Documentatie versiebeheer

Module canvas

Aantekeningen Youtube video “What is Git – A quick introduction ‘

* Door middel van versiebeheer kan een project altijd teruggezet worden naar een vroegere versie
* Van een versie van een project kan een branch afgeleid worden waardoor er simultaan gewerkt kan worden aan een project
* Verschillende branches kunnen ‘gemerged’ worden tot een enkele versie/savepoint
* Een (lokaal) versiebeheer systeem kan online gezet worden om het hele project, inclusief alle opgeslagen versies/branches te delen

Aantekeningen Youtube video “Introduction to Git – Core Concepts”

* Een versiebeheer systeem maakt het mogelijk om veranderingen over tijd inzichtelijk te maken
* Met een VCS kan je versies opslaan en terughalen
* Met een VCS kan je in verschillende aftakkingen werken aan het zeflde bestand zonder dat dat van elkaars bestaan weet
* Met een VCS kan je verschillende aftakkingen in elkaar mergen tot een bestand

Git commit graph:

* Snapshot van een repository is een commit
* Een visuele weergave van deze commits heet een git commit graph
* Hierbij is het makkelijk terug te halen welke aanpassingen er aan de repo zijn gemaakt

3 areas of git repository:

* Working Tree, Staging Area / Index, History
* Working Tree:
  + Waar je in bezig bent: toevoegingen, aanpassingen
* History:
  + History is een git commit graph (zie hierboven) van de repository waarop versiebeheer is toegepast
  + History zit in een hidden file .git directory
* Staging Area:
  + Veranderingen die gemaakt zijn in de working tree kunnen (evt selectief) in de staging area gezet worden
  + Vanuit de staging area kan een commit gemaakt worden

Elke commit heeft een unieke sha1 hasj (40 char)

Met git log kun je de commits terugzien

Met git diff kun je de verschillen tussen de laatste commit en de huidige versie (working tree) inzien (van de getrackte files)

Met git diff –staged kun je de verschillen tussen de laatset commit en de huidige staging area inzien

Met git log -p kun je de aanpassingen per commit inzien

Met git -rm kun je een file verwijderen uit de staging area (en ook de working tree)

Een verandering in de working tree ongedaan maken :

* Met git checkout – (filename) wordt een file teruggezet naar de versie die in de staging area is opgeslagen
* De veranderingen kunnen niet teruggehaald worden, omdat ze nooit opgeslagen zijn geweest in het versiebeheersysteem

Een verandering in de staging area ongedaan maken:

* Met git reset HEAD (filename) kun je de file terugzetten naar de versie van de laatste opgeslagen commit (daar slaat HEAD op, zie latere documentatie voor uitleg)
* Na de reset HEAD (filename) is de status van de working tree onveranderd

Een file terughalen uit eerdere commit:

* Met git log –(filename) kun je alle commits zien waar die file toe behoorde
* Met git checkout (commithasj) – (filename) kun je een file terughalen uit een eerdere commit (commit hasj hoeft alleen eerste 5 characters)
* Na het terughalen van een file wordt deze teruggezet in de working tree, evenals in de staging area

Aantekeningen Youtube video “Introduction to Git – Branching and Merging”

Een nieuwe repository maakt een nieuwe branch aan genaamd master branch

Je zit automatisch in de master branch als je in git werkt

Het nut van branches is dat je aan verschillende instanties van dezelfde file kunt werken zonder dat die van elkaar bestaan weten

Branches resulteren in verschillende versies van hetzelfde bestand

Met git branch (branchnaam) maak je een nieuwe branch aan

De branch wordt afgeleid van de laatste commit van de branch waar de HEAD naar wijst

Met git checkout (branchnaam) veranderd de branch waar de HEAD naar wijst

Mergen

Fast-forward Merge:

* Git merge (branchnaam)
* De branch waarin gemerged wordt wordt geupdate naar de laatste commit van de branch waarmee gemerged wordt

Three way merge:

* Met een three way merge wordt er gekeken naar: de laatste commit waar beide branches van afgeleid zijn, en de laatset commit van beide branches die gemerged dienen te worden
* Een three way merge wordt toegepast als er geen eenduidig pad is tussen de laatste commit op de ene branch en de laatste commit op de andere branch

Met git diff (branchnaam)..(branchnaam2) kun je de verschillen zien tussen de laatste commits van de branches

Branches verwijderen:

* Met git branch -d (branchnaam) kun je een branch verwijderen
* Als een branch niet volledig gemerged is geeft git een waarschuwing

Merge conflicts:

* Komen voor als er veranderingen gemaakt zijn aan regels in dezelfde bestanden in twee branches
* Met git merg –abort kan een merge geannuleerd worden (mits de working tree en staging area leeg zijn)

Deteached HEAD state:

* Als je een vorige commit bekijkt (git checkout (commithasj)) staat de HEAD niet op de huidige versie (ie laatste commit van een branch)
* Dit kan handig zijn om branches af te leiden van vorige versies, maar dan moet je niet vergeten om de HEAD op die nieuwe branch te zetten (ie: git checkout (branchnaam))

Git Stash:

* Als je tijdens het aanpassen van een file (ie verandering in working tree, niet in staging area) wil switchen van branch(git checkout (branchnaam)) verlies je de aanpassingen die je gemaakt hebt
* Met git stash kun je de huidige veranderingen tijdelijk opslaan in de stash
* Met git stash list kun je alle huidige stashes zien
* Met git stash -p kun je alle aanpassingen per stash inzien
* Met git stash apply kun je de laatste stash versie overzetten naar de huidige working tree en staging area
* Met git stash apply stash@{<stashnr>} kun je een andere stash versie overzetten