Subversion

Laboration

Höstterminen 2008
r81

Ronny Kuylenstierna
ronku@ida.liu.se

”[...] Subversion can be used to manage changes to any sort of information—images, music, databases, documentation, and so on. To Subversion, all data is just data.”
subversionboken

Laboration

Du ska nu laborera med Subversion och det görs i ett **bash-skal**. Labben är tänkt att göras i **den ordning som uppgifterna står i**. Vissa uppgifter har givna kommandon som du ska skriva in och köra i bash. Andra uppgifter innebär att du själv får komma på vad kommandot är eller vad du nu uppmanas göra. Varje rad som börjar med en siffra och har grå bakgrund är ett kommando som du ska köra. Det är det fetstilta som du ska skriva in.

Du bör efter labben förstå varje kommando du skrivit in. Använd subversions inbyggda hjälp (*svn help*) och subversionboken flitigt (se länk i svn-instruktioner). Förslagsvis har du subversionboken öppen i ett fönster och där söker upp de kommandon du labbar med.

Innan du börjar med uppgifterna i labben är det bra att läsa första kapitlet, *1. Fundamental Concepts*, i **svn-boken** (se länk svn-instruktioner). Detta för att få en förståelse om vad revisionshantering är och dess fundamentala koncept. Andra kapitlet, *2. Basic Usage*, går igenom subversionklientens grundläggande kommandon och användning av dito och kommer att vara din vän i subversionlaborationen. Labben kommer (för det mesta) hålla sig till kapitel 2.

Frågor som ska besvaras och uppgifter utöver svn är markerade med en tjock linje i vänsterkanten. Svaren skrivs i en fil med filnamn `svn-svar` som du senare i labben kommer checka in i repositoryt.

Förkunskaper: Stone-kursen.

Inled varje subversionkommentar du gör i denna labb med ”*svnlabb:*”! Detta för att laborationsassistenten ska kunna se att du gjort det du ska. Du **ska** skriva kommentar för varje kommit du gör.

Övningarna förbereder dig för kommande IP-projekt där du kommer använda dig av subversion. **Lägg ner energi** på att förstå subversion – det tjänar du på!

Listig information

`svn list`
`svn info`

Moment 1 `svn list`

Ta reda på vilken URL kursrepositoryt har och kör:

```
1 ~ $ svn list URL
```

Resultatet är troligtvis inte speciellt exalterande – det är nog tomt eftersom repositoryt förmodligen är tomt.

Tips 1: returkod

Varje kommando som körs returnerar en kod (i form av ett heltal) när kommandot avslutats. Om den koden är noll kördes kommandot utan att fel uppstod. En annan siffra än noll betyder följdaktligen fel. För att skriva ut senaste körda kommandos returkod kör:

```
~ $ echo $?
```

Testa nu att köra *svn list* för en felaktig URL och sedan skriva ut felkoden.

(1) Vad har ett ”lyckat” kommando för returkod?

(2) Testa nu att köra *svn list* för en felaktig URL och sedan skriva ut returkoden. Vad fick du för returkod?

Besvara frågorna i en fil som heter `svn-svar`. Var filen ligger just nu spelar inte så stor roll. Du kommer senare flytta den till en mapp för svn-labben.

Extra 1: bash prompt

Som extra uppgift (och för att det är ~~häftigt~~användbart) kan du göra så att din prompt i bash visar returkoden för senast körda kommando!

Ta reda på URL:en för subversions eget repository och kör en list på den! Titta i *doc/user*. Där ser du en ”best practices”-guide som du kan läsa vid tillfälle (kanske efter du gjort labben).

(3) Vad har ”best practices”-guiden för URL?

Moment 2 `svn info`

Kör på kurs-URLen:

```
2 ~ $ svn info URL
```

(4) Vad har ”best practices”-guidens senaste revision för nummer?

(5) Vad har ”best practices”-guidens senaste ändrade revision för nummer?

(6) Vem ändrade i ”best practices”-guiden senast?

(7) Vad har kurs-repositoryt för revisionsnummer? Anteckna datum, klockslag och revisionsnummer.

(8) Vad har subversions repository för revisionsnummer? Anteckna datum, klockslag och revisionsnummer.

Checka ut!

`svn checkout`
`svn status`
`svn help`

Moment 3 `svn checkout`

Checka ut repositoryt för kursen (här nedan stående i hemkatalogen, ~).

```
3 ~ $ svn checkout URL KURS
```

Ersätt URL med aktuell url för kursen och KURS med vad du vill att kursens lokala arbetskopia ska heta. Ovanstående skapar en katalog KURS som är en lokal arbetskopia av senaste revisionen i repositoryt (sök på HEAD respektive BASE i subversionboken). Katalogen uppdateras inte automatiskt när repositoryt ändrats - det får man se till att göra själv.

Moment 4 `svn status`

Jämför 1a: Vad får du när du kör följande? (ha kvar resultatet, du ska jämföra det med annat senare)

```
4 ~/KURS $ ls -la              # observera aktuell katalog!
```

```
5 ~/KURS $ svn status
```

Notera katalogen ”*.svn*”.

Tips 2: ls, man

ls -ltr listar filer ordnat efter ändring, senast sist. I KDE kan du köra *man:ls* (K Menu – Run Command, eller tryck Alt-F2) för att läsa manualen för *ls* – annars finns ju alltid kommandot *man* (*man ls)*. För mer info om *man* kan man köra *man man*.

Moment 5 `svn help`

Om du inte tidigare kört `svn help` så gör det nu och kolla upp några svn-kommandon – till exempel `svn help list`.

> (9) Vad har *list* för kortvariant?
>
> (10) Vad har *status* för kortvariant? Vilka kortflaggor (inte långa) finns det till *status*?

svn mkdir
svn commit
svn add
svn update
svn log

Lägga till

Moment 6 svn mkdir, svn commit

Nu ska vi gå vidare med att skapa en katalog för svnlabben.

```
6 ~/KURS $ svn mkdir svnlabb
```

```
7 ~/KURS $ svn commit -m ”svnlabb: ny katalog 'svnlabb'”
```

Vi har nu skapat en katalog lokalt och sedan ”kommittat” den till repositoryt, det vill säga vi har överfört vår lokala katalog till svn-servern. Vi kan också säga att vi ”checkat in” den.

Skapa följande fil i katalogen *svnlabb* där du fortsättningsvis ska jobba.

hello.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def say_hello():
    print("Hello!")

if __name__ == "__main__":
    say_hello()
```

Jämför 1b: Vad får du *nu* när du kör följande? Vilka filer har du i katalogen och vad tycker svn om dem? Jämför med vad du fick tidigare (4-5).

```
8 ~/svnlabb $ ls -la
```

```
9 ~/svnlabb $ svn status
```

Kör följande (man kan också starta pythontolken och där importera *hello*).

```
10 ~/svnlabb $ echo "import hello" | python
```

Kör följande och notera utskrifterna.

```
11 ~/svnlabb $ ls -la
```

```
12 ~/svnlabb $ svn status
```

Jämför med det innan (8-9). Vad har hänt?

Moment 7 `svn add, svn update`

Nu ska vi se till så att vår lokala fil `hello.py` hamnar i repositoryt.

```
13 ~/svnlabb $ svn add hello.py

14 ~/svnlabb $ svn status       # Jämför med utskrift av (12)
```

Vi uppdaterar nu vår lokala arbetskopia före vi gör en kommit – en mycket bra vana!

```
15 ~/svnlabb $ svn update       # En 'update' före 'commit'!

16 ~/svnlabb $ svn commit -m "svnlabb: ny fil 'hello.py'"

17 ~/svnlabb $ svn st           # Jämför med utskrift av (14)
```

Lägg till en enrads-kommentar i filen – förslagsvis ovanför funktionen.
Kommitta (`status, update, commit`)!

Moment 8 `svn log`

Använd `svn help` för att lära dig mer om `svn log` och läs även i subversionboken. Läs dina egna kommentarer som du skrivit hittills genom att använda `svn log`. Spara loggen i en fil som heter `log-r123` där du byter ut `r123` med aktuellt revisionsnummer.
Kommitta filen.

Nu ska du få kolla lite i Pythons repository. Kolla loggen för interpretatorn som finns på följande adress:

```
http://svn.python.org/projects/python/trunk/Python/pythonrun.c
```

> (11) Vad har repositoryt för root?
>
> (12) Hur många kommittar har gjorts i repositoryt totalt?
>
> (13) Hur många rader är loggen för filen? Diskutera gärna med några klasskamrater om hur du ska lösa uppgiften – ni kanske har lite olika idéer ;-)

Två användare

`svn diff`
`svn resolved`
`svn blame`

Nu ska vi jobba med två lokala arbetskataloger för att simulera att två användare gör ändringar i samma repository – i samma fil!

Moment 9 `svn diff, svn resolved`

Checka nu ut *svnlabb* från kursrepositoryt i en **andra** katalog, *svnlabb2* (till exempel i ~/svnlabb2). Öppna gärna ett andra terminalfönster.

I **första** katalogen, *svnlabb*:
Gör en ändring i enrads-kommentaren.
Kör `svn status`.
Kommitta.

I **andra** katalogen, *svnlabb2*:
Testa och kolla upp i subversionboken flaggan *-u* för *status* (*-q* kan vara trevlig också).
Gör en update.

I **första** katalogen, *svnlabb*:
(i) Gör nu en ändring i enrads-kommentaren. Spara filen.
Gör en `svn diff` på filen och lista ut vad som menas med ”@@”, ”-” och ”+”.
Kommitta. Vilket revisionsnummer har nu HEAD?

I **andra** katalogen, *svnlabb2*:
(ii) Gör nu en annan ändring i enrads-kommentaren (annan ändring än vad du gjorde i (i) ovan).
Gör en *svn diff* på filen. Vilken revision jämförs filen med och varför?
Gör inte `svn update` men kommitta! Vad händer? Kör följande:

```
~/svnlabb $ ls -l
```

Nu kan du göra en `svn update`. Kolla vilka filer du har med *ls*.
Nu har vi fått en konflikt – det vill säga vi har en ändring lokalt i vår arbetskatalog som inte kan slås ihop med ändring från repositoryt. Lös problemet! Ta hjälp av subversionboken och `svn help`.
Du ska se till att kommitta den senaste versionen av *hello.py* du gjorde (ändringen du gjorde i (ii) ovan).

Moment 10 `svn blame`

Om man vill se vem som är ansvarig för vad (radvis) i en fil använder man `svn blame`. Kolla upp kommandot i svn-boken och `svn help`. Testa på någon fil. Det här kommandot blir mer intressant när fler personer är involverade – till exempel när du kommer labba tillsammans med någon annan.

Lite mer

`svn import`
`svn export`

Moment 11 `svn import`

Om man redan har befintliga filer och kataloger som man vill revisionshantera kan man använda `svn import` för att importera en trädstruktur till ett repository.
Om du vill testa detta så kan du först skapa en katalog `labbar` i katalogen där du först skapade katalogen `svnlabb` (KURS ovan rad 3) och sedan i den skapar kataloger för respektive labb, till exempel med följande kommande:

```
mkdir -p labbar/labb{1..6}
```

så kan du sen importera denna trädstruktur. Du kan även lägga någon fil där om du vill testa och till exempel använda kommandot `touch`.

Moment 12 `svn export`

Om du vill ha en kopia av en revisionshanterad katalog utan att få med svn-data (.svn) så används `svn export`. Det är ett bra kommando om man till exempel vill göra ett paket av sitt program för distribution. Man kan dels ”exportera” direkt från ett repository dels från en arbetskopia.

> (14) Exportera din svnlabb till en katalog. Gör sedan en packad tarboll av katalogen (man tar) som du lägger tillgänglig i din www-katalog på ida. Se till så att filen är läsbar och mejla labbass länken med ”TDP002” i ärenderaden.
>
> (15) Flytta eller kopiera (valfritt) din fil med svar på frågorna `svn-svar` till svnlabb-katalogen och kommitta filen.
>
> (16) Mejla labbass om att du är klar med ärenderaden ”TDP002: svnlabb klar!”.

Tips 3: svn config, kinit

Ett tips för att få svn att ignorera vissa filer, till exempel alla pythons bytecode-filer (*.*pyc*), är att kolla i filen '~/.subversion/config'. Man kan även använda sig av så kallade *properties* (se svn-boken).
Konfigurationsfilen är välkommenterad och med exempel. Här kan man även ställa in så att subversion inte sparar lösenord man skriver in utan frågar varje gång. På IDA används kerberos som autentisering och har man en ”kerberosbiljett” används den istället för lösenord (kerberos ticket). Tips: 'man kinit', 'aptitude search ~dkerberos' (kerberosautentisering kan vara snabbare att använda).