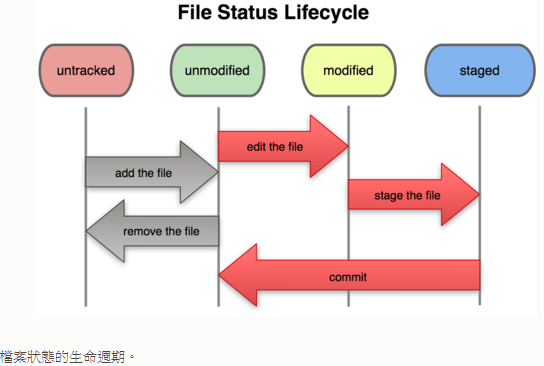
**Git**

**建立專案**

在目錄建立 git 檔案庫, 執行 git init 指令

**檔案庫的狀態**

* 執行 git status 指令可察看當前git狀態
* 當在git 的專案目錄裡建立檔案時，檔案還是untrace，檔案怎麼修改變動，git都不會察覺。
* 執行 git add xxx 指令 ，git就會trace這份檔案。並且直接進入暫存狀態。
* 此時檔案如果有變動，則呈現modify狀態。如確認修改完畢，執行 git add xxx 指令，則會將檔案放入暫存狀態。
* 最後commit將暫存狀態檔案的快照提交到git儲存庫中。

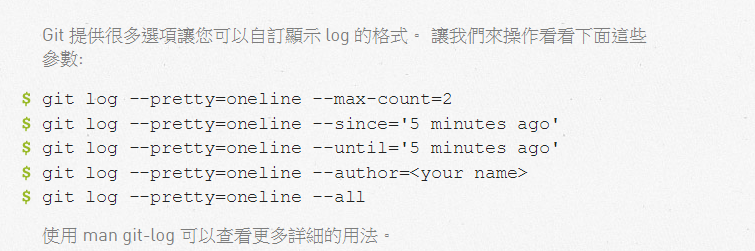


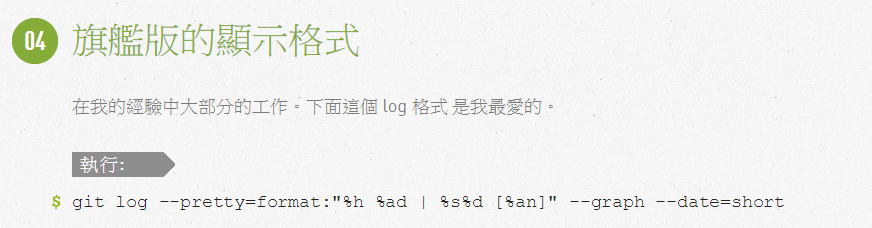
**暫存狀態修改**

如果您決定要放棄送交變更，使用git reset 指令可以將暫存區的紀錄移除。

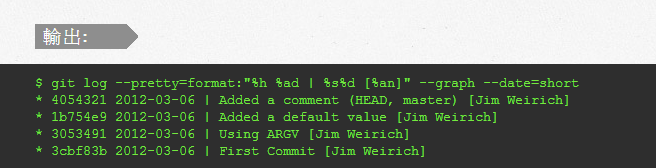
**歷史記錄**

執行git log指令可看歷史紀錄。



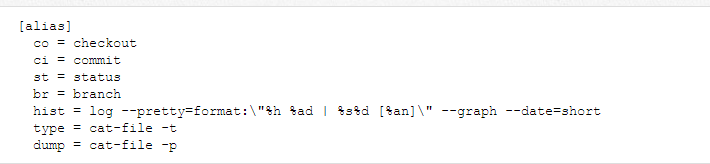


git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short



**別名**

在$HOME目錄下編輯 .gitconfig 檔案，可設定指令的別名。



**取得舊版本**

checkout 指令可以從檔案庫中複製任何時間點的快照（當下時間點的檔案內容和所有記錄）到工作目錄中。

執行git checkout <hash>

每個檔案版本都有一個commit id(一個SHA-1 Hash值)，使用上述的指令就能切換到舊版本。

**建立版本標籤**

可以為檔案版本設立版本標籤來取代hash值，當在切換版本時

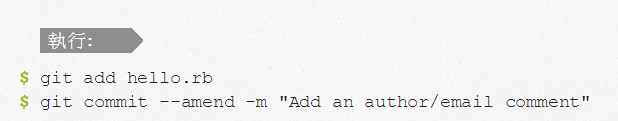
設定標簽（tag）可以方便之後參考該版本代表的意義。

**復原本機修改(暫存前)**

git checkout xxx.txt 恢復編輯前的狀態。

總結一下就是 reset 針對 Staging Area 而 checkout 針對工作目錄，兩者都是使當下狀態跟檔案庫 HEAD 一致。

**修正提交**



**復原已提交的修改**

有時會想移除提交，這時執行git revert HEAD(hash value)，可回到上一個提交版本或是某個版本(hash value)。

舉例：您有版本 A -> B -> C -> D ，版本D 是錯誤的版本，你可以使用 revert HEAD 就會回到版本C在產生一次送交在版本D之後。我們先暫稱為版本E 關係會變成 A -> B -> C -> D -> E(C') 但E的內容跟C是相同的，有些書籍會用C'來表示這跟C的內容是一樣的。

**從分支中移除提交的修改**

revert 這個強大的指令，它可以讓我們回到之前的任何一個版本。不過原本（錯誤）的送交和修正過的送交都會在歷史紀錄（log）中記錄。為了讓歷史紀錄看起來比較簡潔，或者不想讓其他開發者看到這次錯誤地送交

執行reset  標籤(hash value) ，reset 指令會執行的動作 …

* 覆寫目前的分支(可選用參數是否工作目錄內容為指定的版本)並且指定到該次的提交，指的就是在指定版本之後的記錄都會不見（單純執行 git log）。
* 根據選用的參數，暫存區的內容為指定的版本
* 根據選用的參數，工作目錄的內容設定為指定版本

您送錯的版本怎麼呢？其實他們還留在檔案庫的記錄中。我們還是可以回復剛剛那些提交，透過 --all 參數來查看所有提交紀錄。

**探索 GIT 內部: .GIT 目錄**

GIT 目錄它儲存了所有 git 需要的資料。

* git/objects 目錄內一堆名稱是兩個字母的目錄，這些目錄對應 git 儲存的檔案（或目錄），然後 git 使用 sha1 hash 為這些資料命名，您所看到的目錄名稱其實就是這些 hash 的前兩碼。
* 設定檔.git/config，這是隸屬於該專案的設定檔。 當進入這個專案這個設定檔的設定值會蓋過全域的設定檔（該使用者的設定檔），全域的設定檔也會影響這個專案他的路徑在您的家目錄中的 .gitconfig。
* 分支與標籤，.git/refs、.git/refs/heads、.git/refs/tags、tag 目錄下的檔案。每個檔案都對應用 git tag 建立的標簽。heads目錄也一樣，不過是分支的名稱。
* .git/HEAD紀錄目前檔案指向的分支