1 Skaffa sig en lokal kopia av repot

För att skaffa sig en kopia av git repot att jobba emot kör man först ett av följande kommandon:

git clone /proj/tde/projects/projektgrupp20/repo.git tste12 om man jobbar lokalt på någon av ISY:s datorer. Det är lämpligt att checka ut repot i sin hemkatalog och inte i den gemensamma projektkatalogen.

eller git clone ssh://<user>@ixtab.edu.isy.liu.se/proj/tde/projects/projektgrupp20/repo.git tste12 om man jobbar ifrån någon annan dator

Gå därefter in i repot och kör git config user.name '<Yuur full name>' och git config user.email '<user@student.liu.se>' för att man ska kunna se vem som ändrat vad senare.

Därefter är det bara att redigera filer som vanligt tills dess att man är klar med det man håller på med för tillfället i vilket läge det är dags att committa sina ändringar in i repot.

2 Uppdatera det lokala repot med ändringar

Har man inte lagt till några nya filer är det bara att köra git commit -a -m <message> så kommer samtliga ändringar att committas automatiskt. Om man har skapat några nya filer måste man dessutom säga till git att den ska versionhantera dessa filer också, detta göra igenom att köra git -a <file> för de nya filerna varefter det bara är att köra commit som vanligt.

Vill man ha lite mer kontroll över vilka filer man committar, t.ex. om man har ändrat på flera filer men bara vill committa några stycken av dem, kan man istället för att köra git commit -a -m <message> köra först git add <file> på de filer vars ändringar man vill ska committas och därefter köra git commit -m <message> (utan några extra flaggor förrutom committ meddelandet).

För att få information om vad som kommer att committas och vilka filer man har skapat/ändrat sedan senaste committen kan man köra git status. Man skulle då t.ex. kunna få en output som nedan.

```
# On branch master
# Changes to be committed:
#   (use ``git reset HEAD <file >...'' to unstage)
#
#       new file:   git.tex
#
# Changed but not updated:
#   (use ``git add <file >...'' to update what will be committed)
#   (use ``git checkout -- <file >...'' to discard changes in working directory)
#
#       modified:   .empty
#
# Untracked files:
#   (use ``git add <file >...'' to include in what will be committed)
#
#       Design_doc_0.1.odt
#       Req_doc_0.2.odt
```

Skulle man i detta läge köra en git commit skulle den nya filen git.tex att committas, men den modifiera filen .empty skulle inte committas. Om man däremot körde en git commit -a -m <message> skulle både git.tex och .empty committas.

3 Uppdatera det centrala repot med ändringar i det lokala

Viktigt att notera är att alla kommandon som har körts än så länge endast arbetar mot din egen lokala kopia av repot, d.v.s. än så länge kommer ingen annan att kunna se dina ändringar. Detta innebär att man gärna ska göra committas ofta även om saker temporärt inte fungerar eftersom det bara är lokalt dessa ändringar kommer att finnas ändå ända tills man explicit säger åt den att lägga upp det på det centrala repot.

För att göra det kör man git push på sitt lokala repo vilket får alla ändringar man har committat lokalt men som inte finns centralt ännu att föras över till det centrala repot. När man gör detta är det viktigt att koden är i ett fungerande skick. Om någon annan skulle ha gjort push på sina ändringar till repot mellan det att du uppdaterade din lokala kopia senaste och det att du gjorde push kommer git att klaga och inte tillåta dig att föra över dina ändringar dit. För att åtgärda detta måste du först uppdatera din lokala kopia med de senaste uppdateringarna. Detta görs igenom att köra git pull. Samtliga ändringar som finns i det centrala repot kommer då att föras över till ditt lokala repo och automatisk mergas med filerna där om möjligt. Ifall man har otur och har ändrat på precis samma ställe i en fil som kommer detta att misslyckas och måste åtgärdas manuellt. För att minska risken för att detta uppstår bör man alltid innan man börjar arbeta köra git pull för att hämta ner senaste versionen som finns med git pull.

4 Hantering av misslyckad merge

Om en merge misslyckas t.ex. vid en git pull är det viktigt att åtgärda detta. Enklast brukar vara att först köra git diff för att få en snabb överblick över vad som misslyckats och därefter öppna filen/filerna i fråga i sin favoriteditor och korrigera så att det blir rätt. För att veta vad det är som behöver åtgärdas är det bra att veta vad som händer när git misslyckas med en merge. På de ställen i filen som en merge har misslyckats kommer både den lokala versionen av de påverkade raderna samt versionen ifrån det centrala repot att finnas på följande sätt:

```
<<<<<<< HEAD
This is my local version
=======
This is the central version
>>>>>>> origin/master
```

5 HDL designer och versionhantering

Starta HDL designer igenom att köra ./start (en något modifierad variant av tste12proj scriptet) i src mappen i ditt utcheckade repo. Om den inte automatiskt öppnar rätt projekt får du välja öppna projekt och sedan välja MESS.hdp som också ligger i src mappen. För att det ska fungera bra med HDL designer är det viktigt att man endast versionhanterar de filerna som den faktiskta informationen ligger i och inte sådana som autogeneraras utifrån dessa filer. För att förenkla har åtminstonde en del av dessa filer lagts till i en .gitignore file vilket kommer få git att automatiskt ignorera dem, men det är ändå bra att ha koll på ungefär vad som ska vara med och vad som inte ska vara det ifall något missas. I hdl mappen kommer alla .vhd filer att ligga, de som slutar på _struct.vhd och _generatedinstance.vhd är dock autogenerarade och ska inte versionhanteras, övriga filer bör vara med. Mappen vid namn

work autogeneraras vid kompilering och bör därmed inte versionhanteras alls. I hds mappen ser det ut som att alla filer ska versionhanteras förrutom de som slutar på .bak och .lck.

För att saker ska fungera smärtfritt gäller dessutom följande, det är helt okej att inom sin modul ändra runt innuti HDL-designer, lägga till submoduler innuti den, dra ledningar och liknande, så länge man ser till att kommunicera sådana ändringar inom subgruppen på ett bra sätt då ändringar där kan vara svåra att merga i efterhand.

Ändringar på topnivå så som skapande av nya moduler, tilläg av nya signaler och indragning av sådana i modulerna måste koordineras mer nogrannt, innan en sådan ändring görs bör man se till att alla andra är medvetna om att man kommer att göra den så att de kan uppdatera sin version med ändringarna på ett smidigt sätt.

6 Sammanfattning av kommandon

clone remote <local>	Skapar en egen kopia av repot i <remote> på den plats som anges av <lokal>
add <file>	Lägger till <file> att komma med om commit körs. Måste köras på nya filer
commit -a -m <message>	Committar alla ändrade filer med loggmeddelande <message>
commit -m <message> <file>	Committar endast <file>
commit -m <message>	Committar alla filer som lagts till med add
add -u <file>	Samma sak som att köra git add <file> på alla ändrade filer
push	Trycker upp alla committade lokala ändringar till det centrala repot
pull	Hämtar ner alla nya ändringar i den centrala repot till det lokala
log	Visar en lista över alla committs med loggmeddelanden
diff	Visar alla ändrade filer som man inte kör commit eller add på
branch <branchname>	Skapar en ny lokal branch vid namn <branchname>
checkout <branchname>	Byter till branchen vid namn <branchname>
checkout <file>	Återställer filen vid namn <file> till den som man senaste körde commit/add på