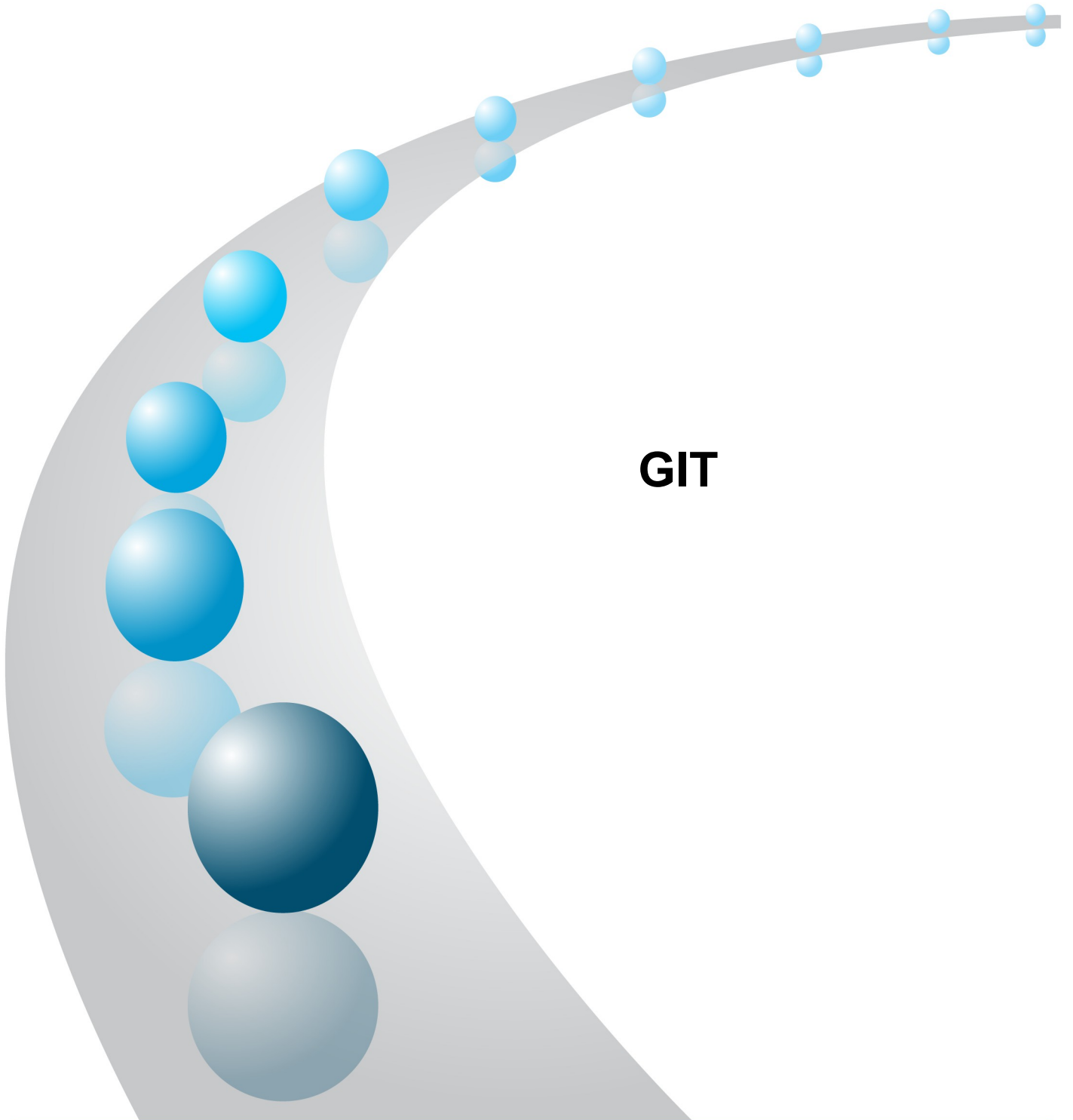




Noël Vaes

Java Trainer & Consultant



GIT

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Vrijwel alle namen van software- en hardwareproducten die in deze cursus worden genoemd, zijn tegelijkertijd ook handelsmerken en dienen dienovereenkomstig te worden behandeld.

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar worden gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de auteur. De enige uitzondering die hierop bestaat, is dat eventuele programma's en door de gebruiker te typen voorbeelden mogen worden ingevoerd opgeslagen en uitgevoerd op een computersysteem, zolang deze voor privé-doeleinden worden gebruikt, en niet bestemd zijn voor reproductie of publicatie.

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

Ondanks alle aan de samenstelling van deze tekst bestede zorg, kan de auteur geen aansprakelijkheid aanvaarden voor eventuele schade die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

17/11/2017

Copyright© 2017 Noël Vaes



Inhoudsopgave

Hoofdstuk 1: GIT	4
1.1 Inleiding.....	4
1.2 Centraal versus gedistribueerd.....	5
1.3 Locking.....	5
1.3.1 Lock-modify-unlock (pessimistic locking).....	5
1.3.2 Copy-modify-merge (optimistic locking).....	6
1.4 Installatie en configuratie van GIT.....	6
1.5 Lokale repositories.....	8
1.5.1 Repository aanmaken.....	9
1.5.2 Toestanden van een bestand.....	9
1.5.3 Indexeren (staging) van bestanden.....	11
1.5.4 Committeren van bestanden.....	11
1.5.5 Verwijderen van bestanden.....	14
1.5.6 negeren van bestanden.....	15
1.5.7 De geschiedenis van het project.....	16
1.5.8 Uitchecken van een project.....	17
1.5.9 Tags.....	19
1.5.10 Vertakkingen of branches.....	21
1.5.10.1 Zijtakken maken.....	22
1.5.10.2 Fast forward merge.....	26
1.5.10.3 Zijtakken samenvoegen	28
1.5.10.4 Zijtak in het verleden maken.....	32
1.5.11 Stashing.....	34
1.6 Remote repositories.....	36
1.6.1 Inleiding.....	36
1.6.2 Soorten remote repositories.....	37
1.6.3 Remote Repository aanmaken.....	38
1.6.4 De Remote Repository aan het project toevoegen.....	39
1.6.5 Pushing.....	39
1.6.6 Cloning.....	43
1.6.7 Fetching.....	45
1.6.8 Pulling.....	47



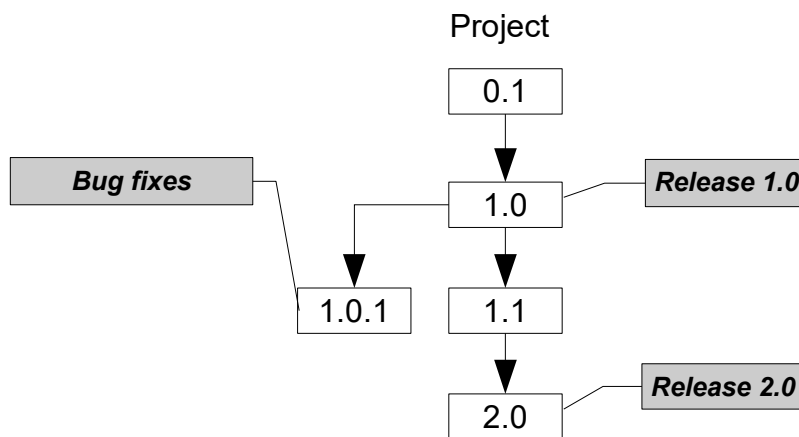
Hoofdstuk 1: GIT

1.1 Inleiding

Bij grotere programmeerprojecten waarbij tevens meerdere personen betrokken zijn, krijgt men te maken met een aantal typische problemen.

Zo doorloopt de ontwikkeling van de software **verschillende stadia**: er ontstaan verschillende versies van de software waaraan telkens wordt verder gebouwd. Op een bepaald moment doet men een *release* en wordt de software vrijgegeven voor uitvoerige testen en uiteindelijk wordt de software geschikt bevonden voor de eindklant. Na deze eerste versie komen er vaak nog aanpassingen die uiteindelijk via de nodige tussenliggende versies resulteren in een nieuwe officiële *release*.

Het eindproduct is de laatste versie in de rij van opeenvolgende versies waarbij telkens aanpassingen gedaan werden aan de bestaande versie.



Daar komt nog bij dat vaak op basis van een bestaande versie **parallel** gewerkt wordt aan kleine aanpassingen aan deze versie (*bug fixes*) en aan de voorbereiding van een nieuwe grote *release* met allerlei nieuwe functionaliteiten. Sommige delen van de code zijn hetzelfde, andere delen worden in het nieuwe traject helemaal herwerkt terwijl de oorspronkelijke code nog wordt onderhouden voor de ondersteuning van de oudere versie.

Verder wordt aan dergelijke grote programmeerprojecten meestal met **meerdere mensen** gewerkt. Samen delen ze dan dezelfde broncode waarbij ieder verantwoordelijk is voor een deel van die broncode. De broncode moet daarom op een of andere manier gedeeld kunnen worden zodat iedere programmeur er toegang toe heeft, de code kan compileren en uitvoeren en er ten slotte zijn eigen wijzigingen in kan aanbrengen. De andere programmeurs moeten nadien deze wijzigingen kunnen overnemen.

Vaak gebeurt het dat twee programmeurs tegelijkertijd bezig zijn met hetzelfde stuk broncode. Dan is het belangrijk dat de wijzigingen die door beide programmeurs gedaan werden, op correcte wijze geïntegreerd worden.

Bij de ontwikkeling van *Open Source* projecten worden deze problemen nog erger. Hier wordt door heel veel programmeurs gewerkt aan hetzelfde project dat meestal ook nog



allerlei stadia heeft met heel wat vertakkingen.

Bovenstaande problemen maken een systeem noodzakelijk dat de broncode van een programmeerproject beheert en de verschillende versies van de software bijhoudt en toegankelijk maakt. Dit noemt men een systeem voor versiebeheer (*version control system of VCS*).

1.2 Centraal versus gedistribueerd

Systemen voor versiecontrole kunnen ingedeeld worden in twee groepen. Zo zijn er de systemen die de bestanden centraal beheren en waarbij iedere deelnemer van het project een (gedeeltelijke) lokale kopie hiervan maakt, vervolgens wijzigingen aanbrengt en nadien deze wijzigingen teruggeplaatst op de centrale locatie zodat ze beschikbaar zijn voor de andere deelnemers. We spreken over *Centralized Version Control Systems* of CVCS.

Het nadeel van dergelijke centrale systemen is evenwel dat gebruikers steeds *online* moeten zijn om de volledige functionaliteit te kunnen benutten.

Gekende gecentraliseerde systemen zijn *ClearCase*, *PVCS*, *SourceSafe* enzovoort. In de *Open Source* wereld hebben we het inmiddels verouderde **Concurrent Versions System** (CVS) en diens opvolger **Subversion**.

Naast gecentraliseerde systemen zijn er ook gedistribueerde systemen waarbij de bestanden niet centraal bewaard worden maar op verschillende plaatsen. Men noemt dit daarom *Distributed Version Control Systems* of DVCS.

GIT is een dergelijk gedistribueerd systeem dat momenteel erg populair is. In deze cursus zullen we leren hoe GIT werkt.

1.3 Locking

Om de werking van versiecontrolesystemen te begrijpen moeten we eerst de problematiek van gedeelde bestanden verder bekijken. Indien twee gebruikers van hetzelfde bestand gebruik willen maken en dit bestand willen aanpassen, kunnen er conflicten optreden.

Gebruiker A kan een bestand uit het versiecontrolesysteem halen, dit aanpassen en het gewijzigde bestand opnieuw wegschrijven in het systeem.

Gebruiker B kan dat zelfde bestand ook lezen en aanpassen. Indien hij echter zijn wijzigingen iets later dan A wegschrijft, gaan de wijzigingen van A verloren.

Om dit probleem op te lossen zijn er twee mechanismen:

1.3.1 Lock-modify-unlock (pessimistic locking)

Een eerste mechanisme is het *lock-modify-unlock*-mechanisme. Men noemt dit ook wel *pessimistic locking*. Hierbij zal gebruiker A het bestand lezen en onmiddellijk vergrendelen. Nadat hij wijzigingen heeft aangebracht kan hij het nieuwe bestand opnieuw wegschrijven en wordt het bestand ontgrendeld. Zolang het bestand vergrendeld is, kan een andere gebruiker het bestand niet wijzigen. Alleen degene die het bestand vergrendeld heeft, kan wijzigingen aanbrengen.

Gebruiker B moet dus wachten totdat het bestand ontgrendeld is vooraleer hij een wijzigbare versie van het bestand kan bemachtigen.

Dit mechanisme heeft echter een aantal nadelen:

1. Indien gebruiker A **vergeet** het bestand opnieuw te ontgrendelen kan gebruiker B



niet verder werken. De beheerder moet dan het bestand manueel opnieuw ontgrendelen.

2. Gebruiker B moet altijd wachten totdat het bestand ontgrendeld is, ook al zijn de wijzigingen die hij maakt **niet** noodzakelijk **conflicterend** met de wijzigingen die gebruiker A maakt. Denken we bijvoorbeeld aan een broncode-bestand waarbij de verschillende programmeurs op verschillende plaatsen wijzigingen aan het aanbrengen zijn, zonder dat die invloed hebben op elkaar. Er is dus heel wat vertraging bij het samenwerken aan hetzelfde bestand.
3. Het vergrendelen van een bestand kan de **valse illusie** wekken dat er niets mis kan gaan. Het kan evenwel gebeuren dat een bestand afhankelijk is van een ander bestand dat niet vergrendeld is en dus wel gewijzigd kan worden door een andere gebruiker. Het uiteindelijke resultaat kan dan toch nog foutief zijn.

1.3.2 Copy-modify-merge (optimistic locking)

Een ander mechanisme is *copy-modify-merge* waarbij de gebruiker een lokale kopie neemt van het bestand en hierin aanpassingen doet. Nadien zal hij het gewijzigde bestand terug proberen te plaatsen in de *repository*. Indien het centrale bestand intussen door iemand anders gewijzigd is, wordt getracht de wijzigingen te versmelten (*merge*). Indien dit niet lukt, moet het conflict opgelost worden door de gebruiker.

Dit systeem is veel flexibeler omdat hier geen bestanden vergrendeld worden en gebruikers tegelijkertijd wijzigingen kunnen aanbrengen. De wijzigingen worden indien nodig automatisch samengevoegd en enkel indien er een conflict optreedt, zal de gebruiker deze samenvoeging manueel moeten doen.

Men noemt dit ook wel *optimistic locking* omdat men er van uit gaat dat er geen conflicten zullen zijn (*optimistisch*) en als ze er toch zijn, moeten ze maar opgelost worden.

GIT heeft geen mogelijkheid tot het vergrendelen van bestanden maar maakt steeds gebruik van deze *copy-modify-merge*.

1.4 Installatie en configuratie van GIT

GIT is een *open source* project dat beschikbaar is op de volgende website <http://git-scm.com>. Daar kan men de versie afhalen overeenkomstig het besturingssysteem. Na installatie kan men GIT gebruiken in een commandovenster.

De meeste IDE's zoals *Eclipse*, *IntelliJ* of *NetBeans* beschikken over grafische mogelijkheden om te werken met GIT. De vormgeving en mogelijkheden verschillen naargelang het product en de versie. In deze cursus zullen we leren werken met GIT via de commandolijn omdat deze benadering het best de werking van GIT uitlegt. Vanuit deze kennis kan men vervolgens makkelijk de grafische mogelijkheden van allerlei andere grafische toepassingen voor GIT begrijpen.

Opdracht 1: Installatie van GIT

In deze opdracht gaan we GIT installeren en configureren op ons systeem. We geven de mogelijkheden voor verschillende besturingssystemen:

- **Windows:** haal GIT af op de volgende website: <https://git-scm.com/downloads>. Kies hierbij de versie overeenkomstig je besturingssysteem. Voer het installatieprogramma uit en kies hier bij telkens de voorgeselecteerde opties.
- **Linux:** installeer GIT met een van volgende commando's:

```
apt-get install git  
yum install git
```
- Open een commandovenster en voer het volgende commando uit:



```
git --version
```

Voor we beginnen met het gebruik van GIT, moeten we een aantal instellingen doen. GIT bewaart instellingen op drie niveaus:

1. **Systeem (system)**: deze instellingen gelden voor alle gebruikers op het systeem. Ze worden opgeslagen in het bestand `/etc/gitconfig`. Bij *Windows* bevindt deze map zich in de installatiemap van GIT.
2. **Globaal (global)**: deze instellingen gelden voor een bepaalde gebruiker. Ze worden opgeslagen in het bestand `.gitconfig` in de *home directory* van de gebruiker.
3. **Lokaal (local)**: deze instellingen gelden voor een specifiek project. Ze worden opgeslagen in de het bestand `.git/config` van het project.

Er bestaat een hiërarchie tussen de instellingen waarbij de instellingen van een lager niveau de instellingen van een hoger niveau kunnen overstemmen. Zo zal een instelling voor de gebruiker (*global*) de overhand krijgen over een systeeminstelling en zal een instelling voor een project (*local*) de overhand krijgen over zowel de gebruiker als het systeem.

Op dit moment moeten we alvast de volgende instellingen doen:

- De **naam** van de gebruiker.
- Het **e-mail-adres** van de gebruiker.

GIT gebruikt namelijk deze informatie bij het bewaren van wijzigingen. Deze instellingen gebeuren dus op het niveau van de gebruiker.

De instellingen van GIT kunnen we wijzigen en opvragen met het volgende commando:

```
git config
```

Dit commando kan gevolgd worden door allerlei opties. Om een volledige uitleg te krijgen over dit commando gebruik je de optie `--help`.

```
git config --help
```

Deze optie `--help` kunnen we overigens ook gebruiken voor alle andere commando's. Zo krijgen we snel de uitgebreide uitleg over deze commando's.

Om een overzicht te krijgen over de huidige instelling gebruiken we:

```
git config --list
```

Hierbij worden de instellingen van alle niveaus weergegeven. Dat is het standaardgedrag.

We kunnen eventueel ook expliciet het niveau aangeven:

```
git config --system --list (systeem)
git config --global --list (globaal)
git config --local --list (lokaal)
```

Hiermee krijgen we enkel de instellingen op dat niveau te zien.

Instellingen kunnen we toevoegen met de optie `--add`. Standaard worden deze toegevoegd aan het lokale niveau tenzij we het expliciet anders bepalen met de optie `--system`, `--global` of `--local` (de standaard).



De naam en het e-mail-adres kunnen we als volgt instellen:

```
git config --global --add user.name "Noel Vaes"  
git config --global --add user.email noel@noelvaes.eu
```

Dit zijn globale instellingen omdat ze specifiek zijn voor de gebruiker.

GIT zal voor het invoeren van boodschappen gebruikmaken van de standaard tekstverwerker. Bij *Windows* is dat bijvoorbeeld *Notepad*. Ontwikkelaars gebruiker vaak de betere variant *Notepad++*.

Indien deze toepassing geïnstalleerd is, kunnen we een extra instelling toevoegen zodat GIT *Notepad++* zal gebruiken:

```
git config --global --add core.editor "'C:\Program Files  
(x86)\Notepad++\notepad++.exe' -multiInst -notabbar -nosession"
```

Opdracht 2: GIT configureren

In deze opdracht gaan we GIT configureren. We zullen onder andere onze eigen naam en e-mail-adres instellen. Tevens verkennen we de instellingen op de verschillende niveaus.

- Open een commandovenster.
- Voer de volgende commando's uit:

```
git config --help  
git config --list  
git config --global --add user.name naam  
git config --global --add user.email e-mail  
git config --list  
git config --system --list  
git config --global --list  
git config --local --list
```
- Optioneel: installeer **Notepad++**.
- Optioneel: Voeg de volgende configuratie toe:

```
git config --global --add core.editor "'C:\Program Files  
(x86)\Notepad++\notepad++.exe' -multiInst -notabbar -nosession"
```

1.5 Lokale repositories

Versiecontrolesystemen maken gebruik van *repositories*. Een *repository* is een opslagplaats waar bestanden en hun verschillende versies bewaard worden. Bij centrale systemen (CVCS) bevindt deze *repository* op een centrale plaats die door alle deelnemers toegankelijk is. De toegang gebeurt hierbij via een of ander netwerkprotocol.

Bij gedistribueerde systemen zoals GIT kan zo'n *repository* zich op verschillende plaatsen bevinden: dit kan op het lokale systeem van de ontwikkelaar zijn maar ook en tegelijkertijd op een of meerdere centraal toegankelijke plaatsen.

In deze paragraaf gaan we eerst werken met een lokale *repository* die zich op ons eigen systeem bevindt. Zodra we de basishandelingen met betrekking tot versiebeheer onder de knie hebben, gaan we zien hoe we gebruik kunnen maken van *remote repositories* en hoe we die met onze lokale *repository* kunnen synchroniseren.

1.5.1 Een repository aanmaken

In tegenstelling tot andere versiecontrolesystemen is het bij GIT gebruikelijk een afzonderlijke *repository* aan te maken voor ieder afzonderlijk project. Er is namelijk ook geen centrale plaats waar alle bestanden van al onze projecten bewaard worden.