DAT290 Datatekniskt projekt

Introduktion till Git

Martin Löfgren

Version 1 – 2018-08-30

Innehåll

1 Inledning

Detta dokument är en introduktion till versionshanteringssystemet Git. Under övningen ska ni skapa ett Git-arkiv som ni sedan skall använda er av i projektet. När ni skapar arkivet så ska ni använda er av katalogstrukturen som detta dokument finns i, dvs. innehållet i `dat290.zip`. Git-delen av övningen görs gemensamt i hela projektgruppen, när ni sedan går över till LaTeX-delen så ska ni dela upp er i grupper om två. Det är då viktigt att alla arbetar mot samma fjärrarkiv[1].

Detta dokument innehåller en hel del listningar från terminalen/Bash-git. Ibland ryms inte all text på en rad, vilket markeras med symbolen ↪ . När ni skriver in motsvarande kommandon eller ser utdatan så kommer då denna symbol *inte* att finnas med.

2 Konfigurera Git

Innan vi börjar använda Git är det viktigt att göra viss konfigurering:

```
$ git config --global user.name "Emilia Emilsson"
$ git config --global user.email "emiemil@student.chalmers.se"
```

Dessa uppgifter kommer att synas vid incheckningar (beskrivs i avsnitt 3.5) och är viktigt för att kunna identifiera vem som har gjort vad i kodbasen.

3 Git lokalt

I denna första del kommer ni att börja arbeta med Git lokalt, på den egna eller på skolans dator. Ni kommer att initiera ett arkiv och lägga till filer till det.

3.1 Initiering

För att börja använda Git i ett projekt behöver det initieras som ett Git-arkiv. För att åstadkomma detta så gå till roten av projektet och ange kommandot

```
$ git init
```

Katalogen `.git` har nu skapats och Git kan börja användas för att versionshantera projektet.

3.2 Status

Såhär långt har vi ett projekt med en uppsättning kataloger och filer och vi har ett Git-arkiv. När vi initierade arkivet fick vi som svar att det är tomt. Vi kan när som helst se skillnaden mellan vår aktuella *arbetskatalog* och vårt arkiv med kommandot `git status`:

[1] Vad det innebär kommer förhoppningsvis att vara tydligt vid slutet av övningen!

```
$ git status
På grenen master

Första incheckning

Ospårade filer:
(använd "git add <fil>..." för att ta med i det som skall
    ↪ checkas in)

README.md
dokumentation/
intro/
kod/

inget köat för incheckning, men ospårade filer finns (spåra
    ↪ med "git add")
```

Vi ser här att vi har en fil och tre kataloger i arbetskatalogen som inte finns i arkivet. Vi måste hela tiden välja *vilka* filer vi vill lägga till arkivet och *när* vi vill lägga till dem.

3.3 Lägga till filer

För att lägga till de saknade filerna till arkivet använder vi oss av kommandot `git add`. Kommandot tar en eller flera filer som argument. För att lägga till samtliga filer och kataloger kan vi ange

```
$ git add .
```

dvs. lägg till den nuvarande katalogen, vilket kommer att inkludera samtliga filer och kataloger som finns i den. Efter att ha lagt till filer till arkivet behöver vi *checka in* dem. Detta görs med kommandot `git commit`. Innan vi checkar in filerna är det dock alltid en bra idé att kontrollera så att vi inte har lagt med några filer som vi inte vill versionshantera:

```
$ git status
På grenen master

Första incheckning

Ändringar att checka in:
(använd "git rm --cached <fil>..." för att ta bort från kö)

ny fil:        README.md
ny fil:        dokumentation/kallelser/.gitkeep
ny fil:        dokumentation/projektplan/.gitkeep
ny fil:        dokumentation/projektrapport/.gitkeep
ny fil:        dokumentation/protokoll/.gitkeep
ny fil:        intro/IEEEtran.bst
ny fil:        intro/README.md
ny fil:        intro/figurer/tidsplan.pdf
ny fil:        intro/git-intro.pdf
ny fil:        intro/latex-intro.pdf
ny fil:        intro/projektplan-mall.pdf
ny fil:        intro/projektplan-mall.tex
ny fil:        intro/referenser.bib
ny fil:        kod/.gitkeep
```

Vi ser här att samtliga filer i projektet har lagts till. Vi vill dock enbart hålla sådana filer under versionshantering som utgör källfiler av något slag, dvs inga kompilerade filer. När vi skriver dokumentation med LaTeX – som nästa del av övningen handlar om – vill vi *inte* inkludera de renderade `.pdf`-filerna.

3.4 Undanta filer

För att ta bort de oönskade filerna från incheckningskön följer vi anvisningen som ges i statusmeddelandet:

```
$ git rm --cached intro/*.pdf
rm 'intro/git-intro.pdf'
rm 'intro/latex-intro.pdf'
rm 'intro/projektplan-mall.pdf'
```

En ny `git status` visar att filerna nu inte ingår bland filerna som skall checkas in. `intro/figurer/tidsplan.pdf` vill vi behålla, eftersom den utgör en figur som används i dokument och sålunda är att betrakta som en källfil.

För att slippa att manuellt hantera vilka filer som skall ingå i versionshanteringen kan vi skriva regler för vilka filer som skall undantas från att ingå i arkivet. Dessa regler sparas i filen `.gitignore`. Denna kan finnas i projektroten eller någon underkatalog och kommer att gälla rekursivt för samtliga underkataloger till den där filen finns. Skapa en ny fil `.gitignore` i rotkatalogen med följande innehåll:

```
*.pdf
!**/figurer/*.pdf
```

Den första raden säger här att vi vill undantag alla .pdf-filer från att ingå i versionshanteringen. Den andra raden säger att vi vill göra ett undantag från

den första regeln: ! innebär att vi *vill* ta med filer som följer mönstret i arkivet. `**/figurer/*.pdf` matchar alla .pdf-filer som finns i en katalog med namn `figurer`, oavsett var i katalogstrukturen katalogen befinner sig.

En förnyad `git status` bör nu visa attt de två .pdf-filerna från ovan inte listas som ospårade filer. Däremot finns `.gitignore` med som ospårad fil. Denn fil bör ingå i versionshanteringen, så lägg till den till listan över filer med `git add .gitignore`. I det fortsatta arbetet så är det viktigt att löpande granska vad som läggs till arkivet och uppdatera `.gitignore` så att oönskade filer inte läggs till arkivet.

Kanske undrar någon över filerna `.gitkeep` som återfinns på olika platser i katalogstrukturen. Dessa finns med för att de i övrigt tomma katalogerna ska tas med i arkivet. Tomma kataloger inkluderas annars *inte*. Filnamnet `.gitkeep` är en konvention som förmedlar just detta: behåll denna katalog!

3.5 Incheckning

Vi är nu redo att göra vår första incheckning. Med varje incheckning skall anges ett meddelande som beskriver incheckningen. Meddelandet kan vara en rad om det räcker för att beskriva ändringarna, men det kan också bestå av många rader. [1] ger några råd om hur incheckningsmeddelanden kan skrivas. Om meddelandet består av en rad kan det anges direkt på kommandoprompten:

```
$ git commit -m "Detta är incheckningsmeddelandet"
 12 files changed, 2984 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 dokumentation/kallelser/.gitkeep
 create mode 100644 dokumentation/projektplan/.gitkeep
 create mode 100644 dokumentation/projektrapport/.gitkeep
 create mode 100644 dokumentation/protokoll/.gitkeep
 create mode 100644 intro/IEEEtran.bst
 create mode 100644 intro/README.md
 create mode 100644 intro/figurer/tidsplan.pdf
 create mode 100644 intro/projektplan-mall.tex
 create mode 100644 intro/referenser.bib
 create mode 100644 kod/.gitkeep
```

4 Fjärrarkiv

Git är ett kraftfullt verktyg då det endast används för ett lokalt arkiv. Det går att följa historiken, gå bakåt till en tidigare version, dela upp historiken i grenar och slå samman dessa igen. En stor del av styrkan är dock möjligheten att arbeta tillsammans och synka sitt lokala arkiv mot ett fjärrarkiv.

4.1 Skapa ett konto

För att kunna arbeta mot ett fjärrarkiv behöver vi först skapa ett konto på någon tjänst som erbjuder detta. I den följande framställningen kommer Github att användas, men det går lika bra med någon annan motsvarande tjänst.

Under denna och andra kurser där ett fjärrarkiv används är det viktigt att det är privat så att andra studenter inte kommer åt er kod. På Github är det förenat med en kostnad, men de erbjuder ett studentabonnemang med kostnadsfri tillgång till privata arkiv. Gå till `https://education.github.com/pack` och registrera ett konto eller lägg till dig som student på ditt befintliga Github-konto.

4.2 Skapa fjärrarkiv

Det har nu blivit dags att skapa ett fjärrarkiv. På förstasidan (`github.com`) finns en knapp där det står New `repository`. Fyll i ett namn på arkivet och eventuellt en beskrivning. Klicka i `Private` och sedan `Create repository`.

4.3 Ställ in fjärrarkiv och *pusha*

Vi har nu ett lokalt arkiv och ett fjärrarkiv. För att synka fjärrarkivet mot det lokala arkivet följer vi anvisningarna på Github:

```
$ git remote add origin https://github.com/emiliaemilsson/ett-
    ↪ arkiv.git
$ git push -u origin master
Username for 'https://github.com': emiliaemilsson
Password for 'https://emiliaemilsson@github.com':
Räknar objekt: 14, klart.
Delta compression using up to 4 threads.
Komprimerar objekt: 100% (10/10), klart.
Skriver objekt: 100% (14/14), 68.59 KiB | 0 bytes/s, klart.
Total 14 (delta 0), reused 0 (delta 0)
To https://github.com/emiliaemilsson/test.git
* [new branch]      master -> master
Grenen master ställdes in att spåra fjärrgrenen master från
    ↪ origin.
```

Den första raden registrerar arkivet på Github som ett fjärrarkiv. Den andra raden *pushar* det lokala arkivet till fjärrarkivet, dvs skriver alla ändringar som har gjorts på det lokala arkivet till fjärrarkivet. Flaggan `-u` kopplar samman `origin` – alltså fjärrarkivet – med grenen `master` på det lokala arkivet. Denna inställning behöver bara göras då en lokal gren skall *pushas* för första gången; fortsättningsvis räcker det med `git push`.

Det är även möjligt att använda sig av `ssh` istället för `https` när man ställer in fjärrarkivet. Formen blir då istället `git remote add git@github.com:emiliaemilsson/ett-arkiv.git`. Fördelen med ssh är att man slipper ange sitt användarnamn vid varje kommando som arbetar mot fjärrarkivet. Se [2] för mer information.

4.4 När fjärrarkivet ligger före

I fallet ovan var det lokala arkivet en incheckning *före* fjärrarkivet innan vi genomförde `git push`. Om någon annan har arbetat mot samma fjärrarkiv och *pushat* något hamnar vi i den omvända situationen: att fjärrarkivet ligger före vårt lokala arkiv. För att synkronisera kan vi då använda `git pull`.

4.5 Klona ett arkiv

Om man *inte* har ett lokalt arkiv, men vill skapa ett sådant utifrån ett fjärrarkiv kan man *klona* det:

```
$ git clone https://github.com/emiliaemilsson/ett-arkiv.git
Klonar till "ett-arkiv"...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 14 (delta 0), reused 14 (delta 0), pack-reused 0
Packar upp objekt: 100% (14/14), klart.
```

För att hitta URL:en till ett arkiv på Github kan man klicka på knappen `Clone or download`. Det går där att välja om man vill använda `https` eller `ssh`.

5 Avslutning

Denna övning beskriver några grundläggande funktioner med Git. Git är ett kraftfullt – och måhända lite krångligt – verktyg. Det finns flera andra kommandon som vi inte har berört här, och otaliga varianter av dessa. Några av de viktigaste är:

- mv, rm, `reset`: Påverka förhållandet mellan arbetskatalogen och aktuell incheckning.
- `log`, `show`: Undersök historiken och vad olika incheckningar består i.
- `branch`, `checkout`: Arbeta med *grenar*.

För att lära sig mer är det bra att gå direkt till källan: Git innehåller en omfattande dokumentation som nås med `git help`. Den innehåller dels dokumentation gällande enskilda kommandon (t.ex. `git help rm`) men också mer vägledande texter, t.ex. `git help tutorial`.

En annan omfattande källa för den som vill lära sig mer är [3] som finns att läsa i sin helhet online utan kostnad.

Referenser

[1] C. Beams, "How to write a git commit message," 2014, [Online; hämtad 19/8 2018]. [Online]. Available: https://chris.beams.io/posts/git-commit/

[2] GitHub Inc., "Connecting to github with ssh," [Online, hämtad 23/8 2018]. [Online]. Available: https://help.github.com/articles/connecting-to-github-with-ssh/

[3] S. Chacon and B. Straub, *Pro Git*. Apress, 2018, [Online; hämtad 23/8 2018]. [Online]. Available: https://git-scm.com/book/en/v2