Workshop 1 : Versionhantering med Git

Om du läser lite om Git innan detta så förstår du bättre. Se länksamling på kursida för några förslag.

Det vi skall åstadkomma är ett centralt s.k. repository (repo = magasin) där projektets gemensamma kod och dokumentation skall lagras (behöver alltså inte skicka mellan gruppmedlemmar). Repositoriet håller reda på alla förändringar (versioner) av allt. Man kan t.ex. ta fram äldre versioner av den kod man har.

Git behöver ingen särskild server. Det räcker om man kan komma åt någon Linuxmaskin (konto) med ssh (m.fl alternativ). När man arbetar med projektet är flödet följande:

- Varje gång man skall börja arbeta hämtar man alla förändringer från det centrala repot till ett lokalt (så att man har den senaste versionen).
- Man arbetar med kod och andra dokument. Då man är klar med något moment sparar men förändringarna i de lokala repot.
- När man är klar för denna gång skickar man upp alla förändringar man gjort i det lokala repot till det centrala (så att andra sedan kan hämta dessa till sina lokal repon).

1 Skapa ett centralt repo

Låt någon gruppmedlem göra följande (allt görs via en terminal). Logga in som vanligt.

1. Gå till grupparea.

   ```
   $ cd /chalmers/groups/ooproj-NN  (NN är ditt gruppnummer, 01-25)
   ```

2. Skapa en mapp med namnet på projektet följt av .git (t.ex. mp.git, använder detta som ett exempel i fortsättningen, ni väljer ett eget namn).

   ```
   $ mkdir mp.git
   ```

   Gå in i mappen

   ```
   $ cd mp.git
   ```

   Initiera mappen så att den kan fungera som ett s.k. ”bare” repository.

   ```
   $ git --bare init
   ```

   Ett antal mappar och filer skapas (dock kan man inte inspektera kod i ett bare repo).

2 Skapa ett första lokalt repo

Låt samma person fortsätta.

1. Lämna grupparean och gå till din hemkatalog. Välj ett bra ställe där det lokala repot skall ligga. Gå dit.

2. Skapa en mapp för det lokala repot (samma namn som centrala men med .src på slutet);

   ```
   $ mkdir mp.src
   ```

3. Gå till in i mappen

   ```
   $ cd mp.src
   ```

4. Initiera mappen så att den kan fungera som ett lokalt git-repo.

   ```
   $ git init
   ```

5. Lägg till det centrala repot som ett s.k. remote. D.v.s. vi skapar en koppling mellan det centrala och det lokala (så att det lokala vet vart det skall skicka (eller hämta) förändringar). Obs! En enda lång rad (cid är ditt login, MM kan vara 11-15).

   ```
   $ git remote add origin cid@remoteMM.chalmers.se:
             /chalmers/groups/ooproj-NN/mp.git
   ```

   Lokalt kan vi i fortsättningen referera till det centrala som ”origin”. Git behöver ingen kontinuerlig kontakt med det centrala repot. Allt sparas lokalt och kan senare skickas till det centrala.

6. Kontrollera status

   ```
   $ git status
   ```

   Git säger ungefär

   ```
   # On branch master
   # ... o.s.v.
   ```

   Git kan jobba med flera olika versioner parallellt (branches). Förvalt skapas automatisk en lokal branch som heter ”master”. Mer om branches nedan.

Låt terminalen vara öppen.

3 Ett initial lokalt projekt

Samma person fortsätter. Skapa ett projekt som skall versionhanteras.

1. Start Eclipse och skapa ett vanligt Java-projekt.

   a) Välj mp.src som katalog för projektet.
   b) Skapa bara en klass Main med main-metoden (tom så länge).
   c) Spara.

2. Gå till baks till terminalen och kör git status (i mp.src). Git säger att det finns "Untracked files" d.v.s. dessa filer hanteras inte av Git (ingår inte i repot). Vi skall strax lägga till dessa men först
   - För att sortera bort sådant som vi inte vill versionhantera (.class, jar-filer, bilder, o.s.v) så skall vi ha en fil .gitignore i katalogen. Du kan hämta en sådan från kursidan (inspektera).
3. Då skall vi lägga till alla filer i projektet. Skriv (lägg till allt);

   ```
   $ git add .      (obs! punkten)
   ```

4. Kör git status igen. Git rapporterar om ett antal nya filer (som kommer att läggas till).
   - OBS! De är inte slutgiltigt inskickade till Git ännu (inte commit:ade på svengelska).
5. Då skall vi slutgiltigt lägga in filerna i Git. Skriv (man måste skriva ett meddelande om vad man skickar in, skall stå efter -m (message));

   ```
   $ git commit -m ''Intital commit''
   ```

   Git säger en del saker. Kör git status. Skall svara "nothing to commit" (vi har ju skickat in allt så inga förändringar finns).
6. Gå till baks till Eclipse projektet. Lägg till i main så att "Hello world" skrivs ut. Spara.
7. Kör git status. Skall säga ett något är modifierat.
8. Vi skickar in modifieringen genom att kombinera add (-a) och commit (kolla med status efteråt).

   ```
   $ git commit -a -m ''Modified main''
   ```

4 Historik

Ofta vill man se vad som har hänt.

1. Test: git log
2. Det finns två fristående program man kan använda (i alla fall i Linux)
   a) Testa att skriva (i terminalen): gitg (titta under fliken Changes)
   b) Testa: gitk

5 Skicka intiala lokala till centrala

Samma person fortsätter. Vi skall nu skicka det lokala projektet till det centrala repot. Görs med kommandot ”push”.

1. Skriv (origin är ju namnet på det centrala och master är den lokala versionen).

   ```
   $ git push origin master
   ```

   I fortsättningen gör man alltid push:en då man avslutar för denna gång (där emellan commit:ar man då och då).

2. Eftersom man inte kan inspektera koden i mp.git (centrala är som sagt ett bare repo) kan man skapa en inspekterbar ”klon” av detta. Gå till mappen där mp.git finns. Skriv

   ```
   $ git clone mp.git mp.check
   ```

   Det skapas en mapp mp.check. Gå in i denna och leta rätt på koden ni skickade in.

6 Skapa flera lokala repon

Personen som har gjort allt ovan är nu klar med sin ”setup”. Nu skall övriga medlemmar skapa lokala repon. D.v.s. man loggar in på sina konton och;

1. Väljer ut var ni vill att det lokala repot skall hamna (mappen som skall innehålla mp.src). Gå till denna och skriv;

   ```
   $ git clone hajo@remoteMM.chalmers.se:

       /chalmers/groups/ooproj-NN/mp.git mp.src   (mellanslag före mp.src)
   ```

2. Skall nu finnas mapp mp.src. Det har alltså skapats ett nytt lokalt repo. Inspektera repot, kör git status.

3. Skall kunna importera projektet till Eclipse (File > Import).

7 Push och pull

När alla har skapat sina lokala repon skall vi testa att det funkar att dela mellan alla i gruppen.

1. Var och en kan fortsätta på det påbörjade Eclipse-projektet. Lägg till varsin klass, commit:a och push:a.

2. När alla är klara så gör alla följande;

   ```
   $ git pull origin master
   ```

3. Inspektera! Alla skall nu ha den nya koden.

Då man inleder sin session kör man alltid git pull innan man påbörjar arbetet.

8 EGit

Detta är alltså optional. Man kan klara hela projektet med det ovan men eventuellt kan man tycka att det är smidigare att använda en Eclipse-plugin, EGit (jag kör dock push och pull från kommandoraden, EGit är för rörig, ointuitivt).

1. Om du redan har ett Eclipse-projekt som heter mp.src måste du ta bort detta (kan inte ha två projekt med samma namn)
2. Öppna EGit repo-fönstret: Window > Show View > Other... > Git Repositories
3. Klicka lilla, lilla ikonen med ett pyttelitet grönt plus.
4. Brows till mp.src, välj.
5. Klicka Seach och Ok. Det dyker upp ett repo i Git Repositories fönstret.
6. Markera ikonen för Git repot (gul cylinder i fönstret). Högerklicka > Import Project...
7. Next > Finish.
8. Ett projekt dyker upp. Projektet har en liten gul cylinder d.v.s. Git hanterar.
9. Alla filer som förändras markeras med en > framför. Då man commit:ar (högerklick > Team > Commit..) försvinner dessa.

Att hantera ”branches” i EGit är mycket smidigt, se nedan.

9 Branches

Detta är också optional. Ofta vill man pröva lite olika saker med koden men man vill ha kvar den gamla koden ifall ändringen inte var bra. Mycket omständligt utan hjälpmedel. Med Git kan man enkelt åstadkomma detta enligt:

- Man skapar en ny branch (gren) med ett valfritt namn. Vi har alltså två brancher, den nya och master.
- Den nya grenen innehåller inledningsvis samma kod som master.
- Man gör alla förändringar i den nya grenen.
- När (om) man är nöjd slår man ihop den nya med master. Är man inte nöjd så går man tillbaks och fortsätter med master (eller skapar en annan gren).

Lokala braches: Här handlar det om lokala grenar. Grenarna finns inte i centrala repot (man kan grena detta också)

De som vill (alla) kan testa detta;

1. Gå till mp.src. Skriv;

   ```
   $ git branch test     (skapa)
   $ git checkout test   (byt)
   ```

   Vi har skapat en gren ”test” och bytt til denna. Kör git status (säger vilken branch vi är på).

2. Ändra en del i koden. Kör git status. Kör commit.

3. Inspektera koden i terminalen (Tips: kommandot less t.ex. $ less src/test/Main.java). Dina ändringar skall finnas med.

4. Kör gitg. Det skall synas två olika ”färgade cirklar”, test och master (som inte är på samma ställe).

5. Byt tillbaks till master med git checkout master.

6. Inspektera koden. Ingen förändring skall synas.

7. Vi antar att förändringen var bra, så vi vill slå ihop grenarna. Se till att du är på master. Skriv;

   ```
   $ git merge test
   ```

8. Git svara med ”Fast-forward” m.m. (en linjär sammanslagning).

9. Kör gitg igen. Nu skall ”cirklarna” vara på samma ställe. Byt mellan grenar och inspektera, all kod skall vara lika.

9.1 Branches i EGit

Om man använder EGit kommer koden man ser i editor-fönstret att bytas direkt, ”automagically” då man byter mellan grenar!

1. Du måste importera projektet via EGit enligt ovan.

2. För att skapa grenar klicka framför projektet i Git Repositories fönstret, välj Branches > Högerklick > Switch to... > New Branch.

3. För att göre merge markera master > högerklick > merge.

4. Man även byta mellan branches under mappen Local (under Branches)

10 Konflikter

Det kan uppstå konflikter mellan olika kodversioner som inte Git kan lösa. Vilken kod är den riktiga? Man måste då gå in och för hand välja den kod som gäller (därefter en commit i terminalen, EGit verkar fastna?).

Både Git status och EGit varnar då man fått en konflikt och säger vilka filer det gäller (stora röda blaffor i Eclipse).

1. Låt två personer ändra samma rad i samma klass till två olika varianter.
2. Person 1 commit:ar allt och push:ar
3. Person 2 commit:ar allt och pull:ar.
4. Git säger något om "merge conflicts" för person 2 och vilken fil det gäller.
5. Gå in i filen och välj vilken kod som skall var den riktiga. Det ser ut som något likanande;

   ```
   <<<<<<< HEAD

   System.out.println("This was the old version");
   =======
   System.out.println("This is the new version");
   >>>>>>> 38fc8527df449fa353fa20e9df68f8fd16952108
   ```

   Ni väljer det som står över eller under ========. Ta bort den andra koden och alla markeringar (===, <<< eller >>>), därefter en ny commit.

TIPS: Har man inget värdefullt och det är väldigt mycket konflikter kan man klona det central igen (zippa ihop katalog för lokala).