GIT 101

Simon Alexander Alsing
201304202
aalsing@gmail.com

15. februar 2016

Resumé

Dette er en lille huskeliste der skal bruges i forbindelse med Projektopgavens kode, når den skal versionsstyres over GIT. Ideen er at du kan finde de mest gængse kommandoer. Når jeg støder på nye som kan være relevante vil jeg opdatere guiden. Guiden vil udelukkende beskæftige sig med kommandoer gennem Git BASH, men bruger du GIT GUI vil denne guide kunne give dig en ide om hvordan det hele fungerer.

Indhold

1 Terminal

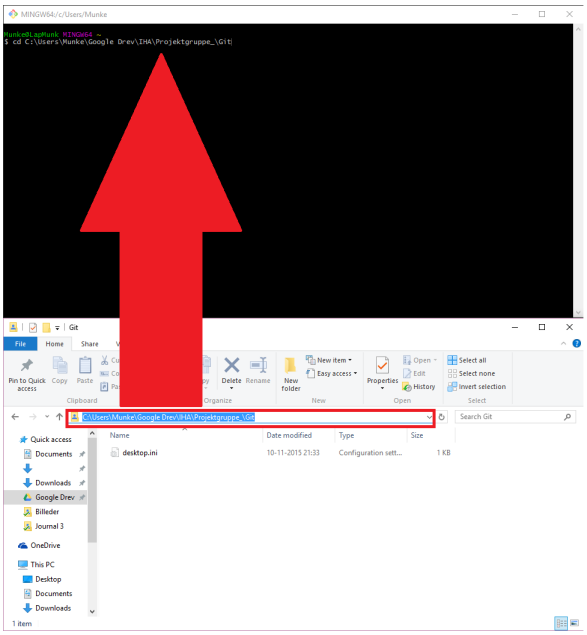
Som nævnt i abstract vil jeg kun bruge GIT BASH i dette dokument, og derfor er der brug for at introducerer terminalen. Kender du allerede til brug af terminal kan du snildt springe ned til afsnit 2. Kender du ikke til det er her nogle hurtige tip.

- Når du bruger terminalen er det vigtigste at forstå, at du navigerer på samme måde som du ellers gør. Når du åbner et dokument du skriver noter i, vil du typisk have dette gemt i en mappe. Når du skal åbne filen navigerer du til denne mappe fra Dokumenter til EnMappe til EnAndenMappe, til du sidst kommer ind til mappen som indeholder dine noter.

  På samme måde gør du i terminal, men her er det mere explicit. Når du bruger git bash, bruger du '/' til at skelne mellem hver mappe, så en stil ville være:

  C:/Dokumenter/EnMappe/EnAndenMappe/

  Stien er den samme, men du fortæller systemet præcis hvordan den finder frem til det.



Figur 1: skifte til korrekt sti, the super easy way

- Du skifter sti med kommandoen cd, som står for 'change directory'. Vi kan altså skifte ind i mappen "EnAndenMappe"ved følgende kommando:

  cd C:/Dokumenter/EnMappe/EnAndenMappe/

Men husk nu på at stierne er Case sesitive, så skriver du med småt, når mappen er med stort så kommer du ikke det rigtige sted hen.

- Nu ved du hvordan man skifter sti, men når du er i gang med at skifte sti - som altså er case-sestive, så er det dejligt hvis nu man kunne få nogle shortcuts til at skrive de lange stier. Her kan du bruge Tap, som altså hjælper med autocomplete når du har skrevet et par af bogstaverne i hvert mappenavn - meget brugbart hvis man skal skrive stien selv

- Er du bange for at skrive stien, kan du åbne mappen du skal finde frem til som du normalt ville. Herfra trykker du op i adresselinjen, og kopierer stien - se figur 1

  Husk på at du ikke kan paste i en terminal, så her bruges højreklik, paste i stedet. Dumme windows..

Mangler der ting til listen så skriv til mig, den vil løbende blive opdateret

2 Begynd et nyt git projekt

Start med at åbne Git BASH og skift stien til mappen som du ønsker git skal gemme dit projekt i. Dette gøres med kommandoen cd, se afsnit 1. Du skal nu fortælle Git at du vil klone mappen med et eksisterende git projekt - dettes gøres med kommandoen:

```
git clone URL
```

Hvor URL erstattes med den url til det pågældende repository.
Tillykke du har nu projektet!

3 Everyday Commands

3.1 Intro

For at udføre et push, dvs dele dine ændringer med holdets nuværende version af koden, har man en fuldstændig fast rækkefølge tingene skal gøres i. Først skal du tilføje de filer som er ændret og du ønker skal med i den version vi alle deler - her skal du huske IKKE at medtage filer der ligger i Debug mappen, da denne kun indeholder compilede versioner af koden og kan variere fra computer til computer(skaber problemer - hertil bruges en .gitignore se **??**).

3.2 Status

Du bruger status kommandoen til at se hvilke ting du har ændret siden sidste PULL, altså siden sidst up har opdateret din version med den nyeste version på github. Kommandoen skrives:

```
git status
```

Status kan fortælle dig om du mangler at up-/downloadloade filer til projektet, men som oplevet kan den fejle - brug derfor altid pull før du stater projektet se 3.6

3.3 Add

Når du har oprettet en ny fil, som skal bruges i projektet, skal denne tilføjes ved hjælp af add kommandoen. Når du bruger add kommandoen, fortæller du git at den nu skal "følge med"i hvordan filen ændrer sig og samtidig at den er en del af projektet. Lad os sige du skal tilføje filen "octocat.png", så er kommandoen:

```
git add octocat.png
```

Havde du mange filer som har samme filtype, kan du bruge stjerne:

```
git add *.png
```

Hermed får du alle de elementer som har filendelsen .png - men det er altså ligegyldigt om de hedder octocat eller carsten.

3.4 Commit

Commit er her hvor du fortæller git hvad du har ændret - i eksemplet fra før havde vi tilføjet nogle billeder, så din besked skal altså beskrive det:

```
git commit -m "tilføjet billeder til ...."
```

husk -m parametren, den fortæller at du skriver commit beskeden i terminalen - ellers kommer du ind i vim og skal bruge vim kommandoer for at navigere.

3.5 Push

Når der skal pushes, altså når du skal give dine ændringer i projektet til de andre, bruges kommandoen:

```
git push -u origin master
```

Læg mærke til buzzword master - det er altså branchen som vi henviser til, skal du pushe til en anden branch, ændres dette til hvad branchen hedder. Se evt kap om branches.

3.6 Pull

Pull henter ændringer ned for en branch.

4 Branches

Med branching skal du tænke på et træ, stammen i et gitprojekt er Master branchen - alt brancher ud fra denne(i hvert fald i vores projekt). Når du er på masterbranchen kan du kun se hvad den indeholder - intet andet! Når du skifter branch vil du kunne se hvad Master indeholder, samt den nye branchs indhold!

4.1 Skift branch

Du skifter branch ved kommandoen:

```
git checkout branch
```

hvor branch erstattes med hvad branchen hedder, hvis du ikke er sikker på hvilke du har adgang til se 4.2

4.2 List branches

Hvis du vil have dig et overblik over hvilke branches der eksisterer bruges kommandoen

```
git branch -a
```

Finder du mod forventning ikke de branches du leder efter, se 4.3

4.3 Opdater listen med branches

Hvis ikke alle branches kommer frem er det garanteret fordi du ikke har opdateret din oversigt over branches, dette gøres ved:

```
git remote update
```

Herefter burde alle virke.

4.4 Ny branch

Du laver en ny branch ved kommandoen:

```
git checkout -b branch
```

så ligesom at skifte branch, bare med parametren '-b' som indikerer at du laver en ny. branch argumentet skiftes selvfølgelig til det som man vil kalde branchen.

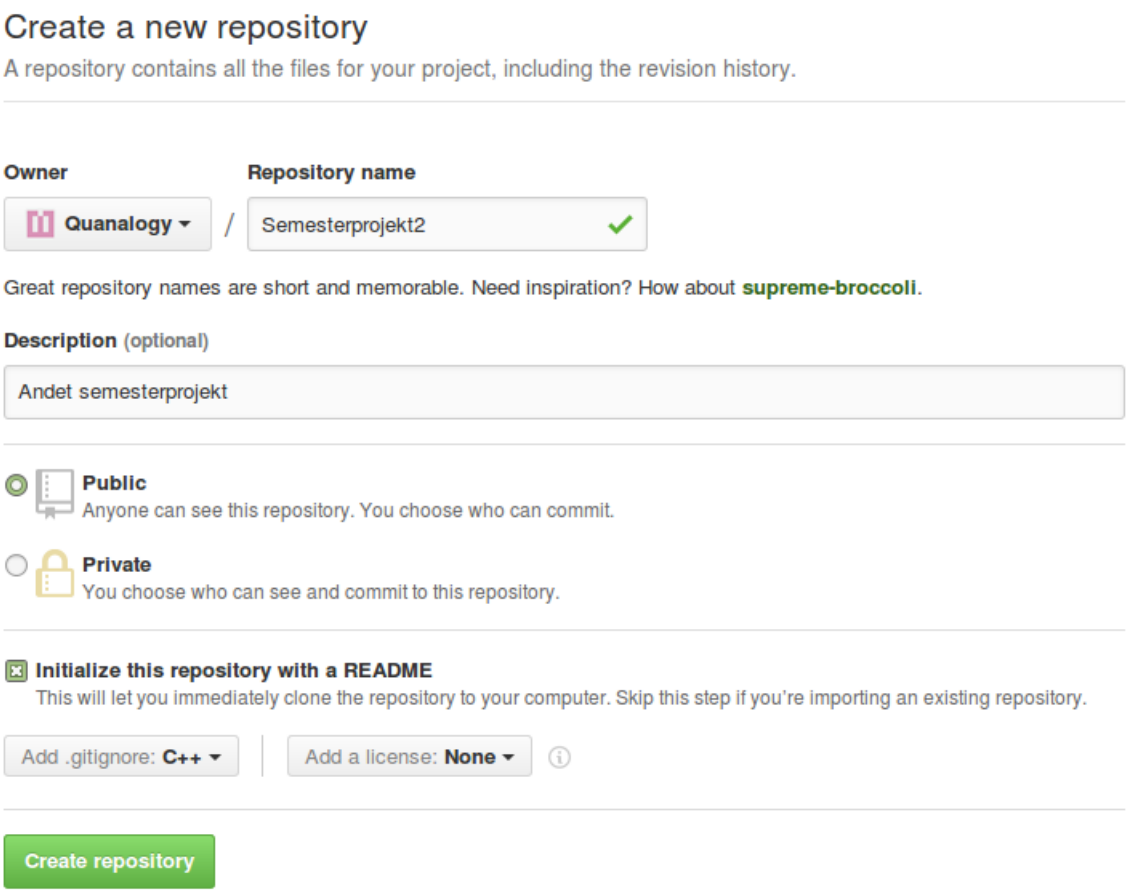
5 Git Ignore

En git ignore fil kan være smart hvis man arbejder i projekter som består af mapper som ikke skal medtages. Dette vil typisk være debug mapper. Der findes mange online gitignore filer som man kan hente, der er tilpasset til forskellige projekter. Selvom de ofte vil få jobbet gjort kan det være nødvendigt at skrive dele selv.

Når der laves en gitignore ville denne typisk laves i starten af projektet. Laves projektet på github har du muligheden for at vælge en tilpasset gitignore fil, som passer til sproget du skriver projektet i - for os typisk `C++`. På figur 2 ser du vinduet i github som laver et nyt projekt eller repository som det kaldes. Læg mærke til den nederste del, der er valgt en gitignore til `C++` og der laves en readme. Readme er en beskrivelse af projektet - bruges typisk i opensource projekter til at give en hurtig beskrivelse af hvad projektet er og hvordan man evt bruger det / installerer det. Be .gitignore og README er noget man typisk vil inkludere i alle projekter.

Helt generelt er en git ignore fil, en fil uden navn med filendelsen '.gitignore'. Filen laves ved at åbne git bash og navigér til mappen med projektet i (se afsnit 1). Herefter bruger vi kommandoen touch til at lave filen:

```
"touch .gitignore"
```



Figur 2: Et nyt repository på git

Filen åbnes en en text editor som IKKE bruger encoding(dvs ikke word), det åbnes derfor i `notepad`,`notepad++` og lignende. I filen skrives stien til en fil, mappe, eller hele filendelser/navne. Her er et eksempel fra den fil som github laver til dig for et `C++` projekt:

```
    ### Compiled Object files
*.slo
*.lo
*.o
*.obj

    # Precompiled Headers
*.gch
*.pch

    # Compiled Dynamic libraries
*.so
*.dylib
*.dll

    # Fortran module files
*.mod

    # Compiled Static libraries
*.lai
*.la
*.a
*.lib

    # Executables
*.exe
*.out
*.app!
```