Distribueret versionskontrol – Mercurial

Jakob Lykke Andersen
jlandersen@imada.sdu.dk

6. december 2012

Indhold

1 Introduktion

Systemer til versionskontrol kan, i korte træk, beskrives som værktøjer til at holde styr på ændringer i filer. Det bliver dermed nemmere at finde gamle version af filer frem og arbejde flere på den samme samling filer samtidig. De fleste versionskontrolsystemer bygger enten på en klient-server-arkitektur eller, som Mercurial, en distribueret arkitektur. Andre distribuerede versionskontrolsystemer (DVCS) inkluderer Git og Bazaar. Et meget brugt ikke-distribueret system er Subversion.

Versionskontrolsystemer er meget anvendelige i mange forskellige scenarier. De kan bruges til at holde styr på ikke bare store software-projekter som Linux-kernen eller operativsystemet Android, men også små projekter som et enkeltmandsprojekt i et programmeringskursus. De fleste open-source projekter bliver udviklet ved hjælp af versionskontrolsystemer. Det er dog ikke kun til ren programmering at versionskontrol kan

bruges, også dokumentation i form af eksempelvis LaTeX-dokumenter kan med fordel versioneres.

Hensigten med denne guide er ikke at forklare alle aspekter af Mercurial, men blot de basale koncepter så man relativt hurtigt kan begynde at benytte versionskontrol i dagligdagen. Mercurial kan installeres på både Mac OS, Windows og Linux-systemer, men opgaverne tager dog udgangspunkt i IMADAs Ubuntu-installation.

1.1 Notation

Der vil i de efterfølgende afsnit være vist udklip fra en terminal der indeholder både kommandoer der bliver kørt og kommandoernes output. Linjer på formen

```
some/path/to/a/folder$ command arg1 arg2 arg3
```

betyder at kommandoen `command` med argumenterne `arg1`, `arg2` og `arg3` udføres i mappen `some/path/to/a/folder`. De resterende linjer er output fra kommandoer.

Når en sti starter med ~ betyder det at stien tager udgangspunkt i brugerens hjemmemappe (`/home/`*`brugernavn`*). Det betyder at stien ~`/Downloads` hos brugeren `john42` skal læses som `/home/john42/Downloads`. Husk at når kommandoen `cd` udføres uden argumenter vil terminalen gå til brugerens hjemme-mappe.

1.2 Hjælp til kommandoer

Mercurial har et indbygget hjælpe-system, så man blot ved at udføre `hg help` *`kommando`* kan få information om kommandoen `hg` *`kommando`*. Vær desuden opmærksom på at for de fleste kommandoer behøver man ikke skrive dem fuldt ud, men blot nok til at Mercurial ikke er i tvivl. Eksempelvis kan man nøjes med at skrive `hg st` i stedet for `hg status`. I de følgende afsnit vil de fulde navne dog blive brugt.

1.3 Installation (på ikke-IMADA-maskiner)

På Ubuntu kan Mercurial installeres på følgende vis.

```
$ sudo apt-get install mercurial
```

1.4 Opsætning

Åben (eller opret) filen ~`/.hgrc` (bemærk at filens navn starter med et punktum) i en tekst-editor og indsæt følgende data (med `Fornavn Efternavn` udskiftet). Husk at filer/mapper hvis navn starter med et punktum også kaldes for skjulte filer/mapper.

```
[ui]
username = Fornavn Efternavn
merge = /home/jlandersen/shared/bin/kdiff3

[extensions]
progress =
color =
```

```
graphlog =
```

Det er dog kun `username` der er strengt nødvendig.

Linjen `merge = ...` fortæller Mercurial at programmet `kdiff3` skal bruges til at sammenflette filer med (se afsnittet om manuel sammenfletning). Her kan man blot indsætte ens eget foretrukne 3-vejs-sammenfletningsprogram, eller linjen kan helt udelades hvis Mercurial selv skal prøve at finde et værktøj.

I skrivende stund er `kdiff3` ikke installeret generelt på IMADAs system, men er dog tilgængeligt via den angivne sti. På systemer hvor `kdiff3` er installeret normalt, vil man kunne nøjes med at skrive `merge = kdiff3`.

Udvidelsen `progress` gør at der kommer status-indikatorer når data overføres. Det er dog primært når et repository tilgås over internettet det relevant.

Den anden udvidelse, `color`, sætter farve på outputtet fra de fleste Mercurial-kommandoer, hvilket blandt andet er nyttigt ved `hg status` og `hg diff`.

Den sidste udvidelse, `graphlog`, gør det muligt at give argumentet `-G` til kommandoen `hg log`, så den også printer træstrukturen for loggen ved siden af det normale log-data.

2 Mini-guide

Når Mercurial skal holde styr på ændringer af filer sker det ved at omdanne en mappe til et *repository*. Eksempelvis

```
hgtest$ mkdir eks
hgtest$ cd eks
hgtest/eks$ hg init
```

vil oprette en mappe `eks` og omdanne det til et repository. Det samme kan også gøres lidt enklere:

```
hgtest$ hg init eks
```

Initialiserings-kommandoen operetter her en mappe `eks/.hg`. Den kommer til at indeholde historikken for filerne i mappen `eks`, inklusiv eventuelle filer i undermapper.

Mercurial skal have at vide hvilke filer der skal holdes styr på, så det er ikke nok blot at lægge filer ind i en mappe med et repository. Lad os starte med to filer:

```
hgtest/eks$ echo "print \"Hello world\"" > hello.py
hgtest/eks$ echo "print 42" > answer.py
hgtest/eks$ ls -a
.  ..  answer.py  hello.py  .hg
```

De første to linjer, på formen `echo "data" > fil`, sletter indholdet af `fil` og skriver `data` til filen, mens kommandoen `ls -a` viser indholdet af en mappe inklusiv skjulte filer og mapper.

Vi kan nu bruge kommandoen `hg status` til at få at vide hvad Mercurial ser som ændringer i mappen `eks`:

```
hgtest/eks$ hg status
? answer.py
? hello.py
```

Hver linje angiver en fil hvor der er sket en ændring, samt et tegn der fortæller hvilken ændring det er. I dette tilfælde står der et `?` ved begge filer, hvilket betyder at repositoriet ikke kender til dem. Vi kan angive at repositoriet skal holde styr på de to filer ved:

```
hgtest/eks$ hg add answer.py hello.py
hgtest/eks$ hg status
A answer.py
A hello.py
```

Et `A` betyder at filen lige er tilføjet.

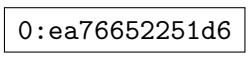
I Mercurial bliver ændringer grupperet i *changeset*s. Vi kan eksempelvis bestemme os for at det at tilføje de to Python-filer er en logisk gruppe af ændringer. Et changeset operettes ved brug af kommandoen `hg commit`:

```
hgtest/eks$ hg commit -m "Addition of some Python scripts"
```

Hvert commit skal have en ikke-tom log-besked. Hvis ikke den bliver angivet som argument (via `-m "besked"`), vil der automatisk blive åbnet et program hvor man skal skrive beskeden.

Vi kan med `hg status` se at der ikke længere er nogle ændringer i mappen. Til gengæld er der blevet tilføjet noget historik til selve repositoriet, der kan ses med kommandoen `hg log`:

```
hgtest/eks$ hg log
changeset:   0:ea76652251d6
tag:         tip
user:        Jakob Lykke Andersen
date:        Mon Dec 03 13:20:41 2012 +0100
summary:     Addition of some Python scripts
```



Figur 1: Grafisk repræsentation af en repository-log med kun et enkelt changeset.

Vi kan også grafisk repræsentere loggen som på figur 1. Ud fra loggen kan vi se at Mercurial kalder vores changeset for `0:ea76652251d6`, hvilket betyder at vi lokalt godt kan se det som changeset 0 (tallet før `:`), men at det korrekte id er `ea76652251d6`. Dette id udregner Mercurial som et hash af ændringerne i changesettet.

Linjen `tag` i loggen angiver alternative navne for et changeset, og man kan selv give tags til enkelte changesets. Dog er `tip` er et specielt tag der bruges af Mercurial til at angive det nyeste changeset.

Et commit i Mercurial er, som nævnt, oprettelsen af et changeset der specificerer hvordan *hele* repositoriet ændrer sig. I terminologien for Mercurial taler man dermed ikke om forskellige version af enkelte filer, men forskellige versioner af et helt projekt.

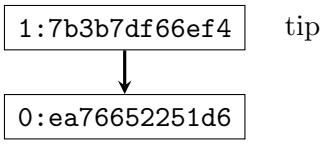
Lad os lave endnu et changeset:

```
hgtest/eks$ echo "data" > note.txt
hgtest/eks$ hg status
? note.txt
hgtest/eks$ hg add note.txt
hgtest/eks$ hg commit -m "Addition of a text file"
```

Her tilføjer vi blot endnu en fil, `note.txt`, og laver et changeset med tilføjelsen. Loggen er nu:

```
hgtest/eks$ hg log
changeset:   1:7b3b7df66ef4
tag:         tip
user:        Jakob Lykke Andersen
date:        Mon Dec 03 14:03:29 2012 +0100
summary:     Addition of a text file

changeset:   0:348348df659f
user:        Jakob Lykke Andersen
date:        Mon Dec 03 13:44:51 2012 +0100
summary:     Addition of some Python scripts
```



Figur 2: Grafisk repræsentation af en repository-log.

Dette er vist grafisk på figur 2, og vi kan se at det nye changeset nu er markeret som `tip`. I den grafiske fremstilling er der også vist en *parent*-pil, der angiver at changeset 1 består af ændringer til changeset 0.

Antag nu at vi laver en ændring til `note.txt` så den indeholder følgende:

```
data, mere data
endnu mere data
```

Vi bruger nu `status`-kommandoen, og da vi har modificeret en fil der allerede er kendt af Mercurial, bliver den markeret med et `M`:

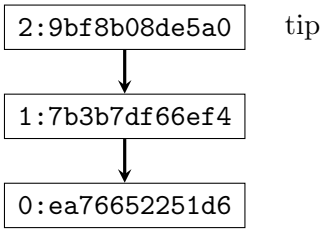
```
hgtest/eks$ hg status
M note.txt
```

Dette comitter vi, og repositoriet har strukturen på figur 3:

```
hgtest/eks$ hg commit -m "Update of notes"
```

Vi kan nu se en gammel version (også kaldet *revision*) af hele repositoriet, eksempelvis lige efter changeset 1 var lavet, ved brug af kommandoen `hg update`:

```
hgtest/eks$ hg update --rev 1
1 files updated, 0 files merged, 0 files removed, 0 files
   unresolved
```



Figur 3: Grafisk repræsentation af en repository-log.

Mercurial fortæller her at for at vise revision 1 er der i mappen `eks` blevet ændret på en enkelt fil, og vi kan se at `note.txt` nu indeholder det gamle data (kommandoen `cat fil` printer indholdet af `fil` til terminalen):

```
hgtest/eks$ cat note.txt
data
```

Vi kan gå ydeligere tilbage:

```
hgtest/eks$ hg update --rev 0
0 files updated, 0 files merged, 1 files removed, 0 files
   unresolved
hgtest/eks$ ls -a
.  ..  answer.py  hello.py  .hg
```

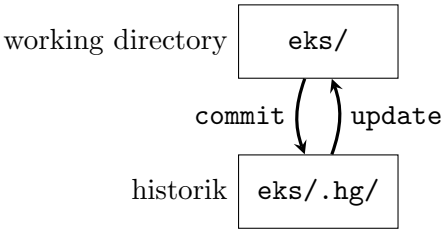
Filen `note.txt` blev først tilføjet til repositoriet i changeset 1, så den er nu blevet fjernet fra `eks`. For at få den nyeste revision frem igen kan vi blot bruge `update` uden argumenter:

```
hgtest/eks$ hg update
1 files updated, 0 files merged, 0 files removed, 0 files
   unresolved
hgtest/eks$ ls -a
.  ..  answer.py  hello.py  .hg  note.txt
hgtest/eks$ cat note.txt
data, mere data
endnu mere data
```

Da dataen for repositoriet ligger i `.hg`-mappen og resten af `eks` er der hvor man som bruger arbejder kalder man `eks` for repositoriets *working directory*. Kommandoen `hg commit` kan dermed ses som data-kommunikation fra et working directory til et repository, og omvendt kan `hg update` ses som data-kommunikation fra repositoriet til et working directory. Man kan bruge kommandoen `hg summary` for, blandt andet, at få at vide hvilken revision man i øjeblikket arbejder ud fra i ens working directory:

```
hgtest/eks$ hg summary
parent: 2:9bf8b08de5a0 tip
 Update of notes
branch: default
commit: (clean)
update: (current)
```

En visualisering af data-strømmen i et repository kan ses på figur 4.



Figur 4: Et repository i mappen `eks`, opdelt i dets working directory og selve historikken. Pilene symboliserer strømmen af data ved `commit`- og `update`-kommandoer.

2.1 Parallelt arbejde

Antag nu at der er flere der skal arbejde på samme datamængde, så en enkelt person, ved navn Alice, har oprette et repository og arbejdet så det ser ud som i slutningen af forrige afsnit. Vi har dermed et repository i mappen `Alice/hgtest/eks`. En anden person, eksempelvis Bob, kan klone repositoriet for at få en eksakt kopi hvor han selv kan arbejde:

```
Bob/hgtest$ hg clone /home/Alice/hgtest/eks
updating to branch default
3 files updated, 0 files merged, 0 files removed, 0 files
   unresolved
Bob/hgtest$ ls
eks
Bob/hgtest$ cd eks
Bob/hgtest/eks$ hg summary
parent: 2:9bf8b08de5a0 tip
 Update of notes
branch: default
commit: (clean)
update: (current)
```

Kun selve repositoriet bliver kopieret, ikke ændringer der ikke er committed i Alice' working directory.

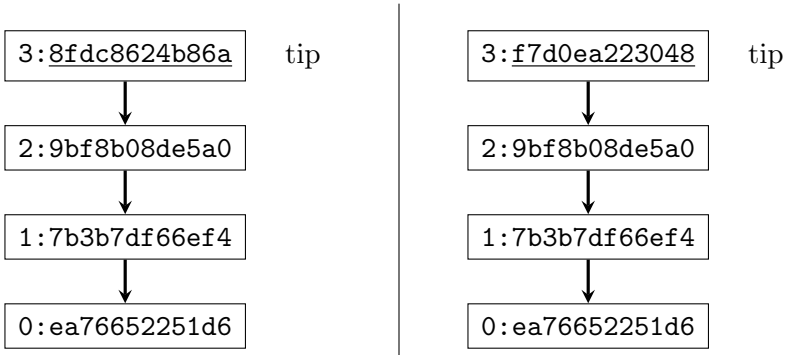
Alice og Bob kan nu hver især arbejde uafhængigt, så lad os antage at Bob synes at `note.txt` i stedet bør hedde `todo.txt`. I stedet for at bruge den almindelig kommando, `mv`, bruger han `hg mv` da Mercurial dermed så også får at vide at filen er blevet flyttet:

```
Bob/hgtest/eks$ hg mv note.txt todo.txt
Bob/hgtest/eks$ hg status
A todo.txt
R note.txt
Bob/hgtest/eks$ hg commit -m "Renamed note.txt to todo.txt"
```

Selv om Mercurial rigtigt nok fortæller ved `R note.txt` fortæller at filen er fjernet ved `A todo.txt` er tilføjet, er der desuden gemt information om at der i virkeligheden er tale om en flytning af en fil.

Samtidig har Alice tilføjet mere data til den fil som i hendes repository stadig hedder `note.txt`:

```
Alice/hgtest/eks$ cat note.txt
data, mere data
endnu mere data
lidt mere data
Alice/hgtest/eks$ hg commit -m "Added data to note.txt"
```



(a) Alice (b) Bob

Figur 5: Grafisk repræsentation af begge repositories. Bemærk at både Alice og Bob har et changeset 3, men at de to changsets id'er er forskellige.

Når vi visualiserer de to repositories får vi figur 5. Her er det meget vigtig af bemærke at for både Alice og Bob der er deres nyeste changeset kaldt 3 (det lokale id), men at de rigtige id'er er forskellige.

Bob kan nu prøve at give sine ændringer til Alice ved brug af kommandoen `hg push`:

```
Bob/hgtest/eks$ hg push /home/Alice/hgtest/eks
pushing to /home/Alice/hgtest/eks
searching for changes
abort: push creates new remote heads on branch 'default'!
(you should pull and merge or use push -f to force)
```

Kommandoen fejler fordi Alice allerede har tilføjet changesets som Bob ikke har. Han bliver derfor nødt til at at hente de ændringer:

```
Bob/hgtest/eks$ hg pull /home/Alice/eks
pulling from /home/Alice/eks
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
```

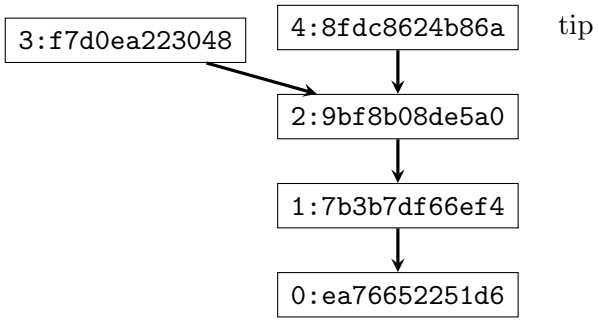
Loggen for Bobs repository har nu også det nye changeset fra Alice (vi bruger argumentet `-l` N til kun af vise de N nyeste changesets):

```
Bob/hgtest/eks$ hg log -l 3
changeset:   4:8fdc8624b86a
tag:         tip
parent:      2:9bf8b08de5a0
user:        Alice
date:        Mon Dec 03 18:00:01 2012 +0100
summary:     Added data to note.txt

changeset:   3:f7d0ea223048
user:        Bob
date:        Mon Dec 03 17:53:09 2012 +0100
summary:     Renamed note.txt to todo.txt

changeset:   2:9bf8b08de5a0
user:        Alice
date:        Mon Dec 03 17:01:13 2012 +0100
summary:     Update of notes
```

Ved Bob bliver det nye changeset fra Alice kaldt `4:8fdc8624b86a`, og da dets forælder ikke blot er changeset 3 bliver der eksplicit skrevet `parent: 2:9bf8b08de5a0`. På figur 6



Figur 6: Grafisk repræsentation af Bobs repository-log.

er Bobs repository visualiseret.

De changsets som ikke er forælder til andre changesets kalder Mercurial for *head* changesets, og loggen for disse changesets kan vises ved kommandoen `hg heads`. I Bobs repository er det changeset 3 og 4 der er heads.

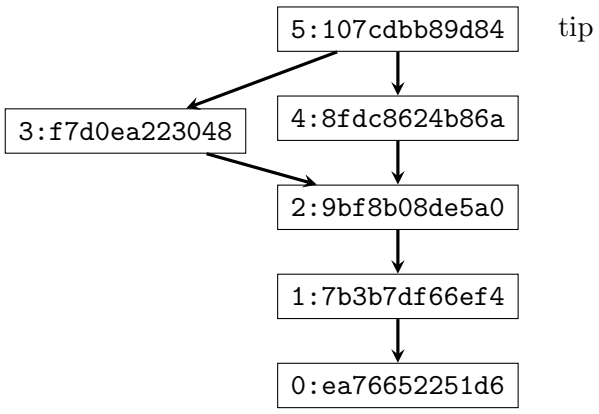
Man kan få Mercurial til at prøve at sammenflette heads ved at bruge kommandoen `hg merge`. I Bobs tilfælde vil Mercurial prøve at sammenflette changesettet fra Alice (der modificerede indholdet af hvad hun så som `note.txt`) og Bobs ændring (flytning af `note.txt` til `todo.txt`):

```
Bob/hgtest/eks$ hg merge
merging todo.txt and note.txt to todo.txt
0 files updated, 1 files merged, 0 files removed, 0 files
   unresolved
```

```
(branch merge, don't forget to commit)
```

Nu indeholder Bobs working directory ændringer der svarer til at sammenflette revision 3 og 4. Denne ændring kan nu committes, og figur 7 er en visualisering af loggen.

```
Bob/hgtest/eks$ hg commit -m "Merge after pull from Alice"
Bob/hgtest/eks$ hg log -l 1
changeset:   5:107cdbb89d84
tag:         tip
parent:      3:f7d0ea223048
parent:      4:8fdc8624b86a
user:        Bob
date:        Mon Dec 03 19:09:26 2012 +0100
summary:     Merge after pull from Alice
```



Figur 7: Grafisk repræsentation af Bobs repository-log efter en sammenfletning.

Når Bob nu bruger `push` vil Alice få opdateret sit repository med de ekstra changesets:

```
Bob/hgtest/eks$ hg push /home/Alice/eks
pushing to /home/Alice/hgtest/eks
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

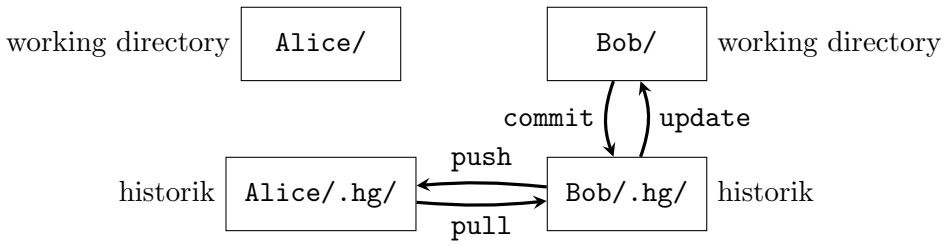
Bemærk at hvad Alice har liggende i sit working directory ikke bliver ændret:

```
Alice/hgtest/eks$ ls
answer.py  hello.py  note.txt
Alice/hgtest/eks$ hg summary
parent: 3:8fdc8624b86a
 Added data to note.txt
branch: default
commit: (clean)
update: 2 new changesets (update)
```

Kommandoen `hg summary` fortæller dog (i `update`-linjen) at der er to nye changesets, så Alice bruger `hg update` for at opdatere sit working directory:

```
Alice/hgtest/eks$ hg update
1 files updated, 0 files merged, 1 files removed, 0 files
   unresolved
Alice/hgtest/eks$ ls
answer.py  hello.py  todo.txt
```

Hvor kommandoerne `commit` og `update` bruges til kommunikation af filændringer mellem et repository og dets working directory, kan man se kommandoerne `push` og `pull` som kommunikation af changesets mellem repositories. Hverken `push` eller `pull` ændrer på working directories, hverken ved afsender eller modtager. På figur 8 er kommunikationen mellem to repositories vist. Bemærk at `push` og `pull` er defineret med udgangspunkt i et specifikt repository, på figuren Bobs repository.



Figur 8: To repositories, `Alice` og `Bob`, opdelt i working directories og historikker. De fuldt optrukne pile symboliserer strømmen af data ved `commit`-, `update`-, `push`- og `pull`-kommandoer udført i `Bob`. Ingen af kommandoerne hverken læser fra eller skriver til det working directory Alice har.

Når man kloner et repository vil stien til der hvor man kloner fra blive gemt i klonen, og hvis man blot bruger `hg push` og `hg pull` uden en sti, vil den gemte sti blive brugt. Det betyder at alle de `push`- og `pull`-kommandoer Bob bruger kan udføres helt uden stien.

2.2 Manuel sammenfletning

Det er ikke altid Mercurial kan finde ud af hvordan to filer skal sammenflettes automatisk. Det kunne eksempelvis være hvis der er én person der har rettet stavefejl i en rapport, mens en anden har omskrevet dele af rapporten. Hvis automatisk sammenfletning ikke er mulig vil Mercurial bede brugeren om manuelt at sammenflette de filer der har konflikter. Har man i sin `.hgrc`-fil indsat en linje med `merge = mergeTool` vil det angivne program blive åbnet.

Det kan være en ganske besværlig procedure at sammenflette filer, men den tekniske del af det kan der læses mere om på følgende adresse: `http://hgbook.red-bean.com/read/a-tour-of-mercurial-merging-work.html`.

3 Anden (u)nyttig information

3.1 Gruppearbejde

Selv om man ikke behøver at have en central server, er det oftest stadig det nemmeste når man skal arbejde flere sammen. Man kan eksempelvis benytte `https://bitbucket.org` som det centrale repository, hvis ikke man er for mange gruppemedlemmer. Der er så vidt vides ikke i øjeblikket en lige så nem og sikker metode til at oprette det centrale repository på IMADAs system.

3.2 Repository-indhold

Det er ikke al data der er lige nem at håndtere i repositories. LaTeX-dokumenter, Python-kode og andre filer der består af tekst er der ingen problemer med, mens filer som Word-dokumenter, PDF-filer og compilede programmer er det der bliver kaldt for *binære filer* (prøv at åbne dem i `gedit`). Problemet er at for at kunne sammenflette binære filer, skal man have kendskab til selve fil-formatet, hvilket de almindelige sammenfletningsprogrammer ikke har. Det er derfor generelt dårlig skik at lægge binære filer i et repository der kan genereres ud fra tekst-filer. Det er eksempelvis PDF-dokumenter genereret fra LaTeX-kode eller compilede Java-programmer (`.class`-filer). Ved andre binære filer skal man blot være ekstra opmærksom.

4 Opgaver

Se først afsnit 1, side 1, om opsætning.

1. Udfør mini-guiden, men brug `hg status`, `hg summary` og `hg log` flittigt for at følge med i alle detaljer. Hvis udvidelsen `graphlog` er slået til, så brug `hg log -G` i stedet for `hg log`.

2. Når filer skal slettes skal Mercurial også have at vide at det er sletning der er intentionen. Kommandoen er `hg rm`.

   (a) Brug `hg rm` til at slette en fil og `hg status` til at se den umiddelbare effekt.

   (b) Commit sletningen. Hvilken effekt har sletningen? Er filen helt væk, og hvis ikke, prøv at find den igen.

   (c) Bonus-opgave: brug nu `rm` til at slette en fil som Mercurial kender til. Hvad er forskellen i outputtet fra `hg status` i forhold til når `hg rm` bruges?

   (d) Bonus-opgave: brug `hg help rm` til at finde en metode til at rette op på fejlen, så Mercurial får at vide at filen er slettet.

3. Hvad gør man hvis der er ændringer i ens working directory man fortryder? (hint: `hg help revert`)

   - Modificer, slet, flyt og tilføj filer og forsøg at fortryde ændringerne.

- `hg revert` kan finde på at oprette filer der ender på `.orig`. Hvad er indholdet af disse filer?

4. Klon et repository, lav ændringer i begge repositories der er i konflikt med hinanden og sammenflet ændringerne. Eventuelt find andre personer, lav ændringer i hver jeres klon og prøv at sammenflet ændringer.

5. Normalt bruger man `cp` til at kopiere filer, men det kan være en fordel at lade Mercurial vide at én fil i virkeligheden er opstået som en kopi af en anden.

   (a) Opret et repository, `A`, og commit en tekst-fil, `test.txt`.

   (b) Klon repositoriet til `B`.

   (c) I repository `A`, lav en ændring i `test.txt`, og commit ændringen.

   (d) I repository `B`, brug `hg cp` til at kopiere `test.txt` til `testCopy.txt`. Commit ændringen.

   (e) I repository `B`, brug `hg pull` (der vil nu være to heads), derefter `hg merge` og til sidst commit sammenfletningen.

   (f) Hvilke ændringer er der sket i `B`?

   (g) Bonus-opgave: gennemfør ovenstående igen, men i stedet for at bruge `hg cp` til kopieringen, så brug `cp` (og derefter `hg add` for at tilføje kopien til repositoriet).

6. Blandt andet når der er sket ændringer i store filer kan det være en fordel at få en præcis angivelse af ændringerne. Udforsk `hg diff` (se `hg help diff`), ved eksempelvis at lav ændringer i et working directory uden at comitte dem, og så kør `hg diff`.

Forslag til andre opgaver af mere omfattende karakter:

- Opret et repository til kursusmaterialer og obligatoriske opgaver (bortset fra gruppeprojekter). Bemærk at kode der er administreret af en IDE (Eclipse, Netbeans etc.) skal behandles specielt, og gennem selve IDE'en.

- Opsæt Mercurial på egen (Linux-)computer og klon repositories via SSH fra IMADAs system.

  Programmet `ssh` kan bruges til at logge ind på andre computere over et netværk, hvis der er installeret en SSH-server på dem (det er der på IMADAs). På Windows kan man benytte programmet PuTTY som SSH-klient. For at logge på IMADAs system kan man benytte følgende, med `username` udskiftet:

  ```
  $ ssh username@login.imada.sdu.dk
  ```

  Man kan klone et repository via SSH:

  ```
  $ hg clone ssh://username@login.imada.sdu.dk/path/on/server
  ```

- Opsæt et gruppeprojekt med et centralt repository.

5 Referencer

Officiel hjemmeside for Mercurial: `http://mercurial.selenic.com/`

Mercurial-guide: `http://hgbook.red-bean.com/`