Chapter 14
版本控制進階

2018/12/27

John.Chen

引言

當團隊人數超過了一定的數量，就會很難讓所有人在同一個版本控制程式碼儲存庫上全職工作了。人們會因為失誤而破壞彼此的功能，還經常會有小衝突。

分支的理由有三

1. 為了發佈新版本
2. 調查與研究新特性或進行重構
3. 有較大幅度的修改

版本控制的歷史

1. Concurrent Version System (CVS)

   是開放程式碼工具之一，它把RCS包裝起來，並提供一些額外的功能。

   例如：架構變成使用者端/伺服器模式，並支援較強大的分支和標籤方式。

2. Subversion (SVN)

   是為了克服CVS的缺點而設計，可以說在任何的情況之下，都比CVS更好用。

商業版本控制

1. Perforce

   超好的性能，可擴展性和完美的支援工具。一些真正大型的軟體開發組織都會選擇使用 Perfocre

2. AccuRev

   它提供了類似ClearCase以流為基礎(Stream-based)的開發能力，卻沒有ClearCase如此大量需管理的開銷和如此差勁的效能。

3. BitKeeper
   第一個真正的 DVCS，也是唯一的一個商業工具。

放棄悲觀鎖

悲觀鎖:

一個元件同時只能讓一個人修改，修改前需要先簽出。如果沒有取得鎖，提交操作就會失敗。

樂觀鎖:

允許多人同時在一個檔案中修改，系統會追蹤其控制之下的所有物件的修改，並提交修改時使用一些演算法來合併修改。如果發生衝突，會要求提交人來解決此衝突。

若版本控制系統支援樂觀鎖，就不要使用悲觀鎖。

分支與合併

1. 實體上

   因系統實體設置而分支，即為了檔案、元件和子系統而分支。

2. 功能上

   因系統功能設置而分支，即為特性、邏輯修改、缺陷修復和改善，以及其它可交付的功能而分支。

3. 環境上

   因系統執行環境而分支，即由於建置和執行時平臺、編譯器 開啟視窗系統、函式庫、硬體或作業系統等的不同而分支。

分支與合併

4. 組織上

   因團隊的工作量而分支，即為活動/任務、子專案、角色和群組而分支。

5. 流程上

   因團隊的工作行為而分支，即為支援不同的規章政策、流程和狀態而分支。

合併

- 建立較多的分支來減少每個分支上的修改。例如，每次開發新特性時就建立分支，這是『提早分支』。

- 謹慎的建立分支，可能是發佈前在建立分支。此即為『延後分支』。

分支、流和持續整合 1/2

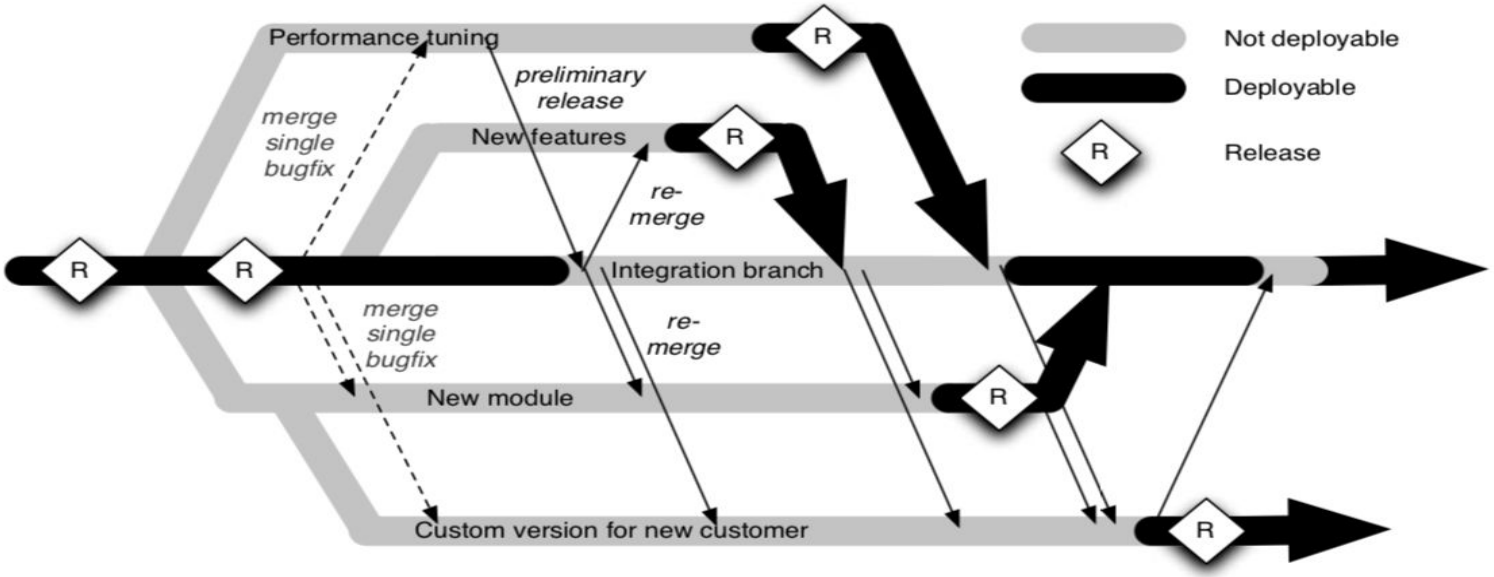


Figure 14.1 *A typical example of poorly controlled branching*

分支、流和持續整合 2/2

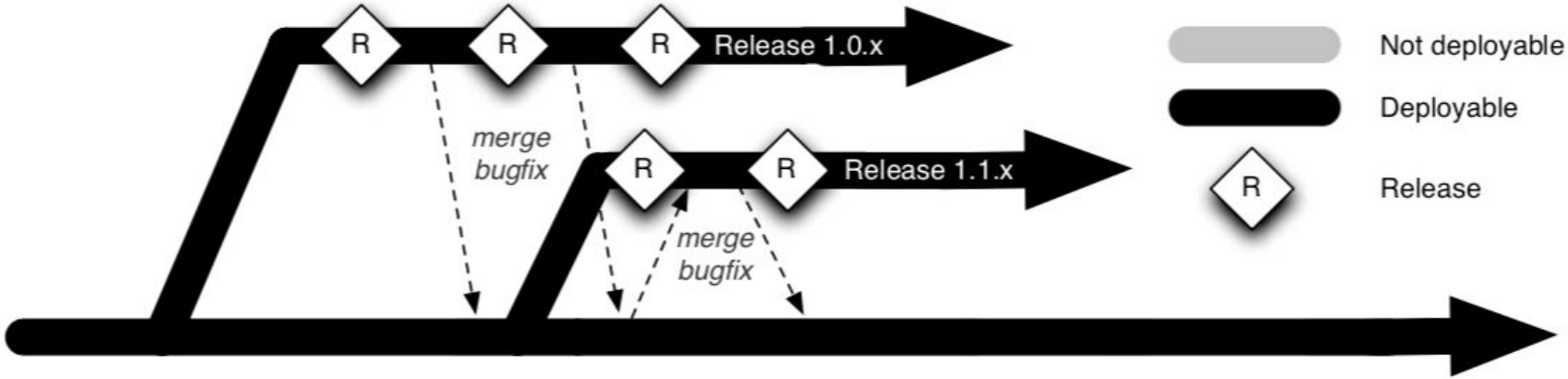


Figure 14.2 *Release branching strategy*

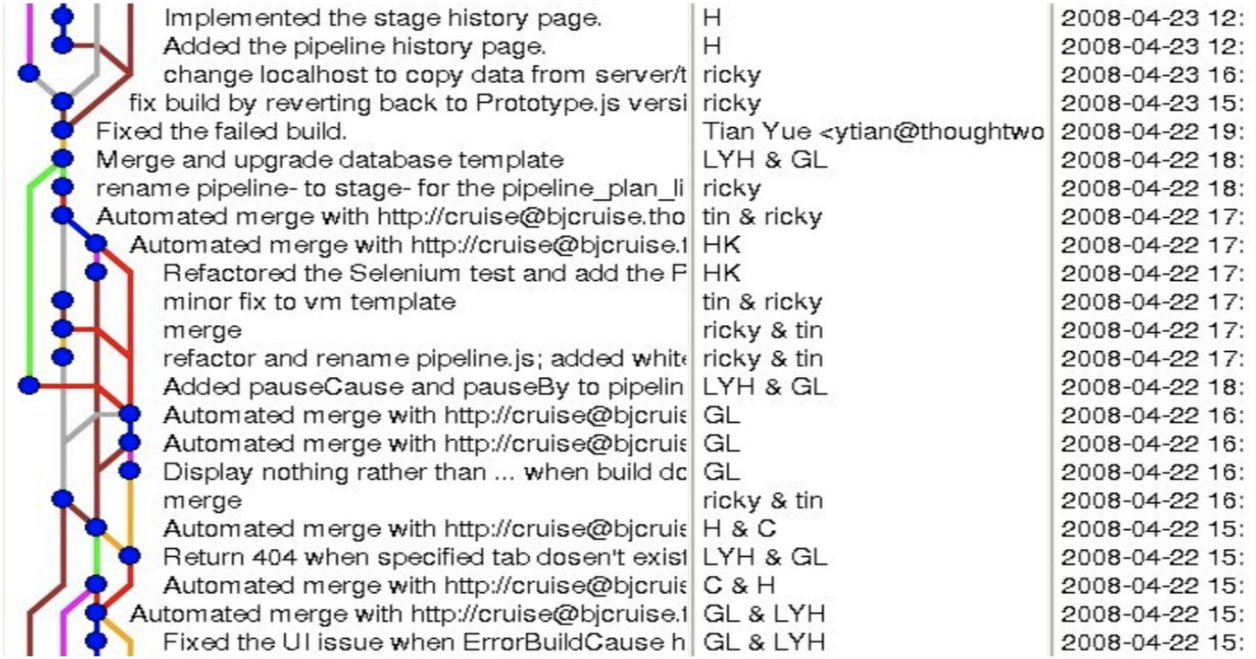
分散式版本控制系統 1/2

DVCS背後的根本性設計原則是，每一個使用者在自己的電腦上都有一個自行包含的一級儲存庫，不需要有個專屬的主儲存庫。

代表作為

1. GitHub
2. BitBucket
3. Google Code

分散式版本控制系統 2/2



Implemented the stage history page.	H	2008-04-23 12:
Added the pipeline history page.	H	2008-04-23 12:
change localhost to copy data from server/t	ricky	2008-04-23 16:
fix build by reverting back to Prototype.js versi	ricky	2008-04-23 15:
Fixed the failed build.	Tian Yue <ytian@thoughtwo	2008-04-22 19:
Merge and upgrade database template	LYH & GL	2008-04-22 18:
rename pipeline- to stage- for the pipeline_plan_li	ricky	2008-04-22 18:
Automated merge with http://cruise@bjcruise.tho	tin & ricky	2008-04-22 17:
Automated merge with http://cruise@bjcruise.t	HK	2008-04-22 17:
Refactored the Selenium test and add the P	HK	2008-04-22 17:
minor fix to vm template	tin & ricky	2008-04-22 17:
merge	ricky & tin	2008-04-22 17:
refactor and rename pipeline.js; added white	ricky & tin	2008-04-22 17:
Added pauseCause and pauseBy to pipelin	LYH & GL	2008-04-22 18:
Automated merge with http://cruise@bjcruis	GL	2008-04-22 16:
Automated merge with http://cruise@bjcruis	GL	2008-04-22 16:
Display nothing rather than ... when build dc	GL	2008-04-22 16:
merge	ricky & tin	2008-04-22 16:
Automated merge with http://cruise@bjcruis	H & C	2008-04-22 15:
Return 404 when specified tab dosen't exist	LYH & GL	2008-04-22 15:
Automated merge with http://cruise@bjcruis	C & H	2008-04-22 15:
Automated merge with http://cruise@bjcruise.t	GL & LYH	2008-04-22 15:
Fixed the UI issue when ErrorBuildCause h	GL & LYH	2008-04-22 15:

Figure 14.3 *Lines of development in a DCVS repository*

企業環境中的分散式版本控制系統

企業反對使用DVCS的意見

1. 集中式版本控制系統在使用者的電腦中只保存了唯一的儲存庫版本，而DVCS則不同，只要有本地儲存庫的副本，即可獲得它的完整歷史。
2. DVCS的王國中，審核與工作流是更不可靠的概念。集中式版本控制系統要求使用者將其全部的變更都簽入中央儲存庫中。而DVCS允許使用者比此交換變更集，甚至允許修改本地儲存庫中的歷史，而不必讓中央系統來追蹤這些修改。
3. Git的確允許修改歷史。而這在受制度監管的企業環境中可能就觸及了『緊戒底線』，但為了記錄全部的內容，就要定期備份儲存庫。

使用分散式版本控制系統

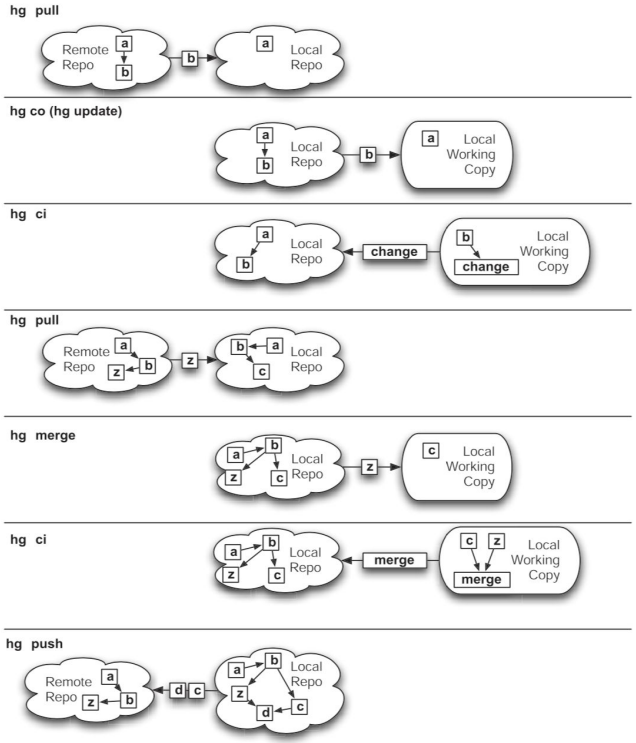


Figure 14.4 *DVCS workflow (diagram by Chris Turner)*

以流為基礎的版本控制系統 1/2

可以把一系列的修改一次性運用到多個分支上，進而減少合併時的麻煩。此種流的方式上，分支被更強大的概年流所代替。其中，最大的區別在於，流之間是可以互相繼承的。因此，某次修改運用指定的流上，所有的子孫都會繼承那些修改。

此方式對以下兩種狀況有幫助

1. 將修復某個缺陷的修補程式運用到軟體的多個版本上。
2. 添加合作廠商函式庫的新版本到程式庫中。

以流為基礎的版本控制系統 2/2

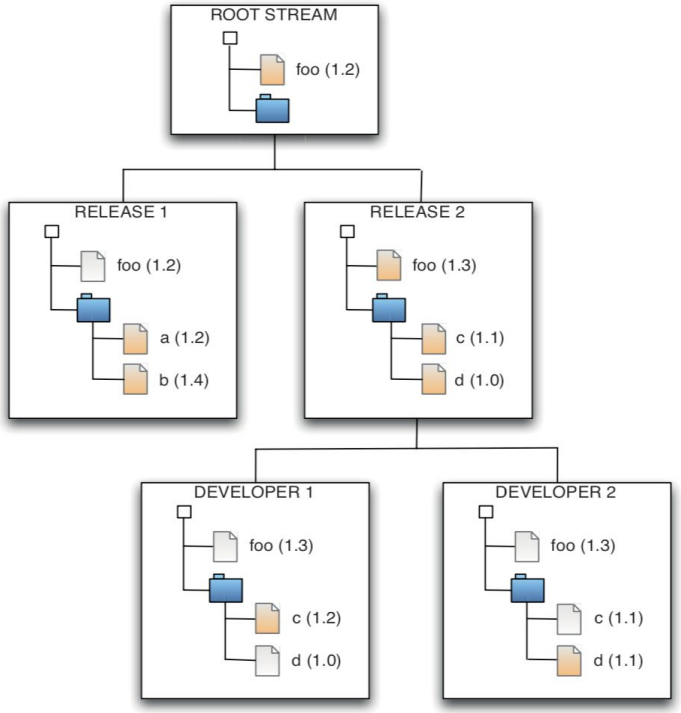


Figure 14.5 *Stream-based development*

主線開發

主線開發有以下三種好處

1. 確保全部的程式碼被持續整合。
2. 確保開發人員即時獲得他人的修改。
3. 避免專案後期的『合併地獄』和『整合地獄』。

缺點

每次簽入主線的並非都是可發佈的狀態

為發佈建立分支

1. 一直在主線上開發新特性。
2. 待發佈版本的所有特性都完成時，且希望繼續開發新特性時才建立分支。
3. 分支上只允許提交那些修復嚴重缺陷的程式，且這些修改必須立即合併回主線。
4. 當執行實際發佈時，此分支可以選擇性的貼標籤。

依特性分支 1/3

此模式的動機是希望能一直保持主線的可發佈狀態。如此一來，全部的開發都會在分支上，而不會被其他人或團隊打擾。

依特性分支 2/3

但有一些前提條件

1. 主線上的所有變更都需每天合併到每個分支。
2. 每個特性分支都應該是短生命週期的，理想情況下應該只有幾天，絕不能超過一次iteration的週期。
3. 活躍分支的數量在任何時間點都應該少於或等於正在開發中的使用者故事數量。除非已把開發的使用者故事合併回主線，否則誰都不能建立新的分支。

依特性分支 3/3

4. 合併到主線之前，該使用者故事應該已由測試人員驗收通過。唯有驗收通過的使用者故事才能合併回主線
5. 重構必須即時合併，才能使合併衝突最小化。此限制非常重要，但也可能非常痛苦，且會限制此模式的使用。
6. 技術負責人的部份職責就是得保證主線維持在可發佈狀態。他應該檢查全部的合併。他有權拒絕可能破壞主線程式的修補程式。

依團隊分支

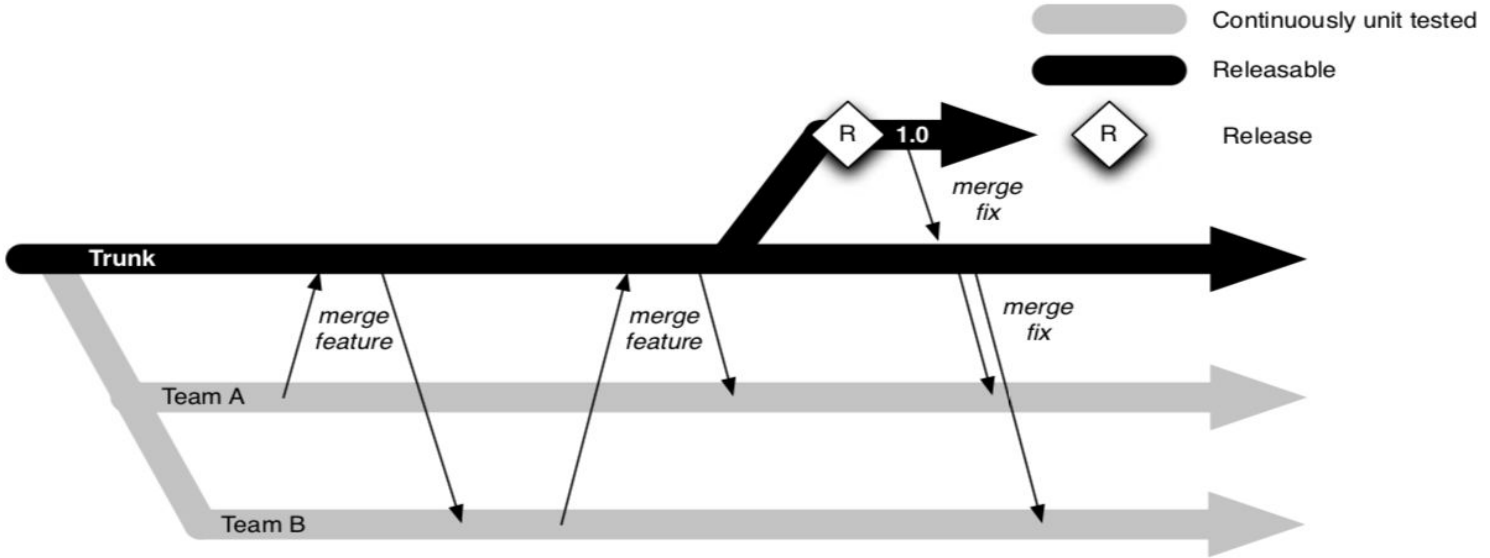


Figure 14.7 *Branch by team*

依團隊分支

1. 建立多個小團隊，每個團隊都有對應的分支。
2. 一旦某個特性或使用者故事完成，就得先穩定該分支，並合併回主線。
3. 每天都應將主線上的變更合併到每個分支上。
4. 對於每個分支，每次簽入後都要執行單元和驗收測試。
5. 每次將分支合併回主線時，都要在主線上執行全部的測試。

小結

1. 現代版本控制系統的複雜性及其良好的可用性使其對於當代以團隊為基礎的軟體開發而言，已具有非常重要的地位。
2. 太差的版本控制實踐是達到『快速且低風險發佈』目標最常見的阻礙之一。
3. 每次建立分支，都要認知到它帶來的成本。此成本的產生在於『增加了風險』，而唯一最小化風險的方式是無論由於什麼理由增加了分支，都要努力保證任何活躍分支能每天合併回主線。若非如此，此流程就不再是持續整合了。