GIT Tutorial

Author by Glee

Table

終端機指令使用

切換目錄

```
> cd 目錄位置
```

回到上一層目錄

```
> cd..
```

查詢檔案夾內容

```
> ls -al
```

查詢目前所在目錄

```
> pwd
```

比較兩個檔案間的不同：

這時會列出檔案2 多了哪些內容 或是 少了哪些內容

```
> diff -u 檔案1 檔案2
```

GIT 的安裝

檢查有沒有安裝 Git，如果輸入下列指令後沒有訊息，代表沒有安裝 Git

```
> git —version
```

如果沒有安裝 Git，可採取手動安裝的方式，請先下載檔案：

http://git-scm.com/download/mac

GIT 初期設定

第一次使用 Git，都會需要設定個人的資訊，讓團隊的其它成員知道你是誰。這個設定只需做一次。

設定姓名

```
> git config --global user.name “glee”
```

設定電子郵件

```
> git config --global user.email glee@example.com
```

檢查設定

```
> git config --list
```

為專案建立 Git

前往到要讓 Git 做為管理的目錄下，並輸入下列指令：

```
> git init
```

只要在目錄下有 「.git」這個資料夾，就代表這個目錄（含以下的子資料夾）都已列入管理。

檢查 Git 狀況

使用 Git 的時候，最常用到的指令就是檢查 Git 的狀況，要檢查哪些檔案還沒被加入 Git 管理、將更改過的檔案加入至 Git 等，都是從這裡判斷。

```
> git -status
```

將檔案加入至 Git 管理

在 Git 中的檔案管理分為「未追蹤」及「已追蹤」兩種，無論是哪一種，只要檔案一經新增或修改，想列入 Git 管理時，都可以使用下列指令完成。

```
> git add 檔案名稱
```

如果想把資料夾下的所有檔案列入 Git 管理，可以輸入：

```
> git add . （小數點）
```

從遠端拷貝 Git 專案

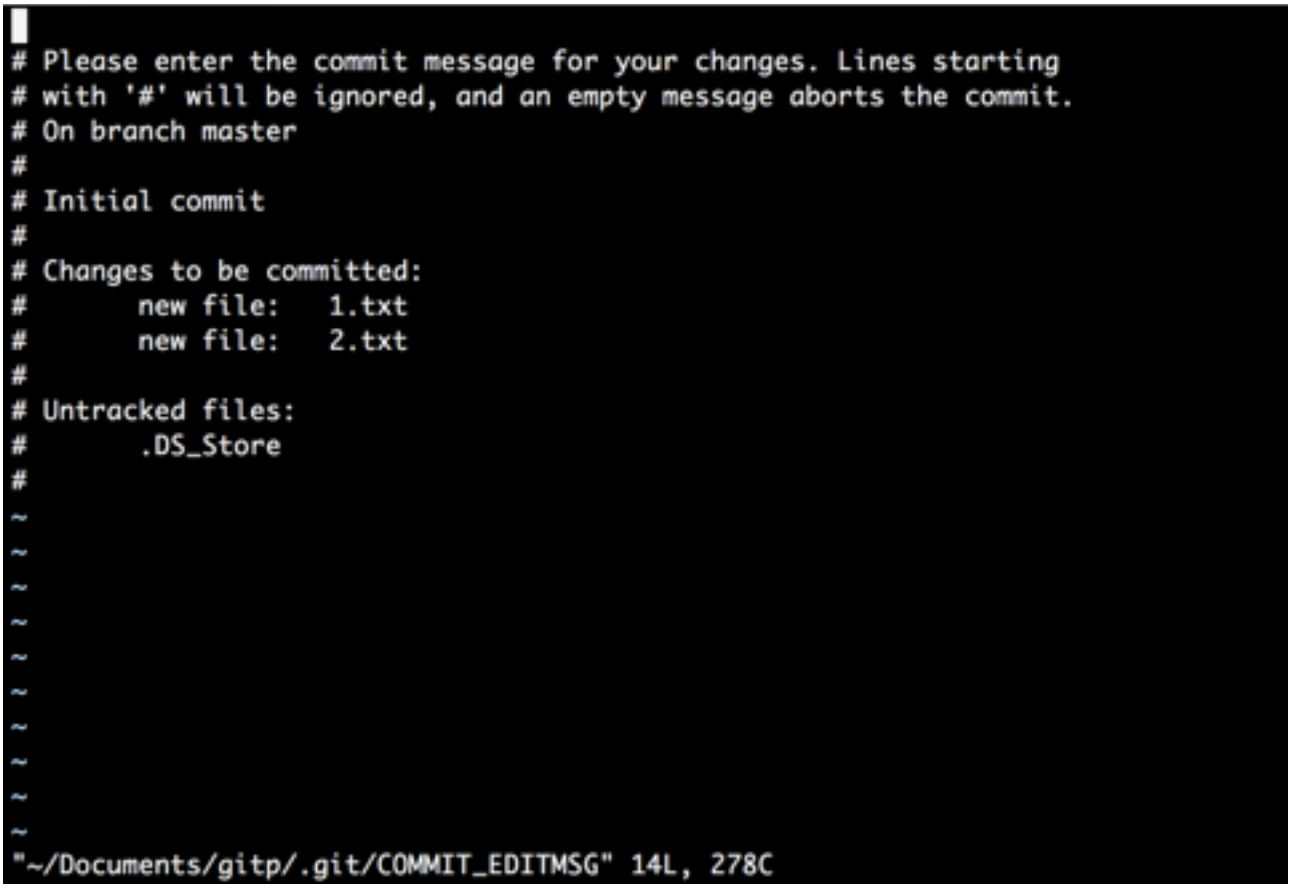
```
git clone 遠端網址
```

送出版本 (Commit)

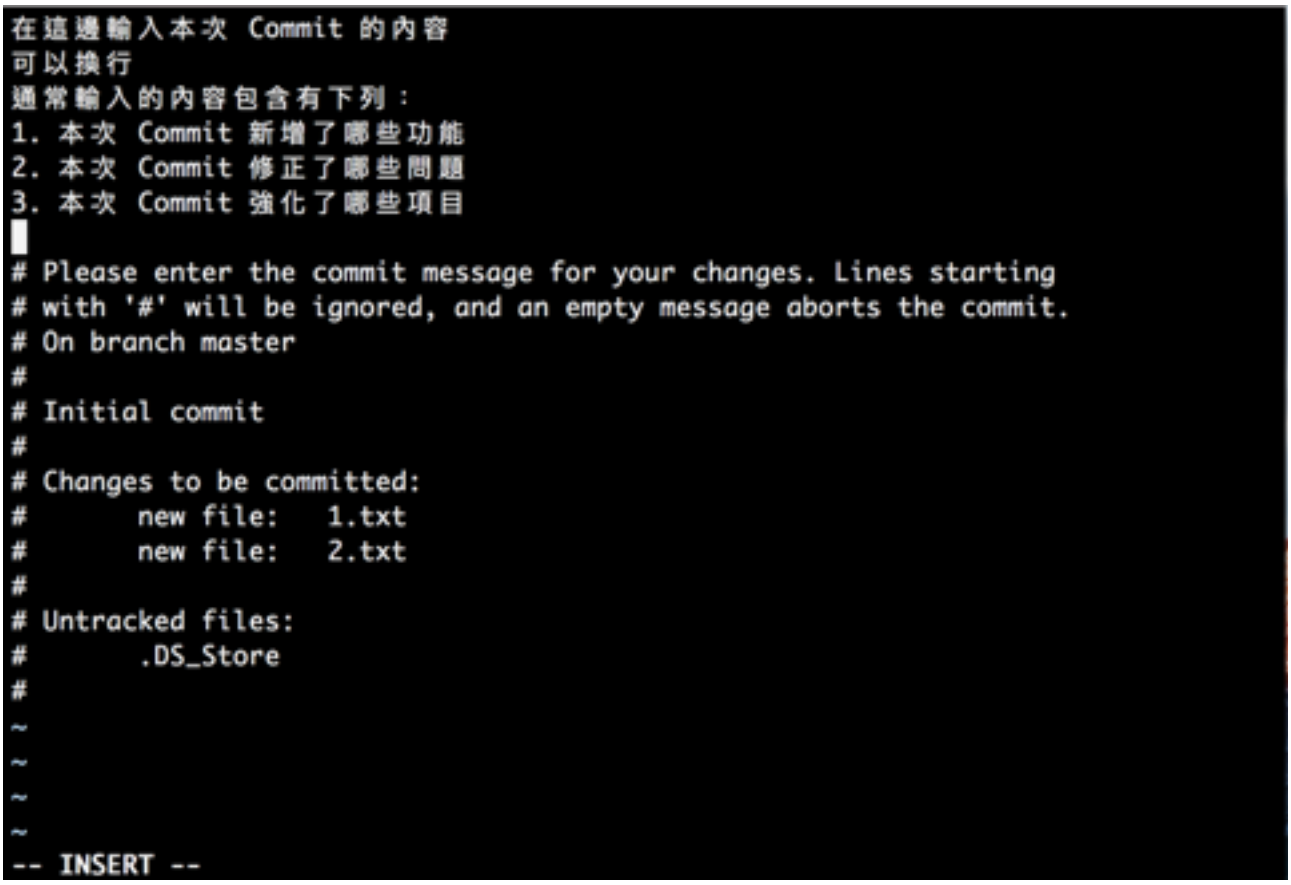
每當一次修改告一段落，可以將現有的結果提交版本出去。這個動作稱之為 Commit。Commit 的時間點不一，依每個工程師習性或團隊的要求有所不同。

```
git commit
```

送出上述指令後，會跳出 Commit 補充訊息提供你輸入。而現在你看到的畫面稱為 vi，也就是 UNIX 下常見的文字編輯器。



要開始在這個編輯器裡面輸入文字，需先按一下「I」，進入編輯模式。



填寫完畢後，就可以按一下 ESC 鍵離開編輯模式。再按冒號（:）進入編輯器指令模式，然後輸入存檔（save）並離開（quit）。

```
:wq
```

檢查本地庫所有版本

若要檢查剛才的 Commit 是否已經成功，或是想查詢之間 Commit 過的版本，可以輸入：

```
> git log
```

每次的 Commit 會有下列資訊，包含 Commit 的 ID（唯一值），Commit 的工程師為誰、時間為何，以及 Commit 的備註等。

繪製 Git 樹

Git 就像一棵樹，每次的 Commit 都是一個節點，節點會結在樹幹上。在這個概念裡，節點被稱為 Commit，而樹幹被稱為 Branch，利用下列指令可以把這個你的 Git 目錄圖像化：

```
> git log --graph --decorate --all
```

管理永不 Git 的檔案

有些檔案，可能是外部的第三方函式庫，我們永遠不要其被 Git 管理，這時你可以把這些檔案都加入到 .gitignore 的檔案裡。 .gitignore 的檔案是一個隱藏的文字檔，使用 Finder 建立會比較費時，所以我們改採以終端機的方式建立。

首先，前往專案的目錄下，直到與 .git 資料夾同層。

輸入下列指令：

```
> touch .gitignore
```

touch 指令是一個在 UNIX 下產生檔案的指令。接著，就可以利用 vi 編輯器修改這份文件。

```
> vi .gitignore
```

在 .gitignore 文件下，你可以將名為 glee.txt 的檔案列入永不管理，只要在文件中輸入 glee.txt 就好。如果要將所有副檔名為 .DS_Store 的檔案都列入永不管理，只需要換一行輸入 *.DS_Store 就可以了。最後記得，.gitignore 的這個檔案一定加入 Git 管理，所以請輸入：

```
> git add .gitignore
```

這樣以後團隊中的所有人，只要維護這份 .gitignore 的檔案即可。

在 XCode 開發環境裡，有些東西我們可以不列入 Git 管理，請參考：

https://www.gitignore.io/api/objective-c

加入遠端 Repository

上述的所有動作都是在本機（Local）完成，也就是說，即使沒有網路的環境下，你一樣可以操作上述的指令，當你確定要把這些 Commit 及 Branch 都送上像是 Github 這種雲端的 Repository 時，就得先告訴你的專案其遠端 Repository 的位置。

```
> git remote add origin 遠端網址
```

PULL

Pull 是把團隊上傳至遠端的結果下載回來，如果與你本機的專案出現程式碼衝突的話，就會進行合併。這是 Pull 的指令

```
> git pull
```

Push

當你解決了本機與網路上的程式碼衝突問題後，就能把自己的程式碼及合併過後的程式碼送上遠端。

```
> git push
```

分支 Branch

分支，是 Git 中非常重要的技巧，它同樣可以在本機作業。當你要開發新的功能時，或是要調整程式碼，卻擔心會大動到原本可以執行的程式碼時，利用分支就能幫你解決這個問題。

在 Git 上，都會有一個主幹，稱為 master。其它的分支你可以自訂名稱。

建立分支。下列這個指令是建立一個 TEST 分支，並切換過去。

```
> git checkout -b “TEST"
```

以某個 commitID 建立 old 分支，並切換過去。

```
> git checkout -b old commitID
```

列出所有的分支

```
> git branch
```

切換到其它分支。請注意，當你要切換到其它分支時，最好確保當前分支的 git status 的工作目錄是乾淨的，再切換過去。

```
> git checkout 分支名稱
```

刪除某個 branch

```
> git branch -D branch的名稱
```

比較與合併程式碼

比較當前未 Stage 的檔案，與上一次 Commit 的檔案：

```
> git diff
```

比較當前已經 Stage 的檔案，與上一次 Commit 的檔案：

```
> git diff --staged
```

比較兩個版本間的不同，SHA 是指 Commit 的 ID

```
> git diff SHA1 SHA2
```

比較某個檔案在兩個版本間的不同

```
> git diff SHA1 SHA2 --filepath
```

使用視覺工具比較與合併程式碼

我們使用 P4Merge 這套軟體，做為視覺化工具。

下載位置：https://www.perforce.com/downloads/helix#product-10

設定 Git 讓其使用 P4Merge 做為比較及合併工具：

利用 vi 編輯器開啟 ~/.gitconfig 檔案。

https://goo.gl/qTv5xp

```
[merge]
	tool = p4mergetool
[mergetool "p4mergetool"]
	cmd = /Applications/p4merge.app/Contents/Resources/launchp4merge $PWD/$BASE
$PWD/$REMOTE $PWD/$LOCAL $PWD/$MERGED
	trustExitCode = false
[mergetool]
	keepBackup = false
[diff]
	tool = p4mergetool
[difftool "p4mergetool"]
	cmd = /Applications/p4merge.app/Contents/Resources/launchp4merge $LOCAL $REMOTE
```

未來就可以用 difftool 取代 diff 指令、及使用 mergetool 取代 merge 指令。

例如：

```
> git difftool 1.txt 2.txt
```

合併 Merge

(master)—A—B—C—E　　\

—G

(feature)—A—B—C—D—F /

在 Git 的合併有兩種。一為 Merge，二為 Rebase。Merge 是將兩條分支合在一起的做法，例如 Feature 分支要合併到 master 主幹時，就可以下這個指令：

```
> git checkout master （先確認在 master 的主幹下）
> git merge feature
```

合併 Rebase

Rebase 前：

(master)—A—B—C—E

(feature)—A—B—C—D—F

Rebase 後：

(master) —A—B—C—E—原本 Feature 上的 D 重新 Commit 為 G — F 變成 H

所以是：

(master) —A—B—C—E—G—H

Rebase 與 Merge 的結果是一樣的，不過在歷史樹的繪製上會比較乾淨，因為 Rebase 像是傢接，將新的 Commit 接在某個 Commit 上，

```
> git checkout feature （先確認在 feature 的下）
> git rebase master (與 master 傢接)
> git rebase —continue (如果出現衝突，解決後要繼續作業)
> git rebase —abort （如果出現衝突，不想繼續作業）
```

參考資源

1. Github learning — https://training.github.com/
2. Github cheetsheet — https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf

好用的 Git 工具

這邊有幾項好用的視覺化工具，但 Glee 仍建議大家初學時別太依賴這些工具，因為在不知道概念的情況下採用這些工具，可能會不小心把 Git 樹搞壞，弄壞自己的程式碼，得不償失啊！

1. Tower 2 — http://www.git-tower.com
2. SourceTree — https://www.sourcetreeapp.com
3. Github Desktop — https://desktop.github.com