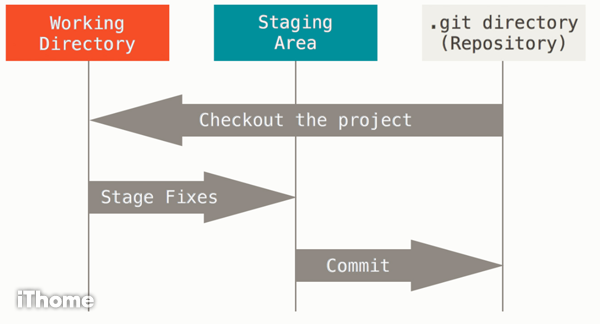
**Git達人教你搞懂GitHub基礎觀念**

使用Git總搞不清楚分支、合併背後邏輯？為什麼專案版本線圖總是亂七八糟？如果Git讓你頭痛，不妨重頭開始，好好釐清Git的基本觀念

文/[王立恒](http://www.ithome.com.tw/users/%E7%8E%8B%E7%AB%8B%E6%81%92) | 2015-04-21發表

對 開發新手而言，想要善用GitHub平臺提供的Git版本控管服務，上手不難，但隨著開發專案開始衍生分支版本後，不少新手往往很容易發生越管版本越混亂 的窘況。開設過多場Git訓練課程，人稱保哥的多奇數位創意技術總監黃保翕觀察，新手最大的迷思是「直接將集中式版本控制軟體的觀念套用在Git上。」他 建議，新手要先了解分散式版本控制系統的運作原理，否則「即便有再好的工具，如果觀念不清楚也沒有用。」

比方說，主幹與分支合併時發生衝突，如果開發者不明就裡地硬下指令解決，就會造成版本控制的錯亂。黃保翕表示：「因為工具都是其次，觀念才是根本。」



在Git工作流程中，使用者先從儲存庫下載某一版本到工作目錄，檔案修改完後，把檔案快照新增到暫存區域，執行提交後檔案就永久存到儲存庫內。圖片來源／Git

**第一步  了解集中式和分散式版本控管的差異**

傳統常見的集中式版本控制系統如CVS、Subversion及Perforce等，採主從式架構，開發專案所有版本的檔案集中儲存在單一臺伺服器 上。而在開發者本機端的儲存庫上，則只會儲存最新版本的歷史紀錄，開發者如果想要提交（commit）新版本、查詢各版本差異或修改歷史紀錄，都要透過網 路連到伺服器才能進行。當專案隨時間變得越來越龐大、版本越來越多，每一個人的每一個動作都要連上伺服器，就會影響了集中式版本控制系統的運作效率。

分散式版本控制系統則不同，黃保翕表示：「Git是分散式版本控制，每個人都有一份完整的本機儲存庫。」也就是說，除了在遠端儲存庫上如 GitHub上，擁有專案各版本的完整程式碼之外，在每一個開發者本機端也還設計了一個本地端儲存庫（Repository），也儲存了所有變更過的檔 案，以及專案各版本的歷史紀錄。遠端儲存庫並非是唯一一份，而是一個供多人同步專案資料用的共享版本。

透過本機端儲存庫上的這份完整專案，開發者不須透過網路便能提交新版本的程式碼到本機端儲存庫。等到需要將本機端儲存庫上資料同步到遠端儲存庫時，才需要使用網路，在Git上，這個指令就是推（Push）。所以，開發者進行版本控制的彈性變大，對網路的依賴也減低。

集中式版本控制系統的優點是，可以在遠端進行集中式的權限控管，限制使用者開啟某些目錄的權限。但在分散式的Git上要做權限控管就稍微麻煩。因為 人人都能取得完整程式碼，要限制存取權限，只能將程式碼依可開放權限，分散在多個遠端儲存庫，限制開發者只能存取符合權限的特定儲存庫。

不過，集中式架構的缺點是，操作版本管理系統高度依賴網路，離線時能使用的版本控制功能有限，例如無法查詢每一個歷史版本，也只能跟本機端最新版本 進行版本差異比較（diff）。而分散式版本控制系統因為在本機端存有一份完整的儲存庫，不會因為沒有網路連線而沒辦法進行查詢、版本差異比較等功能。

黃保翕解釋，版本控制的目的在記錄檔案在某一段時間的變更，方便開發者追蹤原始碼，了解系統軟體的歷史變化。如果開發者修正了3個程式臭蟲，就得有 3個版本被寫入儲存庫中。使用集中式版本控制系統若沒辦法連到網路，就必須同時修畢3個臭蟲後才能提交，而無法追蹤這3次修改的變化處理。

儘管儲存了整套專案，為了減少佔用的磁碟空間，Git處理資料的方式也與其他版本控制系統的作法不一樣，像CVS或Subversion會記錄檔案 隨著時間變更的內容，而Git則只是為當時的專案資料建立快照（Snapshot），如果專案內沒有變更的檔案就不會多儲存一份來佔用磁碟空間，而只是增 加了一筆這個檔案的對應連結，開發者開啟新版本存取這個檔案時，還是開啟先前的舊版檔案，而不是開啟內容相同的新副本。

**第二步  熟悉Git工作流程**

第一步先了解分散式版本控管的特性後，第二步要熟悉Git工作流程。當開發者要建立一個開發專案時，要為這個專案建立一個目錄，以及使用git init指令建立一個這個專案的儲存庫（Repository ）。這個目錄可稱為是這個專案的工作目錄（Working Directory），開發專案所有的檔案都儲存在這個目錄下，而執行與專案相關的Git指令，也都是在工作目錄下執行。而下達建立儲存庫的指令後，則是 會在這個工作目錄下建立一個.git資料目錄，來儲存所有版本變更需要的資訊。

另外，Git還會在.git目錄下建立一個名稱為index的索引檔案，作為記錄專案所有檔案的處理狀態，例如是儲存在本機儲存庫中的已提交 （committed）檔案，或是已變更但尚未提交到本機儲存庫的已修改（modified）檔案，或也有一種狀態是在已修改檔案中標記出要作為提交到下 一版本用的已暫存（staged）檔案。這個索引檔是用來管理檔案修改以及是否要提交到儲存庫的記錄。

在一個常見的Git工作流程中，首先要建立開發專案（包括建立工作目錄和建立儲存庫），接著在專案目錄下新增或修改檔案等開發工作，工作完成告一階 段，就會提交一個專案新版本到Git本地端儲存庫裡。因為提交動作會依據索引檔來進行，所以，開發者得先建立快照來更新索引資料，將需要提交的檔案標記為 staged檔案。然後，才執行提交指令（commit），將staged檔案儲存到儲存庫中。若要使用GitHub來和其他人共享專案，開發者則需進一 步使用Push指令，將本地端儲存庫的特定版本專案，推到遠端儲存庫上整併。下一次要展開開發工作時，則可先從遠端儲存庫將新版程式碼取回本地端儲存庫 （此動作稱為Pull），再從本地端儲存庫放入工作目錄中（此動作稱為Checkout），就可繼續展開下一輪開發。

**必懂開發分版觀念：Master、Branch和Merge**

主幹（Master）與分支（Branch）是稱呼專案的主要版本和分支版本。在GitHub上，通常第一個建立的專案版本會稱為Master版 本。但對Git而言，開發專案的版本並無主幹（Master）與分支（Branch）的分別，每個分支都是等價關係，而開發者自行調整版本命名來區分。一 般習慣將穩定版本稱主幹，其餘的變動、開發中版本則都稱作分支。Branch也是一個建立分支的Git指令，可將某個歷史版本複製一份，獨立成為另一個新 的分支版本，而合併（Merge）指令剛好相反，則是把兩個不同的分支版本，合併到其中一個分支上。

使用Git版本控制時，通常在主幹上開發主要功能，並在主幹分出去的分支上開發副功能。一個開發專案在實際運作時，多半希望主幹的程式碼是穩定、乾 淨的版本。為了避免影響專案穩定性，通常專案開發、更動會在分支上進行。當某分支的功能已經開發完整時，要把分支上的功能套回主幹上，此時就會執行合併， 把分支的版本合併回主幹上。

把分支與主幹放到版本線圖上檢視，分支是從主幹分出去的另外一條路。想要增進版本控制的效率，開發人員必須理解目前自己開發的工作目錄是位於主幹還 是分支，才不致把版本線圖弄得混亂。比方分支上的功能已經開發完整，但是卻把主幹合併到分支上，導致穩定版本上沒有分支上的新功能，分支上反而有了穩定版 本的主功能。

黃保翕解釋，集中式版本控制系統通常沒有引入分支的概念，開發者多半在主幹上開發，但每一個版本都有不同目的、功能，如果只在一條龍的主幹上建立版本，團隊仍舊不易追蹤軟體開發的過程。

Git則提供了一個視覺化的版本線圖（commit graph），來呈現出主幹與各分支連結形成的樹狀架構。不過隨著專案越來越大，版本線圖會變得越來越複雜。黃保翕提醒，當新手不清楚目前自己正在開發哪 個分支版本時，有時會誤將主幹版本合併到分支版本，反而打亂了版本間的關係線。他建議，可以善用合併指令整併不必要的分支，將版本線圖整理清楚。

**必懂專案分版手法：Clone和Fork**

複製（Clone）指令是把專案在遠端儲存庫上的所有內容複製到本地，建立起本機儲存庫及工作目錄，而叉（Fork）則是把別人專案的遠端儲存庫內容複製一份到自己的遠端儲存庫，黃保翕生動的形容：「就像是在餐桌上用叉子把盤子上的一塊肉叉到自己的盤子上。」。

如果在開發者在GitHub上看到有興趣的專案，可以執行Fork指令，把別人專案的遠端儲存庫複製到自己的遠端儲存庫，再執行Clone指令，把自己遠端儲存庫的整個專案的所有內容（包括各版本）複製到本機端儲存庫。

**必懂資料同步指令：Push、Pull、Pull Request**

執行推（Push）指令，可以把自己目前本機端儲存庫的相關檔案，上傳到遠端儲存庫，而拉（Pull）指令則把遠端儲存庫的最新版本下載至自己的本機端，並將遠端分支合併到本地分支，不過，Pull並不像Clone指令會下載完整專案各版本內容。

開發者可以執行Pull取得其他人開放授權的遠端儲存庫上的程式碼，也可以將自己修改的程式碼Push到可存取的第三者遠端儲存庫上。透過推（Push）跟拉（Pull）兩個指令，開發者就能互相分享原始碼。

而Pull Request則是更主動地要求第三方開發者納入自己開發的程式，將本地端儲存庫上的程式碼，整併到對方的儲存庫上。例如A成員可透過Pull Request要求B成員，將A成員修改後的程式碼併入B成員的開發專案中。文⊙王立恒

|  |
| --- |
| ****從Hello-World認識GitHub版本控管**** |
| 「Hello-World」程式範例在電腦學習上有悠久的歷史，引領許多人走入程式語言，幫助我們熟悉由0跟1構成的世界。只要是IT人，對Hello-World程式一定不陌生。開始上手GitHub的第一步就從Hello- World專案開始吧。  在GitHub上開啟專案並不難。首先申請帳號並開啟一個專案儲存庫，並且在本機端安裝SourceTree、GitHub for Windows等Git GUI工具，同步本機端儲存庫與遠端儲存庫，就可以開始進行版本控制。GitHub是以Git為核心的線上協作平臺，使用議題（Issue）、新開分支 （Branch）、維基（Wiki）、拉取要求（Pull Request）等功能，開發者能更有效率的進行版本控制，並且在線上合作、討論專案及透過GitHub分享程式碼。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-1.png  **步驟 1 建立Hello-World專案儲存庫**  建立儲存庫是使用GitHub的第一步。儲存庫是GitHub的基本元件，通常一個儲存庫對應到一個專案。儲存庫裏頭可以放置文件、資料夾與其它專 案所需要的檔案。建議在每個儲存庫中附上描述這個專案的「讀我檔案」，方便自己或者其他開發者想要檢閱這個專案的時候，能快速建立對這個專案的認識。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-2.png  **步驟 2 用議題來管理開發任務的進度**  在GitHub通常會用議題（Issue）功能來管理開發需求，使用者提出一項開發需求或有程式臭蟲要修改，就會發起一個議題來追蹤這個工作的進 度。議題就像是一個工作筆記，可以紀錄某個程式臭蟲，或者說明開發中的功能等等。在GitHub上，可以對議題執行搜尋、標籤，或把議題指派給某個共同開 發者處理，並在議題中開啟對話，讓共同協作的過程更順利。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-3.png  **步驟 3 建立分支管理版本**  在GitHub專案中新增了一個儲存庫後，第一個分支的名稱預設為主幹（Master）。通常主幹是穩定版本，這個版本主要用於存放測試穩定後的專案程式碼，或是最主要的一份程式碼。  如果專案要開發新功能，或者使用者要自行發展自己的專案版本，則會使用Fork指令來增加新的分支版本。  在分支版本上開發程式，等到分支版本上的功能完成開發後，才會將程式碼合併回主幹版本，避免未經測試還不穩定的功能，影響了穩定版本的內容。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-4.png  **步驟 4 執行提交建立新版本**  開發者要將修改檔案後的專案內容，增加為一個新的專案版本，這個動作就稱為提交（Commit）。開發者可以每次開發完一項功能，就先將新版程式 碼，先提交到本機端儲存庫上，再等到多次提交後，才一次將本機端儲存庫上的專案程式碼，一次性發布（稱為Push）到GitHub網站上的雲端儲存庫。通 常執行提交時會輸入提交訊息（commit message），說明這個版本做了些什麼修正。有了這些訊息記錄，專案中的開發者便可以追蹤每個版本間的更動。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-5.png  **步驟 5 用拉取要求提醒他人整併程式碼**  在拉取要求（Pull Request）是GitHub協作開發的重要功能。透過拉取要求，主動要求他人把自己的程式碼納入專案中。比方A開發者看到B開發者的專案，覺得也想參 與開發，便執行Fork指令複製一份到自己的遠端儲存庫。而A開發者完成某個新功能後，向B開發者提出拉取要求，希望能把變更整併到B的專案中。透過拉取 要求，開發者也能做到基本的權限控管，而不是無條件的讓別人的變更合併到自己的專案中。比方在收到拉取要求後，原專案發起者能在GitHub上比對兩個版 本間差異，覺得變更有益於原專案後再合併到自己的專案。許多人使用拉取要求，在GitHub上面開始跟其他開發者展開專案的討論。 |
| http://static4.ithome.com.tw/sites/default/files/images/708-%E5%B0%81%E9%9D%A2-P30-600-6.png  **步驟 6 用Wiki整理專案參考文件**  GitHub也提供了Wiki功能，可供開發者建立專案文件，來記錄專案開發需要的參考資訊。如開發時，特定需求的議題往往會經過多次修改和討論 後，功能變動才會定案，就可以將此議題內的開發者討論過程，整理出精華版本後放入Wiki文件，供未來其他開發者參考，也不用再打開該議題回溯冗長的討論 過程才能得知結果。Wiki也是多奇數位創意技術總監黃保翕推薦的好用功能，黃保翕表示，透過Wiki功能可以好好歸納專案文檔，方便開發團隊成員檢閱。 |

**Git參考資料**

[Git官方網站：http://git-scm.com/](http://git-scm.com/)

[Try Git：https://try.github.io/levels/1/challenges/1](https://try.github.io/levels/1/challenges/1)

[Git-it：http://jlord.us/git-it/index-zhtw.html](http://jlord.us/git-it/index-zhtw.html)

[GitHub Guides：https://guides.github.com](https://guides.github.com)

[保哥30天精通Git版本控管：http://ithelp.ithome.com.tw/ironman6/player/doggy/dev/1](http://ithelp.ithome.com.tw/ironman6/player/doggy/dev/1)

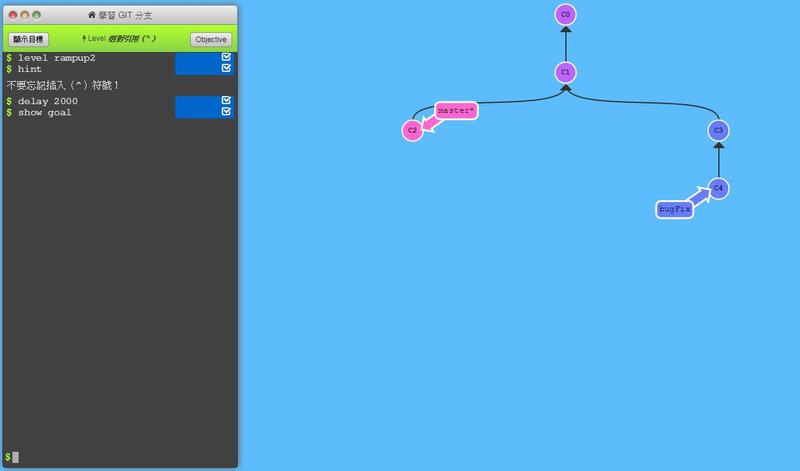
[相關報導請參考：GitHub：IT人必學開發神器」](http://www.ithome.com.tw/article/95280)

## [Git 筆記 - Git初始設定 & Github入門](http://tech.marsw.tw/blog/2013/08/16/git-notes-github)

Git 這個詞聽了好久，也知道好處，但遲遲沒用  
主要就是上手的門檻有點高，要設定實在會搞死人阿@@

不少網路上的神手紛紛寫的Git教學，很久也沒激起想要弄Git的興趣  
直到上了研究所時間更多、也有更多的project對於version控管更加依賴  
加上前陣子看到了這份教學是目前覺得最順眼最簡單易瞭的，開始認真把Git融入資工生活中囉

如果想學更多以圖解方式講解的Git功能，也可以參考 [Learn Git Branching](http://pcottle.github.io/learnGitBranching/) //2014-10-08更新

  
或許看完上面的Slide對Git還是霧煞煞，那就 **直接照著以下筆記try it** ，  
相信操作過對於Slide中所講的概念會更加熟悉!!!

(END 有文字簡易流程)

## Github

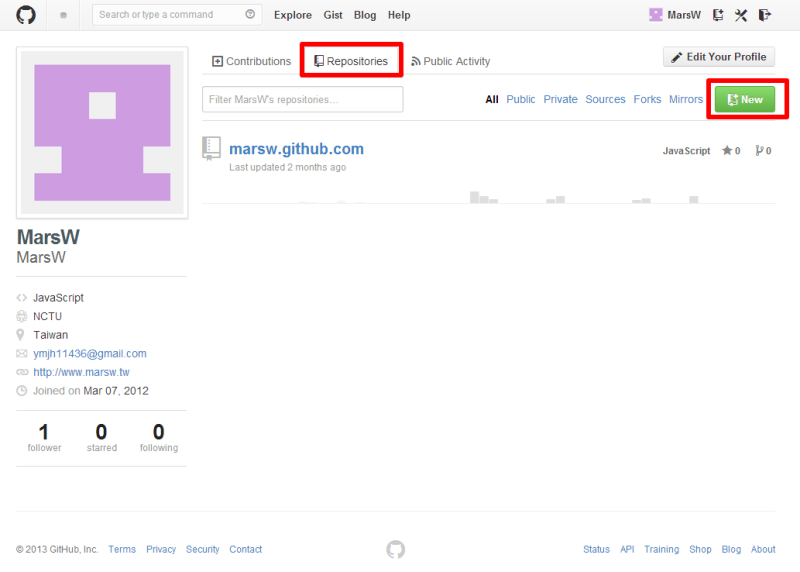
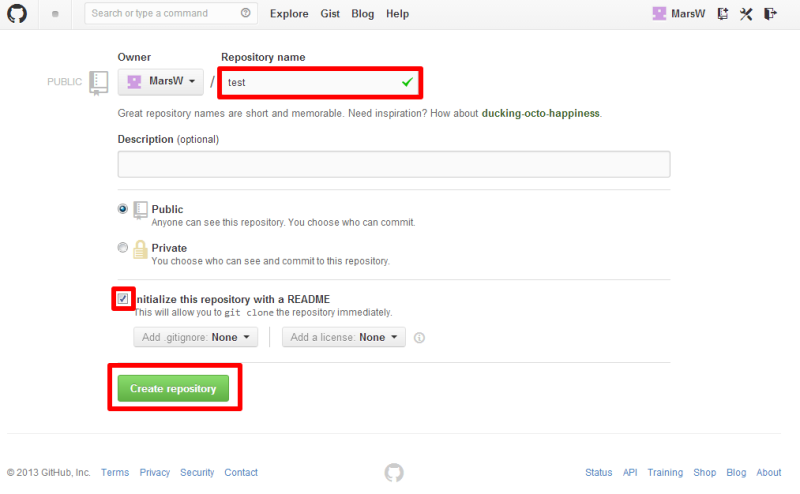
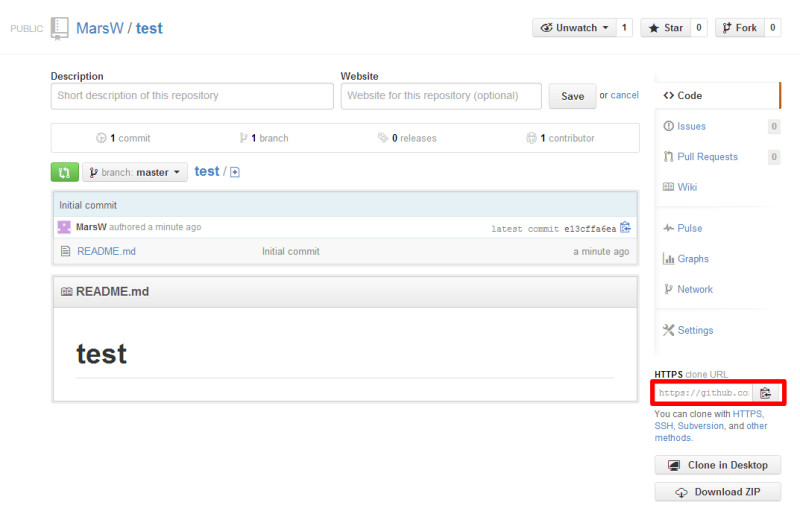
先從最常用的線上Git服務Github來做體驗

首先當然記得要先[申請個Github帳號](https://github.com/)

#### 1.[安裝git](http://git-scm.com/book/zh-tw/%E9%96%8B%E5%A7%8B-%E5%AE%89%E8%A3%9DGit)

//2014.03.02 在Mac中安裝，只要打開Terminal，輸入Git，就會自動導入Xcode安裝，非常貼心！

#### 2.Github上新增一個 repository

  
Github要把repository設成private要付費(所以如果有比較secret的code就不建議使用Github)  
  


#### 3.本機端初始化

Windows的話使用Git Bash (不要用Git GUI) / Unix 的話就直接用本身的command line  
(不管是Windows還是Unix，下的command 都是一樣的)  
先設定你的基本資料(之後commit的時候才容易知道作者是誰)  
git config --global user.name "MarsW"  
git config --global user.email "ymjh11436@gmail.com"

然後移動到本機想放此repository的資料夾底下 (善用ls,cd)

git clone https://github.com/MarsW/test.git 先把code從Github抓下來 (這裡的URL 就是前張圖的clone URL)  
會發覺本機的目錄底下出現一個新目錄test，裡面應該會有之前在Github上建立repository所生成的README.md

#### 4.開始來寫code啦！

然後進到test目錄中  
git branch v1 生一個新branch名叫v1  
git checkout v1 切換到v1這個branch上  
command line應該會顯示

Switched to branch 'v1'

開始來寫code啦!!! 隨意修改增刪檔案

像我這次的修改就是 **在test底下新增一個叫做test.txt的檔**，  
下git status指令command line會顯示以下這樣

# On branch v1

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#

# test.txt

nothing added to commit but untracked files present (use "git add" to track)

有 **Untracked files** 所以下指令 git add test.txt，然後看看git status

# On branch v1

# Changes to be committed:

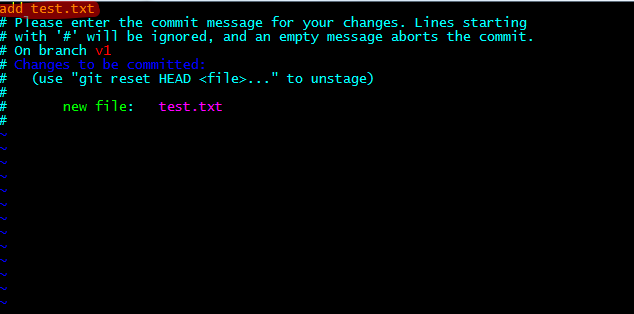
# (use "git reset HEAD <file>..." to unstage)

#

# new file: test.txt

#

這次的修改就這樣，git status也沒顯示有任何修改過的檔案沒有track到的 (代表都add過了)  
就可以下個 git commit  
Windows 中的 Git Bash 是使用vim編輯器   
Unix系統若預設不是vim，可下*git config --global core.editor "vim"*將預設編輯器設成vim

  
上面的紅色標示空行就是讓你輸入這次修改的內容為何，   
**要寫清楚** ，這樣以後在翻code的時候才容易找到想找的內容

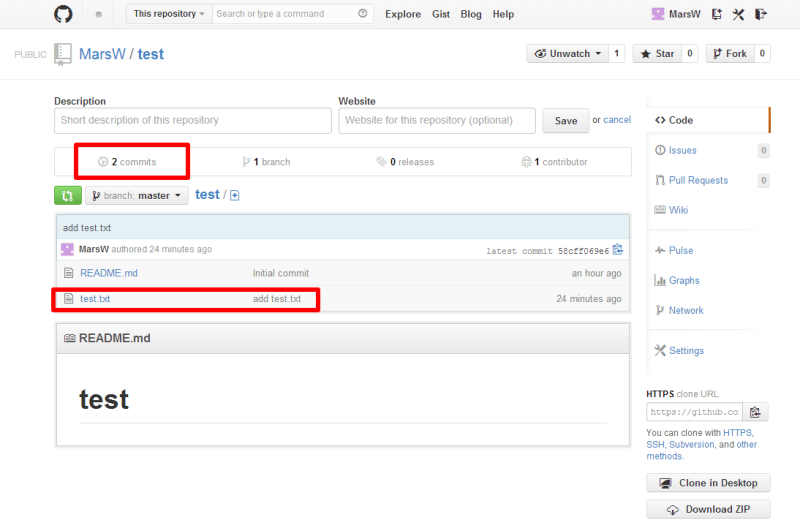
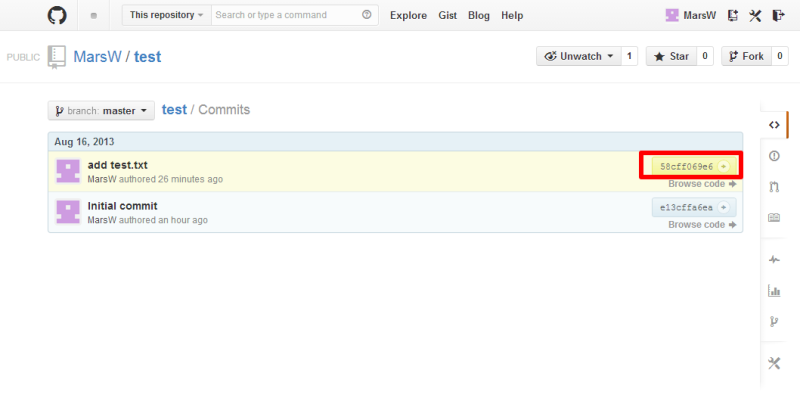
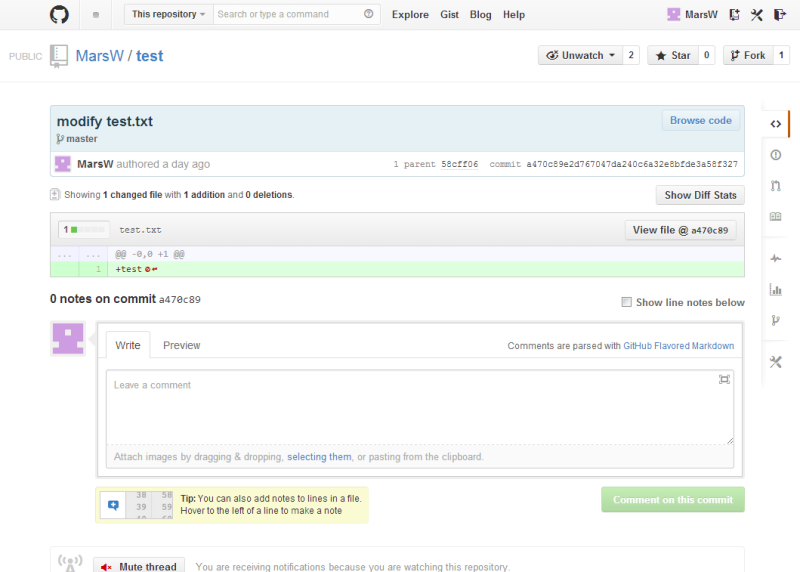
**vim 幾個簡單指令** (更多指令請自行google)

i 進入編輯模式(自由打字啦)  
Esc 退出編輯模式(回到一般模式)  
dd 刪除一行(一般模式下)  
:wq 存檔離開(一般模式下)  
:q! 不存檔直接離開 (一般模式下)

#### 5.合併->push到Github上

git checkout master  
git merge v1 把v1和master merge起來  
git push 然後接著輸入Github的帳號密碼

看看這次新增的test.txt果然同步到Github上了

  
Commits中也能從標題清楚的看到每次的修改摘要，  
也能更進階去看每個Commits中增刪哪些檔案，修改的檔案增刪了哪些行  
  


## 總結

簡單再講一次操作過程

1.第一次初始  
在Github開repo  
git clone [clone URL] 先把code從Github抓下來

2.  
進到該目錄中  
git branch [branch-name] 生一個新branch  
git checkout [branch-name] 切換到此branch上

3.  
git status 看修改了哪些檔案  
git add [修改到的檔案] 一個檔一個檔自己加，因為這樣可以Double check自己到底改了哪些檔  
git commit 註解要寫清楚 完成一個小功能就要commit一次  
重複步驟3.直到 **git status** 顯示沒有任何修改過的檔案

4.  
git checkout master  
git merge [branch-name] 和master merge起來  
git push 然後接著輸入Github的帳號密碼

如果只有一人work，每次的流程就是跑 2.~4.