

# 版本控制系統 — SVK

## 何謂版本控制？

版本控制是一種用來管理開發中的模組（module）的一種技巧，如：工程藍圖、程式原始碼、法規文件等。這些模組可能同時被許多人維護。而版本控制透過文檔控制記錄模組中各個區塊的改動，並為每次的更動加上序號，藉此讓模組在開發的過程中，確保不同人編輯的同一檔案都能保持在最新的狀態。

## 為甚麼需要版本控制？

平常若我們寫個像是 Hello World 之類的小程式，程式碼連一千行都不到，維護者也只有一人，說不定這程式還是那種寫了一次就不會再去更動的小東西，這種情況下要套用版本控制就有點大材小用的感覺了。那版本控制到底要用在哪？當你的專案很龐大時（例如程式碼超過上萬行，且排除掉每一行都是 Hello World 的情況），或是有許多人共同維護這個專案，那這時版本控制就派上用場了。想想看，今天程式被發現有個錯誤，在你費盡千辛萬苦以後找到了這個錯誤，在做出相應的修正以後過一個月，發現你上次做的修正造成了其他的問題，可是你早已經忘記你當時是修正哪邊，也忘記你是怎麼修改的，若這時你沒有使用版本控制，那你就只好帶著眼淚去翻那堆程式碼了。但是套用版本控制以後，以上的痛苦發生的機率就可以大大下降，你可以透過版本控制來紀錄你對程式的修改位置以及之間的差異，從每次的變更記錄中找出程式在經過哪次的修改以後從可正常執行變成了不能正常執行（或是相反），加快找出問題所在並解決問題。如果今天這個專案的維護者人數是在兩人以上，那這時版本控制就更顯得必要了，透過檢視變更記錄可以得知何人於何時對程式碼做出何種改變，你就可以不用花時間去修正另一人已經修正的錯誤。

另一種情況是，在開發軟體的過程中會經常需要同時開發兩個版本（一個沒錯誤且沒新功能的版本與有錯誤也有新功能的版本），這時透過版本控制的技巧，將兩個版本分開進行維護，在開發新功能的同時不去影響到現在發佈的穩定版本。

最簡易的版本控制實行方式就是保留軟體不同版本的數份複本，並加上適當的編號。雖然這種方式簡單易用，但卻很沒效率，一來是因為每個複本之間的差異並不大，浪費了許多儲存空間，二來是這種方法需要高度依賴開發者的自我紀律，因此經常導致錯誤的產生。所以現在多將專案的部份或全部交給自動化的版本控制系統，也就是這篇文章所要介紹的。

版本控制的另一個用處是，你可以去取出（Check-Out）一些有 Open Source 的專案（例如：pcmanx 這些專案），透過追蹤每一次的變更記錄來學習人家怎麼寫程式。如果你喜歡使用新功能、或者是喜歡追版本的數字而你又覺得最新的穩定版本或是測試版本仍無法滿足你的慾望，那你可以取出檔案庫（Repository）中最新的版本來使用，當然後果請自負。

## 較知名的版本控制系統與比較

### 一般資訊的比較

Software	Repository model	許可證	價格	網路通訊協定
CVS	主從式	GPL	Free	pserver, ssh
Subversion	主從式	Apache/BSD style	Free	WebDAV, svnserve
SVK	分散式	Artistic/GPL	Free	

以上三種系統皆支援 Unix-like, Windows, Mac OS X 等平台

其中 SVK 在 Windows 上還未有圖形介面的客戶端，其他如 CVS 與 Subversion 都已有與 Windows shell 整合在一起的客戶端（分別是 TortoiseCVS 與 TortoiseSVN）

## 今日的主角 — SVK

SVK 是建立於 Subversion 之上，使用 Perl 寫成的版本控制系統，他透過作為 Subversion 一個用戶端的程式，使用原本 Subversion 的檔案系統，去加強離線使用的功能。例如在使用 Subversion 時，很多工作必須在使用者有連上伺服器時才能進行（例如你想要檢視每次的 revision 所做的變更都必需跟伺服器連線取得資料），且當使用者一將手上的東西上傳以後，這些東西馬上就會變成一個新的版本（revision），但這些東西其實只有改了一半，也就是處在不一定能正常使用的狀態，這些東西如果是上傳到有許多人共用的伺服器上面，則就很容易影響整個系統，可是如果不上傳有時候也會讓修改的人無法進行版本控制，造成工作上的困擾。

有了 SVK 這樣一個 Subversion 的離線用戶端程式，當你在沒有網路的地方時還是可以正常的工作，等到工作告一段落時再透過網路將成果送回到伺服器上。這是透過 SVK 中的映射功能達成的，也就是說這是把工作用伺服器上面的檔案庫整個映射回自己的電腦裡面。也因此可以透過 SVK patch 的功能來製作 patch file，這使得沒有權限存取檔案庫的使用者還是可以在把檔案庫映射下來後在自己電腦上做版本控制，並透過 SVK 產生 patch file 與其它使用者交流。

SVK 還有一個好處，除了 Subversion 的檔案庫以外，他也能映射其它版本控制系統的檔案庫（例如 CVS, Perforce, arch, cvsbk），對於參加多個使用不同版本控制系統的專案的人來說，就不用每次都使用不同的用戶端程式。

最後要提的一點，你可以自行選擇你想要用來維護程式碼（或是其他你想套用版本控制的東西）的程式，而不用受限於某些開發工具內建的、難以使用的版本控制系統。

## SVK 的安裝方式

因為各人所慣用的平台與需求不同，在不同平台上面有不同的安裝方式，請自行參照 SVK 官方網站尋找你適用於你的需求與平台的建議安裝方式。

## SVK 的用法

SVK 剛安裝好以後我們要為他建立一個檔案庫，請使用以下指令

```
svk depotmap --init
```

他會跳出一個訊息問你是否要建立目錄，請選「是」

然後你就可以將遠端的檔案庫映射到自己的電腦上了

透過以下指令初始化映射的檔案庫路徑

```
~$ svk mirror https://opensvn.csie.org/pcman/ //pcman/mirror
```

然後進行同步化，前面的 -s 12 是指從遠端的第 12 個 revision 開始進行同步

```
~$ svk sync -s 12 //pcman/mirror
```

然後就可以建立本地分支（在 Windows 上執行以下指令時請將 ' 替換成 "）

```
~$ svk copy -m 'create local branch for pcman' //pcman/mirror //pcman/local
```

接著從本地分支中取出（Check-Out）一份工作複本（working copy）到當前目錄中

```
~ $ svk checkout //pcman/local/trunk pcman
```

然後你就可以開始對工作複本進行修改了

將工作複本中的更動送交（Commit）至你取出的檔案庫（若無任何更動則就不會進行送交）

```
~ $ cd pcman/
```

```
~/pcman $ svk ci -m 'test commit'
```

當你正在對你本地的的工作複本進行修改時，遠端的檔案庫可能有了更新的版本，為確保你是在對最新的版本進行修改，則要再進行一次同步化的工作

```
~ $ svk sync //pcman/mirror
```

然後要將更新版本中所做的變更合併（Merge）到工作複本中

先加上 -C 來檢查映射與工作複本之間有無衝突（Conflict）

```
~ $ svk smerge -C //pcman/mirror/trunk pcman
```

然後進行合併

```
~ $ svk smerge //pcman/mirror/trunk pcman
```

沒衝突的話就謝天謝地，有衝突的話就要靠自己手動解決，解決完後要告訴 svk 你已經解決衝突

```
~ $ svk resolved pcman/License.txt
```

然後才能進行送交

```
~ $ cd pcman/
```

```
~/pcman $ svk ci -m 'resolved conflict of License.txt'
```

將本地分支裡更新的 revision 取出到工作複本上

```
~ $ cd pcman
```

```
~/pcman $ svk update
```

要將工作成果上傳到工作伺服器上

```
~ $ svk sync //pcman/mirror/
```

```
~ $ svk smerge -C //pcman/local //pcman/mirror
```

```
~ $ svk smerge //pcman/local //pcman/mirror
```

## 相關連結

SVK Homepage

<http://svk.elixus.org/view/HomePage>

Version Control with SVK

<http://svkbook.elixus.org/>

在 Wikipedia 中對版本控制的介紹

<http://zh.wikipedia.org/w/index.php?title=版本控制&variant=zh-tw>