerd en wil ik hier verwijzen naar hoofdstukken 4.7, 4.8 en 4.9.

3.6.6 Oefeningen

- ✓ Start "Git cmd" en navigeer naar het woordenboek project uit de vorige oefening.
- ✓ Volg het bestand "e.txt" opnieuw.
- ✓ Bekijk de verschillen tussen de huidige versie van e.txt en die in de laatste commit. Probeer de output van "diff" uit te leggen. Leg uit waarom 'git diff e.txt' geen resultaat geeft.
- ✓ Bekijk de verschillen in "a.txt" tussen de derde en de vierde versie. *Dit is de versie met én zonder de tikfout in a.txt.*
- ✓ Verwijder (per ongeluk) a.txt.
- ✓ Zet "a.txt" terug uit de derde commit. (De commit met de omschrijving "A aangevuld".). *Verifieer of de tikfout terug in het bestand is opgedoken.*
- ✓ De letter A is gewijzigd (er werd een oude versie teruggezet) en die wijziging werd nog niet opgeslagen in een versie. Bewaar deze wijziging en die in de letter E in een 7de commit met omschrijving. "E toegevoegd, A oude versie".

4 Branches

4.1 Wat zijn branches?

In de ideale wereld begin je aan een ontwikkeltraject met een eerste versie. Je bouwt verder op die eerste versie en maakt zo een tweede versie. Waarna je weer verder werkt op die tweede versie om een derde versie te maken enz … De versies bouwen netjes verder op de vorige en vormen zo als het ware één tak (of branch).

In de echte wereld echter, is er af en toe nood aan een parallel traject. Op een bepaald moment vertrekken uit één versie twee verschillende nieuwe versies, de tak splitst zich op in twee (of meer) takken.

Een voorbeeld: je hebt een project waarbij de toegangscontrole tot je bedrijf gebaseerd is op tekstbestanden. Voor elke werknemer één tekstbestand met daarin zijn code in de vorm van een MD5-hash. Je baas vraagt je om na te gaan of het mogelijk is de codes vanaf nu te bewaren in SHA256-hashes.

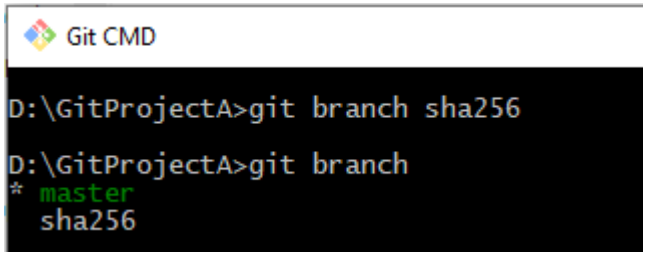
Ingrijpende aanpassingen die wel wat tijd zullen vragen en terwijl je dat doet kan het zijn dat je nieuwe bestanden moet aanmaken of bestaand moet aanpassen met MD5-hashes.

De ontwikkeling van je project verloopt niet meer zuiver lineair. Je werkt op twee verschillende takken. Eén tak waar je het dagdagelijks kunt blijven doen en een tweede tak waar je gaat experimenteren.

Standaard bestaat elk GIT-project uit één branch/tak die typisch 'master' wordt genoemd.

4.2 Een nieuwe branch aanmaken

- ✓ Tik het commando "**`git branch [naam van de nieuwe branch]`**" in.
- ✓ Om te controleren of het gelukt is tik je het commando "**`git branch`**" in.



Je ziet dat er nu twee branches zijn en dat de master branch de actieve branch is. Opgelet, de tak is er nog niet. We hebben enkel aangegeven dat hij er zit aan te komen.

Een branch aanmaken en er onmiddellijk naartoe switchen (zie 4.3) kan:

- ✓ **`git checkout -b [naam van de nieuwe branch]`**

4.3 Switchen tussen branches

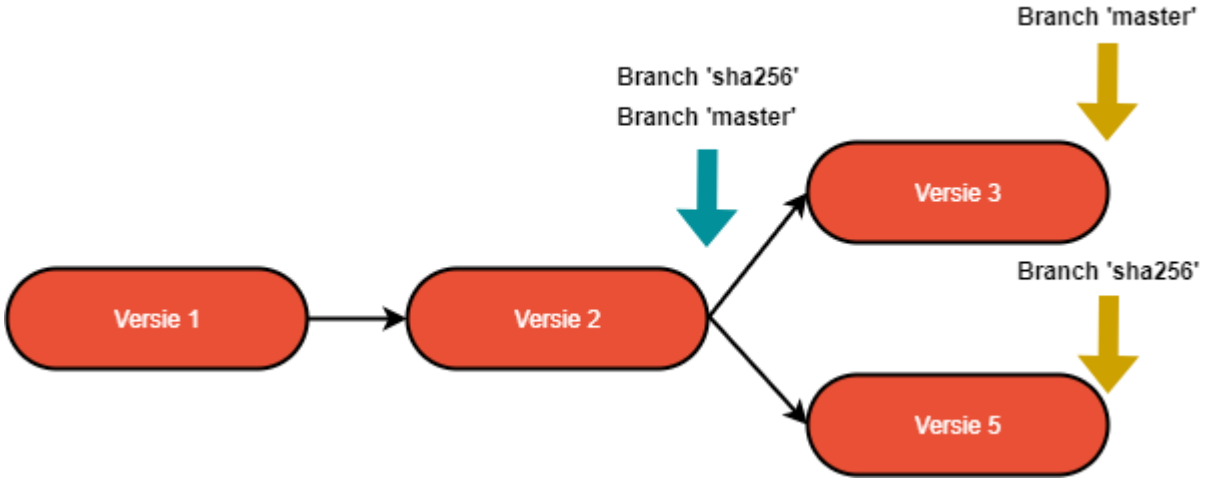
Het is belangrijk om te begrijpen wat er gebeurt als je switcht tussen branches. Enerzijds geef je aan dat commits/versies vanaf nu toebehoren aan de branch die je kiest maar tegelijk zet je alle bestanden in je working directory naar de situatie zoals ze was bij de laatst commit van de gekozen branch.

Om over te stappen naar de nieuwe branch en daarop verder te werken geef je volgende commando in:

- ✓ **`git checkout [naam van de nieuwe branch]`**

Zal dit NU meteen een groot effect hebben op de working directory? Neen! Hoewel alle voorbereidingen getroffen zijn voor de nieuwe tak, is de tak er zelf nog niet. (zie schema hieronder) We bevinden ons op dit

ogenblik daar waar de groen/blauwe pijl zich bevindt. Er zijn twee takken gemaakt en ze bevatten 2 versies. Hier zijn de twee takken nog één. Vanaf nu zal deze tak zich uitsplitsen. Doen we een commit in de ene tak dan groeit die, doen we een commit in de andere tak dan groeit die.



Om tot de bruine pijlen te komen ga ik als volgt te werk:

> *Ik let erop dat "master" de actieve branch is en ik maak 2 nieuwe bestanden aan nl. "bart devos.txt" en "herlinde peeters.txt". Ik voeg de beide bestanden toe aan de staging area en commit de wijzigingen (versie 3). Ik switch naar de nieuwe branch (sha256). De twee nieuwe bestanden verdwijnen automatisch. Dat is normaal ze zaten niet in de laatste commit van sha256. Hoezo? Er is toch nog geen enkele commit in sha256? Alle commits van de gemeenschappelijke tak worden onderdeel van de nieuwe tak. Concreet hebben we in de "sha256" tak dus "versie 1" en "versie 2". Ik wijzig MD5-hash in een SHA256-hash in "dieter depuydt.txt". Ik voeg het bestand toe aan de staging area en commit de wijzigingen. (versie 5).*

Het visualiseren van takken/branches door middel van een pijl is bewust gekozen. Git houdt een pointer (pijl) bij per branch/tak om te weten waar hij volgende versies aan moet vastknopen. Bovendien is er nog een extra pointer die als het ware nog een stapje hoger in de hiërarchie staat nl. de HEAD pointer. Deze houdt bij waar we NU zitten. Typisch is dat bij de pointer van de branch die nu actief is. Uitzonderlijk kan de HEAD pointer elders zitten, we spreken dan van een DETACHED HEAD.

4.4 Een branch stashen

4.4.1 Wijzigingen ongedaan maken en aan de kant zetten

Van branch wisselen kan alleen maar als je working directory 'clean' is. Dat wil zeggen als de working directory of staging area GEEN bestanden bevatten die gewijzigd zijn én nog niet doorgevoerd.

Door te switchen van branch zou je wijzigingen in de working directory verliezen. Bij het switchen van branch geef je immers niet alleen aan dat je vanaf nu versies wil toevoegen aan die branch, je zet de working directory ook terug naar de laatste commit van die branch. (zie opmerkingen hieromtrent aan het eind van dit deeltje).

Git CMD

```
D:\GitProjectA>git checkout master
error: Your local changes to the following files would be overwritten by checkout:
        lieselot vershuere.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

In bovenstaand screenshot geeft Git aan dat we wijzigingen hebben aangebracht aan “lieselot vershuere.txt” en dat we die wijzigingen eerst moeten verwerken (stagen én committen).

Dat is in elk geval de mooiste manier van werken maar soms heb je het te druk en wil je gewoon snel even kunnen switchen. Dan kan je je werk stashen of aan de kant zetten.

✓ Tik het commando “**`git stash push`**” in.

Alle wijzigingen die je hebt gedaan sinds de laatste commit worden ongedaan gemaakt én aan de kant gezet. Het gevolg is dat je geen gewijzigde bestanden meer hebt sinds de laatste commit én je kan switchen van branch.

Git CMD

```
D:\GitProjectA>git stash push
Saved working directory and index state WIP on sha256: ba714f7 Versie 3, eerste versie in sha256

D:\GitProjectA>git checkout master
Switched to branch 'master'
```

Je kan de stash ook een beschrijving (<> naam) geven:

✓ Tik het commando “**`git stash push -m [beschrijving van de stash]`**” in.

De naam van een stash kan je niet wijzigen. Die is altijd van de vorm stash@{XXX}, waarbij XXX een volgnummer is. Hoe groter het volgnummer, hoe ouder de stash. De eerste stash die je maakt krijgt dus nummer 0. Maak je een tweede stash, dan krijgt de eerste stash nummer 1 en de nieuwe nummer 0. Je begrijpt dat een beschrijving van de stash wel een meerwaarde kan zijn. Je kan een lijst opvragen van alle stash’en.

✓ Tik het commando “**`git stash list`**” in.

Git CMD

```
D:\GitProjectA>git stash list
stash@{0}: On sha256: Nu ook met wijziging herlinde peeters
stash@{1}: WIP on sha256: ba714f7 Versie 3, eerste versie in sha256
```

4.4.2 Wijzigingen terug toepassen

Om later de wijzigingen die je al had aangebracht terug toe te passen …

✓ Tik het commando “**`git stash apply [naam van de stash]`**” in.

Git CMD

```
D:\GitProjectA>git stash apply stash@{0}
On branch sha256
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   herlinde peeters.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Je krijgt te zien welke wijzigingen het toepassen van de stash teweeg bracht. In dit geval werden er wijzigingen aan “herlinde peeters.txt” aangebracht. Hoewel de wijzigingen nu terug zijn toegepast blijven ze beschikbaar als stash. Je kan de wijzigingen ook toepassen en die dan als stash verwijderen:

✓ Tik het commando “**`git stash pop [naam van de stash]`**” in.

Heb je de stash al toegepast en wil je hem nu nog verwijderen?

✓ Tik het commando "**`git stash drop [naam van de stash]`**" in.

Of wil je ineens alle stashes verwijderen?

✓ Tik het commando "**`git stash clear`**" in.

Opgelet! Als je bv. 'stash@{0}' pop't of verwijdert dan heb je mogelijks onmiddellijk een nieuwe 'stash@{0}'? De jongste stash is weg, alle oudere krijgen een nieuw nummer (1 wordt 0, 2 wordt 1 …).

4.4.3 Stash conflicten

Af en toe zorgt ook het stashen van wijzigingen voor conflicten. Stel je voert een wijziging door aan het bestand "dieter depuydt.txt". Je stasht deze wijziging (waardoor ze verdwijnt) en even later breng je een nieuwe wijziging aan aan datzelfde bestand.

Wanneer je de wijzigingen in de stash (stapel) opnieuw wil toepassen slaagt Git daar niet in. Het bestand is immers niet meer zoals Git het heeft achtergelaten na de stash push. Wat je sowieso zal moeten doen is de laatst uitgevoerde wijzigingen stagen en committen. Daarna kan je proberen om de wijzigingen in de stash door te voeren. In het beste geval zal dat lukken, in het slechtste geval zal dit zorgen voor een conflict dat je manueel moet oplossen. (zie 5.7.2).

Maak het jezelf niet te moeilijk. Zoals bij het begin van dit deeltje geschreven … Stashen heeft tot doel om wijzigingen **even** aan de kant te zetten, gebruik het niet als een soort tweede manier om te branchen. Met die gedact in het achterhoofd blijven conflicten bij het stashen zeker achterwege.

Het niet kunnen switchen van branch bij een working directory die niet clean is verdient wat nuance. Zolang Git geen wijzigingen dreigt te verliezen blijft het switchen van branch mogelijk.

- Een nieuw bestand aanmaken in een branch zal het switchen naar een andere branch niet verhinderen. Het nieuwe bestand zal in de andere branch ook als een toe te voegen bestand worden beschouwd.
- Ook het wijzigen van een bestand dat in beide branches identiek is zal in beide branches gewoon als een gewijzigd bestand gezien worden.

In de oefeningen komt dit nadrukkelijker naar voor.

4.5 Twee branches samenvoegen

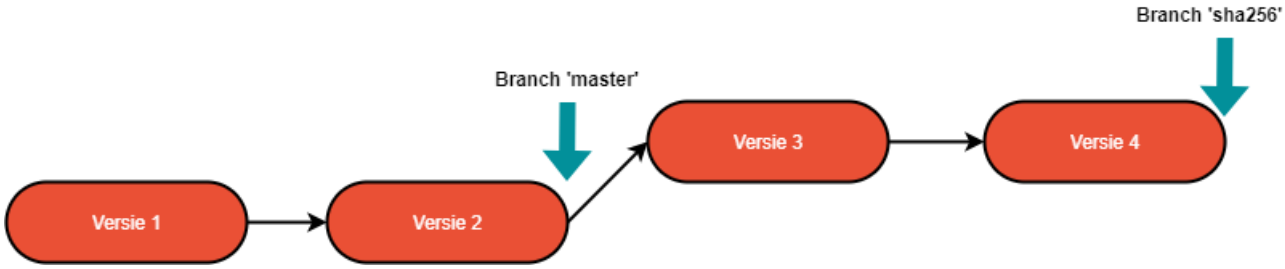
Na een tweetal versies van Sha256 besluiten we dat dit de weg is die we met ons project willen inslaan. Het is niet langer nodig om twee takken van het project te hebben. We gaan de twee versies van het project samenvoegen. We gaan als volgt te werk:

✓ Tik het commando "**`git checkout [tak waarin je wil verder werken]`**" in. *We gaan naar de situatie net voor we begonnen aan de ontwikkeling met sha256-hashes.*

✓ Tik het commando "**`git merge [wijzigingen die je in de andere tak wil invoegen]`**" in. *We voegen alle wijzigingen uit sha256 door in master.*

4.5.1 Fast Forward

In een eerste situatie blijkt dat we eigenlijk niet verder gewerkt hebben aan onze ‘master’ tak, enkel aan onze ‘sha256’ tak.



Als we nu de bovenstaande commando’s gaan invoeren om ‘master’ en ‘sha256’ samen te voegen krijgen we als resultaat sowieso ‘sha256’. Het enige wat Git nu moet doen is de pijl ‘master’ verplaatsen naar waar nu ook de pijl ‘sha256’ staat. We noemen dit ‘fast forward’. Je ziet dit ook in de feedback van de commando’s.

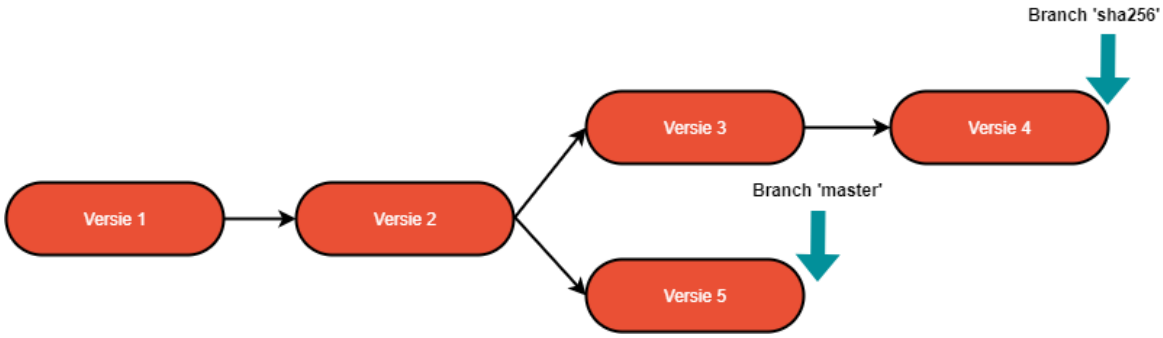
Git CMD

```
D:\GitProjectA>git checkout master
Switched to branch 'master'

D:\GitProjectA>git merge sha256
Updating 5f0dffb..8c3d632
Fast-forward
 dieter depuydt.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

4.5.2 Effectieve samenvoeging (merge commit)

In realiteit is de kans wellicht groot dat we zowel verder gewerkt hebben in ons MD5-hash project als zaken hebben gedaan in ons SHA256-hash-experiment.



Git zal nu echt aan het werk moeten … Het resultaat zal een nieuwe versie/commit (Versie 6) zijn in de ‘master’ tak. Deze Versie 6 zal de samenvoeging zijn van Versie 4 en Versie 5. In het voorbeeld is de omschrijving “Merge branch ‘sha256’.

```
Git CMD - "C:\Program Files\Git\cmd\git.exe" log

D:\GitProjectA>git checkout master
Already on 'master'

D:\GitProjectA>git merge sha256
Merge made by the 'recursive' strategy.
 dieter depuydt.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

D:\GitProjectA>git log
commit 100dffe21de4af4feb2ec9450935fbd6f65ac683 (HEAD -> master)
Merge: 214c21b 8c3d632
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Mon Jul 26 13:41:27 2021 +0200

    Merge branch 'sha256'

commit 214c21b977d64e10bd77e9bc035a5e06af82a30a
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Mon Jul 26 13:40:45 2021 +0200

    Versie 5
```

De kans bestaat dat bij het samenvoegen van deze projecten conflicten zullen ontstaan. *Als je bijvoorbeeld in Versie 4 het bestand 'dieter depuydt.txt' de hash hebt gewijzigd naar een sha256-hash én in Versie 5 heb je datzelfde bestand voorzien van een nieuwe md5-hash dan weet Git nu niet meer wat het moet zijn. Er zal zich een conflict voordoen dat je manueel moet oplossen.*

```
Git CMD

D:\GitProjectA>git merge sha256
Auto-merging dieter depuydt.txt
CONFLICT (content): Merge conflict in dieter depuydt.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Open het bestand (of de bestanden) waarin zich conflicten voordoen en los die op. In ons geval dus "dieter depuydt.txt"

```
dieter depuydt.txt - Kladblok
Bestand  Bewerken  Opmaak  Beeld  Help
depuydt dieter
aanpassing 1
<<<<<<< HEAD
aanpassing 2 - md5
=======
aanpassing 2 - sha256
aanpassing 3 - sha256
>>>>>>> sha256
```

Het conflicht bevindt zich tussen de codes <<<<<<< (7x <) en >>>>>>> (7x >) In de ene tak stond dus "aanpassing 2 – md5", in de andere tak stond er "aanpassing 2 – sha256, aanpassing 3 – sha256". Om verder te kunnen moeten we het conflicht oplossen bv.

```
dieter depuydt.txt - Kladblok
Bestand  Bewerken  Opmaak  Beeld  Help
depuydt dieter
aanpassing 1
aanpassing 2 - sha256
aanpassing 3 - sha256
```

De bestanden die voor een conflict zorgden moeten opnieuw gestaged worden én het hele project moet opnieuw gecommit worden. Als er een conflict is opgetreden worden de bestanden die geen conflict veroorzaakten wel gestaged maar nog niet gecommit!

4.6 Een branch verwijderen

✓ Tik het commando "**`git branch -d [naam van de te wissen branch]`**" in.

```
Git CMD
D:\GitProjectA>
D:\GitProjectA>git branch -d sha256
Deleted branch sha256 (was 8c3d632).
```

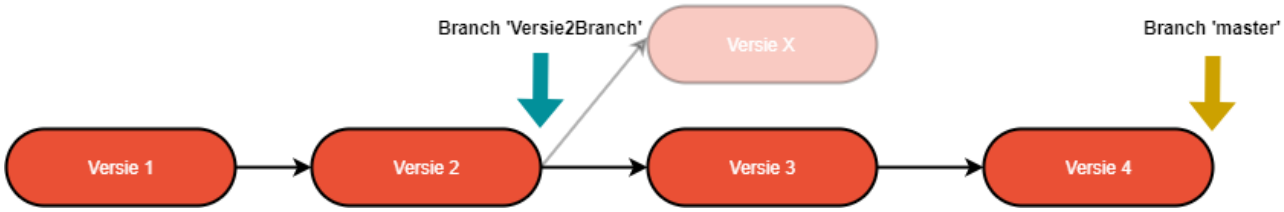
Opgelet, de tak wordt verwijderd maar de versies op die tak blijven in de historiek van git bewaard. Je kan dus als je dat wil terugkeren naar die versie.

4.7 Een branch maken van een (oude) commit

Bij het aanmaken van een nieuwe branch, splitst de tak zich gewoonlijk waar hij gekomen was, bij de laatste commit dus. Soms wil je een nieuwe branch maken op basis van hoe het project ervoor stond enkele commits geleden. Dat kan …

✓ Tik het commando "**`git branch [naam voor de branch] [commit code]`**" in.

De commit code kan je eventueel opzoeken met het commando git log. (zie 3.5.3). Dit is één manier om een project terug te draaien naar een vorige versie zonder dat je grote risico's neemt. Als je deze branch checkout dan zal je working directory eruit zien zoals X commits geleden.




```
Git CMD

D:\GitProjectA>git branch Versie2Branch 1ef538

D:\GitProjectA>git branch
  Versie2Branch
* master

D:\GitProjectA>git log
commit aa5d61648f68860f16a59d23d28c6bd8b20870da (HEAD -> master)
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Tue Jul 27 15:22:49 2021 +0200

    Versie 4

commit ba714f71757173cbd3e03aec62ce8916d365ba71
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Mon Jul 26 17:54:03 2021 +0200

    Versie 3, eerste versie in sha256

commit 1ef5381ec2f6a04bad82be4d7acf44cccabdd14b (Versie2Branch)
Author: Dieter Depuydt <dieter@depuydt.eu>
Date:   Mon Jul 26 17:50:28 2021 +0200

    Versie 2
```

Je kan dit gerust doen want je kan op elk moment terugkeren naar de meest recente situatie door de 'master' branch terug te selecteren (checkout). Verder bouwen op de nieuwe branch kan door na de checkout naar de nieuwe branch een commit uit te voeren. (zie "Versie X" in de visualisatie).

4.8 De tak verlaten (HEAD detached)

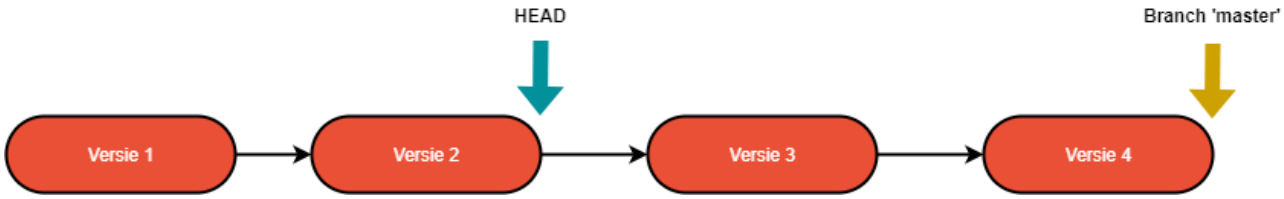
In praktijksituaties gebruik ik dit nooit, er zijn voldoende goede alternatieven zoals bv. 4.7.

In het vorige hoofdstuk maakten we een nieuwe branch op basis van een oude commit. Die oude commit zou dan de plek worden van waar de tak zou splitsen. Soms willen we gewoon de oude commit eens kunnen inkijken zonder daar echt een nieuw parallel ontwikkeltraject te beginnen.

Dat kan …

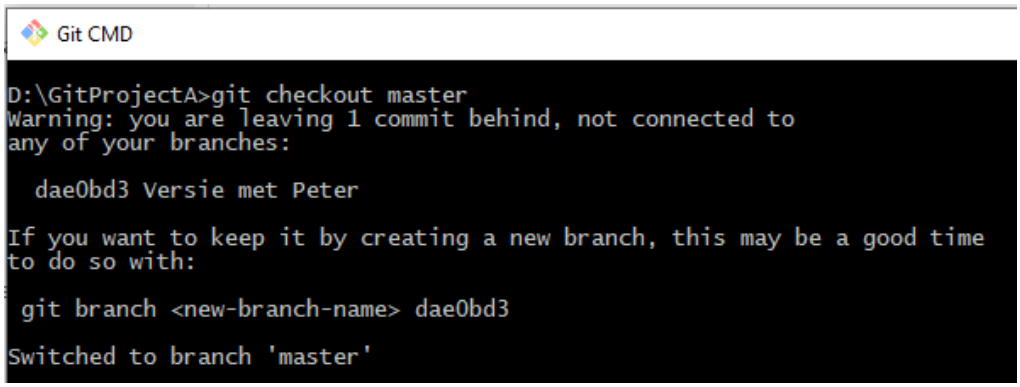
✓ Tik het commando "**`git checkout [commit code]`**" in.

De working directory wordt onmiddellijk teruggedraaid naar de situatie op het ogenblik van de gekozen commit. Het is goed te weten wat er achter de schermen is gebeurd.



Deze manier van werken is wel bijzonder. "HEAD" is de pointer die aangeeft waar in het project we nu zitten. Normaal gezien is dat altijd bij het uiteinde van een tak. *(Ik heb de HEAD pointer in vorige visualisatie nooit getekend)*. "De HEAD is dan attached". We hebben met dit commando onze HEAD losgemaakt van de tak "HEAD is DETACHED".

Hoewel je wat meldingen krijgt is een detached HEAD geen fout. Je kan in deze situatie veel doen binnen Git waaronder bestanden toevoegen aan de stage en die committen. Deze nieuwe versie hangt dan wel een beetje te zweven want herinner je, we zitten niet meer op een tak.



Wanneer je deze "tak" (die er geen is) verlaat word je hiervan nogmaals verwittigd. Je laat X commits/versies achter. Je kan ze vastmaken aan een nieuwe tak als je dat wil.

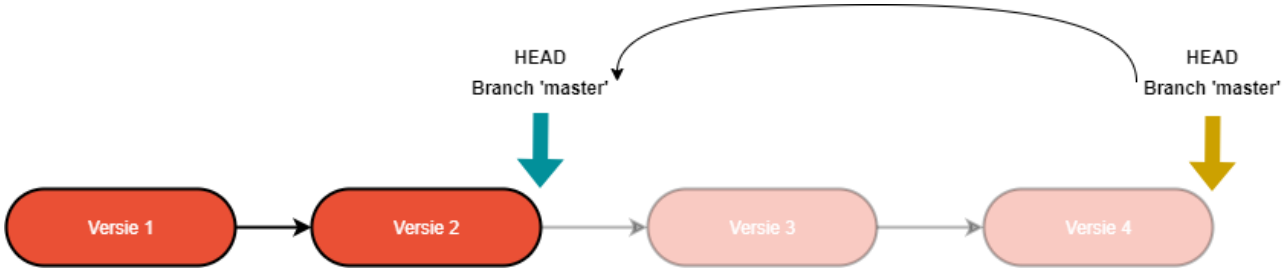
✓ Tik het commando "**`git branch [naam naar keuze][commit code]`**" in.

Doe je dat niet dan blijven die versies/commits rondzweven in je project tot ze opgeruimd worden door de garbage collector.

4.9 De tak een stukje afzagen (reset)

In praktijksituaties gebruik ik dit nooit, er zijn voldoende goede alternatieven zoals bv. 4.7.

Als je helemaal komaf wil maken met het verleden kan je een aantal versies terugkeren en de commits die je niet meer nodig hebt definitief verwijderen. Vergelijk het gerust met het afzagen van een tak. Het hoeft geen groot betoog dat dit een ingrijpende actie is én dat een foutje grote gevolgen kan hebben.



Het is een beetje te vergelijken met wat we gedaan hebben in 4.8. Alleen keren we nu niet eventjes terug naar een bepaalde commit door onze 'HEAD' te verplaatsen. We verplaatsen meteen ook de pointer van de branch. De versies die we na deze versie maakten op deze branch/tak worden als het ware afgezaagd.

- ✓ Tik het commando "**`git reset [commit code] --hard`**" in.

De parameter "--hard" zorgt ervoor dat ook de working directory én de staging index afgestemd worden op de commit die je gekozen hebt. Dat wil zeggen dat nieuwe bestanden uit Versie 3 en Versie 4 verwijderd worden, wijzigingen uit die versies worden ook in de working directory ongedaan gemaakt.

4.10 Oefeningen

- ✓ Maak een nieuwe branch in je project genaamd "woordenlijsten".
- ✓ Controleer of je nieuwe branch effectief is aangemaakt en welke branch actief is.
- ✓ Activeer de branch "woordenlijsten".
- ✓ Voeg de bestanden f.txt, g.txt, h.txt die je bij deze cursus vindt toe aan je woordenboek project in de branch "woordenlijsten".
- ✓ Commit deze wijzigingen in een versie (8) die je de omschrijving "Toevoeging F, G en H" meegeeft.
- ✓ Keer terug naar de "master" branch. *Wat valt je op?*
- ✓ Voeg het bestand i.txt die je bij deze cursus vindt toe aan je woordenboek project in de master branch.
- ✓ Commit deze wijziging nog niet.
- ✓ Keer terug naar de branch "woordenlijsten". *Hoe komt het dat dit lukt ondanks het feit dat de working directory en staging area niet 'clean' zijn?*
- ✓ Keer terug naar de branch "master".
- ✓ Zet het typfoutje in "a.txt" terug recht. (aanduwwen => aanduwen).
- ✓ Stage deze wijziging maar doe nog geen commit.
- ✓ Keer terug naar de branch "woordenlijsten". *Hoe komt het dat dit lukt ondanks het feit dat de working directory en staging area niet 'clean' zijn?*
- ✓ Keer terug naar de branch "master".
- ✓ Commit de wijzigingen met als vermelding "Correctie A, Toevoeging I".
- ✓ Open het bestand a.txt en voeg het woord "analfabeet" toe.
- ✓ Keer terug naar de branch "woordenlijsten". *Denk je dat dit zal lukken? Waarom (niet)?*
- ✓ Stash je wijzigingen, geef als omschrijving mee "aanpassingen aan A". *Controleer of je wijzigingen in bestand A ongedaan werden gemaakt.*
- ✓ Open het bestand b.txt en voeg de woorden "beladen" en "berekend" toe aan het bestand.

- ✓ Stash je wijzigingen, geef als omschrijving mee “aanpassingen aan B”. *Controleer of je wijzigingen in bestand B ongedaan werden gemaakt.*
- ✓ Keer terug naar de branch “woordenlijsten”. *Lukt dit?*
- ✓ Keer terug naar de branch “master”.
- ✓ De wijzigingen in a.txt en b.txt zitten elk in een stapeltje (stash). Voer beide stapeltjes terug uit en let erop dat de stapeltjes meteen gewist worden.
- ✓ Stage de wijzigingen in a.txt en b.txt en commit deze wijzigingen met als commentaar “uitbreidingen A en B”.
- ✓ Keer terug naar de branch “woordenlijsten”.
- ✓ Voeg een bestand toe met als bestandsnaam “i.txt” en geef als inhoud het woord “idyllisch” in.
- ✓ Stage en commit deze wijziging met als naam “Toevoeging I in woordenlijsten”.
- ✓ Keer terug naar de branch “woordenlijsten”.
- ✓ Voer een merge uit van master met woordenlijsten. *Voor je dit doet, zal dit een fast-forward zijn of een echte merge? Verwacht je merge conflicten?*
- ✓ Los het probleem in i.txt op. De inhoud uit de masterbranch (de +/- 16 woorden behouden we, “idyllisch” mag verwijderd worden.
- ✓ Bekijk de status van het project. *Je zou 3 bestanden moeten zien die gestaged zijn en ééntje die gewijzigd is (i.txt).*
- ✓ Stage nu ook bestand i.txt en commit de wijzigingen met als omschrijving “Samenvoeging master en woordenlijsten”.

5 Remotes

Zoals eerder is gebleken kan je Git perfect gebruiken zonder server. Als je aan versiebeheer doet om enkel en alleen op je eigen stappen te kunnen terugkeren dan heeft het ook weinig zin om met een server te werken (tenzij als backup-oplossing?).

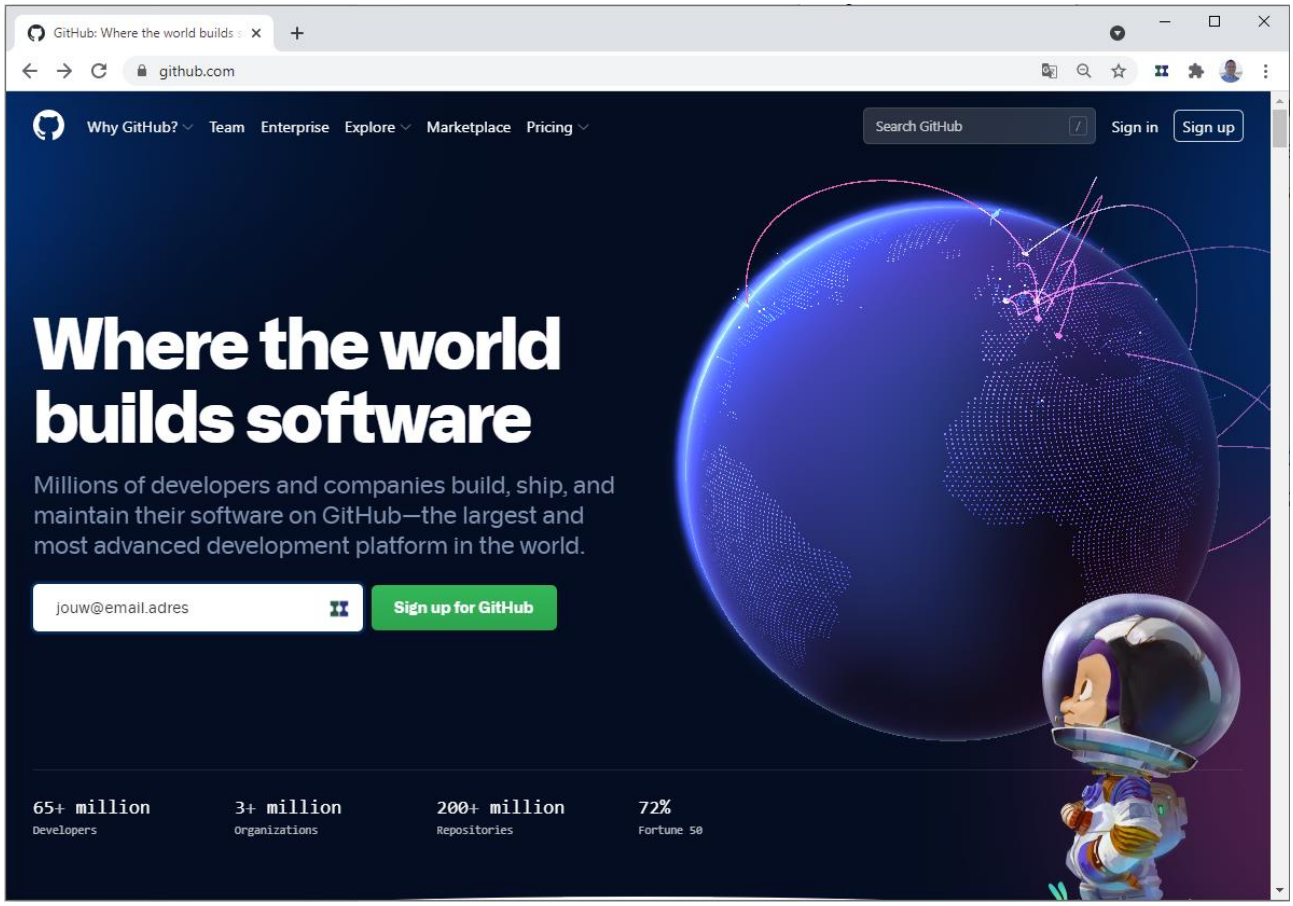
Vanaf het ogenblik dat je met meerdere mensen samen aan een project werkt is een server als "uitwisselcentrum" noodzakelijk. Je kan zelf een Git server opzetten (buiten het bestek van deze cursus) of je kan een stukje "Git serverruimte" huren. Er zijn heel wat bedrijven die Git hosting aanbieden oa. bitbucket.org, gitsrc.com, codebasehq.com en github.com. In deze cursus werken we met GitHub.

5.1 GitHub

GitHub is een bedrijf/project dat online ruimte aanbiedt om Git repositories op te slaan. Ze bieden die ruimte zowel gratis aan als tegen betaling. De gratis ruimte is voor publieke projecten. Deze projecten zijn steeds voor heel de wereld zichtbaar, je kan ze niet beveiligen of afschermen. De betalende ruimte kan je dan weer wel afschermen van de rest van de wereld.

Een account aanmaken is in elk geval gratis en zolang je geen afgeschermde projecten nodig hebt blijft het ook allemaal gratis. (Studenten krijgen via het GitHub student pack ook gratis afgeschermde projecten).

Om een account aan te maken surf je naar de website http://www.github.com en vul je je e-mailadres in en je klikt op de knop "Sign up for GitHub".

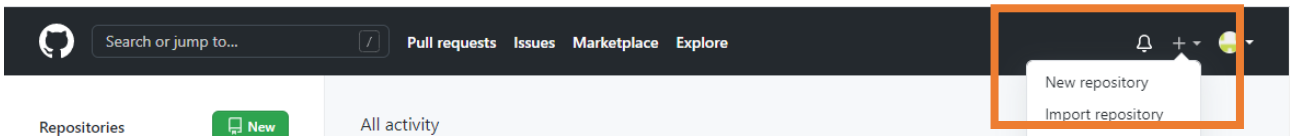


Vervolgens kom je op een nieuwe pagina (een beetje starwars stijl) waarin gevraagd wordt naar een wachtwoord, gebruikersnaam … Klik tenslotte op "Create account".

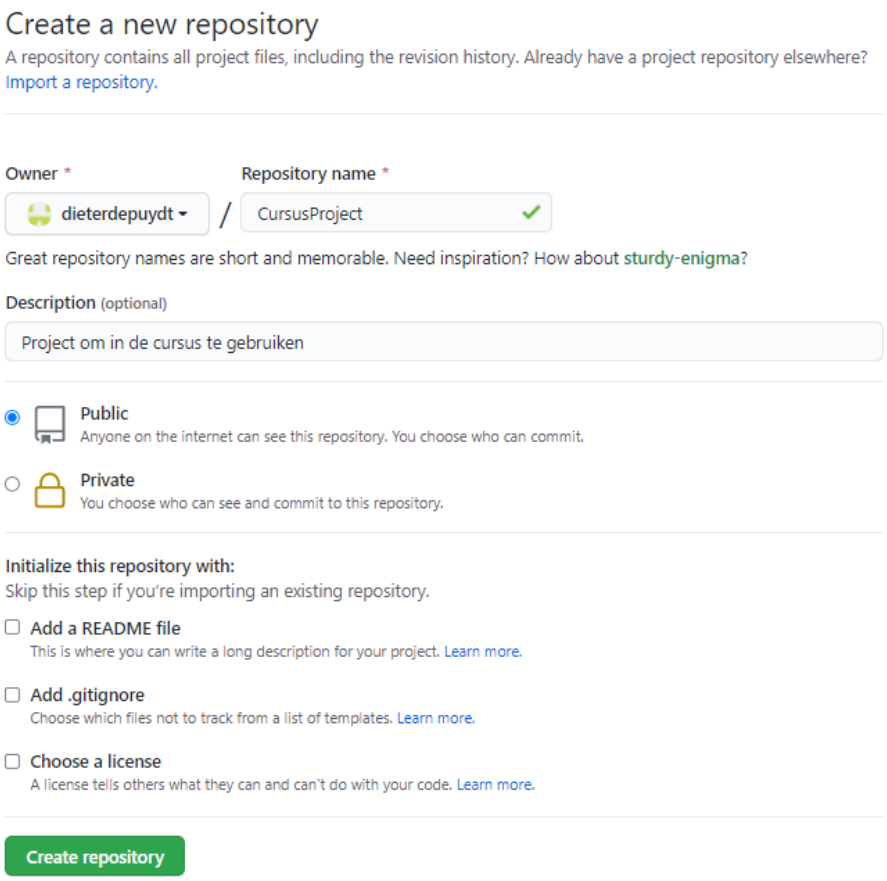
5.2 Een Git project aanmaken bij GitHub

Voor elk Git project dat je online wil hosten bij GitHub moet je een project aanmaken. Ga als volgt te werk:

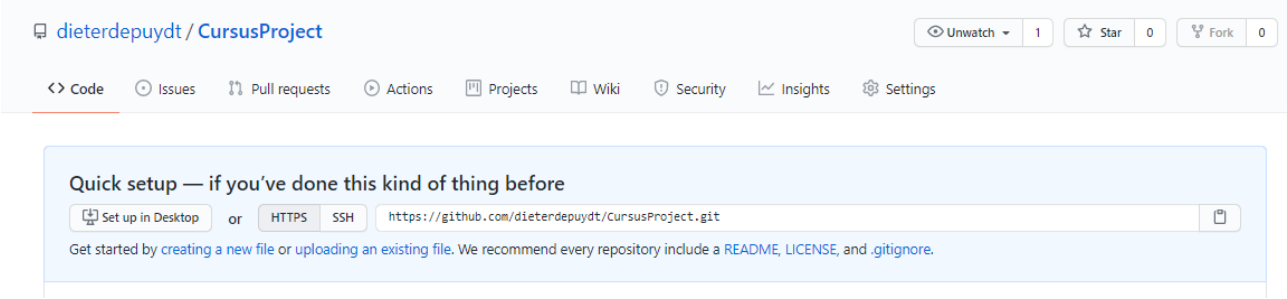
✓ Klik bovenaan rechts in de titelbalk op het "+" icoontje en kies uit het menu dat verschijnt "New repository"



✓ In het scherm dat verschijnt geef je het project een naam en eventueel een omschrijving. Het type project laat je ingesteld staan op "Public". Dat wil zeggen dat iedereen het project zal kunnen zien (niet wijzigen!) maar enkel dit type project is gratis. Bevestigen doe je met de knop "Create repository"



Het project wordt nu aangemaakt en kan onmiddellijk gebruikt worden. Zolang het project nog niet in gebruik is krijg je een woordje uitleg te zien telkens je de projectpagina bezoekt. Erg belangrijk is de project url, die zal je zo dadelijk nodig hebben.



5.3 Remote toevoegen aan een project

Een plekje op een Git server noemen we ook wel een “remote”. We kunnen één of meer “remotes” toevoegen aan een Git project:

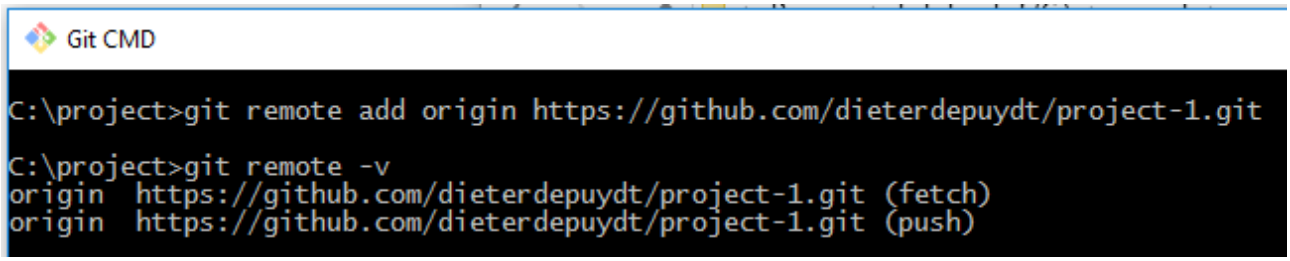
✓ Tik het commando “**`git remote add origin [git url]`**” in.

“Origin” is de naam van mijn remote, ik had die even goed “SmurfProject” mogen noemen maar “Origin” is een veelgebruikte naam voor een remote in Git.

Wanneer je een project niet zelf bent gestart maar je hebt het gecloned van een server (zie 3.3.2) dan heb je al een remote in het project nl. de server waar je het project hebt gehaald.

Om te weten welke remotes al gedefinieerd zijn voor een Git project kan je als volgt te werk gaan:

✓ Tik het commando “**`git remote -v`**” in.



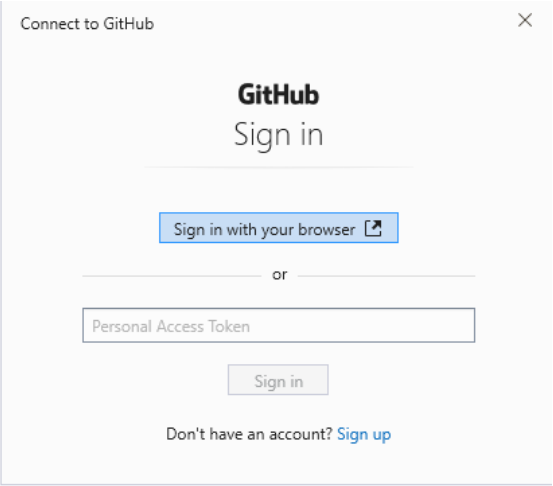
Wil je een bepaalde remote niet langer gebruiken dan kun je die uit het Git project verwijderen. Opgelet, je verwijdert enkel de link met de remote. De eventuele versies die je al op de remote hebt geplaatst blijven daar gewoon staan!

✓ Tik het commando “**`git remote remove [naam]`**” in.

5.4 Versies naar de remote kopiëren

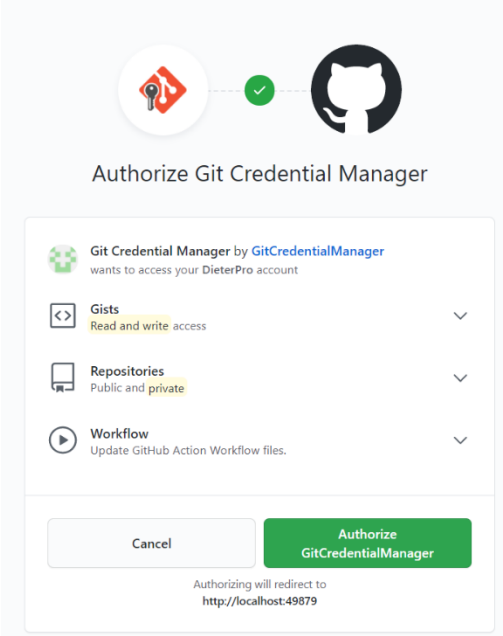
Van zodra er een remote gedefinieerd is in je Git project kan je er alle versies van je project naartoe kopiëren zodat je collega’s ze daar kunnen gaan ophalen. Een project kopiëren naar de remote doe je als volgt:

✓ Tik het commando “**`git push origin master`**” in.



Als dit de eerste keer is dat je iets publiceert naar de online repository dan zal de Git Credential Manager Core (zie tijdens installatie van Git) verschijnen waarmee je op een veilige manier kan aanmelden op GitHub. Deze credential manager zorgt voor de veilige opslag van wachtwoorden.

Klik op “Sign in with your browser”. Je browser wordt geopend en als je nog niet aangemeld bent op GitHub krijg je nu de kans om aan te melden. Eens je bent aangemeld op GitHub krijg je een scherm te zien zoals hieronder (volgende pagina, linksbovenaan). GitHub vraagt of je zeker bent dat dat je de Credential Manager toegang wil geven tot je GitHub omgeving. Klik op “Authorize GitCredentialManager” om aan te geven dat je dit wenst.



"Origin" is de naam van je remote, "master" is de naam van je branch. Wil je ineens alle branches naar GitHub kopiëren? Gebruik dan dit commando:

✓ "**`git push origin --all`**

Opgelet! In GitHub kan je gratis public projecten aanmaken. Deze zijn toegankelijke voor de hele wereld MAAR alleen jijzelf kan schrijven in dit project. Dus het pushen van versies naar de server is voorlopig enkel voor jezelf mogelijk. Wil je andere GitHub gebruikers SCHRIJF rechten geven op je project, dan moet je ze aan het project toevoegen als "collaborator".

5.5 Oefeningen

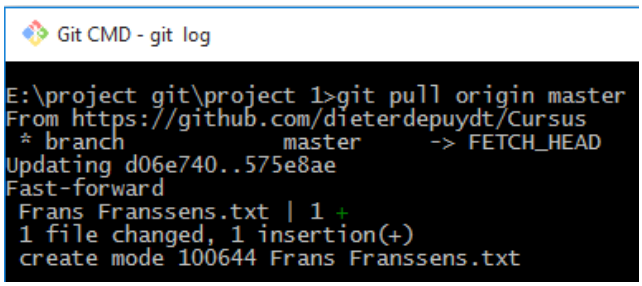
- ✓ Maak een (gratis) account aan op GitHub zoals beschreven in 5.2.
- ✓ Maak een (public) project aan zoals beschreven in 5.3. Geef het project de naam "woordenboek".
- ✓ Voeg dit GitHub project toe aan je project als remote met als naam "origin".
- ✓ Controleer of de remote goed is aangemaakt.
- ✓ Controleer nu of je bestanden, de verschillende versies … zichtbaar zijn in GitHub.
- ✓ Push nu ook alle andere branches (ok, er is maar één andere) naar GitHub en controleer of de branch daar nu ook zichtbaar is.
- ✓ Verwijder de remote opnieuw.

5.6 Versies naar je cliënt kopiëren

Indien je het project niet zelf start dan moet je een initiële versie van het project op je computer zien te krijgen. Dit kan je doen met het commando "git clone". (zie 3.3.2). Eens het project op je computer staat kan je het behandelen alsof je het project zelf bent begonnen. We gaan dan niet meer "clonen" om nieuwe versies binnen te halen maar "pullen" zoals hieronder verder beschreven.

Wanneer je met meerdere mensen aan een project werkt is het niet alleen belangrijk om regelmatig je wijzigingen te committen en op de remote te plaatsen maar ook om de wijzigingen (de commits, de versies) van anderen regelmatig binnen te halen. Dit kan je als volgt doen:

- ✓ Tik het commando "**`git pull origin master`**" in. *Origin verwijst naar de bron, de remote (zie 5.3). Master verwijst dan weer naar de tak, de branch die we willen binnen halen.*



In het deeltje “Omgaan met conflicten” zal het belang van regelmatig committen, pushen en pullen duidelijk worden.

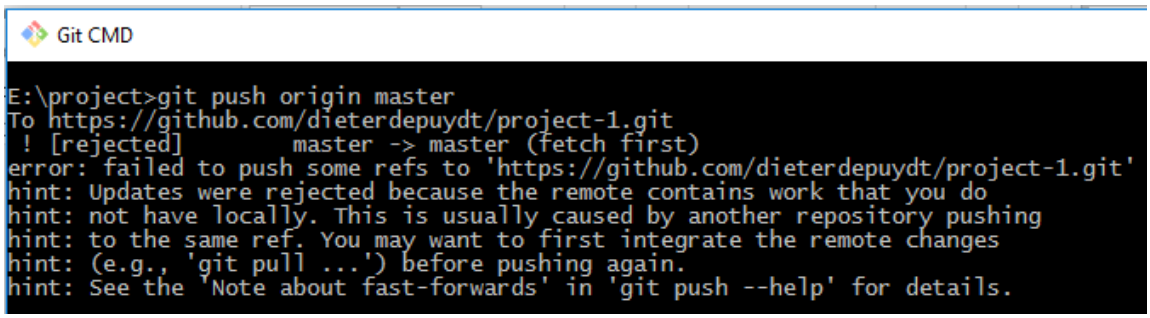
5.7 Omgaan met conflicten

5.7.1 Hoezo conflicten?

Het versiebeheer van Git is eigenlijk een opeenvolging van commits (of versies). Elke commit heeft precies één “vorige commit” (behalve de allereerste commit natuurlijk) en een commit heeft nul of meer “volgende commits”. Dit laatste vraagt wat extra uitleg.

Stel we (gebruiker X) hebben een commit (versie) “A” aangemaakt en op de remote geplaatst. Gebruiker “Y” heeft dit project binnengehaald op zijn computer en beschikt nu dus ook over commit (versie) “A”. Beide gebruikers (X en Y) voeren een aantal wijzigingen door, stagen die en committen die. Elke gebruiker heeft dus een commit die volgt op commit “A”.

Wanneer de eerste gebruiker zijn commit op de remote wil plaatsen (zie 4.4 voor het commando) dan zal dit probleemloos lukken. Wanneer de tweede gebruiker nu zal proberen om zijn commit op de remote te plaatsen zal hij een foutmelding krijgen.



Git vertelt ons dat we onze commit niet zomaar kunnen publiceren omdat wij niet beschikken over de meest recente versie (commit) van het project. Dat klopt we hebben de laatste commit van de eerste gebruiker niet op onze computer. We zullen onze commit NOOIT kunnen publiceren voor we eerst de update van de andere gebruiker hebben binnengehaald. Wat zullen we dus moeten doen?

✓ Tik het commando “**`git pull origin master`**” in.

Git zal de updates van de eerste gebruiker downloaden en kijken of die te verzoenen zijn met de updates die wij (de tweede gebruiker) hebben uitgevoerd. Stel dat de eerste gebruiker een bestand “jan janssens.txt” heeft aangemaakt en wij hebben een bestand “marc peeters.txt” aangemaakt dan zijn die twee ingrepen perfect samen te brengen. De twee commits worden door Git automatisch samengevoegd of “gemerged”.

Bemerk in dit hele deel trouwens dat “eerste gebruiker” en “tweede gebruiker” om het even welke gebruiker kunnen zijn. De gebruiker die zijn commit het eerst op de remote zet is de “eerste gebruiker” en de andere is de “tweede gebruiker”. Het gaat er dus niet om wie het eerst een aanpassing aan het project gedaan heeft, wie het eerste gestaged heeft en zelfs niet wie het eerst een commit heeft uitgevoerd. Het gaat erom wie het eerst zijn commit op de remote heeft geplaatst. Begrijp je nu waarom regelmatig pushen/pullen zo belangrijk is.

5.7.2 Conflicten: als samenvoegen niet lukt

Maar wat als Git er niet in slaagt om de wijzigingen van de eerste gebruiker en de tweede gebruiker met elkaar te verzoenen.

Stel beide gebruikers maken een bestand aan “john doe.txt”, de ene gebruiker plaatst erin “John Doe” en de andere plaatst erin “John Doe, adres onbekend”. Git zal nu niet weten wat er uiteindelijk in het project moet komen “John Doe of John Doe, adres onbekend”.

Opnieuw wanneer de eerste gebruiker zijn commit op de remote plaatst: geen probleem. En eveneens opnieuw als de tweede gebruiker zijn commit op de remote wil plaatsen zal hij een foutmelding krijgen. De tweede gebruiker moet eerst een update binnenhalen (pullen) voor hij verder kan.

Alleen zal er nu bij het pullen een foutmelding verschijnen: een CONFLICT.

Git CMD

```
E:\project>git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/dieterdepuydt/project-1
 * branch            master     -> FETCH_HEAD
   e77721c..4d62382  master     -> origin/master
Auto-merging john doe.txt
CONFLICT (add/add): Merge conflict in john doe.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Git meldt hier dat beide gebruikers een bestand “john doe.txt” hebben aangemaakt met een afwijkende inhoud. Git weet niet wat hiermee te doen en vraagt de tweede gebruiker om dit op te lossen en daarna de bestanden in kwestie opnieuw te stagen en opnieuw te committen.

Om het conflict op te lossen openen we het bestand (of de bestanden, meervoud) waarmee problemen gemeld worden.

john doe.txt - Kladblok

Bestand Bewerken Opmaak Beeld Help

```
<<<<<<< HEAD
Doe John
=======
John Doe
>>>>>>> 4d623821f283a93260e188240bb266f719c67588
```

Conflicten worden door Git in bestanden aangeduid. Het begin van het conflict begint met <<<<<<< (7 x <) en eindigt met >>>>>>> (7 x >). Tussen deze codes staat het stuk tekst van onszelf en van de andere gebruiker van elkaar gescheiden door middel van ======= (7 x =). Het is de bedoeling dat wij, de tweede gebruiker, het conflict oplossen: dus alles tussen <<<<<<< en >>>>>>> wegnemen en vervangen door wat wij denken dat juist is. Daarna moet het bestand (of bestanden) opnieuw gestaged worden, er moet een nieuwe commit gebeuren en dan pas zal er kunnen geüpload worden.

Er kunnen meerdere conflicten in één bestand zitten en er kunnen conflicten in meerdere bestanden voorkomen. Het hoeft niet veel uitleg dat conflicten oplossen géén leuk werkje is. Zeker als je na een lange en vermoeiende dag nog snel je commits wil op de remote plaatsen en dan pas merkt dat je commits voor 317 conflicten zorgen … Je wil dit vermijden!!! En je kan dit enkel vermijden door VAAK en REGELMATIG te committen en je commits op de remote te plaatsen.

6 Alfabetisch overzicht commando's

Commando	Zie …
`git add [bestandsnaam]`	3.4.1 / 3.4.4
`git branch`	4.2
`git branch [naam nieuwe branch] [commit-code]`	4.7
`git branch -d [branch]`	4.6
`git checkout [bestandsnaam]`	3.6.3
`git checkout [branch]`	4.3
`git checkout -b [branch]`	4.2
`git clone [url] [afwijkende naam]`	3.3.2
`git commit –amend`	3.5.4
`git commit -m [boodschap]`	3.4.2
`git config --global core.editor [editor]`	2.3
`git config --global user.email [emailadres]`	2.3
`git config --global user.name [naam]`	2.3
`git diff [bestandsnaam]`	3.6.2
`git init`	3.3.1
`git log`	3.5.3
`git merge [branch]`	4.5
`git push [naam] [branch]`	5.4
`git push [naam] –all`	5.4
`git remote add [naam] [url]`	5.3
`git remote remove [naam]`	5.3
`git remote -v`	5.3
`git reset [commit-code] –hard`	4.9
`git restore –staged [bestandsnaam]`	3.5.1
`git rm –cached [bestandsnaam]`	3.5.2
`git show [commit-code]:[bestandsnaam]`	3.6.1
`git stash apply [naam stash]`	4.4.2
`git stash clear`	4.4.2
`git stash drop [naam stash]`	4.4.2
`git stash list`	4.4.1
`git stash pop [naam stash]`	4.4.2
`git stash push -m [boodschap]`	4.4.1
`git status`	3.3.1