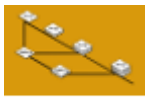
GIT een dom versiebeheerssysteem

Handleiding van Helpmij.nl

Auteur: Erik98

april 2017



Beheer

Wat is nou dat magische versiebeheer? Eigenlijk is de basis heel eenvoudig. Dat hebben we allemaal al wel eens gedaan. De meest eenvoudige vorm is dat je een kopie maakt van een bestand voordat je het gaat veranderen. Zo kun je de veranderingen die je maakt vergelijken met de oude versie en, nog veel belangrijker, je kunt weer opnieuw beginnen vanaf de oude versie als er iets mis gaat.

Versies

Zo'n 40 jaar geleden was dat precies de manier waarop programmeurs aan versiebeheer deden. Op zich werkt dit goed, maar er zijn wel wat beperkingen aan deze methode. Stel dat je bewerking een aantal stappen doorloopt en dat je elke keer na zo'n stap de versie wilt bewaren. Dan heb je op een gegeven moment heel veel versies van hetzelfde bestand die je ergens moet opslaan. Bij programmeren zit je snel aan honderd of meer stappen. Op een gegeven moment wordt het steeds moeilijker om de juiste versie terug te vinden in de brij van bestanden. Aangezien een computerprogramma vaak uit tientallen tot honderden verschillende bestanden bestaat, kun je je voorstellen dat het vrijwel ondoenlijk is om dan nog alles bij te houden. Daarom zijn er programma's ontwikkeld om dat voor je te doen.

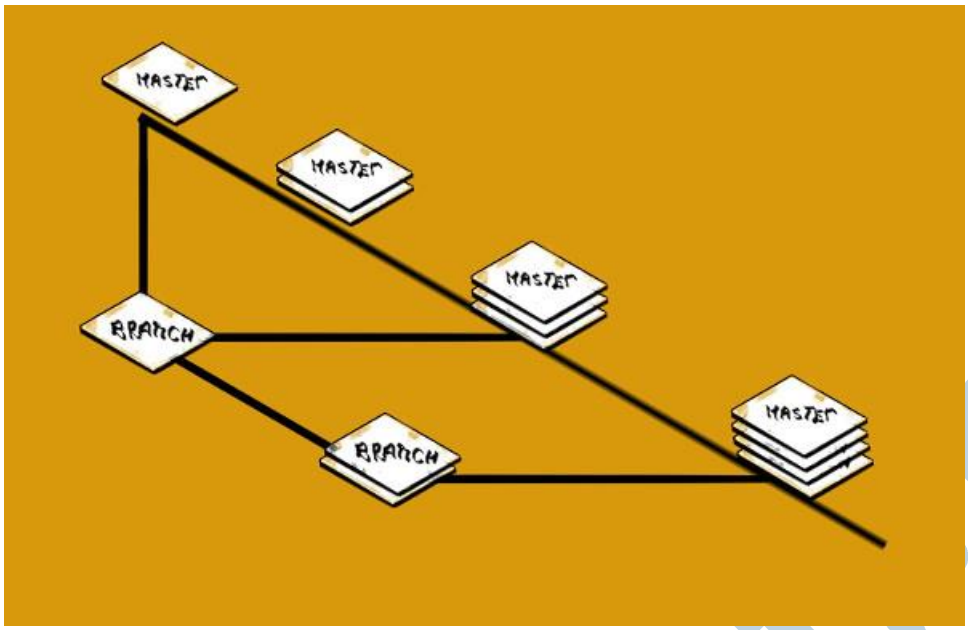
Programma

Een van de eerste versiebeheerprogramma's ('Version Control System' of 'VCS') is RCS (Revision Control System). Het doet eigenlijk precies wat ik hier boven beschrijf, maar dan automatisch. In plaats van steeds een kopie te maken, slaat dit programma alleen op wat er veranderd is tussen de verschillende versies (de 'diffs'). Elke keer dat je een versie wilt opslaan doe je een commit. Je vertelt RCS dat je de huidige versie wilt opslaan. Bij een commit vraagt Revision Control System om een 'annotatie', een korte beschrijving van de versie. Alle veranderingen worden in een speciaal bestand (repository) bijgehouden. Als je een bepaalde versie wilt bekijken doe je een 'checkout' van die versie. Revision Control System volgt dan alle beschreven veranderingen van het begin tot de versie die je wilt hebben en genereert het bestand.

Project

Dit principe wordt nog steeds toegepast in de moderne versiebeheerprogramma's. Revision Control System is nog steeds zeer geschikt om b.v. versies van je configuratie bestanden bij te houden. Het wordt nog steeds veel gebruikt door systeembeheerders. De mogelijkheden om met meerdere mensen aan hetzelfde bestand te werken is echter beperkt. Verder kan Revision Control System alleen individuele bestanden bijhouden en niet de status van een heel project met vele bestanden. Daarom zijn uit Revision Control System vele nieuwe programma's ontwikkeld.

Groot



De bekendste hiervan is Subversion (SVN). Hierbij worden de veranderingen opgeslagen op een centrale server zodat programmeurs over de hele wereld kunnen samenwerken. Subversion is een uitstekende versiebeheerprogramma en wordt nog steeds gebruikt in gevallen waarbij het team van programmeurs niet te groot is. Het is een vorm van centraal versiebeheer. Subversion heeft echter nog steeds zijn beperkingen. Zo gaat Subversion uit van een chronologische ontwikkeling van een programma. Dit zorgde voor problemen voor zeer grote open source projecten zoals de Linux kernel. In deze projecten werken soms duizenden programmeurs parallel aan elkaar. Dat is voor Subversion bijna niet bij te houden.

Parallel

Daarom werd een nieuwe generatie versiebeheerprogramma's ontwikkeld, de gedistribueerde versiecontrole systemen (DVCS). Hiervan zijn Mercurial en vooral GIT de meest bekende varianten. De werking van Mercurial en GIT is vergelijkbaar. De programma's zijn onafhankelijk van elkaar door verschillende teams ontwikkeld. De eerste versies van beide programma' werden vrijwel op hetzelfde moment uitgebracht. GIT lijkt op dit moment het meest populair, hoewel Mercurial ook nog veel gebruikt wordt, o.a. voor de ontwikkeling van de programmeertaal Python.

Video

Erik98: Zeg mke21, ik liep tegen een prachtige video * aan met een interview van Linus Torvalds.
* https://www.linux.com/news/event/open-source-leadership-summit/2017/2/video-linus-torvalds-how-build-successful-open-source-project

De tekst erbij luidde:
De man die Linux maakte, Linus Torvalds, vertelde op het Open Source Leadership Summit van de achtergronden van één van de grootste en meest succesvolle open source projecten in de wereld.

Na 25 jaar ontwikkeling telde de Linux kernel vorig jaar meer dan 22 miljoen regels code en hebben meer dan 5000 ontwikkelaars van ongeveer 500 bedrijven er aan bijgedragen volgens het Linux Kernel Ontwikkelingsrapport van 2016.

Linux 1991-2016
20.000.000 regels software - 5000 man - 500 bedrijven

5000 man, 500 bedrijven. En hoeveel landen dan? Wat een ongelooflijke getallen zijn dat! En dat loopt allemaal via GIT?

mke21: Dat klopt. Ik heb overigens de GIT repo voor linux geanalyseerd en zag dat er voor de laatste twee puntuitgaven (versie 4.8 en 4.9) commits zijn van 1866 verschillende programmeurs. Sinds 2005 zijn er meer dan 20.000 programmeurs geweest die aan de kernel hebben bijgedragen. Echt het grootste software project in de wereld.

GIT

De eerste versie van GIT werd ontwikkeld door Linus Torvald, de man die bekend is als de bedenker van Linux. Hij was op zoek naar een versiebeheersprogramma die geschikt was voor de ontwikkeling van de Linux kernel. Dit is namelijk het grootste software project in de wereld. In de loop der tijd hebben zo'n 21.000 programmeurs bijgedragen aan de kernel. Op het moment van schrijven zijn er zo'n 1800 verschillende actieve ontwikkelaars.

Hij kreeg problemen met de ontwikkelaar van BitKeeper, het programma dat hij tot dan toe gebruikte. Concurrent Versions System **(**CSV) (de voorganger van SVN) was er niet voor geschikt. Met zijn gebruikelijke voortvarendheid schreef hij vervolgens GIT. De eerste versie werd in een recordtijd van 3 weken geschreven. Hij noemde het GIT, wat in de Engelse straattaal 'vervelend persoon' betekent. Hij zei hierover het volgende: "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'GIT'". Er zitten nog meer van dit soort grappen in. Dit staat bijvoorbeeld in de handleiding van GIT:

$ man git

NAME
    git - the stupid content tracker

GIT is dus de "domme inhoudsvolger".

GitHub

Buiten het feit dat GIT gebruikt wordt bij Linux heeft GIT zijn populariteit ook te danken aan GitHub. GitHub is een hosting service waar mensen hun repositories kunnen delen en waar programmeurs met elkaar kunnen samenwerken om opensource software te ontwikkelen. Op GitHub wordt dit uitsluitend met GIT gedaan. De site geeft ontwikkelaars allerlei extra tools. Voor de communicatie met gebruikers biedt GitHub bugtrackers en feature requests. Dit zijn een soort discussie forums waar je een probleem kunt voorleggen of een nieuwe functie kunt aanvragen.

Je kunt vervolgens volgen of en hoe aan je aanvraag wordt voldaan. Verder wordt het programma vaak als zip bestand aangeboden, al zal de installatie niet altijd gemakkelijk zijn. Op GitHub kun je vaak ook documentatie vinden over de programma's en hoe je ze bijvoorbeeld moet installeren. GitHub is dus niet alleen voor programmeurs interessant, maar ook gewoon voor eindgebruikers die graag de laatste versie van een programma willen hebben. Om een voorbeeld te geven, de GitHub pagina van de Linux kernel kun je vinden op https://github.com/torvalds/linux.

Parallelle tijdlijnen

Zoals de naam al aangeeft kan versiebeheer helpen om versies van je bestanden bij te houden. Als zodanig werkt het eigenlijk als een uitgebreide UNDO functie. Alles wat je gedaan hebt kun je weer ongedaan maken en je kunt opnieuw beginnen vanaf een eerdere versie. Versiebeheerprogramma's als GIT kunnen echter veel meer. Ze kunnen namelijk parallelle tijdlijnen ondersteunen. Dit klinkt natuurlijk erg zweverig en science fiction, maar het is eigenlijk een erg logisch en erg nuttig concept.

Artikel

Een mooi en simpel voorbeeld is hoe dit artikel tot stand is gekomen. Normaal gesproken wordt een artikel eerst geschreven, vervolgens kijkt een redactie ernaar en geeft verbeteringen aan, dan gaat het terug naar de schrijver die de verbeteringen doorvoert. Dit proces wordt een aantal keren doorlopen totdat zowel de redactie als de schrijver tevreden zijn. Vervolgens wordt het artikel opgemaakt. In dit artikel over versiebeheer hebben we heel passend gekozen voor een methode met GIT. Ik ben begonnen met dit artikel en heb steeds mijn versies opgeslagen in de hoofdtijdslijn van GIT.

```
erik98@erik98-Lat-xub-1604:~/Documenten/artikel$ git log
commit a84664796cda889b80bc32b0edc8c11ea571164a
Author: mke21
Date:   Tue Feb 21 21:49:29 2017 +0100

    verbeteringen, taal etc.

commit 9ee9dcbe9da434cc25b8f8cca152100728161477
Merge: 70f0ff8 35c729a
Author: Erik98
Date:   Tue Feb 21 21:07:18 2017 +0100

    Merge branch 'redactie'
```

laatste stuk uit de log van GIT over dit artikel (iets vereenvoudigd)

Branch

Een tijdlijn heet in versiebeheerprogramma taal een 'branch' (in het Nederlands 'tak'). De hoofdtijdslijn heet de master branch. Nadat ik een stukje had geschreven en de eerste versie had opgeslagen hebben we een tweede 'redactie' branch gemaakt. Dit is gedeeld met de redactie. De redactie kan daardoor veranderingen aanbrengen, zoals spelfouten verbeteren en koppen en witregels toevoegen en die opslaan in hun eigen branch, zonder dat ik daar wat van merk. Terwijl zij bezig zijn met hun werk kan ik gewoon doorwerken aan de tekst.

Mergen

Als de redactie klaar is met het eerste stuk kan ik hun versie met die van mij vergelijken en vervolgens hun verbeteringen doorvoeren in mijn versie in de master. Dit heet 'mergen'. Dit mergen is een grotendeels automatisch proces en veel eenvoudiger dan met de hand knippen en plakken tussen twee versies van een bestand. De nieuwe versie sla ik vervolgens op in de master. De redactie kan dan weer mijn nieuwe versie met nieuwe tekst naar hun redactie branch brengen en weer verder gaan met de nieuwe tekst te bewerken. Zo hoeven we dus niet op elkaar te wachten en kan het artikel klaar gemaakt worden voor publicatie terwijl het nog geschreven wordt.

Vork (Fork)

Hierboven is een zeer eenvoudige workflow beschreven met behulp van branching. Deze methode is zeer krachtig en wordt door de meeste ontwikkelteams gebruikt. Een voorwaarde voor deze

methode is dat alle ontwikkelaars elkaar kennen en in dezelfde GIT repo kunnen schrijven. Bij opensource ontwikkeling wordt echter iedereen uitgenodigd om een bijdrage te leveren en al die vreemde mensen wil je natuurlijk niet rechtstreeks op je centrale repo laten schrijven. Hiervoor biedt Github de mogelijkheid van forking. Bij forking maakt een programmeur een kopie van de volledige GIT repo voor zijn eigen gebruik. Hij kan dan vervolgens onafhankelijk zelf het programma verder ontwikkelen.

Pull request

Meestal wil zo'n programmeur uiteindelijk dat zijn werk weer terecht komt in de code basis van het het oorspronkelijke programma. Hij kan daarvoor zijn veranderde versie van de code kenbaar maken aan de hoofd ontwikkelaars door middel van een 'pull request'. Dit is niets anders dan dat hij zegt: "Ik heb deze veranderingen gemaakt en je mag het hier ophalen als je het kunt gebruiken". De hoofdontwikkelaars kunnen dan die code naar zich toe halen ("pullen") en dan vervolgens samenvoegen ("mergen") met hun programma code. Soms echter besluit een programmeur om zelfstandig verder te gaan. Zo ontstaat een 'fork' die vervolgens een nieuw programma wordt. Bekende forks zijn bijvoorbeeld Libre Office (een fork van Open Office) en MariaDB (een fork van MySQL).

IJsberg

Dit is slechts het topje van de ijsberg van wat er allemaal met een versiebeheerprogramma gedaan kan worden. Ik hoop dat duidelijk is dat versiebeheerprogramma's een zeer krachtig hulpmiddel is voor allerlei dingen, niet alleen voor programmeren, maar ook voor andere toepassingen als het schrijven van een stuk tekst. Ben je geïnteresseerd dan kun je de interactieve tutorials (in het Engels) op GitHub proberen. Op https://try.github.io/levels/1/challenges/1 kun je de basis van GIT leren. Ga naar http://learngitbranching.js.org/ voor een introductie in branching.

Auteur

mke21 is werkzaam in de Bio-Informatica en is als zodanig professioneel betrokken bij het ontwikkelen van analyse programma's. Hij gebruikt hierbij voornamelijk de talen Python en C. Zowel in zijn werk als thuis gebruikt hij Linux. Daarnaast is hij een enthousiaste fotograaf, waarvoor hij onder andere DarkTable gebruikt.