# Från CVS till SVN - en utvärdering av SVN

Kalle Benéus och Johan Fänge
{d04kb, d04jfa}@student.lth.se

20 februari 2007

**Abstract**

This paper aims to examine the Subversion software configuration management tool and compare it to the Concurrent Versions System. We try to give an idea of the benefits and disadvantages in switching from Concurrent Versions System to Subversion, with a focus on how the tools are used in the course called "Programvaruutveckling i grupp" at the Faculty of Engineering at Lund University. We try to present this information both from the viewpoint of the users and from that of the administrator.

# Innehåll

# 1 Inledning

Denna rapport ämnar undersöka konfigurationshanteringsverktyget Subversion (SVN) och vad det skulle innebära att byta till detta från Concurrent Versions System (CVS). Mer specifikt undersöker vi huruvida SVN verkar vara fördelaktigt att använda som standard i kursen Programvaruutveckling i Grupp (PVG) som ges på Lunds Tekniska Högskola (LTH).

Frågan är då ifall studenterna som går kursen har någon användning av de nya funktionerna som erbjuds av SVN. Den andra sidan av det hela är hur systemet ser ut från administratörens synvinkel. För närvarande finns en fullt fungerande installation av CVS på LTH:s servrar som bevisligen har fungerat bra. Det finns därför flera faktorer inblandade i ett eventuellt byte av system, bland annat hur svårt det nya systemet är att konfigurera och underhålla.

Till att börja med vill vi sammanställa en jämförelse mellan funktionerna i de båda programmen både från användarens och administratörens synvinkel. I denna del kommer vi att fokusera på användning som den ser ut i PVG-kursen. Därefter kommer en sektion med en undersökning av vad de studenter som läser PVG-kursen tycker om de båda verktygen. Med detta som bas skall vi slutligen göra en rekommendation för huruvida ett byte till SVN är motiverat.

# 2 Bakgrund

I ett XP-projekt är en av de mest väsentliga delarna att man har tillgång till ett välfungerande verktyg för konfigurationshantering, där filer versionshanteras i en gemensam lagringsplats — ett repository[1]. Detta kan bli en avgörande faktor för huruvida projektet lyckas eller inte. I dagsläget är det vanligast att man använder sig av open source-verktyget Concurrent Versions System, åtminstone i mindre projekt. Idag finns det dock ett alternativ till CVS som är på stark uppgång och kan vara värt att se över.

Subversion, även det open source, skapades ursprungligen med avsikt att göra en bättre version av CVS. Man har här sett över flera saker som man ansåg saknades i CVS, däribland möjligheten att versionshantera kataloger samt ett nytt system för att hantera versioner av filerna. Man har även som delmål att göra det lätt för en användare att byta från CVS till SVN.

# 3 SVN vs CVS, ett användarperspektiv

## 3.1 Överblick

SVN gjordes som sagt med avsikt att förbättra CVS. Målet var dock inte att revolutionera konfigurationhantering, utan snarare att ta ett överlag bra program och göra det bättre. Därmed är möjligheterna och det grundläggande systemet mycket likt det som man finner i CVS, varför övergången mellan de båda för en användare är relativt smärfri. SVN har bland annat som filosofi att om en funktion redan finns i CVS så skall användargränssnittet för denna vara så likt

[1] Vi har valt att kalla repositiriet (med engelskt uttal) för: ett repository, flera repositories, för att efterlikna vårt dagliga språkbruk. Ett enklare alternativ vore att använda: ett repositorie (repositoriet, repositorier, repositorierna osv) som ett svenskt (försvenskat) ord, men detta är vi (ännu) ej helt bekväma med.

som möjligt det som finns i CVS, undantaget att gränsnittet är en av de saker man anser behöver förbättras.

Som användare märker man snabbt av denna filosofi när man kör SVN. Alla de grundläggande funktionerna fungerar som man väntar sig. I Subclipse[2], det SVN-plugin (användargränssnitt) för Eclipse som vanligtvis används i LTH:s kurser, är det mesta är sig likt. Vid första anblicken är den enda skillnaden att menyerna för SVN utnyttjar ikoner mer, vilket är en sund idé kognitionsmässigt.

Skillnaderna är här uppdelade i nya sätt att implementera de från CVS välkända funktionerna samt helt nya funktioner. Det bör nämnas att även det nya i stor utsträckning utgörs av nya sätt att använda gamla funktioner. För en användare som inte besvärar sig med att leta upp alla nyanser kommer denne sannolikt knappt märka någon skillnad mot CVS.

De följande sektionerna kommer gå igenom det nya i SVN samt teoretiskt utvärdera hur användbara de skulle vara i PVG-kursen.

## 3.2 Gamla funktioner, nya implementationer

### 3.2.1 Versionsnummrering

Det mest grundläggande i konfigurationshanteringen för mjukvara är förmågan att hålla reda på tidigare versioner av sitt producerade material. Detta är även det område där den mest grundläggande och viktiga skillnaden mellan CVS och SVN förekommer. CVS hanterar versioner på filnivå. Det vill säga att varje fil hanteras separat och får ett eget versionsnummer. Detta kan innebära vissa problem vad gäller överblick över commits, då det inte finns någon inneboende koppling mellan olika filer som ändrades och lades upp samtidigt. Man måste då vända sig till commit-kommentarer för att se kopplingarna, vilket sällan är tillförlitligt nog.

I SVN har man istället valt att låta varje enskilt tillägg — varje utförd commit — innebära att hela projektet får ett nytt versionsnummer (man numrerar helt enkelt commits som 1,2,3...). Detta gör att man enkelt kan se vilka filer som lades upp samtidigt, och därmed underlättas letandet efter sammankopplade ändringar. Det underlättar också återgången till ett stadie då allt fungerade, då detta inte längre behöver göras på filbasis. Istället räcker det med att ange ett revisionsnummer då systemet fungerade.

Under PVG-kursen är utvecklarna, förståeligt nog, på en rudimentär nivå när det gäller både XP och konfigurationshantering. Generellt sett sker inte i någon större utsträckning att man går tillbaka i historien för att lista ut vad som ändrats. Dock kommer man någon gång sannolikt att behöva göra det, till exempel efter en större spike på refaktorisering, och i detta läge skulle denna funktionalitet underlätta betydligt. Detta särskilt när man får problem — "Hur gjorde vi tidigare?", "När fungerade x senast?". Med revisionsnummer blir det enklare att se hur systemet förändrats. Väljer man "Show History" får man för en fil eller en mapp eller hela projektet en praktisk lista med alla commits som

[2]Ett annat SVN-plugin till Eclipse är Subversive, som flera anser vara mer lättanvänt. Vi har tittat på och använt Subclipse, som är det plugin med längst historia. Eclipse-projektet är intresserat av att inkludera ett SVN-plugin i sin standarddistribution — precis som det har ett väldigt bra CVS-plugin — men det är idag inte helt klart vilket projekt man kommer välja som bas för detta. När ett SVN-plugin väl har integrerats i Eclipses standarddistribution kommer detta säkerligen vara det som mest används och dit mest utvecklarresurer fokuseras.

gjorts och vilka filer de påverkade. Se sist i appendix för exempel på vilken information man får med Show History.

Överlag får man aningen bättre kontroll över historiken, även om det är tveksamt hur stor skillnad det egentligen gör. Men, bara det faktum att SVN underlättar ett korrekt beteende är anmärkningsvärt och bör tas i beaktning.

Det nya systemet påverkar flera andra aspekter av SVN. Detta tas upp ytterligare i sektion 3.2.4 och 3.3.1.

### 3.2.2 Commit

Denna mycket grundläggande funktion, som ibland kallas kallas check-in i andra system, har ändrats på ett signifikanta sätt. Detta är att en commit i SVN är en atomisk operation, vilket innebär att den sker enligt en princip av "allt eller inget". I CVS är det möjligt att en commit-operation avbryts innan den färdigställts, vilket kan leda till problem. I SVN kan man dock vara säker på att alla filer man lägger upp faktiskt kommer upp tillsammans.

Detta är helt klart fördel för SVN, men samtidigt skall sägas att detta ytterst sällan faktiskt blir ett problem i CVS. Påverkan för PVG skulle vara mycket låg.

### 3.2.3 Branches

Branches i SVN och CVS är implementerade på vitt skilda sätt, men funktionaliteten är i grund och botten densamma. SVN har dock gått den enkla vägen och löser det hela genom att utnyttja en simpel kopieringsfunktion. Att skapa en ny branch i SVN innebär att man kopierar projektet till en mapp i branchkatalogen, där man kan fortsätta jobba på den som vanligt genom att ändra vilken katalog SVN kommunicerar med. SVN har alltså ersatt CVS:s helt verktygsstödda funktion branches med en funktion med simplare verktygsstöd och mer konventioner. Detta sätt att lösa problemet på ökar flexibiliteten har också en fördel i att det är mycket enkelt för användaren att förstå sig på.

Detta är dock ett ämne som argumenteras mycket bland användare av de olika systemen. Inbitna CVS-användare har ofta svårt att ta till sig tankesättet att man kan skapa en branch bara genom att kopiera sina filer till en ny katalog.

Tack vare en enklare struktur skulle SVN kunna uppmuntra till att branches används i större utsträckning, till exempel när man skall göra en release. Man är inte lika rädd för att göra något fel i SVN då metoden är simpel och hanteringen av branches i allmänhet är mer flexibel. Man kan t.ex. utan problem radera en felaktig eller utdaterad branch (men med möjlighet att återskapa den senare). Denna möjlighet har även en stor fördel i att den underlättar översikten över sitt repository. Än en gång vill vi att det uppmärksammas hur SVN i och med detta uppmuntrar till ett mer effektivt användande av konfigurationshantering.

### 3.2.4 Tags

Ett ämne relaterat till branches är hur SVN hanterar tags — en fryst version av repositoriet, där man valt ut specifika versioner av filer. Denna funktion är nämligen i princip helt ersatt av det nya versionssystemet, där man kan hålla reda på ett revisionsnummer istället för att explicit skapa en tag. Detta har dock nackdelen att man inte kan ge sin tag ett specifikt namn för att underlätta kommunikationen. Den lösning för detta som förspråkas är en konvention att ha

en mapp som heter tags. När man ska skapa en ny tag kopierar man helt enkelt alla filer i den aktuella revisionen dit, med namn valt efter behag. Denna kopia är dock på intet sätt fryst som man kunde önska, utan fungerar snarare som en branch som läggs i en annan huvudkatalog. Detta kan förstås vara en fördel om man behöver uppdatera vad som ska ingå i en tag, men då har man väl å andra sidan också omdefinierat betydelsen av en tag. En annan fördel med detta över CVS variant är dock att det hela går mycket snabbt i SVN medan CVS, i större projekt, kan ta en ibland oacceptabelt lång tid för att utföra processen.

För PVG gäller här i princip samma förhållanden som de som tidigare beskrevs för branches. Den enkla och lättförstådda arkitekturen är väl anpassad för nybörjare. För tags fungerar det tyvärr inte riktigt lika bra då man här faktiskt saknar en del av det som vanligtvis utgör en tag.

### 3.2.5 Lagringsutrymme mot bandbredd

SVN har implementerats med filosofin att lagrinsgutrymme är billigare än bandbredd, varför de varit frikostiga med den förstnämnda för att dra ner på den andra. Mer specifikt lagrar SVN alltid en orörd version av varje fil på servern lokalt. Detta innebär att man kan utföra en diff utan att behöva kontakta SVN-servern, och likaså återgå till den tidigare versionen av filen med kommandot revert. På detta sätt minskar man de antal gånger som servern behöver kontaktas under utvecklingen. Nackdelen är då förståss att de utcheckade filerna tar upp dubbelt så mycket utrymme på den lokala disken.

Om detta är en fördel eller en nackdel beror på vilken av de ovan nämnda parametrarna som prioriteras högst. I fallet för PVG-kursen borde det räknas som en smärre fördel, då projekten aldrig blir så stora att de tar upp en på något sätt signifikant mängd utrymme.

## 3.3 Nya funktioner i SVN

### 3.3.1 Stöd för kataloger och binärfiler

Den kanske viktigaste nya funktionaliteten i SVN, allmänt sett, är dess förmåga att stödja både kataloger och binärfiler, något som CVS saknar. Mycket tack vare det nya versionsnummersystemet kan man nu tala om att en katalog har ett visst revisionsnummer. Detta innebär framförallt ökad flexibilitet i hur man hanterar sin katalogstruktur.

I CVS är en katalog inte mer än en del av en eller flera filers sparade namn. För att ta bort en mapp krävs där att man får radera alla filer som ingår i den. Även efter detta kommer mappen att ligga kvar i översikten av CVS repository, vilket efterhand gör det svårt att navigera. SVN ger där mycket större frihet att ordna till det efter behag. Potentiellt farliga kommandon som delete tillåts tack vare vissheten att man alltid kan plocka fram en gammal version även av hela kataloger.

Detta är kanske i grunden en liten sak, men som helhet resulterar detta i att det blir lättare att hantera sitt repository. Tack vare den ökade mängden information man kan få ut av systemet och de utökade hanteringsmöjligheterna är det lättare att våga testa sig fram istället för att bli fast i rädslan för att göra fel. Detta är en mycket positiv sak för nybörjare på området, vilket de flesta i PVG-kursen är. Jag skulle vilja framhålla detta som en tydlig fördel för SVN.

### 3.3.2 Move och rename

En av de användarmässigt viktigaste nya funktionerna i SVN är möjligheten att ge versionshanteringsstöd även för filer som man flyttar eller döper om. Ett vanligt klagomål på CVS är att dessa två grundläggande funktioner inte har ett ordentligt stöd. Det har länge varit så att en omdöpt fil i CVS förlorar sin historik. Detta har delvis åtgärdats på senare tid, men det saknar ännu fullgott stöd.

Move och rename är två av de viktigaste kommandona vid refaktorisering av kod. Att tro att alla klasser man skapar redan har optimalt namn och position är naivt, varför man kan förutsätta att dessa kommandon kommer behövas. Det är här SVN skiner, då båda dessa stöds där med all historik intakt. I Subclipse har stödet för funktionerna integrerats med Eclipse vanliga gränssnitt.

I PVG-kursen kan man förutsätta att utvecklarna kommer behöva både flytta och döpa om sina filer. Vad som dock inte är uppenbart är huruvida förlusten av en fils historia skulle ha överdrivet stor påverkan i nuläget. Filhistoriken används i mycket liten utsträckning, och även då är det än mer sällan som en utvecklare behöver gå mer än ett eller två steg tillbaka. Det är därmed osannolikt att bristen i CVS har särskilt stor slagkraft i kursen. Däremot vill jag framhålla att i PVG-kursen finns ett fokus på både korrekt konfigurationshantering och en hög grad av refaktoriseringar. Här har SVN bäst stöd för bägge.

# 4 SVN för administratörer

## 4.1 CVS till SVN med bibehållen data

Har man redan en mängd versionshanterad data i ett CVS-repository man vill konvertera kan man använda sig av `cvs2svn`, vilket är ett Pythonskript gjort för att konvertera ett helt CVS-repository till ett SVN-repository. Vi har inte tittat på detta, då det inte tycks särskilt aktuellt för kursen i fråga. I kursen börjar man om från början varje år, med tomma repositories.

## 4.2 FSFS och Berkeley DB

Det finns två format för repositories, FSFS och Berkeley DB. Berkeley DB är baserat på en Berkeley DB-databas och var det ursprungliga filsystemet. Det har en del problem, exempelvis går det inte köra på en nätverksenhet, och det går inte att flytta en mapp med en repository mellan olika processorarkitekturer. FSFS är nyare och simplare. Det är filbaserat, där en ny fil skapas för varje revision, och det är inga problem att flytta ett repository eller använda något nätverskfilsystem. FSFS verkar i allmänhet vara det alternativ som rekommenderas idag.

## 4.3 Installationsmöjligheter

SVN kan installeras på flera olika sätt. Vilket man bör välja beror på hur stort projekt man har, vilken tidigare infrastruktur man har och vad man har för krav på autentisering och konfidentialitet (d.v.s. kryptering). Idag finns i huvudsak 3 sätt att komma åt ett repository: Som en separat serverprocess, som en modul i en Apache-server eller som en mapp på en (ev. delad) hårddisk.

### 4.3.1 Apache och `mod_dav_svn`

Apache i kombination med SVN-modulen `mod_dav_svn` är den metod som ger mest möjlighet till autentisering (authentication) och behörighetskontroll (authorization). Man har här i princip tillgång till allt som redan finns till Apache, och man kan integrera repositories med andra redan existerande websystem. Man ansluter via WebDAV (en utökning av HTTP), och tack vare detta kan man komma åt senaste versionen av ett repository via en vanlig webläsare[3]. WebDAV gör dock också åtkomsten till ett repository märkbart långsammare. Adressen till ett repository ser ut som "`http://server/repository`" eller som "`https://server/repository`". Inte oväntat är denna installation också den som är besvärligast att konfigurera.

För PVG-kursen känns detta alternativ som mer avancerat än nödvändigt, samtidigt som det kräver mer konfiguration. Det är tveksamt om de ökade möjligheterna tillför något till kursen. Vill man ha webgränssnitt, så kräver ViewVC heller inte fysisk tillgång till ett repository. Den extra konfiguration som krävs är dock i regel per server och inte per repository, varför det är mer av ett engångsproblem.

### 4.3.2 `svnserve`

Motsvarigheten till pserver-protokollet i CVS kallas i SVN helt enkelt `svn`. Det stödjs av `svnserve` som är en simpel server som kör som en separat process, med ett eget protokoll. Den konfigureras genom en handful filer, alla med samma relativt simpla format, som ligger i mappen conf i ett repository. Man kan ange användare och lösenord (dessa ligger listade i en fil, i plaintext). Man kan även ange generella eller specifika åtkomsträttigheter till olika delar av ett repository (dessa listas i en annan fil, efter sökväg).

Detta alternativ ger snabbast åtkomst till repository, men erbjuder ingen möjlighet till kryptering, även om själva lösenordet aldrig skickas över nätverket[4]. Adressen till ett repository ser ut som "`svn://server/repository`". Detta låter som den enklaste lösningen för PVG-kursen. Det är den som är mest lik den CVS-baserade lösning man har idag, och den kräver minst konfiguration.

För PVG-kursen, som behöver ett område per team skulle man kunna tänka sig att antingen lägga allting i samma repository, eller att ha ett eget repository per team. Ett gigantiskt repository för alla team skulle innebära att konfigurering blir enklare, och att det blir lättare att få översikt över alla teams mappar om det behövs. Dessa ligger då helt enkelt synliga (om än ej nödvändigtvis läsbara för andra team) som underkataloger till repository-rooten. Å andra sidan skulle teamen få svårare att räkna antalet commits p.g.a. systemet med gemensamma revisionsnummer för hela repositoriet. Med flera repositories slipper man dessa problem, men konfigurering blir besvärligare. Den går dock att automatisera utan större svårigheter. Eventuella andra problem kan också tänkas slå mer lokalt för ett team.

[3]Detta interface är dock väldigt simpelt. Det finns mer avancerade och kompletta alternativ så som ViewVC om ett webinterface önskas, `www.viewvc.org`.

[4]Det protokoll för autentisering som `svnserve` stödjer idag är CRAM-MD5. Kortfattat innebär detta att servern skickar en challange i form av en mängd slumpmässig data (nonce) som klienten tillsammans med lösenordet skickar genom en hashfunktion, varpå resultatet skickas till servern som gör samma sak själv och sen jämför svaren.

#### 4.3.3 `svn+ssh`

Motsvarigheten till CVS:s extssh (någon okrypterad version som ext finns ej) är svn+ssh. I detta alternativ ansluter man till datorn med SVN-servern via ssh. Detta innebär att SVN loggar in på servern med ssh och kör igång svnserve som modifierar repositoryfilerna direkt, och därefter loggar ut igen. Det krävs alltså att alla användare av har ett konto på servern de vill kontakta. Skriv och läsrättigheter bestäms av vilka rättigheter användaren har till filerna på systemet, och hanteras alltså inte av SVN. Adressen till ett repository ser i regel ut som "`svn+ssh://server/repository`".

Detta alternativ är mer decentraliserat och kräver alltså inte någon dedikerad server, till skillnad från de andra alternativen. Det räcker med att användarna har tillgång till ett ssh-konto där man kan komma åt repositoriet. Underhåll av rättigheter och dylikt överförs i princip från den som sköter SVN-servern till den som sköter SSH-servern, vilket kan vara positivt om man redan har ett system konfigurerat med grupper och liknande färdigt, men kan också tänkas innebära en flaskhals då SSH-admininstration oftare sker på central nivå (med medföljande byråkrati).

Detta alternativ ger också användarna mer kontroll över sitt eget repository. Man har därmed ökad möjlighet att utföra mer avancerade operationer på repositoriet (t.ex. göra en dump), men detta kan också vara riskabelt om man inte är försiktig. Exempelvis kan en användare råka ta bort ett helt repository, varvid all historik[5] skulle gå förlorad (men detta kan man förstås inte göra om man inte loggar in på kontot via en vanlig ssh-klient). Ett vanligt problem med detta alternativ, särskilt för Berkeley DB, är skriv- och läsrättigheter till filer i repositoriet. Man bör försäkra sig om att man hanterar detta korrekt och ger nya filer rättigheter så att andra kan läsa och skriva till repositoriet.

I PVG-kursen borde detta alternativ fungera relativt bra, och kanske underlätta underhållet för kursledningen genom att undvika en dedikerad server. Emellertid är verktygsstödet för detta alternativ inte lika välutvecklat som för de andra. (Särskilt på Windows hade vi problem med PuTTY, som var icke-trivialt att konfigurera tillsammans med SVN och Eclipse.) Det är därmed möjligt att det blir svårare för utvecklarna att komma åt repositoriet hemifrån. Den ökade kontrollen teamen får över sitt repository skulle också kunna innebära att repositorierna inte blir lika enhetligt installerade, eller att de får underlig layout. Det är exempelvis inte otänkbart att man råkar skapa ett repository inuti ett annat repository, något som inte egentligen behöver ställa till några problem, men som skulle göra det svårare för coacher att få en bra översikt. Kort sagt finns det mer utrymme för besvärliga problem, vilket kräver god kommunikation.

## 5 Utvärdering från studenterna

Under PVG-kursen vårterminen 2007 lät vi studenterna använda SVN för sin konfigurationshantering istället för CVS, som är det vanliga alternativet. Detta i syfte att få höra deras reaktioner på verktyget och få en känsla av till hur stor användning SVN:s funktioner kan vara. Tyvärr hade de flesta av dem inte tidigare använt CVS, vilket omöjliggjorde en direkt jämförelse ur deras perspektiv.

[5]Det är osannolikt att mer än historik skulle gå förlorad då alla har en lokalt utcheckad version av repositoriet i sin workspace.

Samtliga ansåg att SVN har varit lätt att arbeta med under kursen, och åsikterna var överlag mycket positiva. För ungefär hälften av studenterna gällde att de inte hade använt sig av history alls, eller enbart vid ett fåtal tillfällen. De tillhör därmed den grupp som med största sannolikhet inte har haft någon större fördel av SVN över CVS. Vi fick dock kommentaren att branches mycket riktigt var väldigt enkla att arbeta med.

Den andra halvan av studenterna, en förvånande stor siffra, använde sig mycket av kommandot "show history" och tyckte att detta gav en mycket bra överblick i SVN. De reagerade negativt på CVS implementation då vi beskrev hur history fungerar där, men det är inte helt uteslutet att vi kan ha påverkat dem till denna slutsats. Studenterna uttryckte även åsikten att det var en viktig fördel med SVN att den stödde move och rename.

Det gavs även vissa negativa kommentarer om SVN. Dessa var framförallt relaterade till installationen. En av studenterna som själv installerat en SVN-server ansåg inte att manualen var tillräckligt tydlig, framförallt i beskrivningen av vilket filsystem som bör användas. Det fanns även en allmän skepsis till kommandot "cleanup", som i praktiken tycks vara något av ett magisk kommando som löser de problem som ibland uppstår med servern.[6] Denna skulle i användargränsnittet behöva en tydligare förklaring och motivering, då den skär sig med den annars så transparenta implementationen. Det är inte intuivt uppenbart vad den gör, varför den behövs och framförallt inte varför den inte kan köra det själv om det nu behövs.

# 6 Sammanfattning och slutsatser

Ur en användares perspektiv, då specifikt en användare i PVG-kursen, innebär SVN nästan bara fördelar. De flesta som går kursen har ingen tidigare vana av konfigurationshanteringsverktyg, men även för de som har det (särskilt då dessa oftast har erfarenhet med CVS) är SVN enkelt att komma in i. Detta tack vare sin allmäna likhet till CVS samt det välfungerande användargränssnittet i Subclipse. CVS har dock två utpräglade fördelar över SVN. Den första är att tags fungerar så som de skall, d.v.s. att de faktiskt utgör en frysning av koden. Den andra är att CVS ännu är det mest populära konfigurationshanteringsverktyget för mindre projekt, varför det kan finnas en poäng med att det är detta man får lära sig. Det är dock värt att notera att SVN är på uppgång, så det är inte uteslutet att detta förhållande kan komma att vända.

Administrativt sett skiljer det sig inte mer än i detaljerna. Ett repository är annorlunda uppbyggt, men är fortfarande bara en mapp med allting. I SVN finns stöd för mer detaljerad kontroll av läs- och skrivrättigheter per katalog, till skillnad från CVS, men detta är inte något som vi förväntar oss ha någon direkt använding av i kursen (särskilt i.o.m. Collective Code Ownership).

Som nämnts tidigare i rapporten kommer de flesta av SVN:s fördelar var för sig troligen endast i mycket ringa grad märkas för utvecklarna i kursen. Sammantaget utgör dessa fördelar dock en viktig poäng med SVN, nämligen hur dess implementation i mångt och mycket är väldigt intuitiv i jämförelse med hur samma sak gjorts i CVS. Det må inte vara nämnvärt lättare att använda,

[6] Vad cleanup egentligen gör är att den går igenom de loggar som svn skapar innan den gör något och återställer låsta filer, och kan behöva köras om något kommando inte lyckas köra färdigt.

men det mesta är enklare att förstå för en nybörjare. Mer förståelse medför vanligtvis att man vågar använda verktyget fullt ut istället för att vara rädd att göra fel. Detta tillsammans med den markant bättre översyn som ges av den nya versionshanteringen är i vår mening skäl nog, användarmässigt, för att byta till SVN.

# 7 Referenser


- Version Control with Subversion http://svnbook.red-bean.com/ By Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato First Edition June 2004, O'Reilly Media
- CVS Manual, http://ximbiot.com/cvs/wiki/index.php, 2007-02-09
- Advantages Of SubVersion Over CVS When Working With XP Projects, Marcus Eliasson, http://www.cs.lth.se/EDA270/Djupstudier/Articles/2004-2005/R-08-SubVersion.pdf, 2007-02-09
- The Top Ten Subversion Tips for CVS Users, Brian W. Fitzpatrick, http://www.onlamp.com/pub/a/onlamp/2004/08/19/subversiontips.html, 2007-02-09
- My Experiences With Subversion, Simon Tatham, http://www.chiark.greenend.org.uk/ sgtatham/svn.html, 2007-02-09
- Subversion Homepage, http://subversion.tigris.org, 2007-02-17
- CVS to Subversion Migration - a war story http://twasink.net/blog/archives/2006/08/cvs_to_subversi.html, 2007-02-09

# A Installation av en simpel SVN-server

## A.1 Konfigurering

Här är ett exempel på hur man kan installera och konfigurera en SVN-server med `svnserve` och två repositories. Exemplet körs på ett UNIX-system.

```
> cd /path/to/repositories
> svnadmin create school --fs-type fsfs
> svnadmin create stuff --fs-type fsfs
> ls
school/  stuff/
> ls school
conf/  dav/  db/  format  hooks/  locks/  README.txt
> svnserve -d -r /path/to/repositories
```

Nu kör `svnserve` på port 3690 och väntar på anslutningar. URL till ett repository kan exempelvis vara "svn://servername/school". Så till inställningar. Inställningar görs i `svnserve.conf`, där man bl.a. anger en fil som innehåller lösenord. Här nere ändrar vi `svnserve.conf` och skapar `passwd`.

```
> cd /path/to/repositories/school/conf
> ls
svnserve.conf
> emacs svnserve.conf
> emacs passwd
> ls
passwd  svnserve.conf
> cat svnserve.conf
[general]
anon-access = none
auth-access = write
password-db = passwd
> cat passwd
[users]
d04jfa=terriblesecretofspace
d04kb=gnu
```

Repositoriet är nu konfigurerat och man arbeta med det, även på en annan dator. För att avsluta installationen bör man se till att `svnserve` startas automatiskt.

## A.2 Användning

Nedan följer exempel på några commits och updates. Här använder vi commandline-verktyget `svn` som följer med SVN. (Detta görs som användaren d04jfa.) Det hade förstås också gått bra att göra det med Subclipse och Eclipse.

```
> cd /some/other/path
> mkdir pvg
> mkdir pvg/trunk
> mkdir pvg/branches
```

```
> mkdir pvg/tags
> svn import pvg svn://localhost/school -m "Creating directory layout"
Authentication realm: <svn://localhost:3690> [...]
Password for 'd04jfa':
Adding         pvg
Adding         pvg/trunk
Adding         pvg/branches
Adding         pvg/tags

Committed revision 1.
```

Nu har vi skapat en grundstruktur för projektet. Av konvention ser strukturen ofta ut såhär, där huvudarbetet sker i `trunk`. Nu checkar vi ut den och börjar arbeta lite:

```
> rm -r pvg
> svn checkout svn://localhost/school/pvg/trunk .
Checked out revision 1.
> ls -a
./  ../  .svn/
> echo "update! update! update! commit!" > bendix.txt
> svn status
?      bendix.txt
> svn add bendix.txt
A         bendix.txt
> svn update
At revision 1.
> svn commit -m "der kampfsangen"
Adding         bendix.txt
Transmitting file data .
Committed revision 2.
> echo don\'t forget the tests! >> bendix.txt
> cat bendix.txt
update! update! update! commit!
don't forget the tests!
> svn update
At revision 2.
> svn status
M      bendix.txt
> svn commit -m "fixed bug in der kampfsangen"
Sending        bendix.txt
Transmitting file data .
Committed revision 3.
> svn update
At revision 3.
> svn status
```

Till slut tittar vi på vad vi gjort. Notera hur SVN visar alla filer som är del av en commit när man tittar på history såhär. Tittar man på en fil eller en mapp visas de revisioner som påverkat denna.

```
> svn log -v
------------------------------------------------------------------------
r3 | d04jfa | 2007-02-18 17:45:12 +0100 (Sun, 18 Feb 2007) | 1 line
Changed paths:
   M /pvg/trunk/bendix.txt

fixed bug in der kampfsangen
------------------------------------------------------------------------
r2 | d04jfa | 2007-02-18 17:40:31 +0100 (Sun, 18 Feb 2007) | 1 line
Changed paths:
   A /pvg/trunk/bendix.txt

der kampfsangen
------------------------------------------------------------------------
r1 | d04jfa | 2007-02-18 17:32:06 +0100 (Sun, 18 Feb 2007) | 1 line
Changed paths:
   A /pvg
   A /pvg/branches
   A /pvg/tags
   A /pvg/trunk

Creating directory layout
------------------------------------------------------------------------
```