中國科技大學

資訊學院-資管系-日間部

系專業選修

【版本控制-Git】

(Version Control-Git)

【NSPDxxxxx】【選修】【2】

**資管系二年A班**

**教室:格致樓708**

**週三:6、7節**

教師:李村林

**中華民國108年 5月15日**

目錄

[版本控制系統（Version Control System）[4] 1](#_Toc9024661)

[Git安裝 1](#_Toc9024662)

[使用者設定 3](#_Toc9024663)

[Git基本操作 4](#_Toc9024664)

[刪除、修改檔案 7](#_Toc9024665)

[.gitignore 10](#_Toc9024666)

[比較檔案差異(KDiff3) 10](#_Toc9024667)

[取出檔案 12](#_Toc9024668)

[暫存目前工作資料 12](#_Toc9024669)

[清理repository 13](#_Toc9024670)

[detached HEAD 13](#_Toc9024671)

[Git專案管理實務 15](#_Toc9024672)

[Android Studio專案 15](#_Toc9024673)

[Visual Studio專案 17](#_Toc9024674)

[使用圖形介面 17](#_Toc9024675)

[專案的分支(Branch) 22](#_Toc9024676)

[合併分支 25](#_Toc9024677)

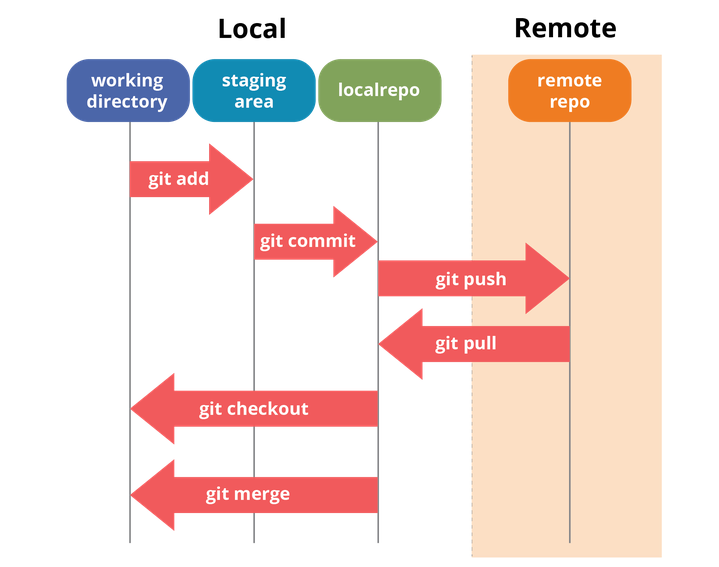
[參考資料 26](#_Toc9024678)

# 版本控制系統（Version Control System）[4]

版本控制系統是一種軟體工程的開發技巧，可以透過這個系統讓每位成員的軟體版本可以方便同步和維護管理。

一般在軟體開發中又分為中央式系統（例如：Subversion、CVS 等）與分散式系統（例如：Git、BitKeeper、mercurial 等），中央式版本控制系統的工作主要在一個伺服器進行，由**中央管理**存取權限「鎖上」檔案庫中的檔案，**一次只能讓一個開發者進行工作**。而分散式系統讓不同開發者直接在各自的本地檔案庫工作，並容許多個開發者同時更動同一個檔案，而每個檔案庫有另外一個合併各個改變的功能。分散式系統讓開發者能在沒有網路的情況下也能繼續工作，也讓開發者有充分的版本控制能力，而不需經中央管理者的許可。

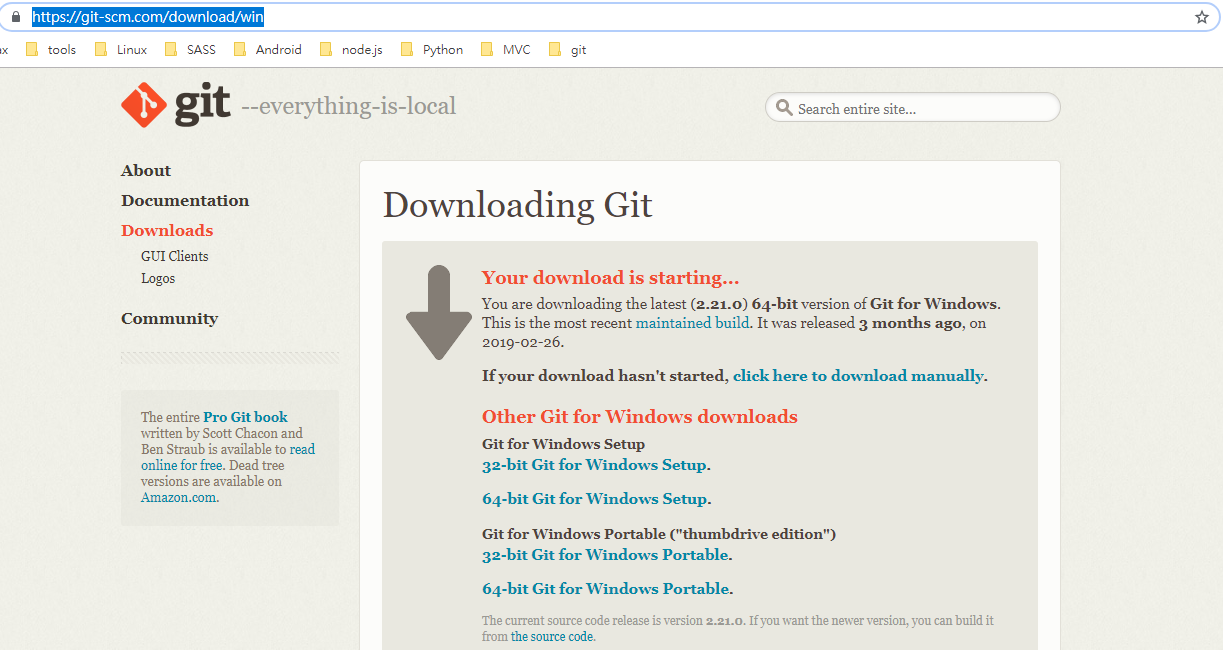
Git 是一個分散式版本控制軟體，最初由 Linus Torvalds 創作（也是作業系統 Linux 系統的開發者），其最初目的是為更好地管理 Linux kernel 開發而設計，其具備優秀的 merge tracing 合併程式碼的能力（使用程式碼 snapshot 來比較歷史版本差異）。

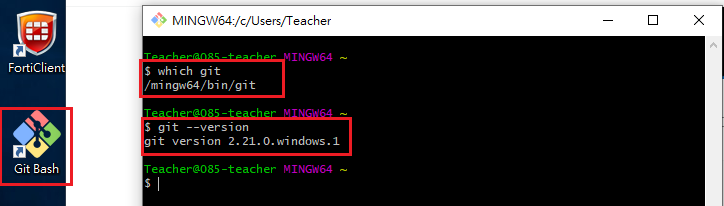


Git 可以分為 Local（本地）和 Remote（遠端）兩個環境，由於 Git 屬於分散式的版本控制系統，所以開發者可以在離線 local 環境下開發，等到有網路時再將自己的程式推到 Remote 環境或 pull 下其他開發者程式碼進行整合。在 Local 中我們又分為 working directory（工作資料夾）、staging area（暫存區）和 repositories（檔案庫）。

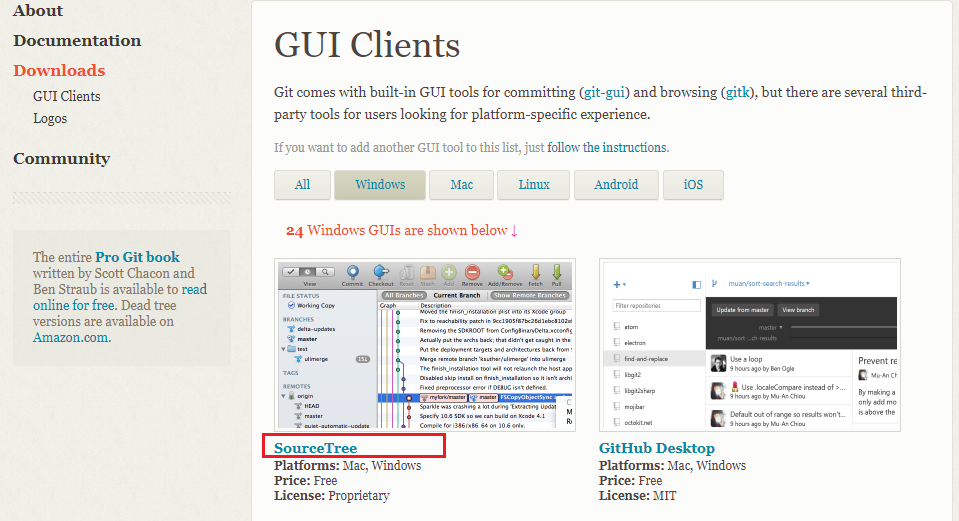
## Git安裝

Windows OS

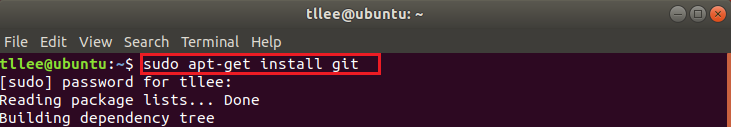




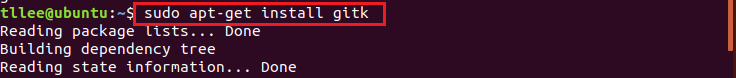
圖形化介面: SourceTree



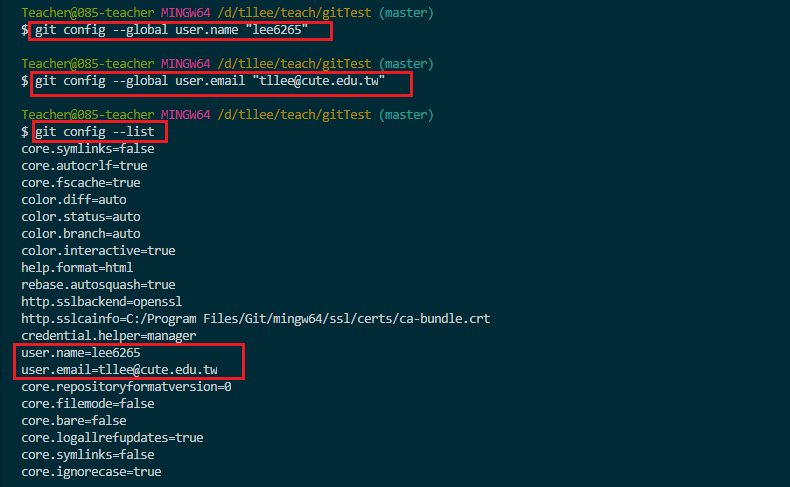
Ubunut OS

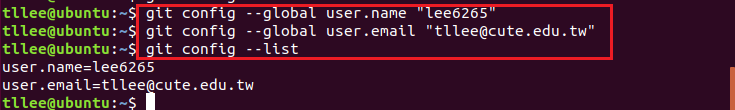


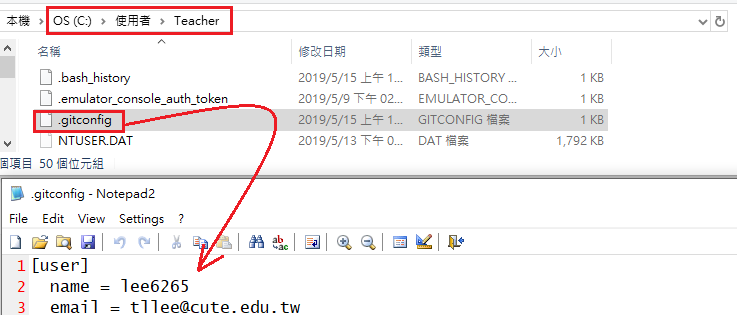




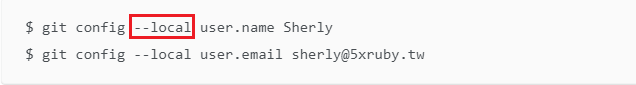
## 使用者設定







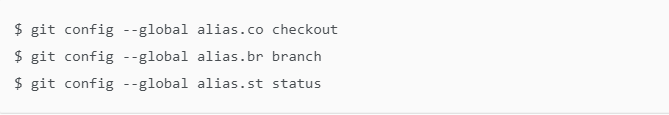
可以每個專案設定不同的作者: 要幫特定的專案設定不同的作者時，可以在該專案目錄下進行 Git 設定的時候，加上 --local 參數



更換編輯器



Alias 設定

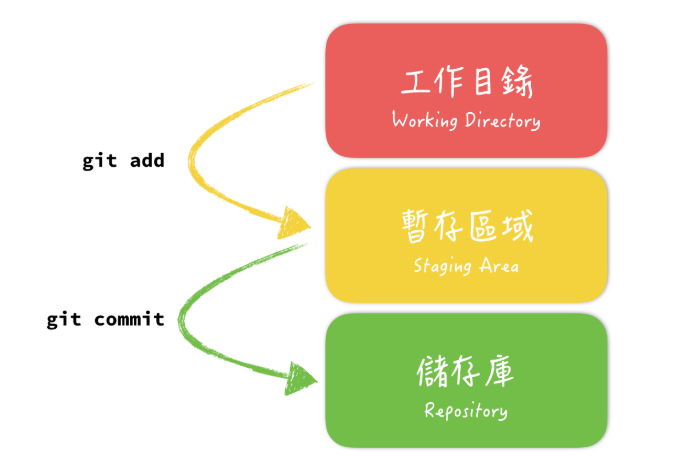


也可直接到 ~/.gitconfig 裡修改



目錄若不想再被 Git 控制，只要把那個 .git 目錄(隱藏目錄)移除就可以了。

## Git基本操作



git add 指令把檔案從工作目錄移至暫存區（staging area或索引）

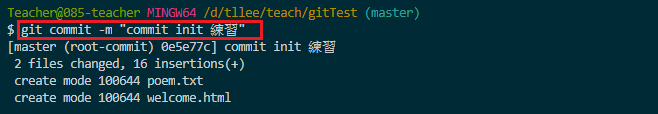
1. git add welcome.html
2. git add \*.html
3. git add –all (git add .)

-all 跟 . 參數有什麼不一樣？

git add . 這個指令會把目前當下這個目錄，以及它的子目錄、子子目錄、子子子目錄…裡的異動全部加到暫存區，但在這個目錄的以外的就不歸它管了。而 git add --all 指令就沒這個問題，這個指令不管在專案的哪一層目錄執行，效果都是一樣的，在這個專案裡所有的異動都會被加至暫存區。

git commit 指令把暫存區的內容移至儲存庫

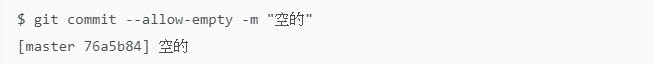
Commit: Git 每次的 Commit 都只會處理暫存區(staging area)裡的內容，沒被加到暫存區裡的檔案，就不會被 Commit 到儲存庫裡。



訊息很重要要注意幾個原則:

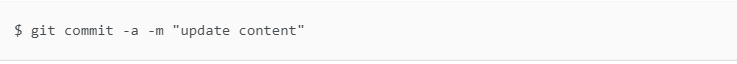
1. 不要使用太過情緒性的字眼
2. 英文、中文皆可，要簡單、清楚
3. 儘量不要使用像 bug fixed，但可以使用#21 bug fixed

沒東西也是可以 Commit ，例:



-a 參數

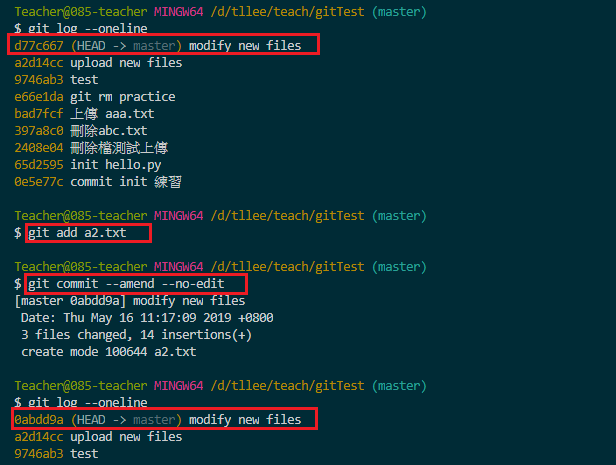
即使沒有先 add 也可以完成 Commit。但要注意的是這個 -a 參數只對**已經存在 Repository 的檔案有效**，對還是新加入的檔案（也就是 Untracked file）是無效的。



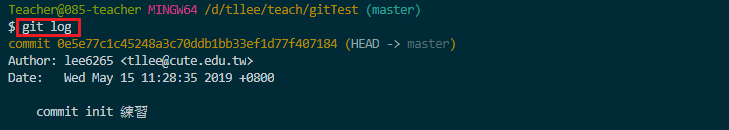
什麼時候要 Commit？常見的 Commit 的時間點如下：

1. 完成一個任務的時候
2. 下班的時候：雖然可能還沒完全搞定任務，但至少先 Commit 今天的進度和備份檔案。

追加檔案到最近一次的 Commit(但是SHA已改變了): **--amend --no-edit**

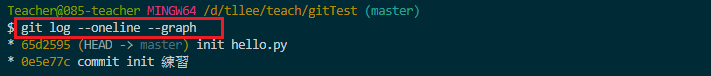


檢視紀錄:

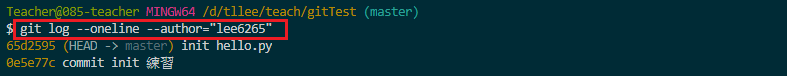


在 Git 裡每串這種看起來像亂碼的文字，是使用 SHA-1（Secure Hash Algorithm 1）演算法所計算的結果。Git 使用這樣的字串做為識別。每個 Commit 都有一個這樣的值，不會重複。

輸出的結果就會更為精簡，可以一次看到更多的 Commit: --oneline --graph



想要找某個人或某些人的 Commit: --author



可以再用 |（中文「或者」的意思） 來查詢「Sherly 以及 Eddie 這兩個人的 Commit 紀錄」



Commit 訊息裡面有寫訊息”.py”



找到哪些 Commit 的檔案內容有提到 “中文” 這個字？



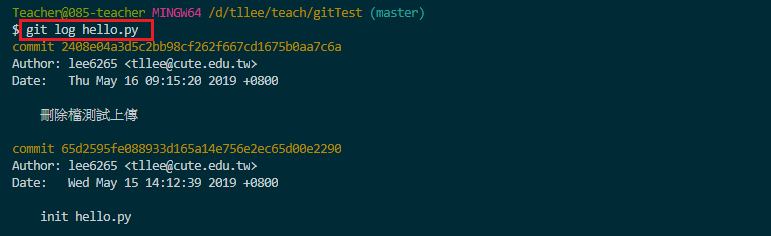
今天早上 Commit 了什麼



從 2017 年 1 月之後，每天早上 9 點到 12 點的 Commit



檢視特定檔案的 Commit 紀錄



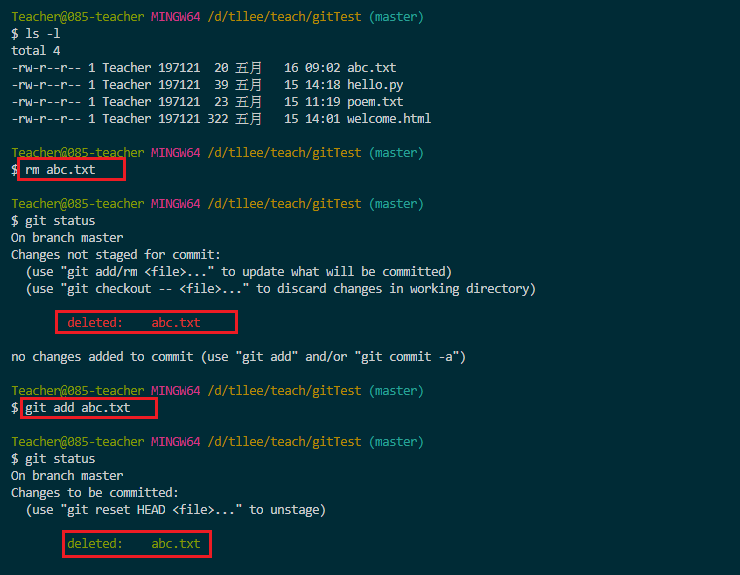
如果想看這個檔案到底每次的 Commit 做了什麼修改，可以再給它一個 -p 參數



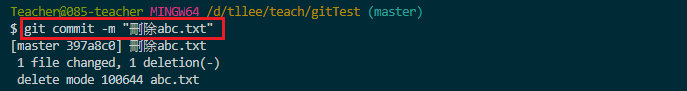
## 刪除、修改檔案

在 Git 裡，不管是刪除檔案或是變更檔名，對 Git 來說都是一種「修改」。

直接刪除**rm=>git add=>git commit**



要commit後才會真得在Repository中刪除

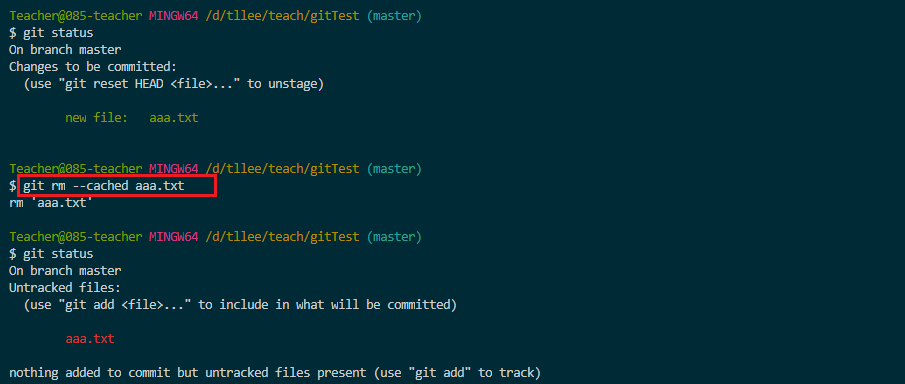


上述兩段式也可改為一段式刪除: git rm

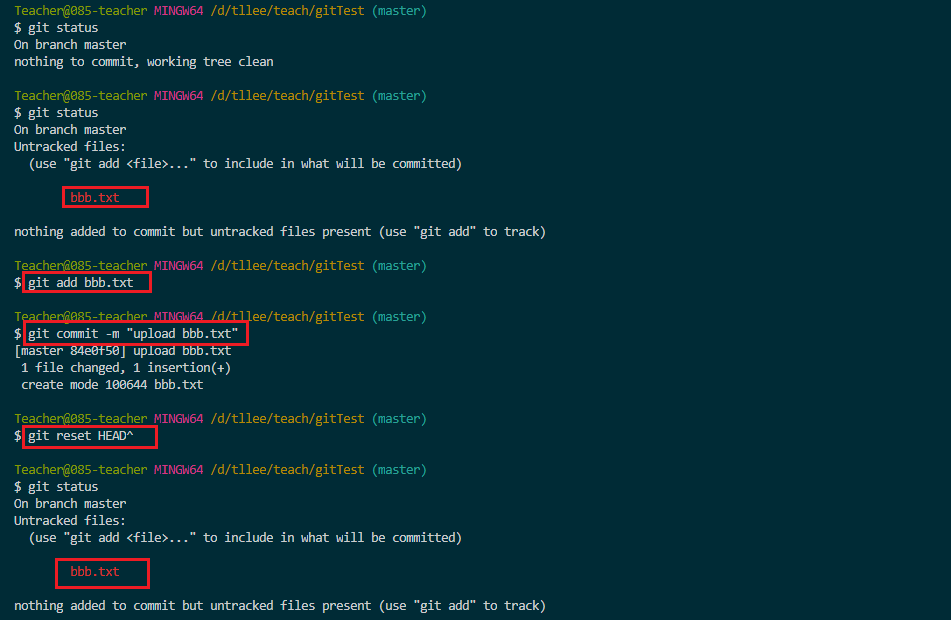


不管是系統的 rm 或是 git rm 指令，都會真的把這個檔案從工作目錄裡刪掉，但如果只是「我不是真的想把這個檔案刪掉，只是不想讓這個檔案再被 Git 控管了」的話，可以加上 --cached 參數：

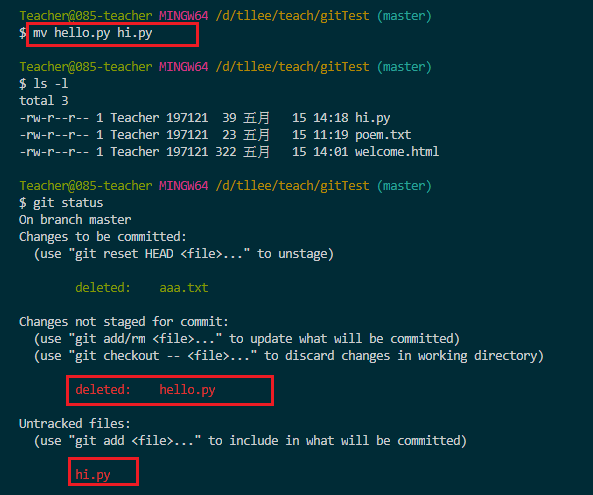
反悔不想把檔案加入追蹤: **git rm --cached**

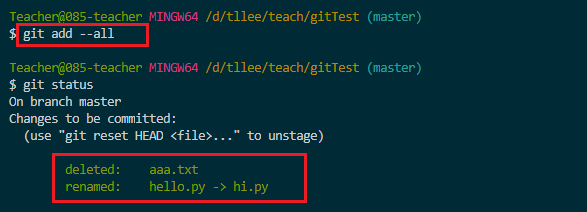


若檔案已經在 repository 內: git reset HEAD^

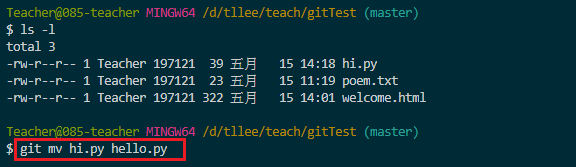


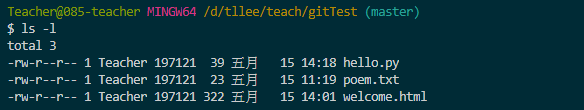
變更檔名:改檔名但對 Git 來說會被認為是兩個動作，一個是刪除 hello.py 檔案，一個是新增 hi.py 檔案（變成 Untracked 狀態）, **mv=>git add --all**





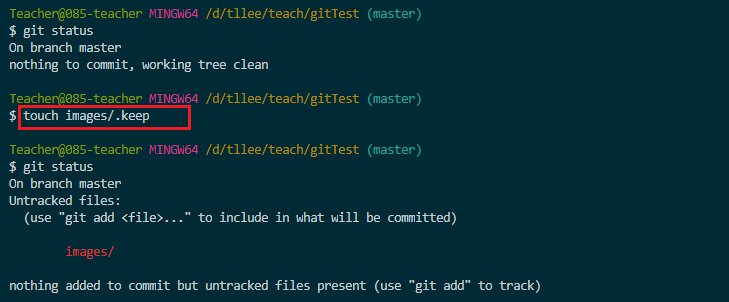
也可用**git mv**





Git 是根據檔案的「內容」去算出那個 SHA-1 的值，所以 Git 不是很在乎你的檔案叫什麼名字，只在乎檔案的內容是什麼。所以當你進行更改檔名的時候，Git 並沒有為此做出一個新的 Blob 物件，而僅是指向原來舊的那顆 Blob 物件。但因為檔名變了，所以會為此做出一顆新的 Tree 物件喔。

Git 在計算、產生物件的時候，是根據「檔案的內容」去做計算的，所以光是新增一個目錄，Git 是沒辦法處理它的。慣例上可以放一個名為 “.keep” 或 “.gitkeep” 的空檔案，讓 Git 能「感應」到這個目錄的存在。



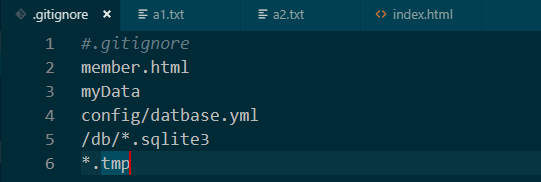
## .gitignore

只要 .gitignore 這個檔案存在，即使這個檔案沒被 Commit 或是沒被 Push 上 Git Server 就會有效果。但這個檔案會建議 Commit 進專案並且推上 Git Server，這樣一來整個專案一起開發的人可以共享相同的設定。

GitHub 上有整理了一份各種程式語言常見的 .gitignore 檔案

<https://github.com/github/gitignore>

注意: .gitignore 檔案設定的規則，只對在規則設定之後的檔案才有效。



若要忽略這個規則



## 比較檔案差異(KDiff3)

設定檔案比對程式為KDiff3





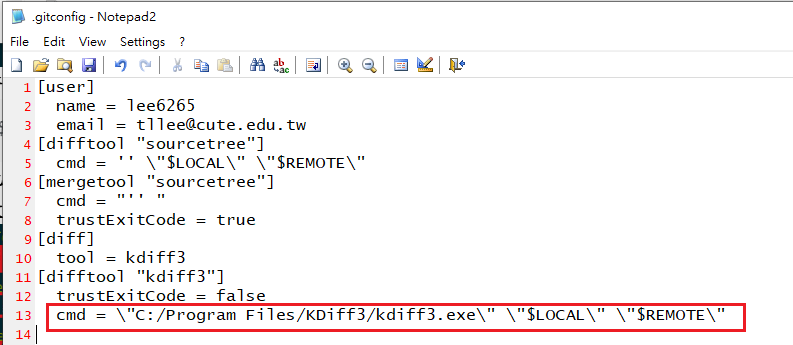


檢查.gitconfig



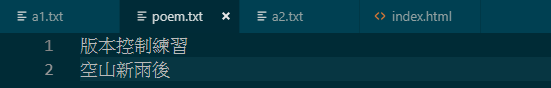
內容修改如下:

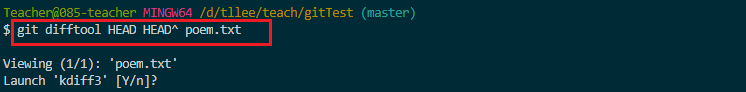
**\"C:/Program Files/KDiff3/kdiff3.exe\" \"$LOCAL\" \"$REMOTE\"**

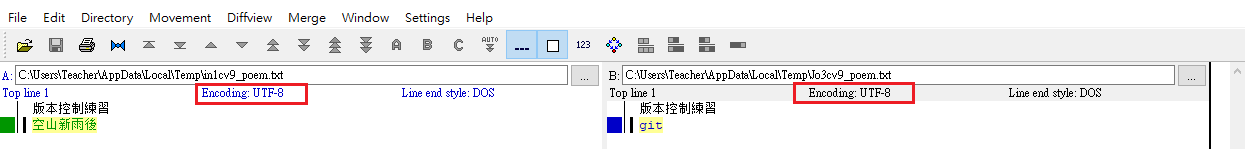


執行測試

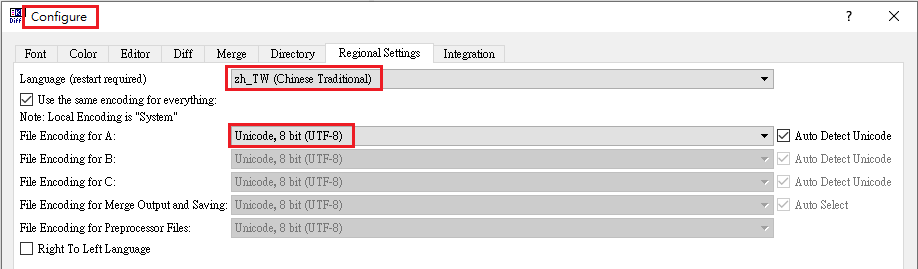
先將比對檔案poem.txt commit一次再修改如下:



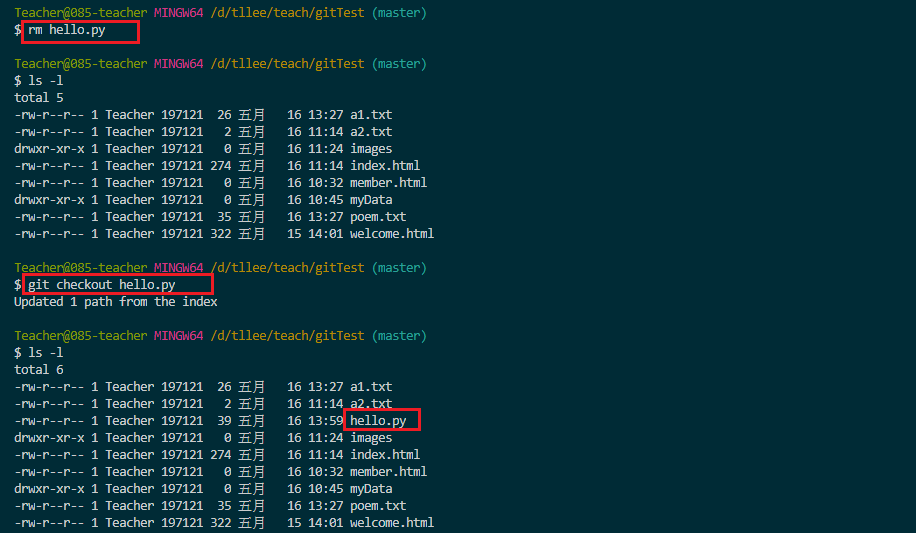




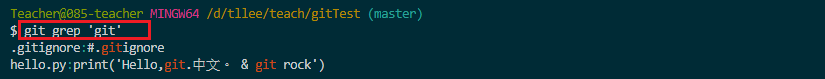
註:KDiff3可修改設定如下，否則中文會出現亂碼



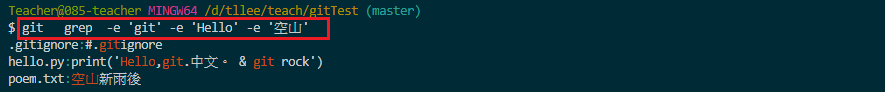
## 取出檔案



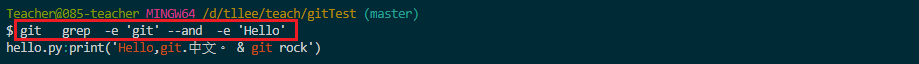
找出含有某字串的檔案



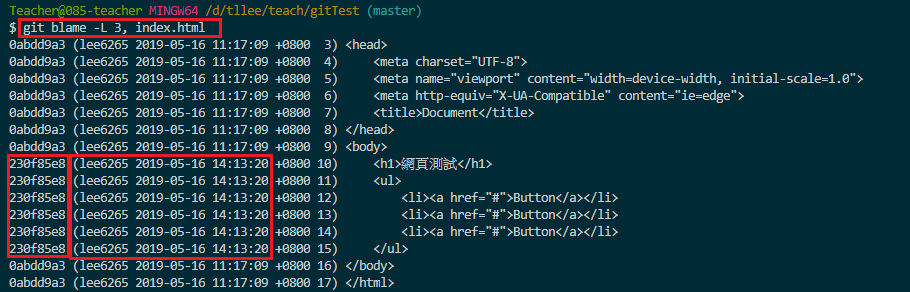
要找的字串不只一個,以**-e**分開表示或運算



若要同時出現兩個字串要用**--and**

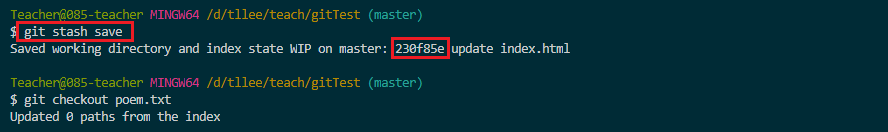


這行程式誰寫的？**git blame -L n1,n2 filename**

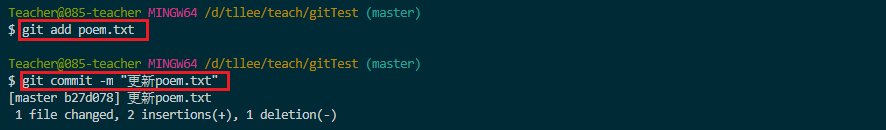


## 暫存目前工作資料

**git stash save**並取出poem.txt檔案暫時性工作(必需先完成)



完成poem.txt 工作並commit

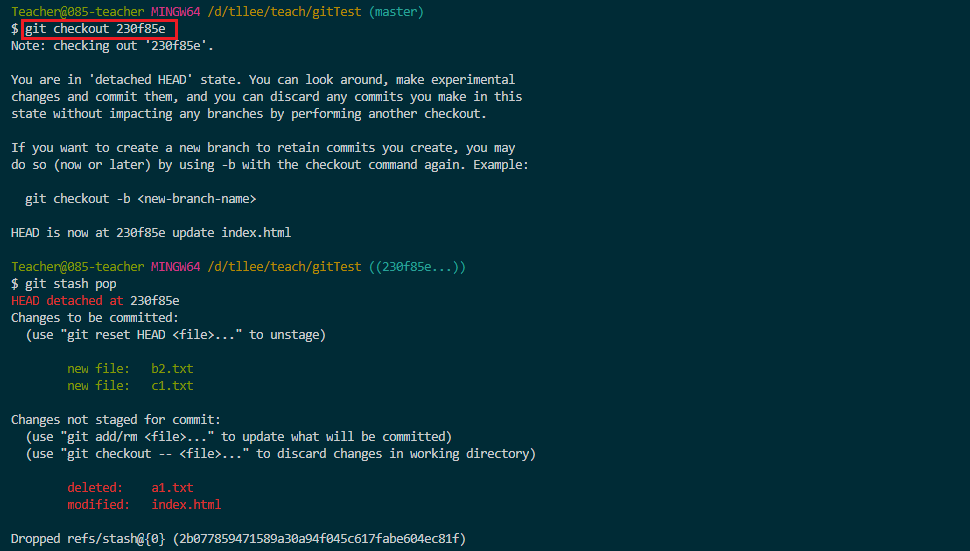


取出先前的工作狀態

先顯示commit 的版本: 230f85e **git stash list**



再取出 **git checkout xxxxx**



## 清理repository



--aggressive 仔細檢查及清理耗時較長，只可偶爾為之

--no-prune 不清除不會用到的檔案只整理它們

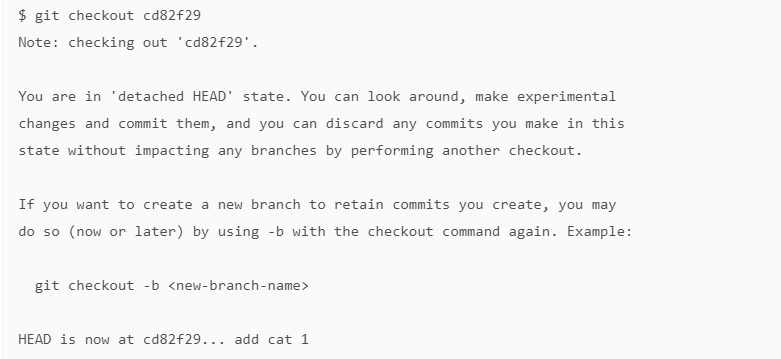
## detached HEAD

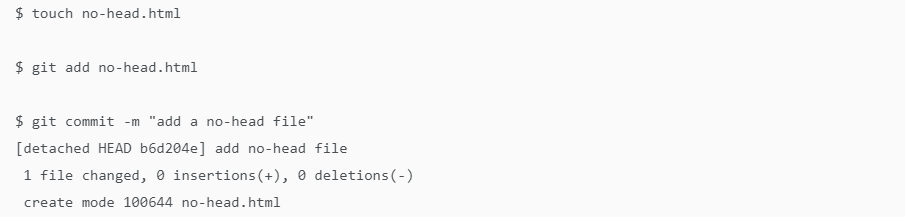
正常情况下，HEAD 會指向某一個分支(Branch)，而分支會指向某一個 Commit。但 HEAD 偶爾會發生「沒有指到某個分支」的情況，這個狀態的 HEAD 便稱之「detached HEAD」。

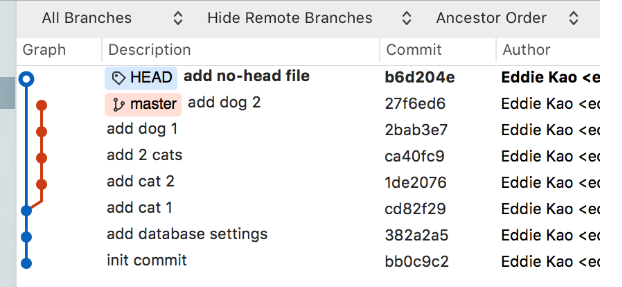
可能發生這個狀態的原因有：

1. 使用 Checkout 指令直接跳到某個 Commit
2. 切換到某個遠端分支的時候

其實這個狀態沒什麼特別的，就只是 HEAD 剛好指向某個沒有分支指著的 Commit 罷了，所以一樣可以跟平常一樣的操作 Git，一樣可以進行 Commit。







現在這個 Commit（b6d204e）還「沒有名字」，更正確的說是還沒有分支指向它，目前僅有 HEAD 指著它而已。這有什麼影響嗎？影響就是當我的 HEAD 回到其它分支之後，這個 Commit 就不容易被找到（除非你有記下這個 Commit 的 SHA-1 值），如果一直沒人來找它，過久了之後就會被 Git 啟動的**資源回收機制給收掉**了。

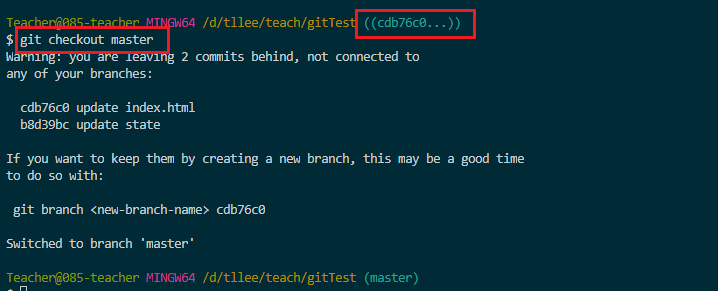
所以，如果還想留下這個 Commit，就給它一個分支指著它就行了。



這個 Commit 以後就可以透過 tiger 這個分支來找到它了。你也可以使用 Checkout 指令配合 -b 參數，建立分支後直接切換：

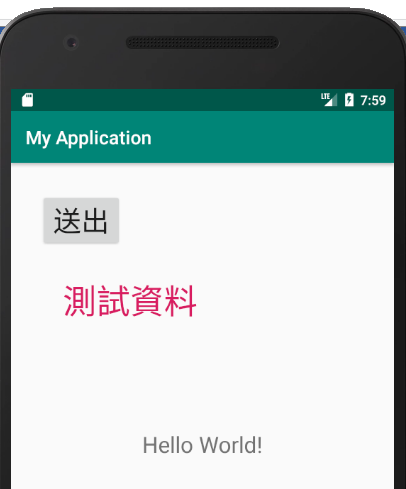
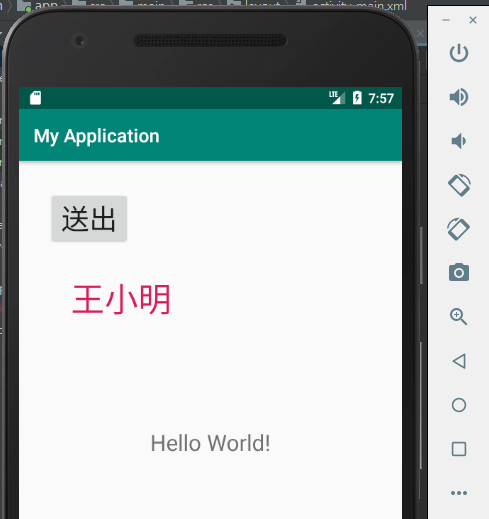


要脫離這個狀態，只要讓 HEAD 有任何分支可以指就行了，例如讓它回到 master 分支: **git checkout master**

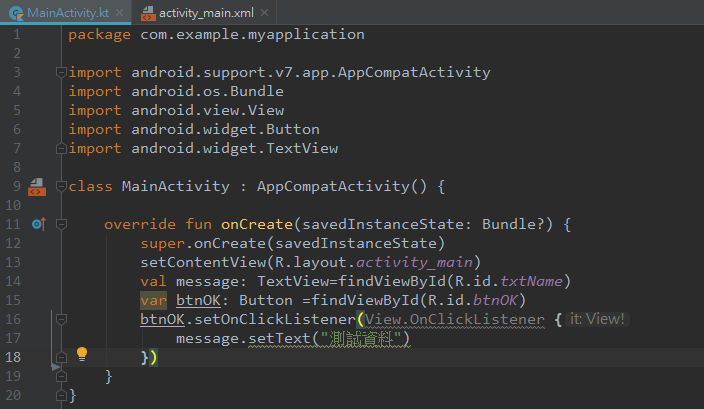


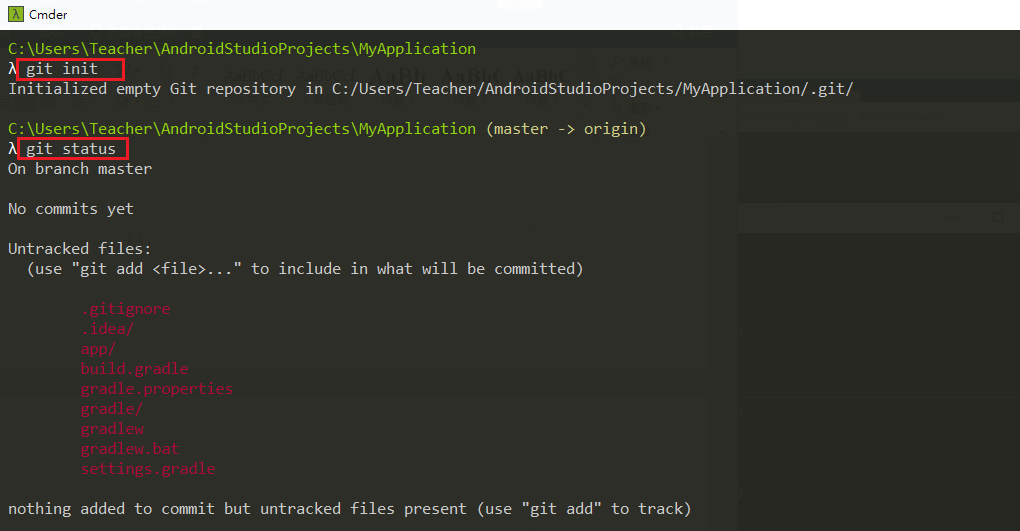
# Git專案管理實務

## Android Studio專案

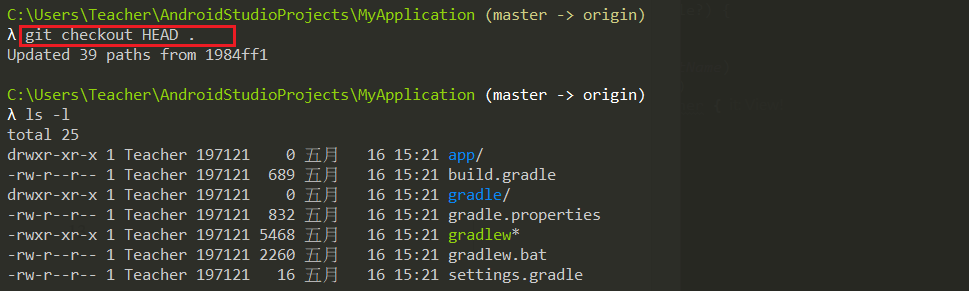


程式寫得很差，kotlin不懂，待查….

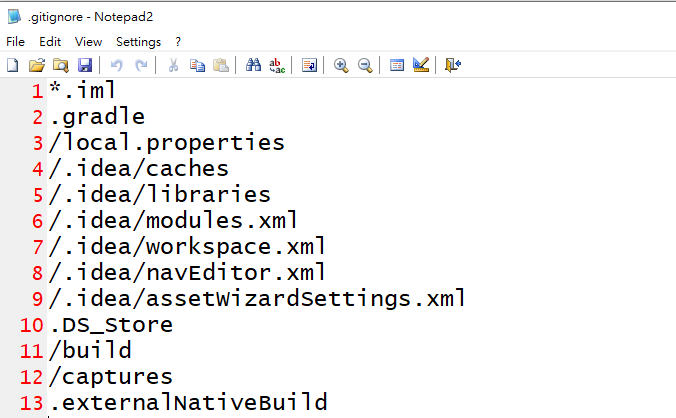




測試，先全部刪除所有檔案(.git除外)，再取出檔案(復原)OK



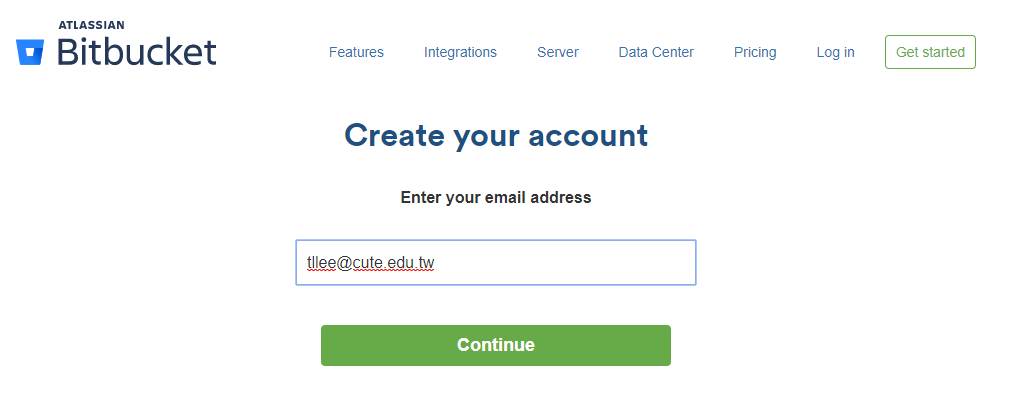
Android Studio預設(專案會自動產生)的.gitignore

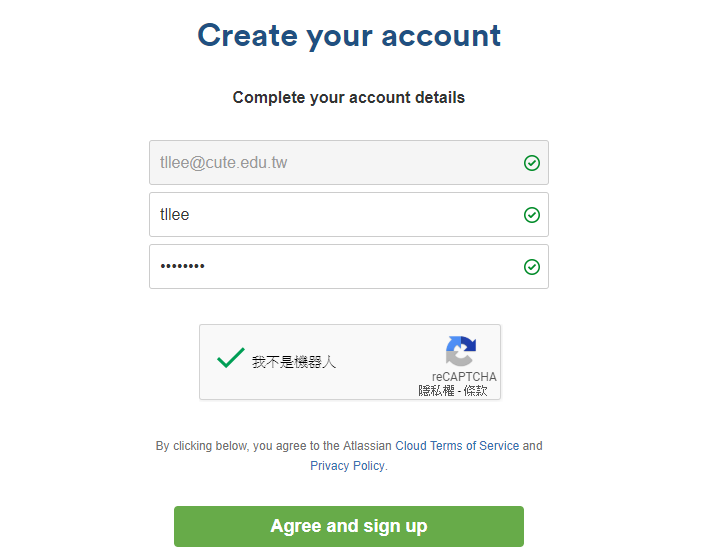


## Visual Studio專案

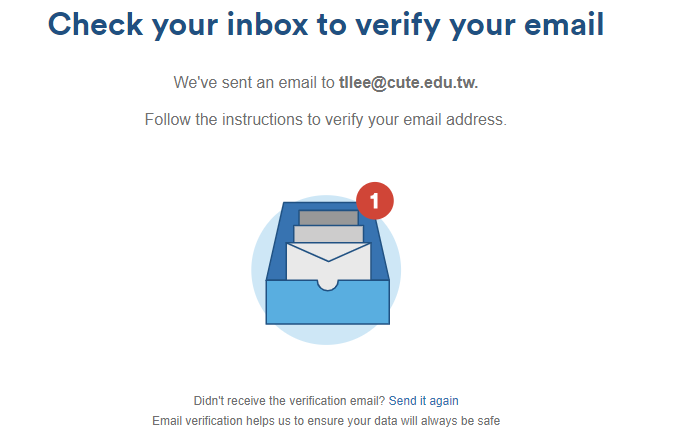
## 使用圖形介面

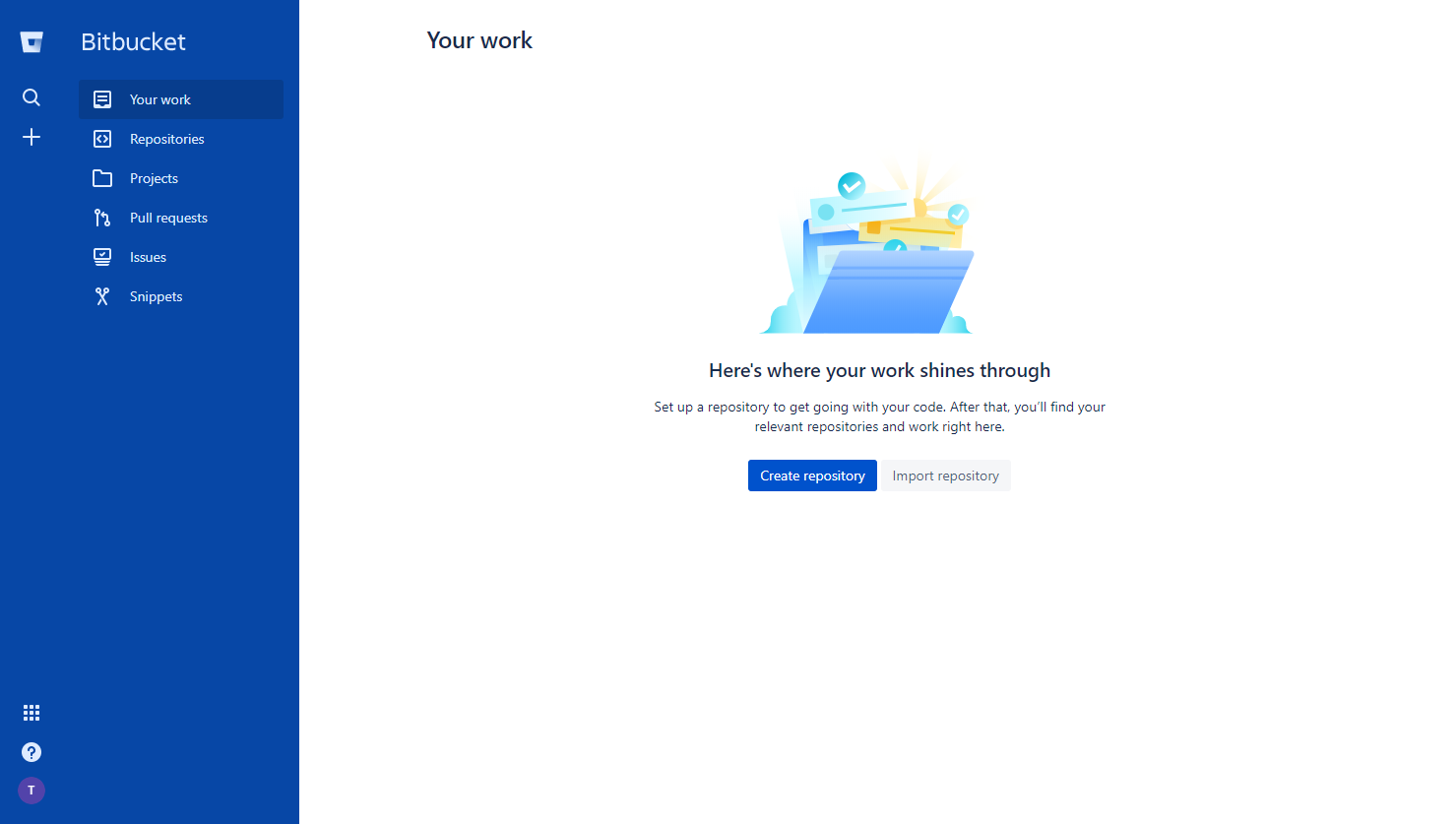
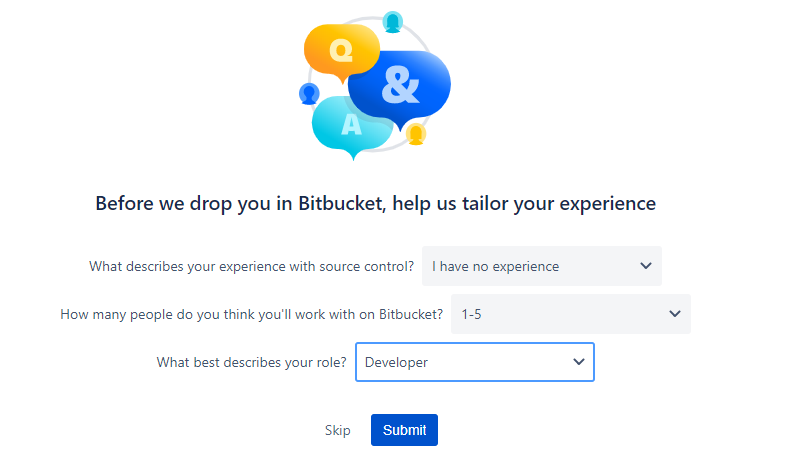
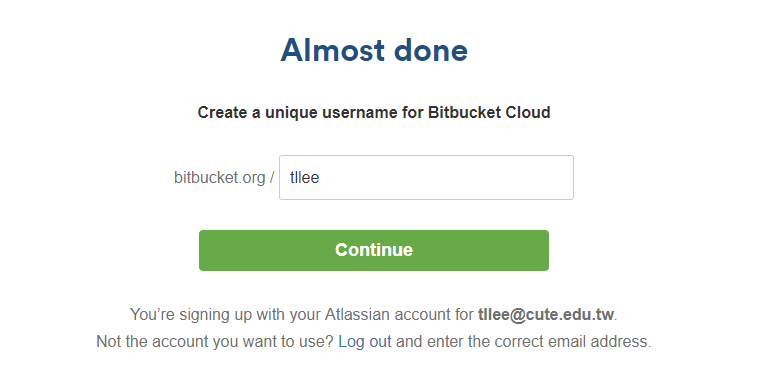
Sourcetree安裝[6]

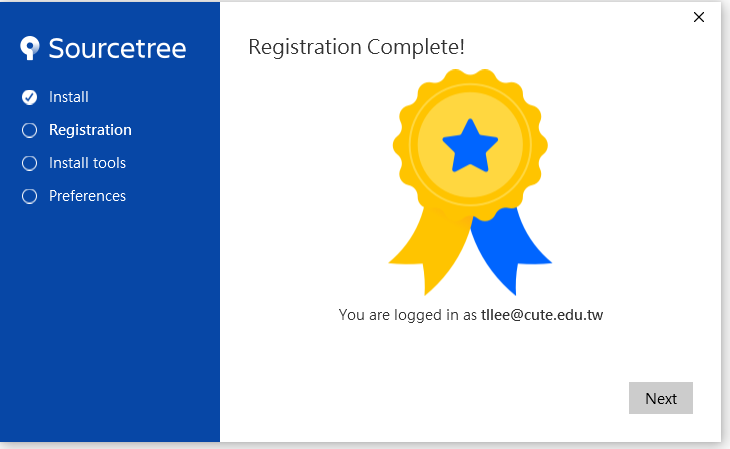


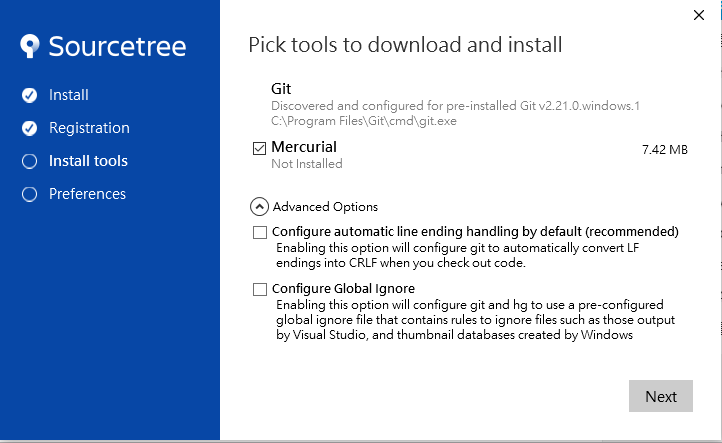


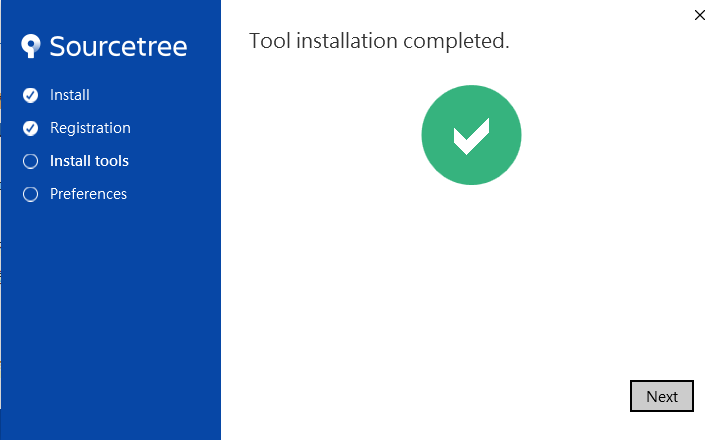
[tllee@cute.edu.tw](mailto:tllee@cute.edu.tw) fabien98

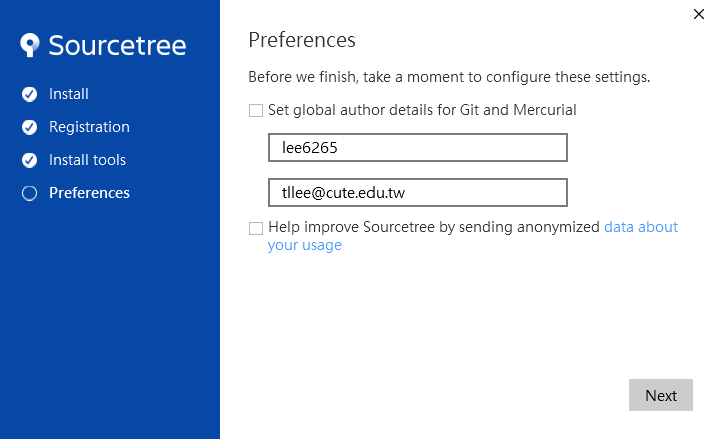




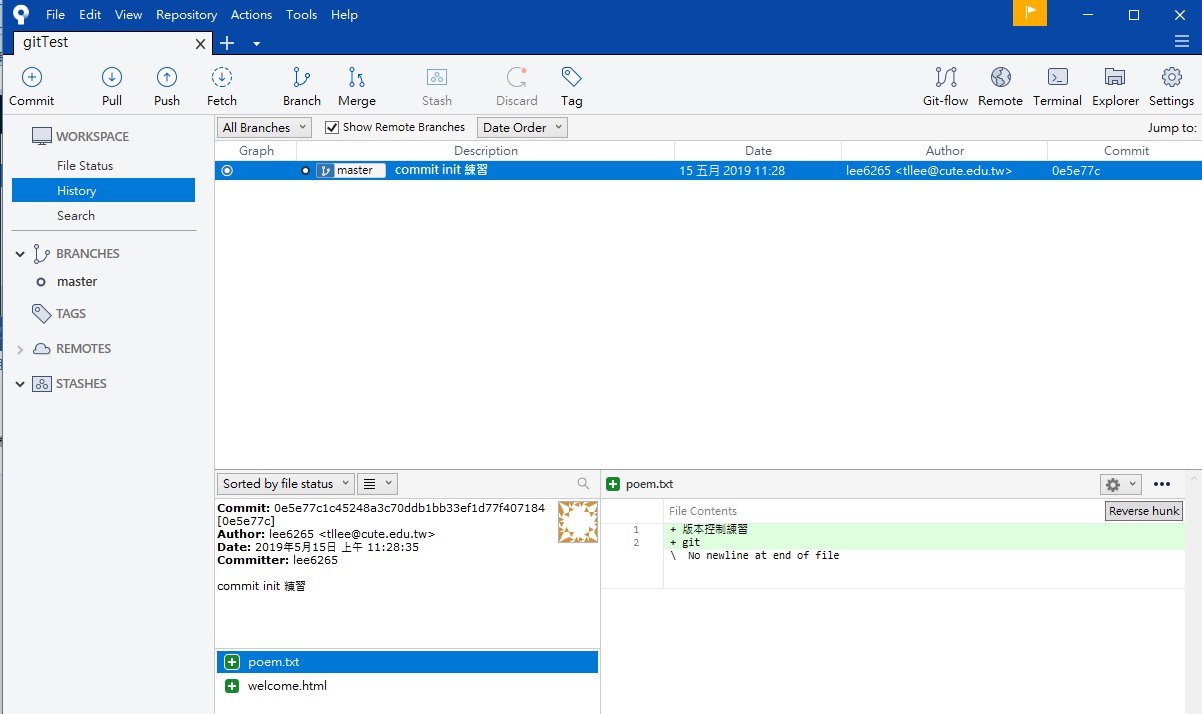








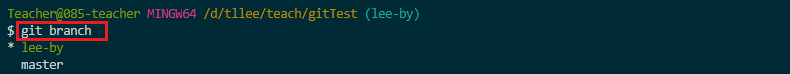
安裝OK

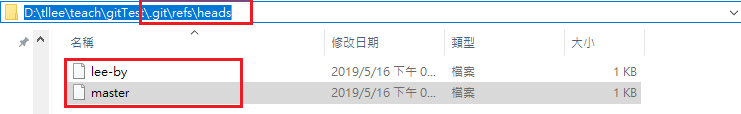


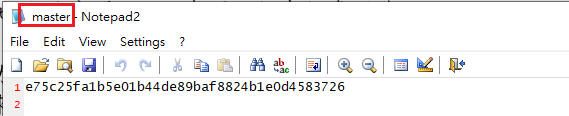
# 專案的分支(Branch)

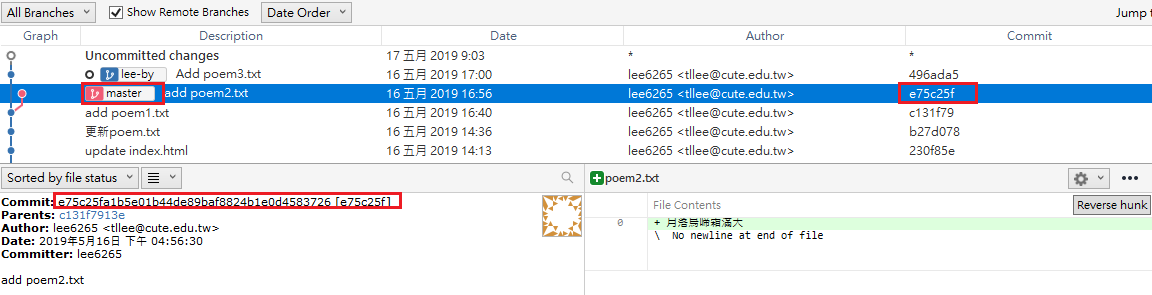
在開發的過程中，一路往前 Commit 也沒什麼問題，但當開始越來越多同伴一起在同一個專案工作的時候，可能就不能這麼隨興的想 Commit 就 Commit，這時候分支就很好用。例如想要增加新功能，或是修正 Bug，或是想實驗看看某些新的做法，都可以另外做一個分支來進行，待做完確認沒問題之後再合併回來，不會影響正在運行的產品線。

所謂的分支，其實就只是一個有 40 個字元的檔案，它藏在 .git 目錄裡面。如果 git branch 後面沒接任何參數，它僅會印出目前在這個專案有哪些分支。Git 預設會幫你設定一個名為 master 的分支，前面的星號 \* 表示現在正在這個分支上(lee-by)。







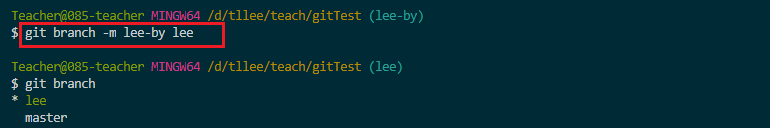


上圖中，在lee-by字樣前面的空心小圓圈表示目前正在這個分支上（也就是 HEAD）。

新增分支



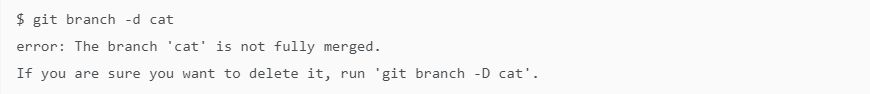
分支改名字 **git branch -m**



刪除分支 **git branch -d**



但如果要刪的分支還沒被完全合併，Git 會有貼心小提示，因為 cat 的內容還沒被合併，所以使用 -d 參數不給刪。這時只要改用 -D 參數就可以強制刪除。





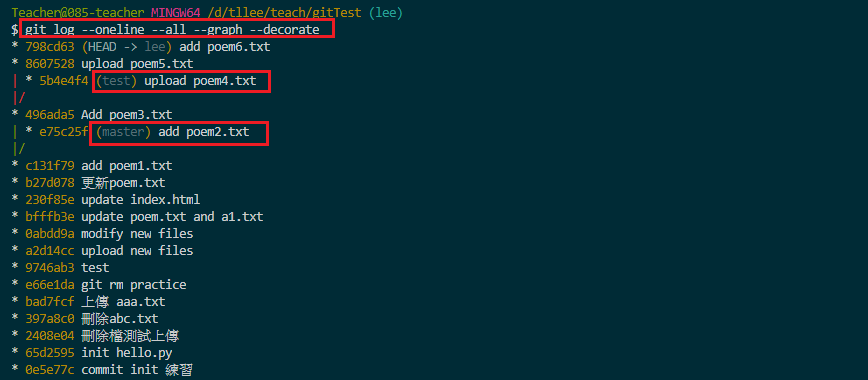
若不小心誤刪了，救得回來嗎？

請先記一下這個訊息「Deleted branch cat (was b174a5a).」，待會可能會用到它。刪掉分支，那些 Commit 還是在，只是因為你可能不知道或沒記下那些 Commit 的 SHA-1 值，所以不容易再拿來利用。只要有SHA-1值就可回復了

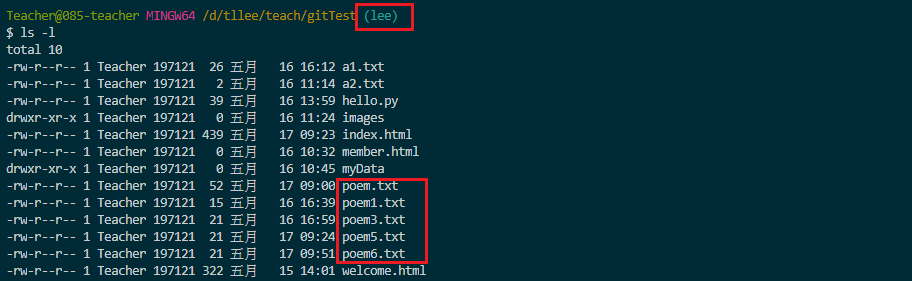


觀察各分支: **git log**

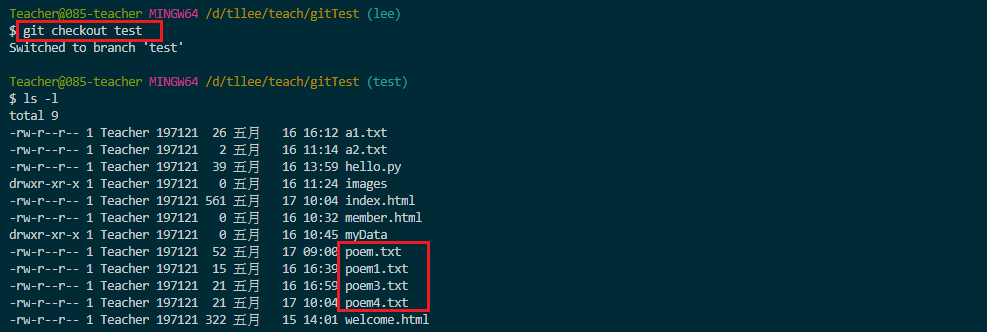
參數: **--graph** 畫出commit節點演進圖、**--all**顯示所有分支、**--decorate** 標示分支名稱、**--oneline**以最精簡的方式顯示



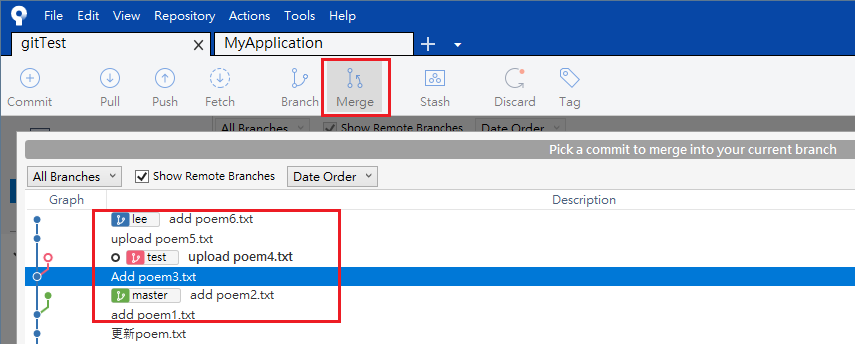
注意檔案的改變情況，現在在**lee**分支，所以就沒有poem2.txt(master)和poem4.txt(test)兩個檔案。



切換到**test**分支，缺poem2.txt(master)和poem5.txt、poem6.txt(lee)3個檔案。

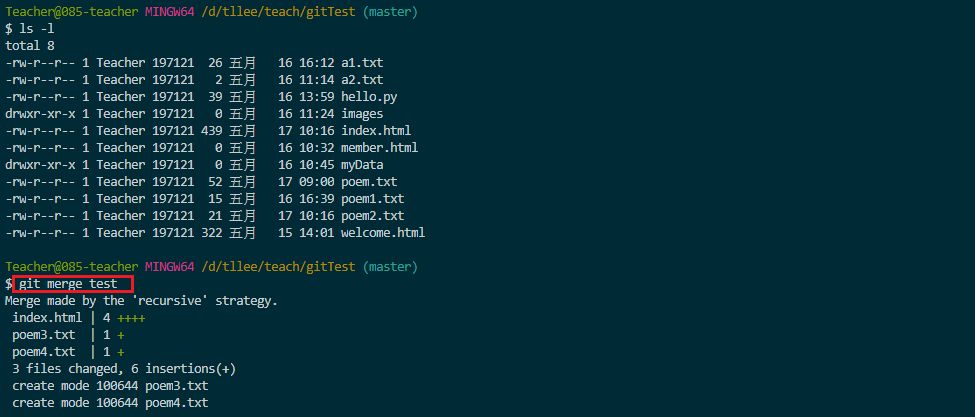


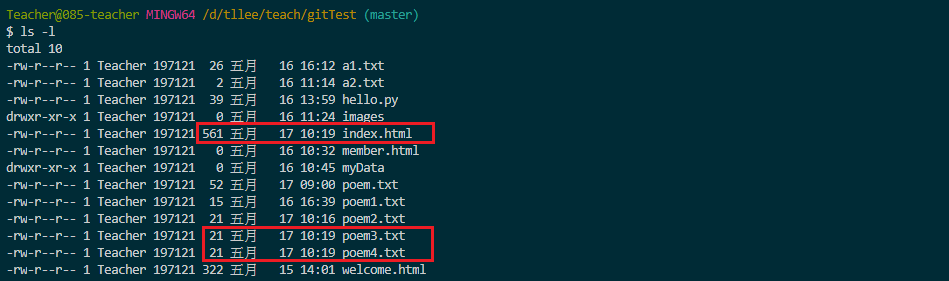
在Sourcetree GUI軟體中要選merge才可以看到各分支的詳細情況

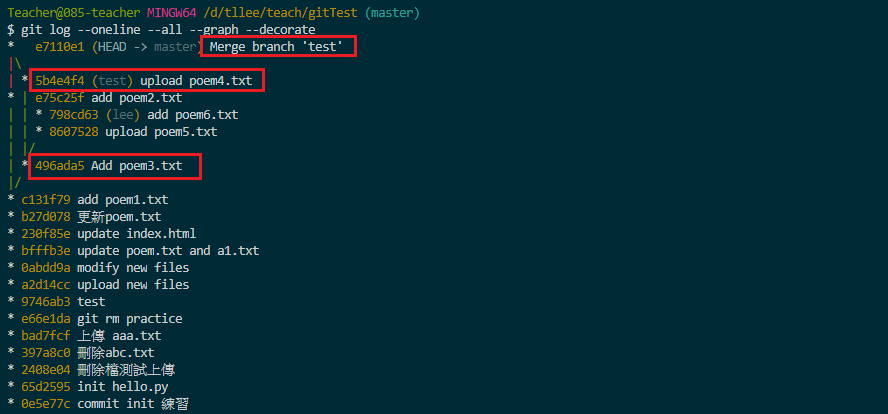


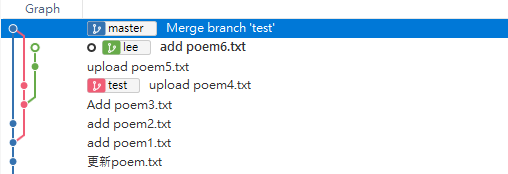
## 合併分支

任務執行的差不多了，就要準備合併回來了。如果我想要 master 分支來合併 test 分支的話，我會先切回 master 分支，接下來，合併分支是使用 **git merge**





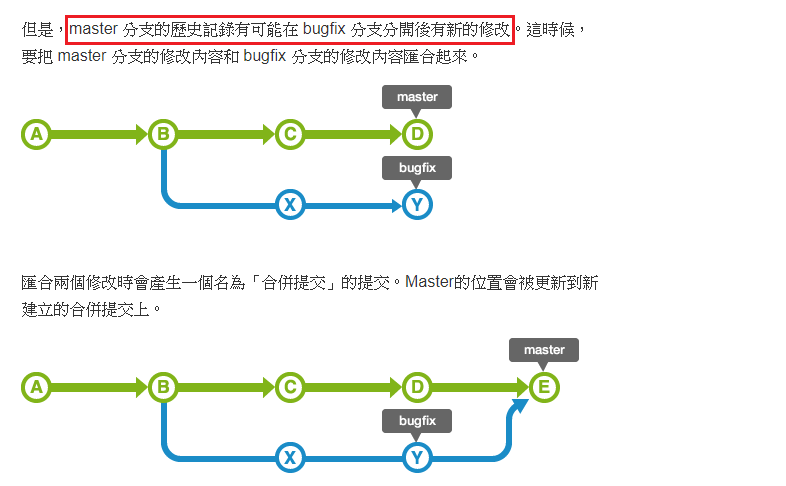




fast-forward merge



3-way merge



# 參考資料

1. <http://book.lib.ksu.edu.tw/blog/1911/%E5%A6%82%E4%BD%95%E8%AE%93word%E8%87%AA%E5%8B%95%E7%94%A2%E7%94%9F%E7%9B%AE%E9%8C%84> 如何讓Word自動產生目錄
2. <https://gitbook.tw/> 為你自己學 Git，高見龍 @ 五倍紅寶石
3. <https://github.com/doggy8088/Learn-Git-in-30-days/blob/master/zh-tw/01.md> Learn-Git-in-30-days
4. <https://blog.techbridge.cc/2018/01/17/learning-programming-and-coding-with-python-git-and-github-tutorial/>
5. <https://git-scm.com/download/win>
6. <https://ithelp.ithome.com.tw/articles/10206852> Sourcetree安裝
7. <https://bitbucket.org/dashboard/overview> Bitbucket網站
8. <https://git-tutorial.readthedocs.io/zh/latest/installation.html> Practical guide for git users
9. <https://github.com/github/gitignore> gitignore for programming languages
10. <https://backlog.com/git-tutorial/tw/stepup/stepup1_4.html> fast forward merge
11. ...
12. …

全文完