Laborationsrapport

Laboration / Moment 1, Versionshantering & Git
DT173G, Webbutveckling III

Författare: Susanne Wolter, suwo1800@student.miun.se
Termin, år: HT, 2019

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

Sammanfattning

Uppgiften, Moment 1, Versionshantering med Git ingår i kursen Webbutveckling III.

Syftet med uppgiften är att förstå syftet med versionshanteringssystem, kunna använda Git för versionshantering, använda någon molntjänst för att publicera Git-baserade datasamlingar(”repositories”).

Laboration 1 baseras på teoretiska frågor som besvaras i rapporten.

Innehållsförteckning

1 Fråga 1 - Versionshantering

1.1 Förklara kortfattat syftet med versionshantering

Med hjälp av versionshantering kan man återskapa tidigare versioner av dokument, program, webbsidor och källkodsfiler. Man kan även se ändringar som gjorts i filerna, vem som gjort ändringen samt stega tillbaka till tidigare versioner.

Med hjälp av versionshanteringsprogram kan man även skapa/utveckla parallellt vilket är bra när många i projekt är inne och arbetar med samma dokument/kodfiler.

Kortfattat ger versionshanteringssystem möjlighet att synkronisera filer mellan datorer, jämföra olika versioner, man kan backa tillbaka och spåra ändringar och det indikerar var persons bidrag till ett projekt[1][27]

1.2 Beskriv kortfattat tre stycken olika system för versionshantering(”version control systems”) som är användbara för oss som utvecklare. En av dessa ska vara Git, de övriga två är valfria. Beskriv syftet, utvecklare, utvecklingsår, licens (om det är baserat på öppen källkod eller ej) och annat du finner intressant.

1.2.1 Git

Syfte: Initialt skapades Git av Linux skapare Linus Torvalds för att förbättra arbetet med att bygga Linuxkärnan.

Idag används Git flitigt bland system- och webbutvecklare för att kunna spåra källkod under utveckling av mjukvara. Underlättar för användare att återskapa, gå tillbaka i projekt samt att samarbeta i projekt där många utvecklare arbetar med samma kod trots att de ej är uppkopplade till samma nätverk.

Utvecklare: Linus Torvalds

Utvecklingsår: initierades April 2005

Licens: GNU GPL (General Public License), gratis att använda, Git hub finns i betalversion som startar på cirka 61 kronor per månad och utvecklare.Går att köra på lokal / egen server också[2][3].

1.2.2 Mercurial

Syfte: Är ett verktyg för utvecklare som arbetar med mjukvara. Man kallar det för ett distribuerat revisionskontrollverktyg för programutvecklare. Man har arbetat mycket med designen och det innebär hög prestanda och skalbarhet. Programvaran innehåller ett integrerat webbgränssnitt. Mercurial är kommandostyrt men det finns således även grafiska användargränssnitt. Versionshanteringsprogrammet är skrivet i Python och C.

Utvecklare: Matt Mackall

Utvecklingsår: 19 April 2005

Licens: GNU GPL (General Public License), gratis att använda[4][5]

1.2.3 Monotone

Syfte: Framtaget i C++ för att utvecklare ska kunna arbeta antingen genom lokalt repository eller att synkronisera med andras genom ett effektivt nätverksprotokoll, utformat för att kunna arbeta med utan nätverksuppkoppling och då genom det lokala repositoryt. Programvaran arbetar med att lägga fokus på förändringar i filer istället för att se varje fil separat, man kan även här slå ihop olika grenar likt Git. Man kan identifiera olika versioner/revisioner och även vem som gjort vad[6] [7].

Utvecklare: ”The Monotone Team”, Richard Levitte har medverkat i monotone’s senare utveckling 2004

Utvecklingsår: 6 April 2003

Licens: Gratis distribuerat versionshanteringsprogram

2 Fråga 2 – Git

2.1 Förklara följande begrepp:

2.1.1 Workspace

När det kommer till utveckling av mjukvara består ofta ett “workspace” av filer eller bibliotek som tillåter användaren att lagra/samla olika källkodsfiler och källor och låter användaren arbeta med dom som en sammanhängande enhet. Workspace är oerhört användbart när man arbetar med komplexa projekt där underhåll kan vara en utmaning[8].

2.1.2 Staging area

Staging area är en fil i Git directory som innehåller information om vad som kommer att följa med i nästa “commit”. Således arbetar man med filer lokalt i working directory för att sedan “stage files” ladda in filerna i staging area innan dom slutligen blir “commit” till git directory(repository)[9][10].

2.1.3 Local repository

Är det bibliotek där filer sparas lokalt på den egna datorn när man arbetar innan man väljer att pusha filerna till ett remote repository[9][10].

2.1.4 Remote repository

Filer som blivit “commit” från staging area till server och eller molntjänster (filsystem någon annan stans) remote repository, kan med “pull”, hämtas ner till lokal dator för att kunna arbetas vidare på. Detta kan ske enkelt via en “länk” till den version man väljer att ladda ner[9][10].

2.1.5 Branch

Är ett kommando för att skapa en ny gren av ett existerande repository. Originalfilen skulle man kunna kalla stammen i versionshanteringsprogrammet, när man vill göra ändringar skapas sedan en ny “branch” av originalfilerna från master branch[11][12].

2.1.6 Merge

Är ett kommando för att slå ihop nuvarande gren(branch) med en annan som man anger efter “git merge” [namn][11][13].

2.2 Beskriv följande kommandon (vad de gör, hur de fungerar etc):

2.2.1 git config

Är ett kommando för att ställa in globala inställningar. Initialt när installation/nedladdning av Git sker till operativsystemet tillämpas sedan initialt kommandot git config för att exempelvis ställa in user.name och e-post för att git ska veta vem som är användare. Används till setup och konfigurering[14].

2.2.2 git init

Tillämpas för att skapa ett nytt tomt Git repository eller initialisera ett befintligt. Används för att hämta och skapa projekt således[15].

2.2.3 git status

Tillämpas för att visa arbetsträdets status. Kommandot visar de förgreningar och struktur över det som skiljer sig från indexfilen. Git status visar vilka filer som har ändrats och som ännu inter har commitats utan finns i reposition[16].

2.2.4 git add

När man tillämpar git add anger man vilka filer man vill lägga till och gör filerna redo gör att commitas. Här anger man antingen alla ändrade filer med en . eller så anger man filnamnet på den fil som önskas läggas till. Man väljer därmed vilka filer som ska ingå i en och samma commit. Filer läggs till i index[17].

2.2.5 git commit

Med hjälp av git commit uppdateras filerna i repository och man skapar därmed en punkt som man kan återställa ifrån om man senare vill gå tillbaka till en

tidigare version. Man kan här även med git commit -m skriva “ett meddelande som beskriver vad man gjort för ändring/uppdatering/förbättring i denna version”[18].

2.2.6 git push

Laddar upp filerna från lokala repository till remote repository så som exempelvis Github för att det ska bli åtkomligt publikt för andra att ladda ner. Remote filer uppdateras med de nya lokala filerna som läggs upp[19].

2.2.7 git checkout

Låter användaren växla mellan olika branches och återskapa fungerande filer i arbetsträdet. Om man har två branches(grenar) i reposition så kan man växla gren/branch med detta kommando. Om man skapar en branch, hoppar man till den med git checkout, om man sedan till tillbaka till master branch skriver man istället git checkout master[20].

2.2.8 git pull

Används för att kunna hämta hem/ladda ner dom senaste ändringarna från remote repository till lokala repository[21].

2.2.9 git merge

Kommandot tillämpas för att slå samman/ena två eller fler branches till en. Man anger den förgrening som man vill slå samman med den nuvarande. Således kan man ha hoppat mellan branches med checkout för att när man är nöjd med uppdateringen/utvecklingen slå samman de två med merge till en ny version[22].

2.2.10 git fetch

Används när man vill dela och uppdatera objekt. Man laddar ner objekt och filer från en annan repository. Man kan hämta branches eller tags från en eller flera olika repositories[23].

2.2.11 git log

Visar en logg över alla commits som gjort, bra när man vill inspektera och jämföra vad som hänt i ett projekt[24].

2.3 Hur går vi tillväga för att ignorera vissa filer eller kataloger att indexeras med Git?

För att ignorera, hoppa över vissa filer eller kataloger i repository kan ma skapa en fil .gitignore i projektmappen. När man skapat filen skriver man följande i den[25][26].

```
# Notera att brädgård “#” används för att skriva
kommentarer så som att dela upp vad som är en katalog
eller ett dokument som ska ignoreras.

# Ignorera följande kataloger
katalog1/
katalog2/

# Ignorera följande filer
anteckning.txt
privat_bild.jpg
```

3 Webbtjänst för versionshantering

3.1 Skapa ett konto på någon av servrarna Github eller Bitbucket. Skapa ett nytt remote repository och projekt på Github eller Bitbucket, antingen tomt (som bara betsår av en README.md-fil) eller genom att klona ett existerande projekt. Skapa README.md-filen med *markdown* och ge en kort beskrivning av uppgiften/repositoriet(formatera med överskrifter, listor och så vidare)

Vilket segsrdg

3.2 Vilket kommando behöver man använda i terminalen/kommandoprompten för att klona repository till dator?

För att ladda ner/kopiera ett remote repository så tillämpas kommandot *git clone*. Repositoryt laddas då hem från remote repository till en lokal dator.

Inne på Github kan man klivka in på det repository man vill ladda ner och sedan klicka på clone or download remote reository(grön knapp) för att få en länk som går att klistra in det på terminalfönstret eller välja att klicka på ladda ner för att få en zip.

Man kan också öppna terminal och stega fram till den mapp man vill lägga in filen på. Sedan skriver man git clone + adressen och sedan enter så kopieras/klonas filerna till den plats du navigerat till

```
git clone https://github.com/Suwo12/Moment1.git
```

4 Slutsatser

Jag hade tidigare ingen kunskap om versionshanteringsprogram, det har varit oerhört lärorikt då jag nu förstår hur bra verktyg det är. Det var även givande att få inblick i hur många olika versionshanteringsprogram det finns och att git verkar vara ledande på marknaden då det är svårt att hitta information om en del andra versionshanteringsprogram. Dock liknar ju många varandra.

5 Källförteckning

Här följer exempel på hur en källförteckning kan utformas enligt Vancouver-systemet. Den är automatiserad enligt metoden numrerad lista och korsreferenser. Radera denna text, samt ersätt källorna med dina egna.

[1] Wikipedia, ”Version control”
https://en.wikipedia.org/wiki/Version_control
Hämtad 2019-09-04

[2] Git, ”Git fast version control”
https://git-scm.com/
Hämtad 2019-09-04

[3] Wikipedia, ”git”
https://sv.wikipedia.org/wiki/Git_(datorprogram)
Hämtad 2019-09-04

[4] Wikipedia, ”Mercurial”
https://en.wikipedia.org/wiki/Mercurial][https://www.mercurial-scm.org
Hämtad 2019-09-04

[5] Mercurial, ”Mercurial”
https://www.mercurial-scm.org
Hämtad 2019-09-04

[6] Wikipedia, ”Monotone”
https://en.wikipedia.org/wiki/Monotone_(software)
Hämtad 2019-09-04

[7] KTH, ”Monotone”
https://ths.kth.se/news/forelasning-om-monotone
Hämtad 2019-09-04

[8] Wikipedia, ”Workspace”
https://en.wikipedia.org/wiki/Workspace
Hämtad 2019-09-04

[9] Git, ”Getting started Git basics”
https://git-scm.com/book/en/v1/Getting-Started-Git-Basics
Hämtad 2019-09-04

[10] Git, ”Getting started Git basics”
https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F
Hämtad 2019-09-04

[11] Git, ”Branching Branches in a Nutshell”
https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
Hämtad 2019-09-04

[12] Git-scm, ”Branch”
https://git-scm.com/docs/git-branch
Hämtad 2019-09-04

[13] Git-scm, ”Merge”
https://git-scm.com/docs/git-merge
Hämtad 2019-09-04

[14] Git-scm, ”Git Config”
https://git-scm.com/docs/git-config
Hämtad 2019-09-04

[15] Git-scm, ”Git init”
https://git-scm.com/docs/git-init
Hämtad 2019-09-04

[16] Git-scm, ”Git status”
https://git-scm.com/docs/git-status
Hämtad 2019-09-04

[17] Git-scm, ”Git add”
https://git-scm.com/docs/git-add
Hämtad 2019-09-04

[18] Git-scm, ”Git Commit”
https://git-scm.com/docs/git-commit
Hämtad 2019-09-04

[19] Git-scm, ”Git push”
https://git-scm.com/docs/git-push
Hämtad 2019-09-04

[20] Git-scm, ”Git Checkout”
https://git-scm.com/docs/git-checkout
Hämtad 2019-09-04

[21] Git-scm, ”Git pull”
https://git-scm.com/docs/git-pull
Hämtad 2019-09-04

[22] Git-scm, ”Git merge”
https://git-scm.com/docs/git-merge
Hämtad 2019-09-04

[23] Git-scm, ”Git fetch”
https://git-scm.com/docs/git-fetch
Hämtad 2019-09-04

[24] Git-scm, ”Git log”
https://git-scm.com/docs/git-log
Hämtad 2019-09-04

[25] Git-scm, ”Git ignore”
https://git-scm.com/docs/gitignore
Hämtad 2019-09-04

[26] 13pixlar, ”Versionshantering när den är som bäst”
http://13pixlar.se/artiklar/guide-till-git-versionshantering-nar-den-ar-som-bast/
Hämtad 2019-09-04

[27] Föreläsning 1, Git
https://elearn20.miun.se/moodle/pluginfile.php/708552/mod_resource/content/0/git_ht19.mp4
Hämtad 2019-09-04