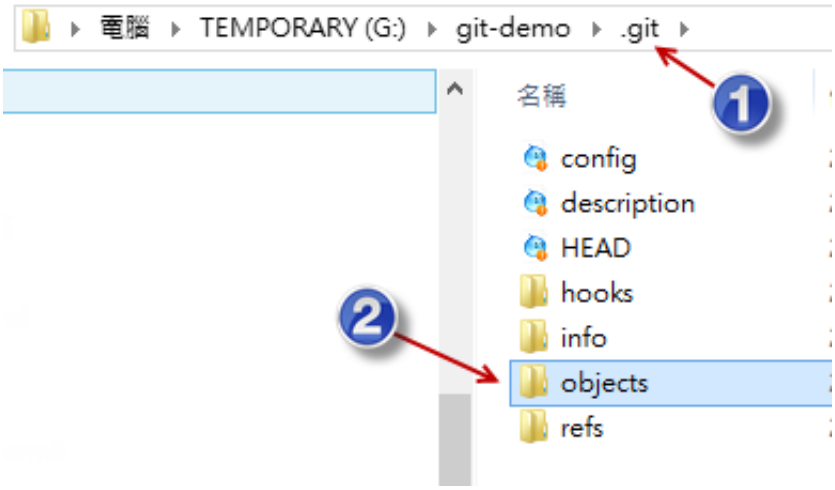
Show

第 06 天：解析 Git 資料結構 - 物件結構

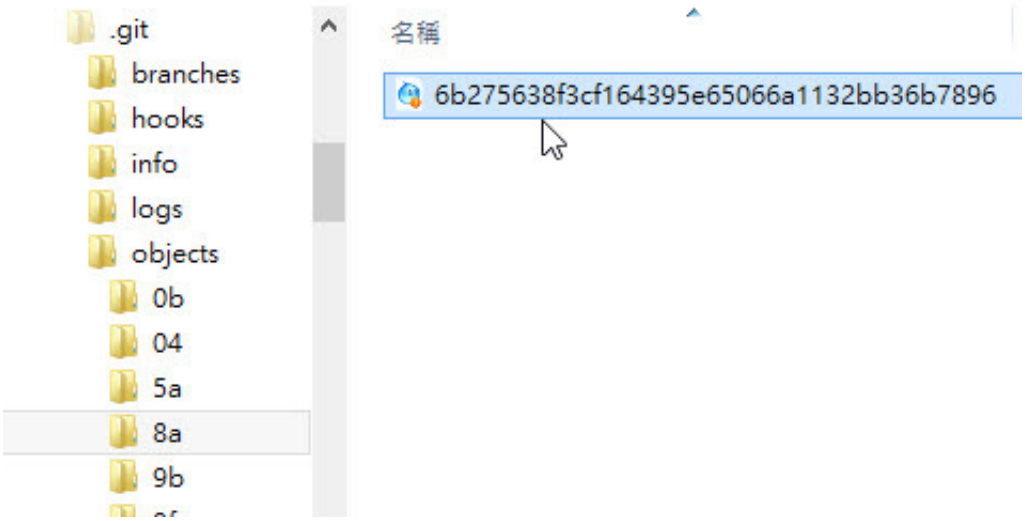
在 Git 的資料結構中，「物件」是一種「不可變的」 (immutable) 檔案類型，所有儲存在「物件儲存區」的檔案通常只進不出，也不會被修改內容。原因在於，如果你竄改了檔案了內容，新的內容所運算出來的 SHA1 雜湊值將會與原有物件的檔名不一樣，這導致 Git 無法繼續執行，相對地也對 Git 儲存庫產生了一定程度的保護作用。本篇文章，將更加仔細地介紹 Git 的物件結構，最後也會透過一則影片，詳細解說整個物件被產生的過程與邏輯。

關於物件資料庫

前一篇文章有提到，無論 blob 物件與 tree 物件，這些都算是物件，這些物件都會儲存在一個所謂的「物件儲存區」 (object storage) 之中，而這個「物件儲存區」預設就在「儲存庫」的 objects 目錄下，如下圖示：



然而 Git 儲存庫中的每一個「物件」，都是以「檔案內容」進行 SHA1 雜湊運算出一個 hash 值，並用這個 hash 值當作物件的名稱 (檔名)。我們以 8a6b275638f3cf164395e65066a1132bb36b7896 為例，Git 會先拿前兩個字元(8a)當作目錄，然後把剩下的 hash 值當成檔名 (6b275638f3cf164395e65066a1132bb36b7896)，這些物件的實體目錄與檔案也都會放在 .git\objects 目錄下，如下圖示：



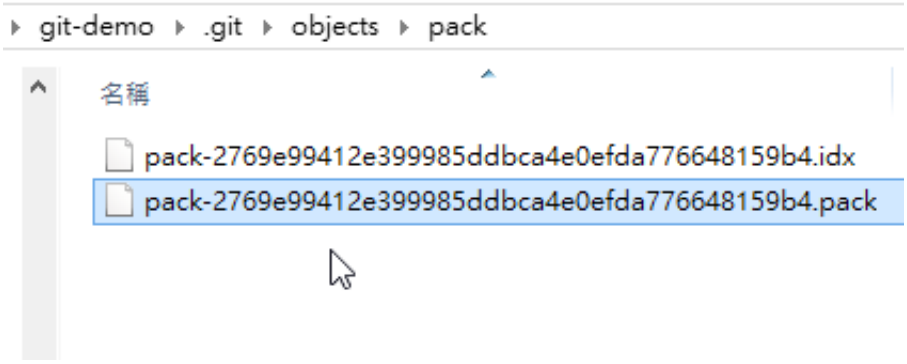
物件的類型

在這些「物件資料庫」裡面，又包含了 4 種物件類型，分別是：

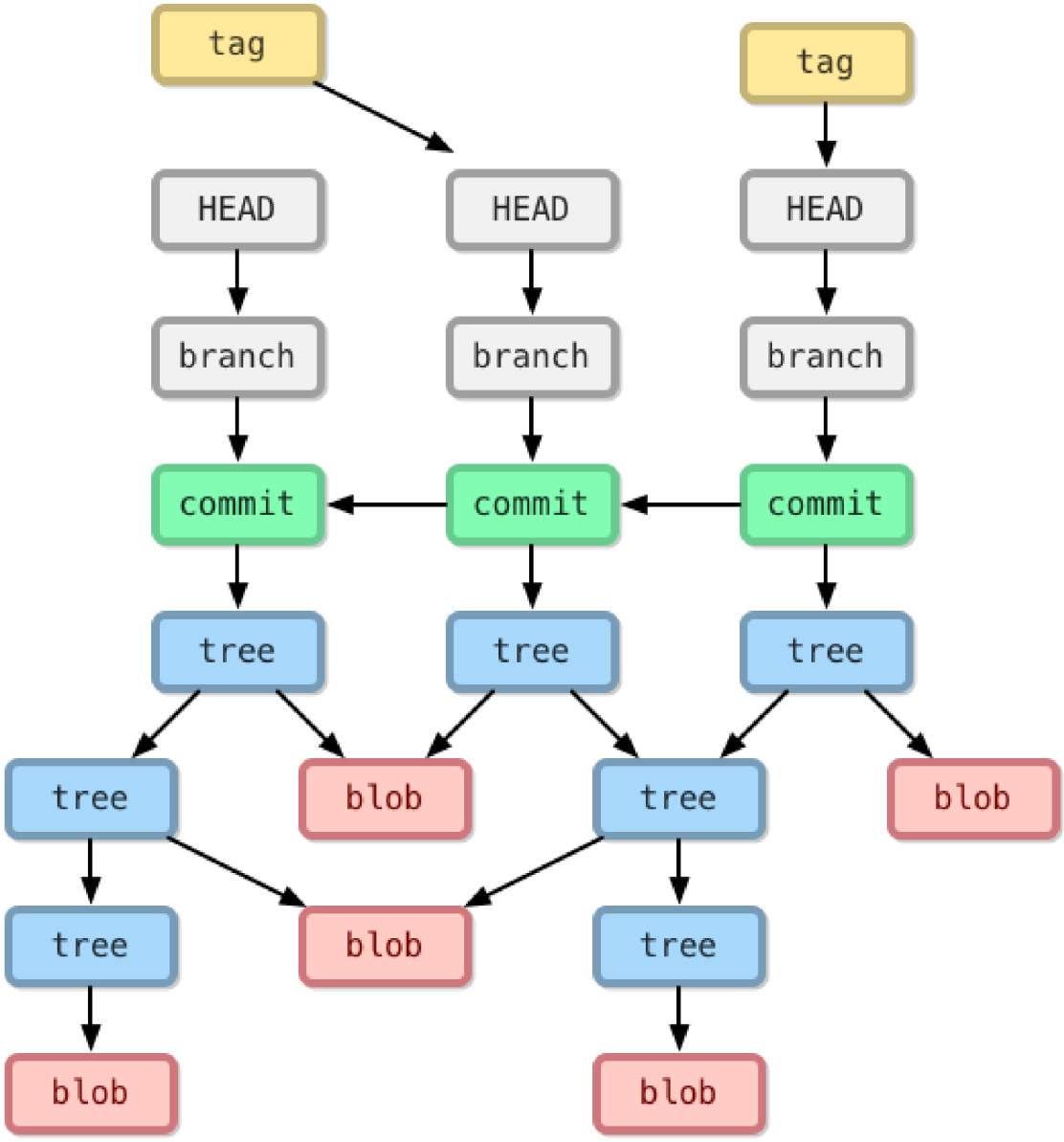
1. **blob** 物件：就是工作目錄中某個檔案的 "內容"，且只有內容而已，當你執行 git add 指令的同時，這些新增檔案的內容就會立刻被寫入成為 blob 物件，檔名則是物件內容的雜湊運算結果，沒有任何其他其他資訊，像是檔案時間、原本的檔名或檔案的其他資訊，都會儲存在其他類型的物件裡 (也就是 tree 物件)。
2. **tree** 物件：這類物件會儲存特定目錄下的所有資訊，包含該目錄下的檔名、對應的 blob 物件名稱、檔案連結(symbolic link) 或其他 tree 物件等等。由於 tree 物件可以包含其他 tree 物件，所以瀏覽 tree 物件的方式其實就跟檔案系統中的「資料夾」沒兩樣。簡單來說，tree 物件這就是在特定版本下某個資料夾的快照(Snapshot)。
3. **commit** 物件：用來記錄有那些 tree 物件包含在版本中，一個 commit 物件代表著 Git 的一次提交，記錄著特定提交版本有哪些 tree 物件、以及版本提交的時間、紀錄訊息等等，通常還會記錄上一層的 commit 物件名稱 (只有第一次 commit 的版本沒有上層 commit 物件名稱。
4. **tag** 物件：是一個容器，通常用來關聯特定一個 commit 物件 (也可以關聯到特定 blob、tree 物件)，並額外儲存一些額外的參考資訊(metadata)，例如: tag 名稱。使用 tag 物件最常見的情況是替特定一個版本的 commit 物件標示一個易懂的名稱，可能是代表某個特定發行的版本，或是擁有某個特殊意義的版本。

Git 會將每一個版本中的檔案建立一個對應的 blob 物件，一樣的，該 blob 物件的檔名就是用上述的方式計算出來的，從這些 blob 檔案，你看不出跟版本有任何關係，你必須透過 tree 物件 (資料夾的快照) 與 commit 物件 (每一個版本的快照) 才能關聯出這些 blob 與版本的關係。

所有的物件都會以 zlib 演算法進行壓縮，不但可以有效的提升檔案存取效率，在日後進行封裝(pack)的時候也可以利用差異壓縮(delta compression)演算法來節省空間。他會自動找出相似的 blobs，並自動計算出 blob 之間的變化差異，再將這些差異儲存在一個名為 *packfile* 的檔案中，這樣就可以大幅節省磁碟空間的耗用)。通常 *packfile* 會置於 .git\objects\pack 目錄下，如下圖示：



上述這四種物件之間的關係，可參考以下圖示：



然而，光是觀看文字與圖示，或許還是難以看出這幾種物件類型之間的關係，沒關係，筆者特別錄製了一段教學影片，試圖用 git 指令的方式解釋 Git 的物件結構與產生物件的過程，也讓各位更清楚的了解到底 Git 如何產生與管理這些檔案。

YouTube 影片連結：認識 Git 資料結構中的物件資料庫與物件之間的關係 (http://www.youtube.com/watch?v=PZbSRy_ow0U)

物件結構的優點

你應該可以漸漸了解 Git 的「物件」設計是如此的漂亮，我們在第一篇文章曾經提到幾個 Git 重要的設計，我們重新列出幾點與「物件」特性有關的設計來看看：

- 有效率的處理大型專案
  - 不僅僅是完整的版本庫會複製(clone)一份在本機，由於所有的 blob 物件都是透過「內容」來做定址的 (content addressable)，因此，若在不同版本之間找尋相同的內容，效率是非常高的。
- 歷史紀錄保護
  - Git 版控的過程，每次提交變更都會產生一個 commit 物件，而這個 commit 物件的名稱又是透過 commit 物件的內容產生。再者，commit 物件會關連到 tree 物件，tree 物件的名稱又是透過 tree 物件的內容所產生。tree 物件又會關聯到 blob 與 tree 物件，這些物件的名稱也是透過內容產生。就這樣一層一層的關聯下去，如果你今天真的想竄改某個版本的歷史紀錄，困難度也是挺高的！

    - 由於 Git 儲存庫經常會被 clone 或 fork，只要是被 clone 過的儲存庫，來源的儲存庫只要任何一個物件被修改，這些 clone 出去的儲存庫就很難再合併回來，所以你幾乎不可能任意竄改版本紀錄。
- 定期的封裝物件
    - 我們在 Git 中提到的 "物件" 其實就是代表版本庫中的一個檔案。而在版本異動的過程中，專案中的程式碼或其他檔案會被更新，每次更新時，只要檔案內容不一樣，就會建立一個新的 "物件"，這些不同內容的檔案全部都會保留下來。
    - 你應該可以想像，當一個專案越來越大、版本越來越多時，這個物件會越來越多，雖然每個檔案都可以各自壓縮讓檔案變小，不過過多的檔案還是會檔案存取變得越來越沒效率。因此 Git 的設計有個機制可以將一群老舊的 "物件" 自動封裝進一個封裝檔(packfile)中，以改善檔案存取效率。
    - 那些新增的檔案還是會以單一檔案的方式存在著，也代表一個 Git 版本庫中的 "檔案" 就是一個 Git "物件"，但每隔一段時間就會需要重新封裝(repacking)。
    - 照理說 Git 會自動執行重新封裝等動作，但你依然可以自行下達指令執行。例如: git gc
    - 如果你要檢查 Git 維護的檔案系統是否完整，可以執行以下指令: git fsck

今日小結

Git 裡的「物件」十分重要，其特性也十分重要，雖然我們在操作 Git 指令的過程中通常不太需要直接接觸這些檔案，不過了解這些物件的存在，也確實有助於讓你更加理解 Git 的運作模式，與 Git 獨到的設計概念。

參考連結

- Git Internals - Git Objects (http://git-scm.com/book/en/Git-Internals-Git-Objects)
- Pro Git Book (http://progit.org/)
- Git Magic - 繁體中文版 (http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/zh_tw/)
- Git (software) - Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/Git_(software))