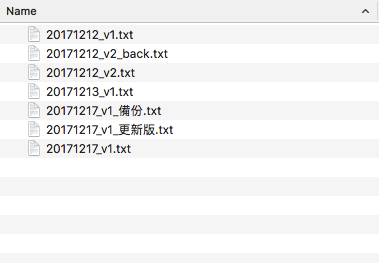
# Git與Github版本控制基本指令與操作入門教學



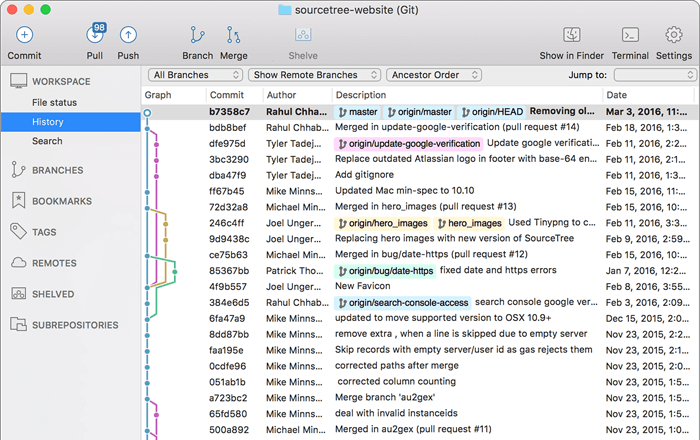
## 前言

要成為一個真正的軟體工程師(Software Engineer)除了掌握基礎開發能力外，更重要的是和其他工程師和開發者團隊合作和溝通的能力，所以若你能培養出掌握Git等版本控制操作和Git server架設的能力，你會更容易參與開放原始碼(open source)的社群和提昇自己在職場上的價值。

什麼是版本控制系統(Version Control System)？



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(不然要用email或是其他工具傳送和管理十分麻煩，尤其程式又常常會有不同版本修改的問題！)。在沒有版本控制系統時，我們常會在編輯檔案前複製一個備份，或是在更新檔案後產生許多重複檔案，非常不便且難以維護。因此，使用版本控制系統的需求就應運而生！



一般在軟體開發中又分為中央式系統(例如：[Subversion](https://zh.wikipedia.org/wiki/Subversion" \t "_blank)、[CVS](https://zh.wikipedia.org/wiki/%E5%8D%94%E4%BD%9C%E7%89%88%E6%9C%AC%E7%B3%BB%E7%B5%B1" \t "_blank)等)與分散式系統(例如：[Git](https://git-scm.com/" \t "_blank)、[BitKeeper](https://blog.techbridge.cc/2018/01/17/learning-programming-and-coding-with-python-git-and-github-tutorial/BitKeeper)、[mercurial](https://zh.wikipedia.org/zh-tw/Mercurial" \t "_blank)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，但分散式系統仍然可以有檔案上鎖功能。

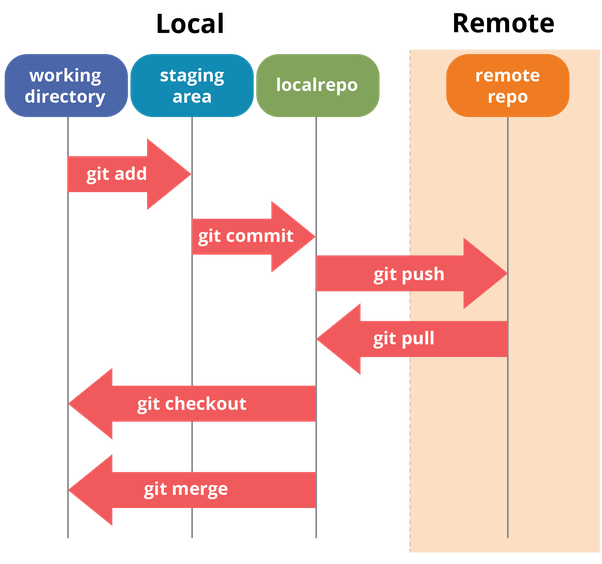


## 什麼是Git？什麼是Github？

[Git](https://zh.wikipedia.org/wiki/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](https://zh.wikipedia.org/zh-tw/Linux" \t "_blank)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比較歷史版本差異)。

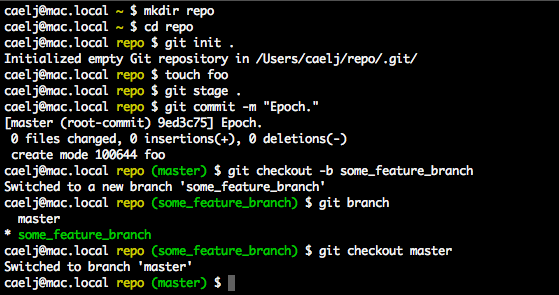
[Github](https://github.com/)則是一個支援git程式碼存取和遠端托管的平台服務，有許多的開放原始碼的專案都是使用Github進行程式碼的管理。若是讀者未來有志於從事程式設計相關工作的話，建議可以熟悉掌握Git和Github的使用，並建立自己的Github profile作品集。

## Git基本觀念



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)。

當自己開發時會在工作資料夾工作，當要進入檔案庫之前會先將檔案加入暫存區，確認沒問題則commit到檔案庫中，最後push上去remote環境。在Git中若是有和其他開發者一起合作，則會需要處理不同branch之間conflict和merge的問題。



## Git與Github實戰操作入門教學

接下來我們用一個實際的實戰範例讓讀者可以快速的掌握那些基礎必知必會的Git/Github操作技能：

### 安裝並且設定Git

任務1：安裝Git到你的電腦上，並且設定好Git內部的使用者名稱和電子信箱

Linux若是使用Debian基礎的作業系統可以在終端機輸入以下指令安裝git：

$ apt-get install git

若是Fedora系列可以輸入：

$ yum install git-core

若是windows可以下載安裝[cmder](http://cmder.net/" \t "_blank)這個模擬Linux terminal終端機時選擇完整版本就會順便安裝或是到[Git官網安裝](https://git-scm.com/" \t "_blank)。當然在市面上有許多免費[GUI圖形化的Git操作軟體](https://git-scm.com/download/gui/linux)，若是初學者則建議先熟悉整個git工作模式和指令，再去使用圖形化工具會比較好，這樣你才比較知道圖形化程式背後做了什麼事情。Mac則是可以到[Git官方網站](https://git-scm.com/downloads" \t "_blank)選擇對應作業系統，按照步驟完整下載安裝。另外也可以參考[中文安裝教學](https://git-scm.com/book/zh-tw/v1/%E9%96%8B%E5%A7%8B-%E5%AE%89%E8%A3%9D-Git)。

由於我們會選擇Github當作遠端托管程式的環境，所以我們也可以根據作業系統安裝[Github桌面版](https://desktop.github.com/" \t "_blank)當作操作工具(內建安裝Git)。



若是安裝完成，打開終端機(terminal)或cmder輸入以下指令，若成功顯示版本(建議Git版本在2以上)，代表安裝成功囉！terminal(終端機)是一個可以下指令讓電腦做事情的互動介面(例如：新增檔案、移動到資料夾等)

//其中$為提示字元，在輸入指令時不用輸入該符號，否則會錯誤，若是使用windows cmder預設是λ

$ git --version

接下來設定你的帳戶，讓Git知道這台電腦做的修改要連結到哪一個使用者(待會我們要在Github上註冊帳號，建議使用一致的帳號和電子信箱)：

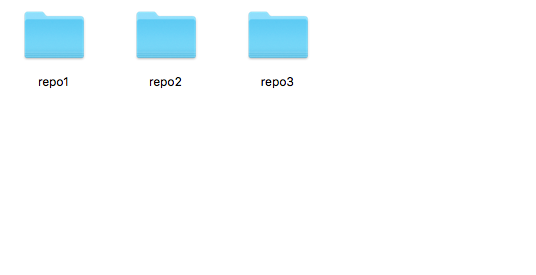
$ git config --global user.name "<Your Name>"

設定電子郵件：

$ git config --global user.email "<your@gmail.com>"

### 建立一個本機的repository

任務二：在自己的電腦上建立一個新的local repositories(本地檔案庫)



repository(檔案庫)是一個什麼樣的概念呢？事實上repository就是一個專案，又簡稱repo。以電腦的檔案資料管理來看，我們通常會把同一個專案的資料放到同一個資料夾下，所以我們也可以把repository看成一個資料夾。

//建立一個hello-git資料夾

$ mkdir hello-git

//移動到hello-git資料夾

$ cd hello-git

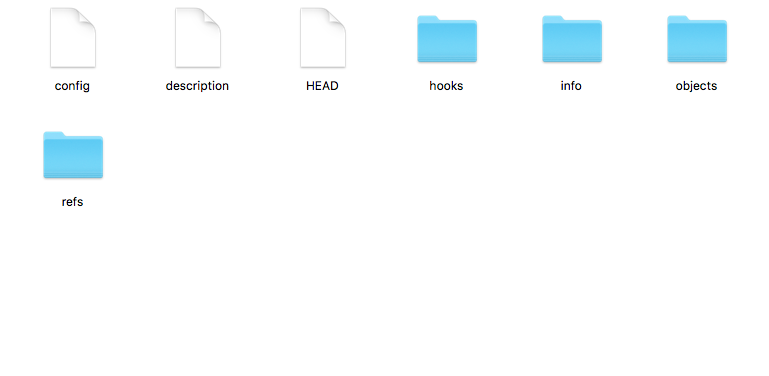
//將專案資料夾建立成git repository

$ git init

//列出專案資料夾下的檔案和資料夾(-l參數為列出詳細資料，-a為列出隱藏資料夾)

$ ls -la

當你執行git init後，你可以發現多出了.git這個隱藏資料夾，可以看到裡面檔案和資料夾如下：



到這邊恭喜讀者，完成你第一個git專案資料夾改造計畫！

### 檢視狀態、新增或修改commits

任務三：在你的repository檔案庫中建立一個新檔案，新增一些內容到該檔案並且將那些檔案修改提交commit到Git中。

接著我們使用文字編輯器，新增一個hello.py的檔案(裡面是一個會印出hello python & git字串的python程式)：

hello.py

print('hello python & git')

然後在終端機的專案資料夾下輸入git status顯示目前工作環境狀態：

$ git status

On branch master

Initial commit

Untracked files:

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

hello.py

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

我們會發現因為我們有新增新的檔案，但是還沒進到git追蹤範圍中/暫存區，所以我們要使用git add hello.py加入追蹤，這樣之後檔案有修改就可以追蹤到。

$ git add hello.py

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: hello.py

若是確認沒問題我們就準備commit進去repository囉！

//比較現在檔案和上次commit之間的差異，也就是說你做了哪些修改

$ git diff

//-m為輸入commit message，也就是說這個commit內做了哪些事情

$ git commit -m "Init hello.py"

[master (root-commit) ad6d328] Init hello.py

1 file changed, 1 insertion(+)

create mode 100644 hello.py

//commmit完成

$ git status

On branch master

nothing to commit, working tree clean

若是想反悔不想把檔案加入追蹤呢？

//檔案尚未加入過追蹤時使用，即可恢復到檔案尚未加入暫存區

$ git rm --cached hello.py

//若檔案已經在repository內，則使用以下指令

//repository與stage的檔案都會被還原到HEAD，但working directory內的檔案不變

$ git reset HARD

當你追蹤後修改了檔案，例如把hello.py的內容改成：

hello.py

print('hello python & git rock')

若有檔案修改，記得要再add修改的檔案(這是新手比較容易忘記的部分)，若是要放棄修改可以使用git checkout -- 檔案名稱

//比較現在檔案和上次commit之間的差異，也就是說你做了哪些修改

$ git diff

//查看目前工作狀態

$ git status

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: hello.py

commit這個修改時簡寫會寫成這樣(-a是add，-m為message簡寫，後面接訊息資訊)：

$ git commit -a -m "修改了hello.py"

### 註冊GitHub帳號

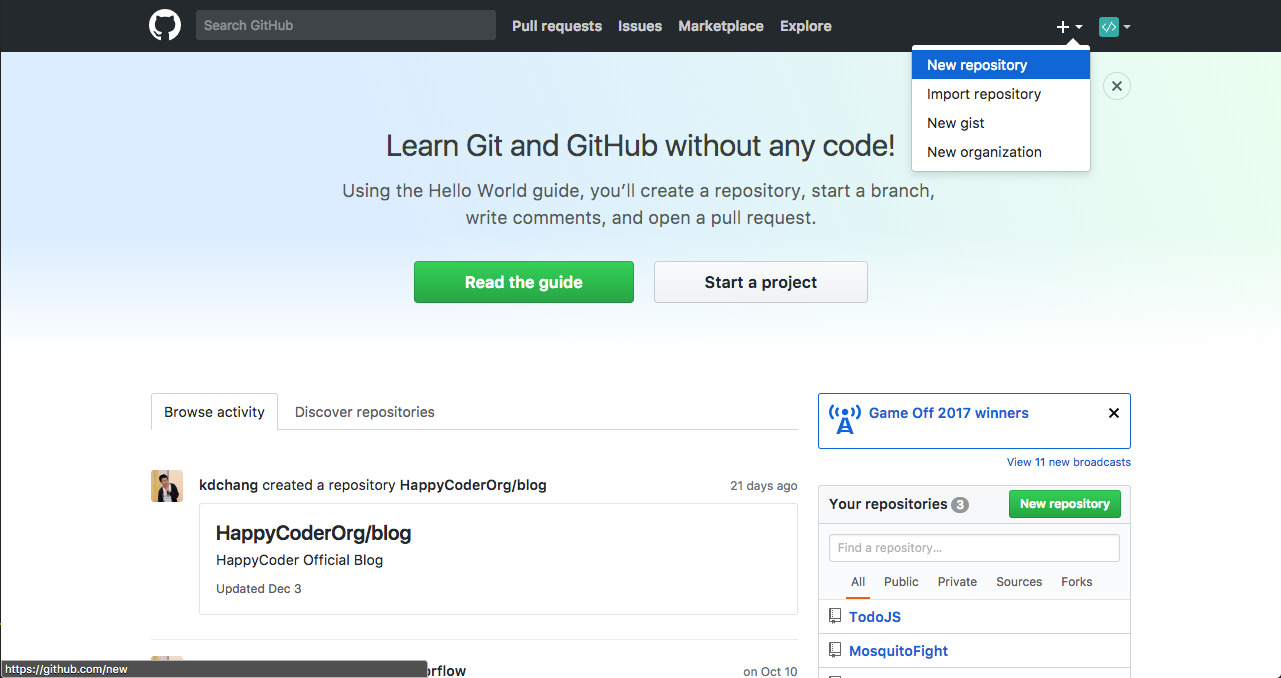
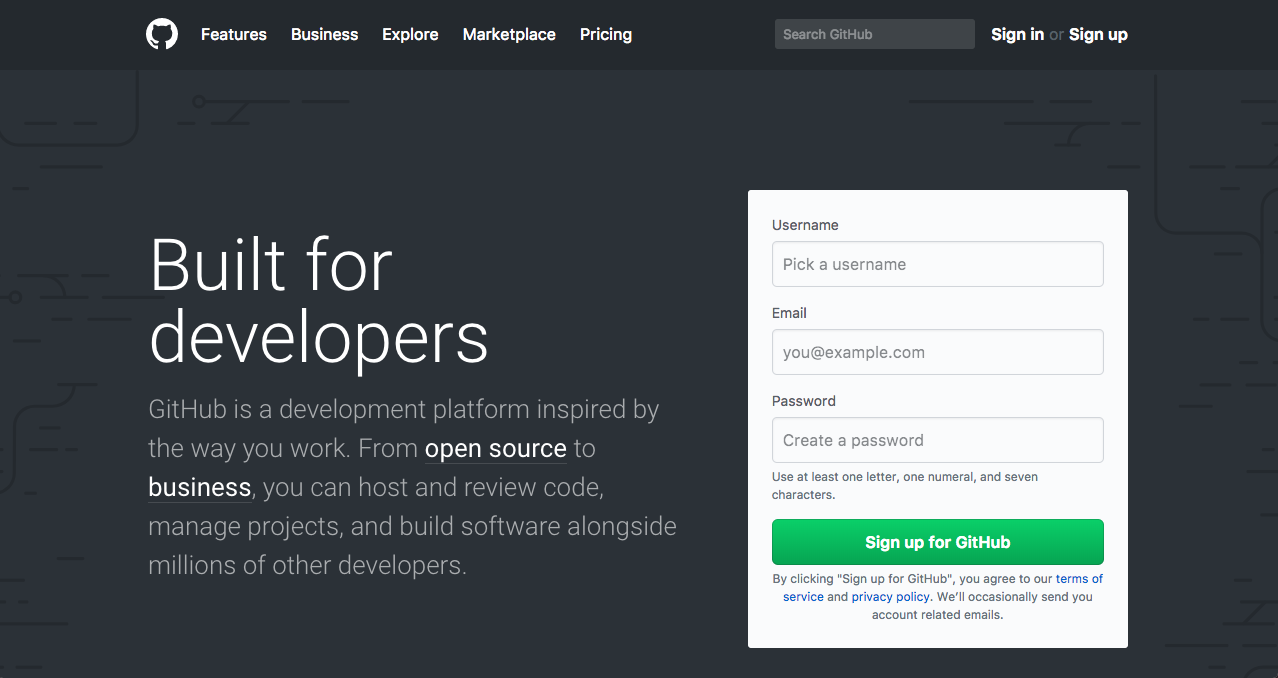
任務四：建立一個GitHub帳號，並在Git設定中加入使用者帳號

到目前為主我們程式主要是在我們local工作環境中操作，但還記得我們在Git基本觀念中有提到，git分為local和remote，若是我們想要和全世界其他開發者合作或是貢獻開放原始碼的話，我們可以透過github來當作我們remote工作環境，去管理我們程式碼，同樣的也可以透過github平台和其他開發者一起合作，參與開放原始碼的開發。github若是公開的repo是無限制數量免費的，但是若是想使用private repo可以[參考付費方案](https://github.com/pricing" \t "_blank)。

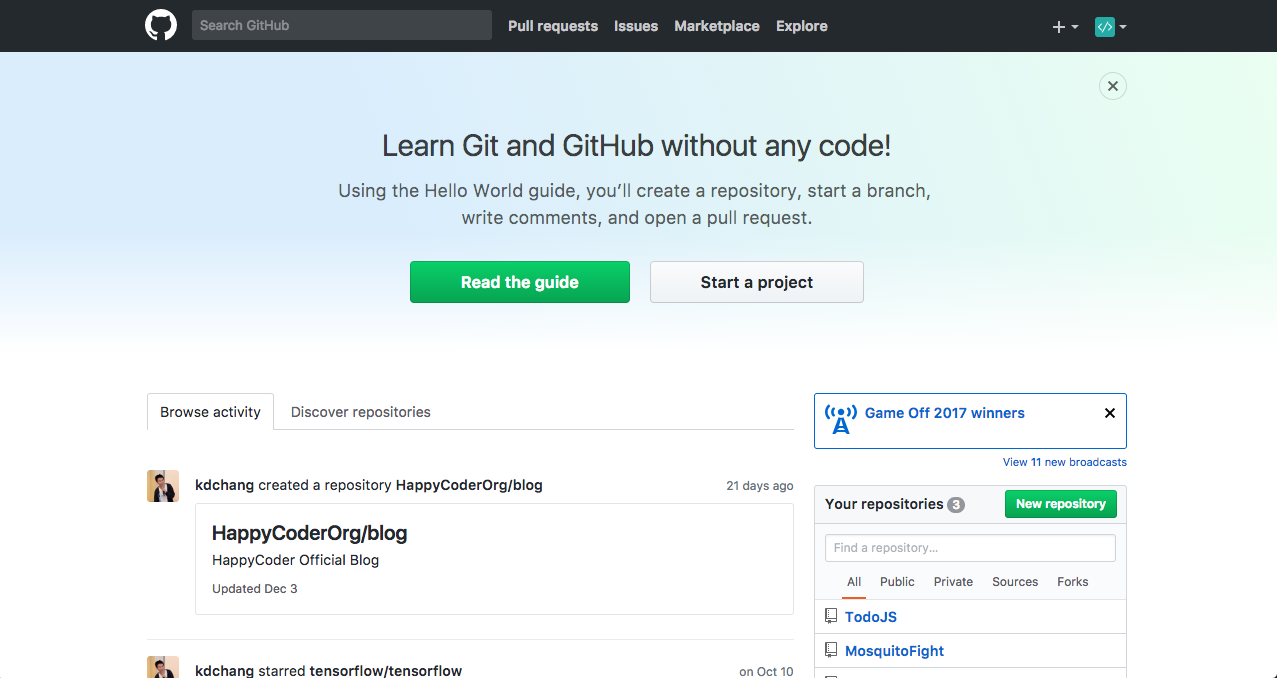
//注意大小寫要一致

$ git config --global user.username <你的github使用者名稱>

先到[github.com](http://github.com/)註冊帳號：



點選右上角+來新增new repository(檔案庫)：

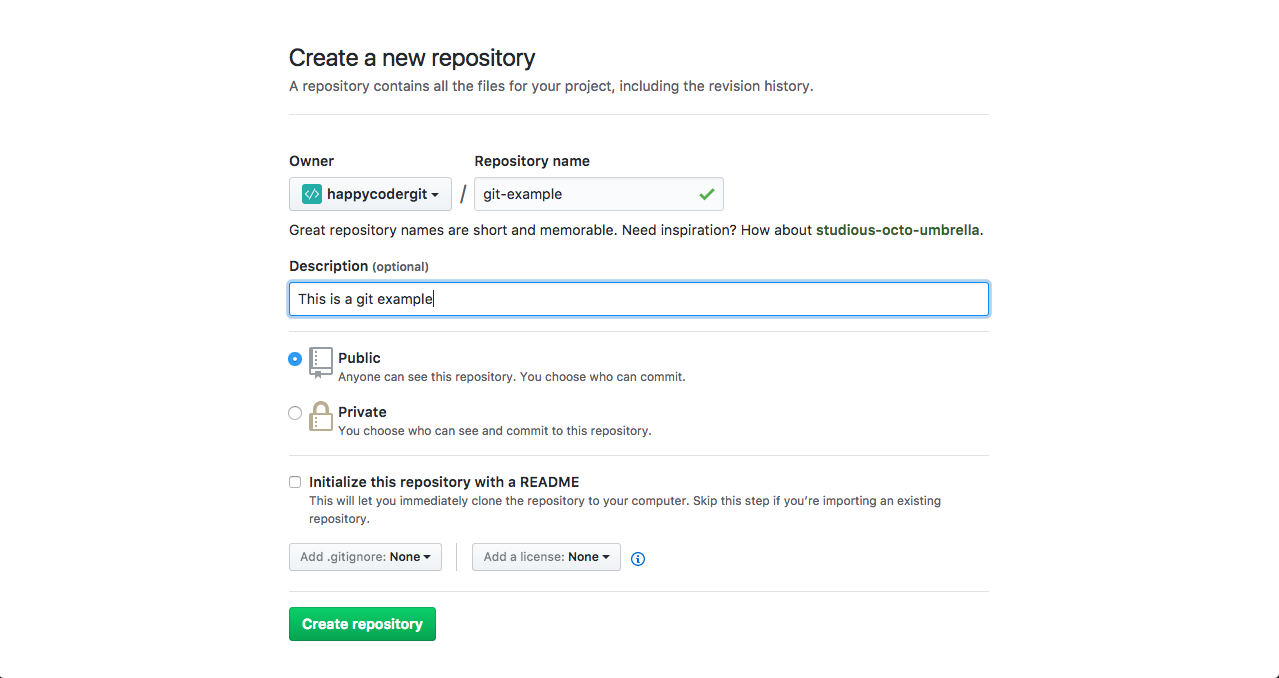


輸入repository name(你要取的專案名稱)，最好跟電腦local專案一致，我們這邊輸入git-example。然後輸入簡短專案描述並先不勾選初始化README和，也先不要選擇.gitignore和License授權(不然會造成本地端和遠端不一致會需要額外一些處理)：

.gitignore：要忽略的檔案清單，這是用來告訴Git，當在做版本控制記錄的時候，忽略這些檔案。通常一些機密資料，如資料庫帳號密碼或是server IP位置等，記得要加入。也可以參考github上面的[一些範本](https://github.com/github/gitignore" \t "_blank)在新增repository時選取對應的程式語言

README.md：repository介紹和使用方式說明(例如：使用方法、參與專案方式等)，使用markdown語法撰寫。另外通常有CONTRIBUTING.md額外說明如何參與貢獻。

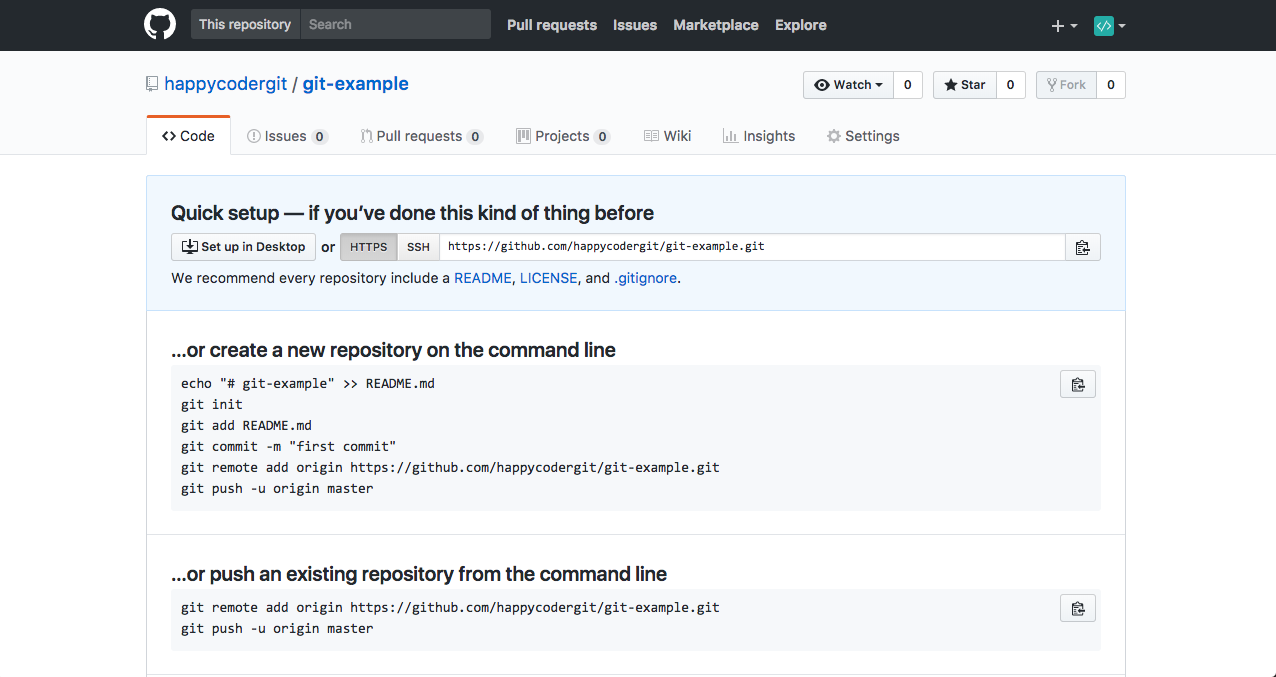
LICENSE：專案使用何種授權方式，例如：MIT、BSD等



接著按create按鈕！恭喜你新增成功，創建了自己第一個github repository(遠端檔案庫)！

### 將repository做本機和遠端的連結

任務五：把電腦裡Local(本地端)的repository(檔案庫)和remote(遠端)的repository(檔案庫)連結起來，並push電腦上的修改



因我們已經在專案有把hello.py修改加入追蹤並commit到local檔案庫，所以我們可以參考下方的指令說明把remote網址加入：

//本地端專案知道origin對應到遠端網址

$ git remote add origin <remote網址>

接者準備將本地端程式push到遠端檔案庫：

//觀看情況

$ git status

//將本地端程式push到遠端檔案庫

$ git push -u origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 239 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To <https://github.com/happycodergit/git-example.git>

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

參數-u等同於--set-upstream，設定upstream可以使分支開始追蹤指定的遠端分支。事實上，只要做過一次$ git push -u <remote name> <branch name>，並且成功push出去；本機端的master就會被設定去追蹤遠端的<remote name>/<branch name>分支。只要成功設定好upstream後，第二次以後要上傳分支時，就只需要透過git push就可以了，不必再帶<remote name>跟<branch name>等參數。例如：$ git push。

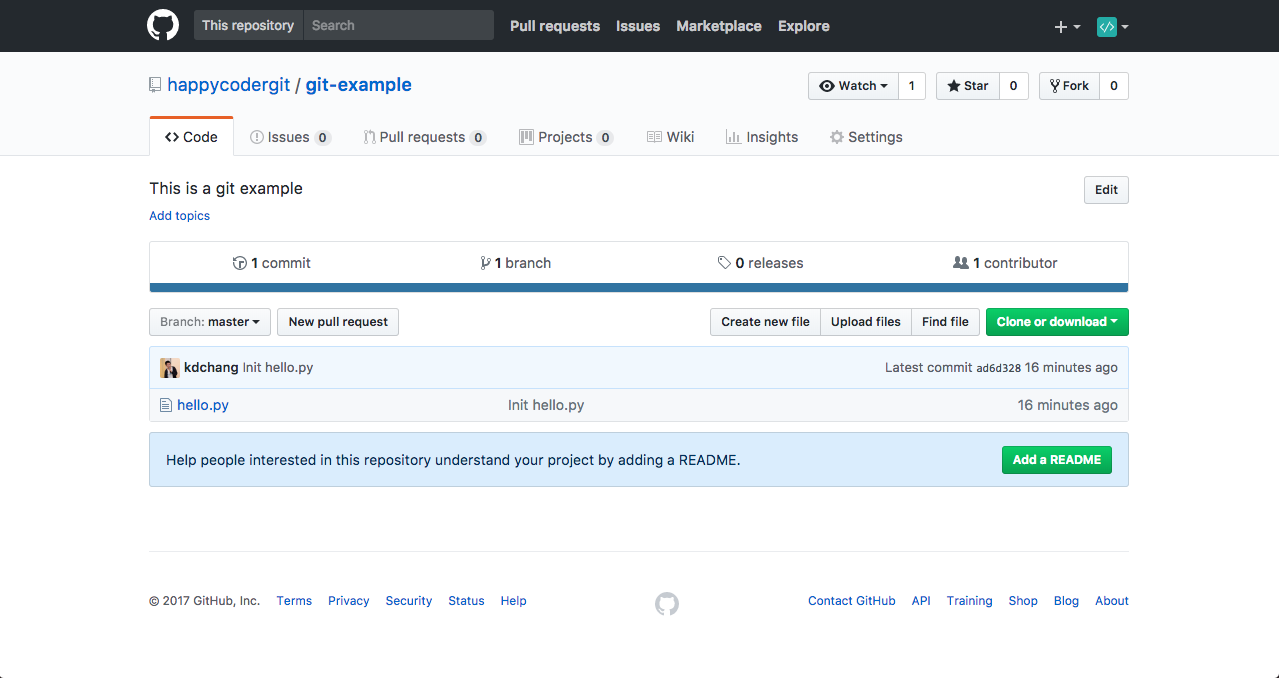
事實上，$ git push -u origin master 可以拆解成：

$ git push origin master

$ git checkout master

$ git branch -u origin/master

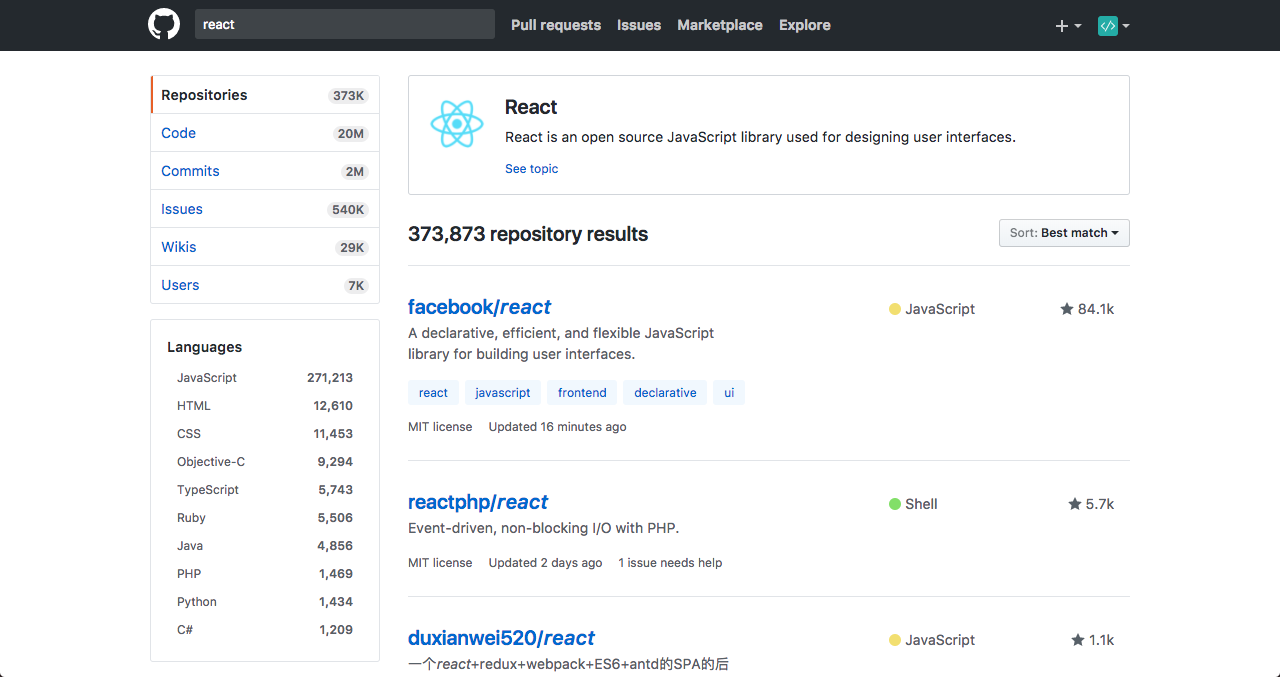
恭喜你成功把你的專案推送上去github啦！



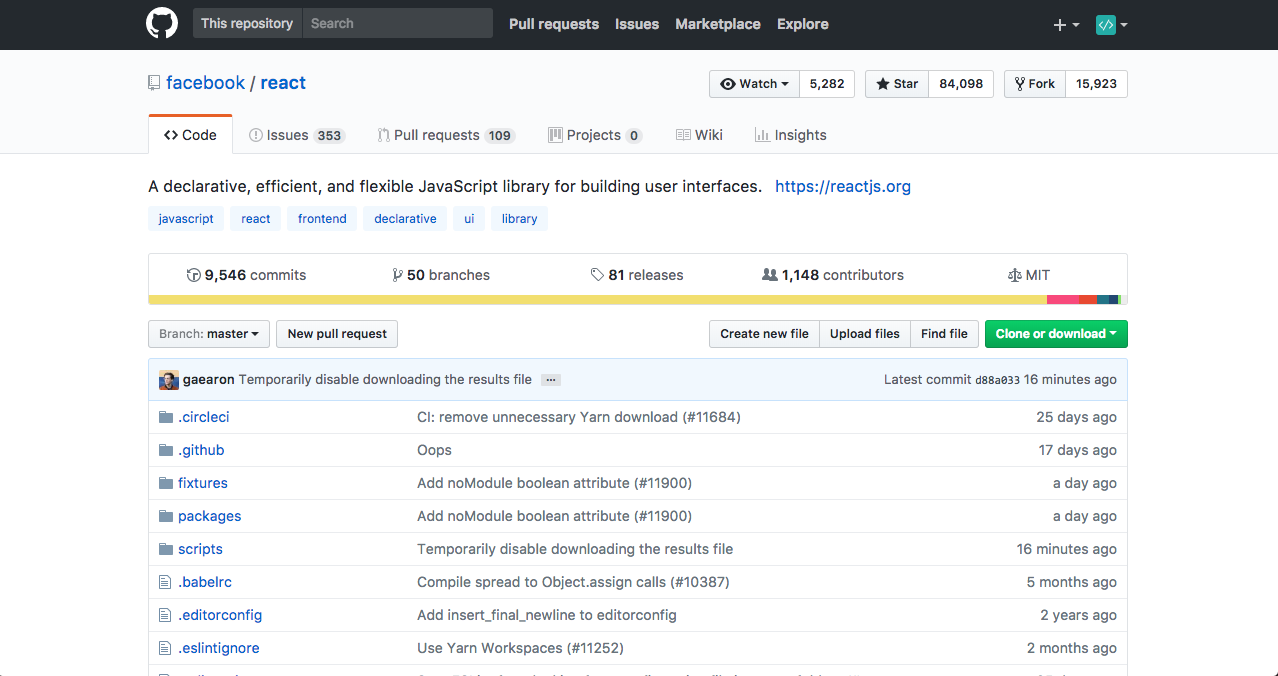
### Fork和clone一個open source(開源)的計畫

任務六：從[GitHub.com](http://github.com/)建立專案，複本fork，並下載clone到電腦上

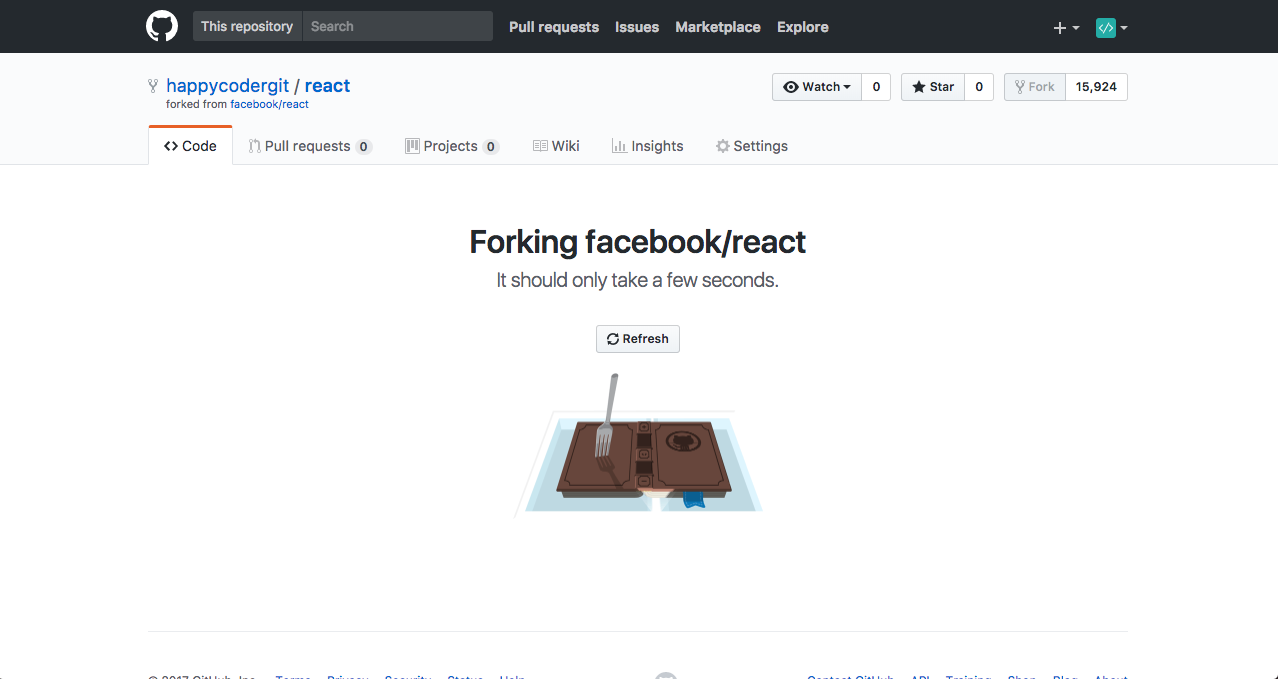
點選左上角github icon回到首頁，我們從上面搜尋欄搜尋react：



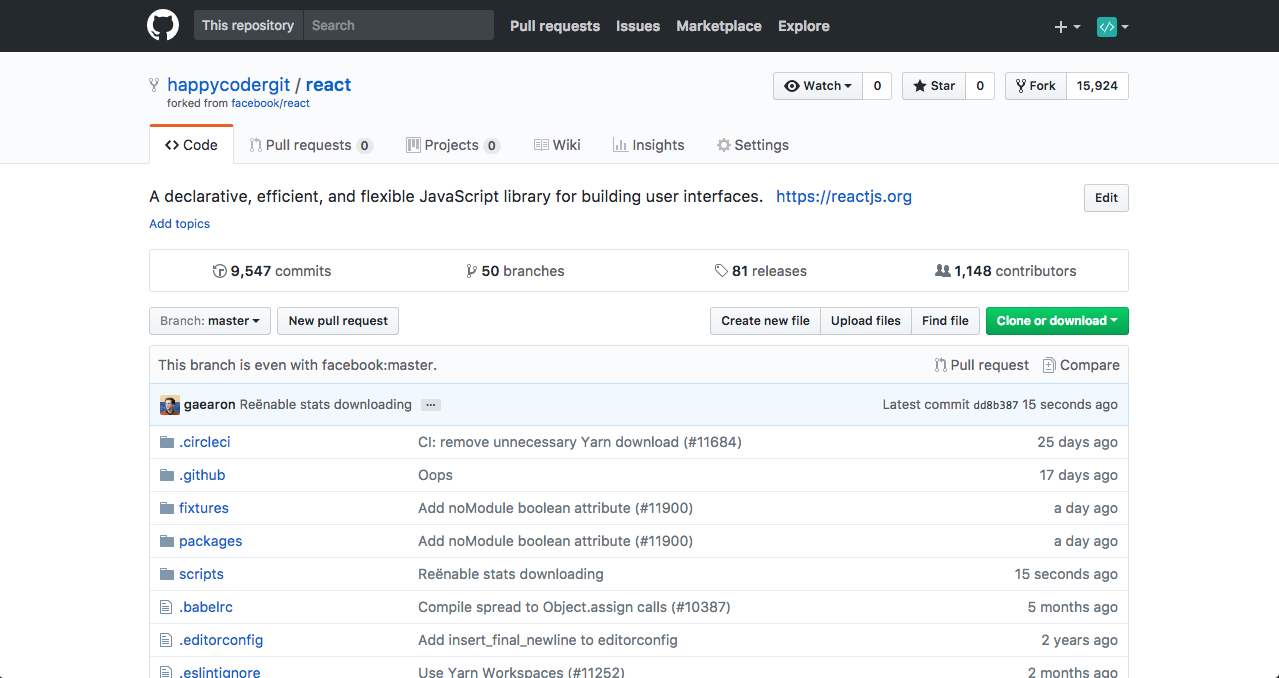
點選右上角fork按鈕，複製一份專案到我們這：



等待fork中：



點選右邊綠色按鈕clone download複製HTTP網址：



//複製到本地端

$ git clone <https://github.com/happycodergit/react.git>

//移動到react資料夾

$ cd react

//切出自己的新分支(使用-b)

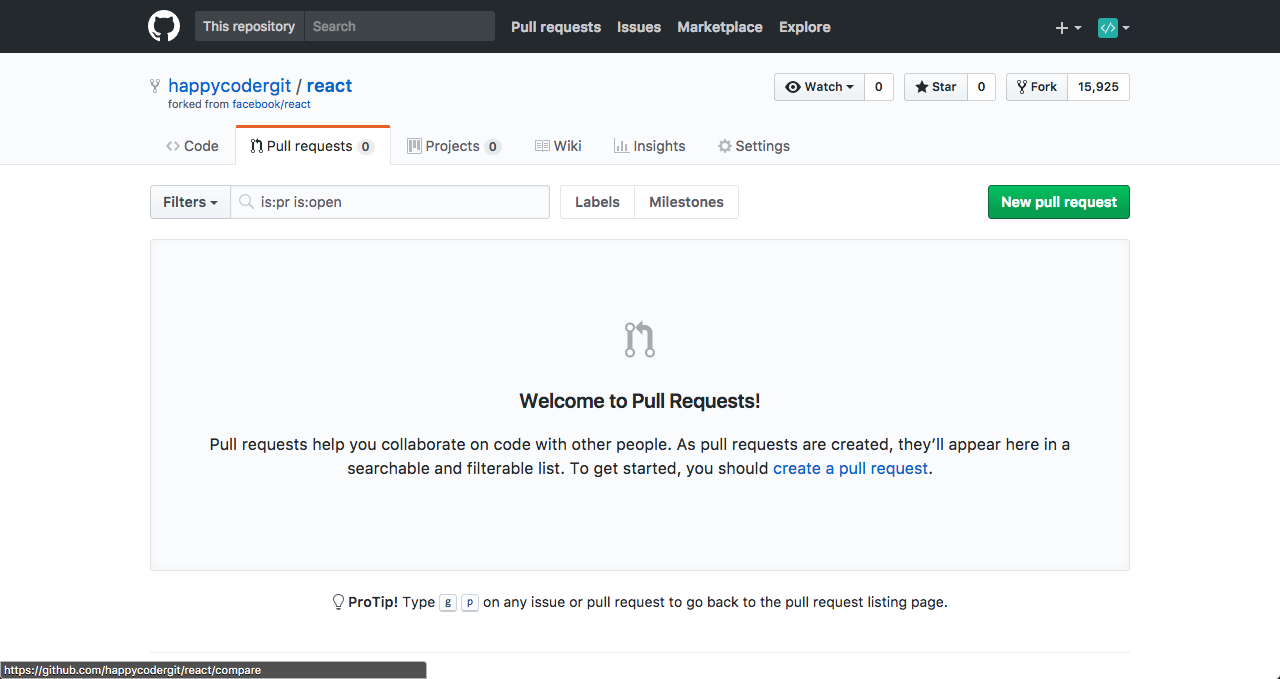
$ git checkout -b happycoder@feature\_branch

//做一些README.md檔案修改，然後commit到自己fork過來的專案

$ git commit -a -m "Update README"

$ git push origin happycoder@feature\_branch

事實上，每個開放原始碼都有他自己貢獻的方式，記得要先了解。例如可以[參考react貢獻說明](https://reactjs.org/docs/how-to-contribute.html" \t "_blank)！



你可以透過將自己的修改commit到自己fork過來的專案，然後到原始專案頁面點選new pull request按鈕發pull request(會比對程式碼的差異)。若對方review完後接受就可以將自己的程式碼合併到原始專案中，為開放原始碼做出第一步貢獻！

//若完成pull request記得讓master(或是合併進去的branch)保持同步

$ git pull upstream master

### 練習建立一個feature branch

任務七：回來原來的git-example專案新增feature branch(分支)

//建立一個名為dev的branch

$ git checkout -b dev

在hello.py最上面多加一行 # hi, this is comment 註解後存檔

//commit到本地端更新

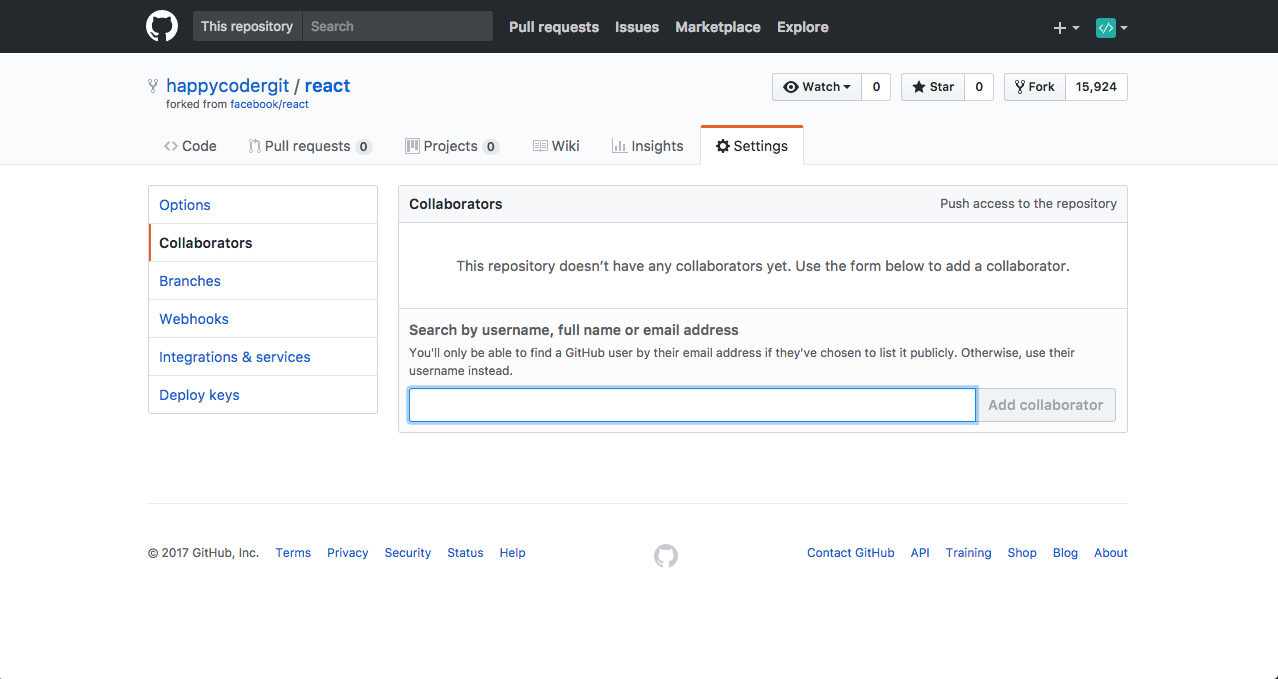
$ git commit -a -m "Init dev branch"

$ gut push origin dev

### 邀請別人和你合作

任務八：在專案新增夥伴collaborator

在右上角進入setting可以選擇到collaborator新增合作者：



### 利用push和pull來和[GitHub.com](http://github.com/)同步

任務九：用pull來和其他collaborators(合作者)同步更新，確保程式是最新的版本

請其他開發者git colone下來你的git-example專案並checkout到devbranch，並完成新增一個README.md檔案後發pull request過來，若沒問題就按同意並合併。我們則透過git pull來保持本地端和遠端程式碼同步：

$ git pull origin dev

### Merge(合併)和刪除branches

任務十：在本機上merge合併你的branch(分支)，刪除舊的branch(分支)

//移動到master branch

$ git checout master

//合併dev到master

$ git merge dev

//刪除dev branch

$ git branch -d dev

//將合併後的master推送到遠端

$ git push origin master

## 總結

以上就是Git與Github版本控制基本指令與操作入門教學！希望讀者可以動手一起操作，漸漸就能感受到git的威力和好處，同時也有能力參與開放原始碼(open source)的社群和提昇自己在職場上的價值，朝成為一個真正的軟體工程師(Software Engineer)邁進！

參考文件

[30天精通Git版本控管](https://github.com/doggy8088/Learn-Git-in-30-days/blob/master/zh-tw/README.md)

[Git官方網站](https://git-scm.com/)

[TryGit](https://www.codeschool.com/courses/try-git)

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

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

[如何將檔案從stage移除?](http://oomusou.io/git/git-remove-stage/)

[Git教學](https://www.gitbook.com/book/zlargon/git-tutorial/details)

# Git簡易使用教學

## 前言

版本控制一直是軟體開發中非常重要的工具，而Git與Subversion、CVS不同的地方在於Subversion及CVS是屬於Centralized VCS，Centralized VCS的共同缺點是做什麼事都要跟伺服器連線，這樣開發會比較慢，且只要伺服器壞掉，就無法工作了。

Git則屬於分散式版本控制系統，讓本地端也維護完整的Repository，即使沒網路，照常可以commit和看history log，伺服器的Repository可以在將來有網路連線時再同步更新。

## 安裝設定Git

[Github](https://github.com/)上有[各大平台完整的安裝及設定教學](https://help.github.com/articles/set-up-git)，建議直接參照這個教學來設定就可以了。

其中請特別注意設定好提交者的name及Email，Git會記錄每個commit是由誰提交的，這在版本控制上是很重要的資訊。

我們可以使用以下的指令來進行設定：(--global</code>表示是全域設定)

$ git config --global user.name "Fukuball Lin"

$ git config --global user.email "fukuball@gmail.com"

設定完成後可以用以下指令來觀察是否有設定完成

$ git config --list

user.name=fukuball

user.email=fukuball@gmail.com

### git init

當Git安裝設定好之後，就可以開始使用Git版本控制，假設現在你有一個Hello-World的資料夾，那在這個資料夾底下下以下指令就可以開啟一個Git Repository：

$ git init

Initialized empty Git repository in /Users/fukuball/Projects/Hello-World/.git/

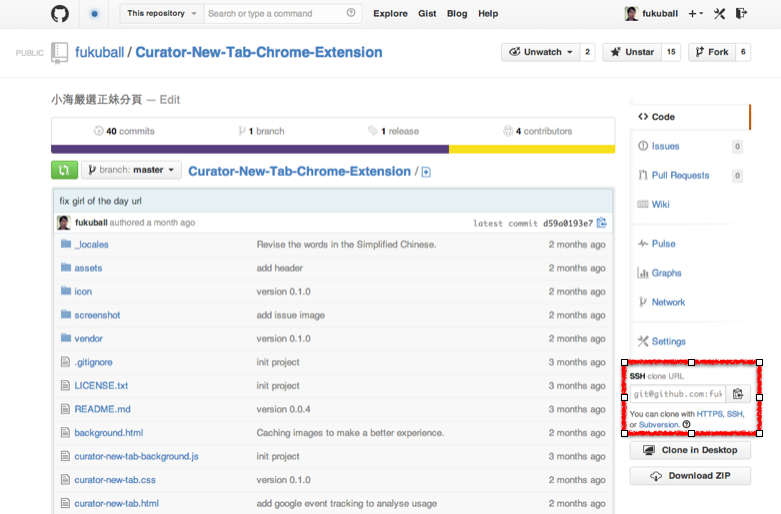
請注意開啟Git Repository之後只是在自己的local端開啟了一個版本控制資料庫，雖然可以正常在local端進行所有版本控制的功能，但因還未連結至Git Server，他人並無法加入共同開發。

目前比較紅的Git Server服務是Github，Github上有[完整的教學](https://help.github.com/articles/create-a-repo)說明如何開Git Repository，並連結至Github上的Git Server服務。

### git clone

當團隊中有人已開啟一個在Git Server的Git Repository，就可以使用Git clone將這個Repository抓來自己的local端一起進行開發。

首先找到Git Repository的位址：



使用以下指令進行Clone：

$ git clone https://github.com/fukuball/Hello-World.git

特別注意如果有寫入權限的話(被加入成Collaborators)，就可以用SSH協定Clone下來：

$ git clone git@github.com:fukuball/Hello-World.git

使用SSH Clone會比較方便，可以不必每次都輸入帳號密碼，但需要事先綁定SSH Key，如何[綁定SSH Key在Github上也有完整的教學](https://help.github.com/articles/generating-ssh-keys)。

### git status

我們可以使用git status來觀察Git Repository的狀態，比如目前所在的branch及哪些檔案還沒commit等等。

$ git status

# On branch master

nothing to commit, working directory clean

### git add (stage)

使用git add可以將新增檔案加入git版本控制，但我通常就直接使用git add.來將所有剛剛修改過或新增加的檔案一次Add進stage狀態，大部份人不推薦這樣做，認為太暴力，但既然都有版本控制系統了，我個人習慣就不這麼婆婆媽媽的了。

$ git add .

### git commit (commit)

stage狀態的檔案的下一步就是準備提交了，一個commit在Git中就是一個節點，這些commit的節點就是未來可以回朔及追蹤的參考。當檔案都加入到stage了，那就可以使用以下指令來commit：

$ git commit -m "這次 commit 的適當描述"

每個commit有個適當的描述是非常重要的，這樣要回朔時會比較容易查找。

當還有檔案沒有進stage就下commit指令，那就不能commit，這時可使用git commit -a -m這樣的暴力法來一次加入檔案至stage然後進行commit，大部份人不建議這麼做，但我個人習慣不這麼婆婆媽媽。

$git commit -a -m "這次commit的適當描述"

### git push

當已經連結了Git Server，就可以用git push來將local端的commit更新到Server上，請注意有修改的檔案還沒commit那就無法使用git push，所以一定要將所有更新都commit之後，才有辦法使用git push。

$ git push

### git pull

當已經連結了Git Server，我們就可以使用git pull來將遠端更新的code抓回來，同樣如果local端有任何更新，一定都要commit之後才  
有辦法使用git pull。

$ git pull

### git log

我們可以使用git log的指令查看過去commit的紀錄，例如commit的版號、作者等等。

$ git log

.gitigore

log檔及build出來的檔案及系統產生的檔案如.DS\_Store等等，我們並不需要commit上去Repository，所以我們會在Repository編寫一個.gitignore文字檔來忽略這些檔案。

範例.gitigore如下：

.DS\_Store

\*.log

## 結語

以上是一些Git指令的簡易使用教學，但我平常用還是用GUI比較多，個人推薦Github出品的GUI工具，工具只要簡單易用就好了，不太需要什麼複雜的功能啊。

# git常用指令

團隊常常需要切換或合併branch，底下是常用git指令

## git pull

拉最新的master程式碼到目前branch。不管是在任何branch都是執行底下指令

$ git pull --rebase origin master

## git reset

發送Pull Request後，又需要修改目前Branch程式，請善用git reset來回到前一次commit紀錄

$ git reset --soft HEAD^

HEAD^代表回到上一個commit，如果要回到多個commit請用HEAD~4，其中4請換成您要的數字

## git branch

用來看目前有哪些branch或者是強制刪除壞掉的branch

$ git bracnch -D xxxx

-D是強制刪除

## git checkout

用來切換branch

### 切換到遠端branch

要review別人的pull request，請使用底下指令

#讀取最新的origin程式碼

$ git fetch origin

$ git checkout -b test origin/test

這樣就可以切換到遠端test branch繼續操作

### 單純切換local branch

$ git checkout master

## git rebase

在PR內合併多個commit內容，這是用在Code Review完成後，原作者新增多個commit，需要將全部commit合成一個

$ git rebase -i commit\_id

底下是常用的狀況

f, fixup = like "squash", but discard this commit's log message

s, squash = use commit, but meld into previous commit

把要squash的commit id前面都改成f，然後:wq存檔。遇到衝突解決後，可以下

$ git add xxxx

$ git rebase --continue

要跳出git rebase的話，請下

$ git rebase --abort