Git版本控制

Part 1
Git基本介紹

蘇維宗(Wei-Tsung Su)
suwt(at)niu.edu.tw
Department of Computer Science and Information Engineering
National Ilan University

歷史版本

版本	說明	日期	負責人
v1.0	中文化初版	2019/02/26	蘇維宗
v1.1	新增前置工作(移除安裝Git)	2024/01/23	蘇維宗
v1.2	新增安裝GitLab Server	2024/01/26	蘇維宗
v1.3	新增回溯版本(reset)	2024/02/01	蘇維宗
v1.4	將GitLab部分拆出來	2024/08/20	蘇維宗

參考文件

1. https://en.wikipedia.org/wiki/Version_control
2. http://homes.cs.washington.edu/~mernst/advice/version-control.html
3. https://git-scm.com/book/en/v2
4. https://nulab.com/zh-tw/learn/software-development/git-tutorial/

內容

- 前置工作
- Git的歷史
- Git的版本管理方式
- Git本機檔案狀態
- Git基本操作
- Git版本切換
- 展示操作

前置工作

- 安裝Git
  - https://git-scm.com
- **(選要)**安裝編輯工具(例如，Visual Studio Code)
  - https://code.visualstudio.com/
- **(選要)**安裝Sourcetree (**視窗介面工具**)
  - https://www.sourcetreeapp.com/
- **(選要)**安裝自行管理的GitLab伺服器

Git的歷史

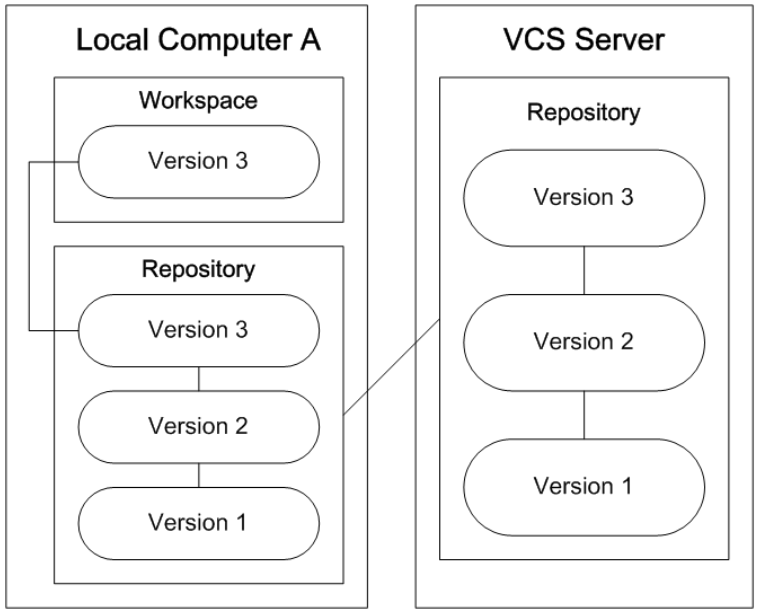
- 早期Linux核心專案被維護在BitKeeper分散式版本管理系統中。然而，因為BitKeeper不再免費提供服務，所以Linus Torvalds於2005年嘗試開發自己的版本管理系統工具，即現在的Git。
- Linux核心(與許多知名的開放原始碼)專案目前都維護在知名的GitHub平台中

https://github.com/torvalds/linux

為何Git適合用來管理大型專案？

Git會在本地端保存專案的所有版本變化，所以可以直接在本機上(不需要連網)完成所有版本控制的動作，需要時再將所有版本變更一次上傳到遠端伺服器上。

- **優點**是可以在本機進行版本管理
- **缺點**是佔用較多的本機空間

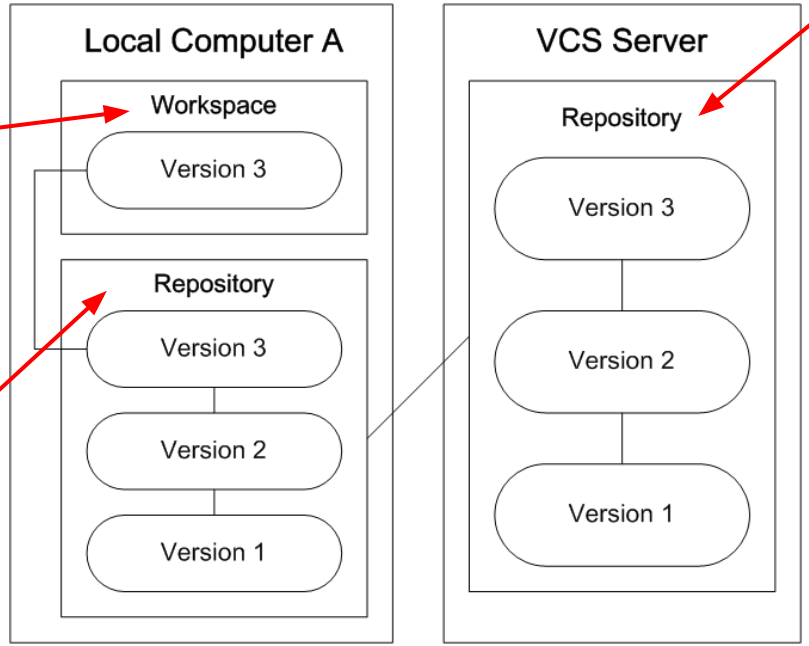


Git的專有名詞

在Git伺服器上的專案稱為一個**Repository**(簡稱**Repo**)。

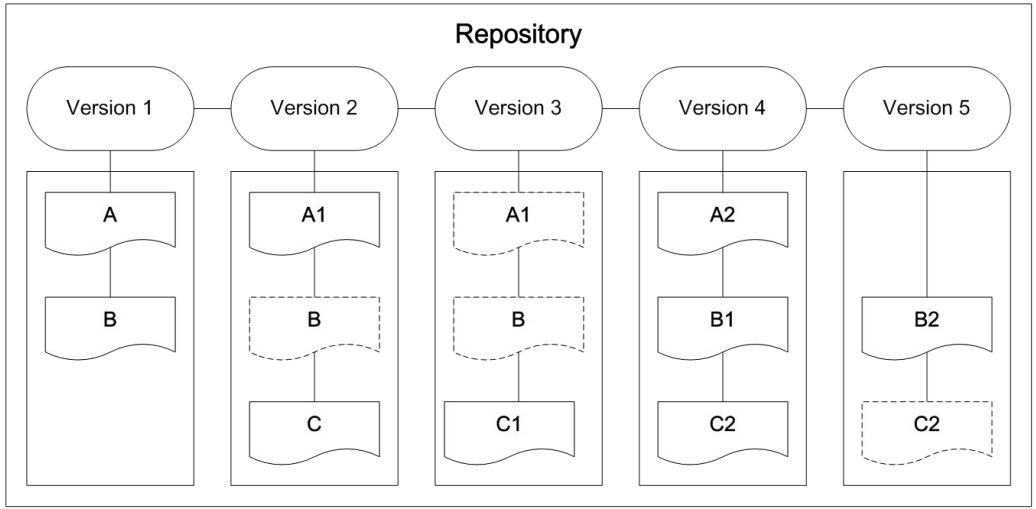
Workspace為在本地端目前的工作版本

可以將Gi 伺服器上的**Repo**完整的下載到本地端。



Git的版本管理方式

Git儲存的是**版本快照 (snapshots)**，而不像SVN儲存是版本之間的差異。



Git環境設定

Git的環境設定分三個層級

- **Local**: 針對專案(**必須在專案目錄中**。設定檔`[repo]/.git/config`)
- **Global**: 針對使用者(設定檔`~/.gitconfig`)
- **System**: 針對系統(**需要有系統權限**。設定檔`/etc/gitconfig`)

當有多個層級具有相同的環境設定時，其優先順序為

Local > Global > System

Git環境設定(續)

設定開發者資訊(當更新版本時，這些資訊會被記錄(**重要！這不是Git帳號**)

```
$ git config --[system/global/local] user.name "[name]"
$ git config --[system/global/local] user.email "[email]"
```

設定預設的文件編輯器

```
$ git config --[system/global/local] core.editor "[editor]"
```

檢查環境設定

```
$ git config --[system/global/local] --list
```

動手練習

Git環境設定

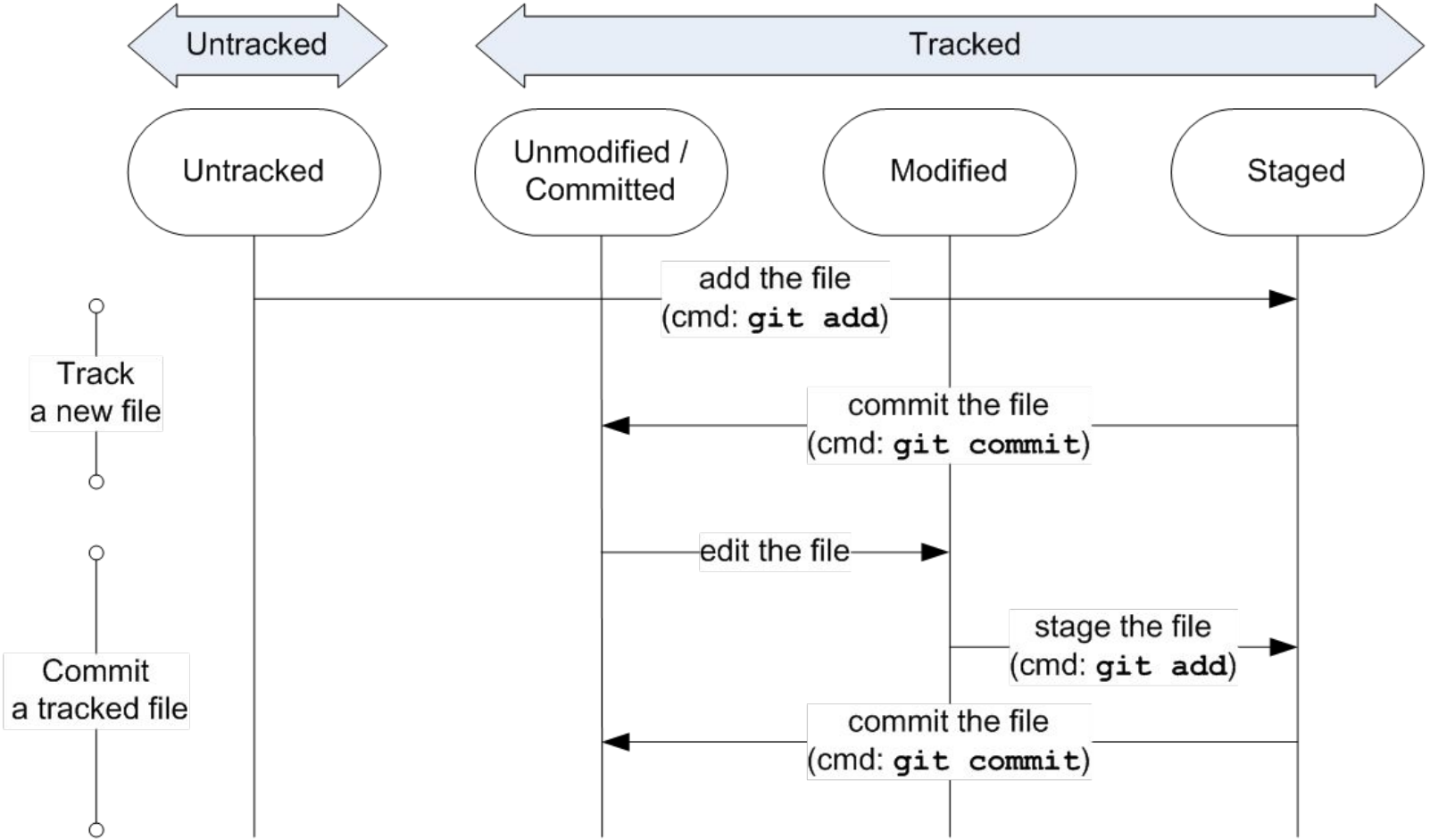
請透過Git命令在本機終端機上設定Global層級的開發者資訊，包含

- 使用者名稱
- 使用者電子郵件

重要！Git本機檔案狀態

在Git管理下的**本機檔案**會有以下幾種不同的狀態

- 未納管的檔案(untracked files)
  - **`Untracked`**: 未納管的檔案(通常是新增的檔案)
- 已納管的檔案(tracked files)，又分為以下幾種狀態
  - **`Unmodified / Committed:`** 未變動過(變動過但已更新版本)的檔案
  - **`Modified:`** 變動過但尚未更新版本的檔案
  - **`Staged:`** 準備納入新版本(新增或修改過)的檔案


Untracked
Tracked
Untracked
Unmodified / Committed
Modified
Staged
add the file
(cmd: git add)
Track a new file
commit the file
(cmd: git commit)
edit the file
stage the file
(cmd: git add)
Commit a tracked file
commit the file
(cmd: git commit)

Git基本操作

- 從Git伺服器下載專案(repo)

  ```
  $ git clone [Git Repo URI]
  ```

- 例如，從 GitHub 下載 Repo

  ```
  $ git clone https://github.com/coachsu/learn-git.git
  ```

備註: 在Git專案目錄(以 `[repo]` 代表)中有一個 `.git` 隱藏目錄紀錄了所有歷史版本資訊。

Git基本操作(續)

- 查看專案資料夾內檔案的狀態

```
[repo]$ git status
```

- 檔案狀態說明
  - **Nothing to commit, working tree clean:** 所有檔案狀態都是 unmodified / committed
  - **Untracked files:** 處於 untracked 狀態的檔案
  - **Changed not staged for commit:** 處於 modified 狀態的檔案
  - **Changes to be committed:** 處於 staged 狀態的檔案

Repo檔案處理(git add/rm/mv)

- 將新增檔案(或修改已存在的檔案)

  [repo]$ `git add [file]`

- 刪除已存在的檔案

  [repo]$ `git rm [file]`

- 變更已存在檔案的檔名(或路徑)

  [repo]$ `git mv [src] [dst]`

備註: 新增、刪除、或修改檔名 (或路徑)都會被視為是一次專案變更而被記錄下來。執行後檔案的狀態都會自動從 modified 變成 statged。

動手練習

Git基本操作

請透過Git命令完成以下任務

1. 下載Git專案並檢查檔案狀態
   https://github.com/coachsu/learn-git.git
2. 在專案中新增檔案並檢查檔案狀態
3. 在專案中修改檔案並檢查檔案狀態
4. 在專案中刪除檔案並檢查檔案狀態
5. 在專案中修改檔名並檢查檔案狀態

忽略檔案(.gitignore)

- 有時候會需要避免一些檔案進入到版本控制當中，例如
  - 暫存檔、測試檔
  - 機敏性資料(金鑰、密碼..)
- 在Git專案目錄中加入.gitignore檔案可以指定在進行版本控制時可以被忽略的檔案。例如，想忽略對builds資料夾進行版本控制
  - `# Ignore committing builds directory`
  - `builds`

在本地端新增版本 (git commit)

- 將目前所有狀態為staged 的檔案變成本機端上新的版本

  [repo]$ git commit -m '[新版本變更描述]'

  或開啟預設編輯器來撰寫新版本變更描述。

  [repo]$ git commit

注意！ 新版本變更描述的撰寫非常重要，因為這是讓開發者知道 **開發過程脈絡的資訊** 。

備註: 目前這些新增的版本目前都只會存在本地端

查詢歷史版本(git log)

- 查詢專案的歷史版本更新紀錄

  [repo]$ git log
- 加入--oneline參數只顯示各個版本的變更描述。例如

  [repo]$ git log --oneline

  版本雜湊　版本變更描述

  cd822c5　文件更新：新增安裝Git

  ccb32e3　revise README

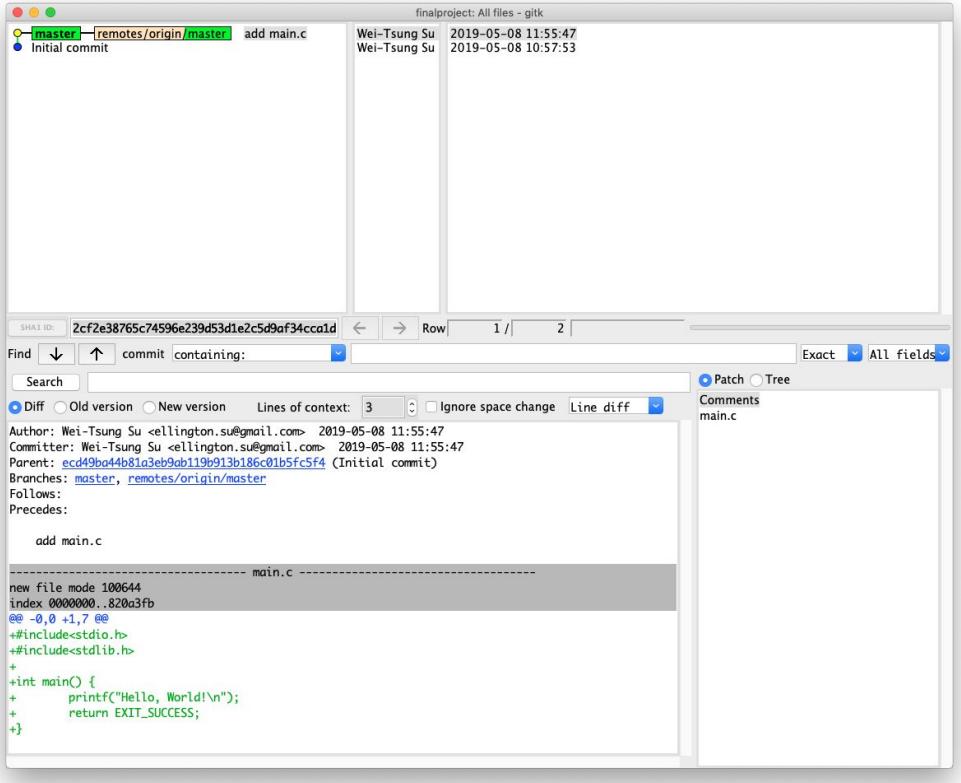
  bc7f216　Initial commit

備註：查詢專案歷史版本更新紀錄是重要的，因為通常會需要這些資訊來進行 **版本切換**(或 **回溯版本**)。不過實務上，我們會利用視覺化工具來取得這個資訊。

視覺化工具

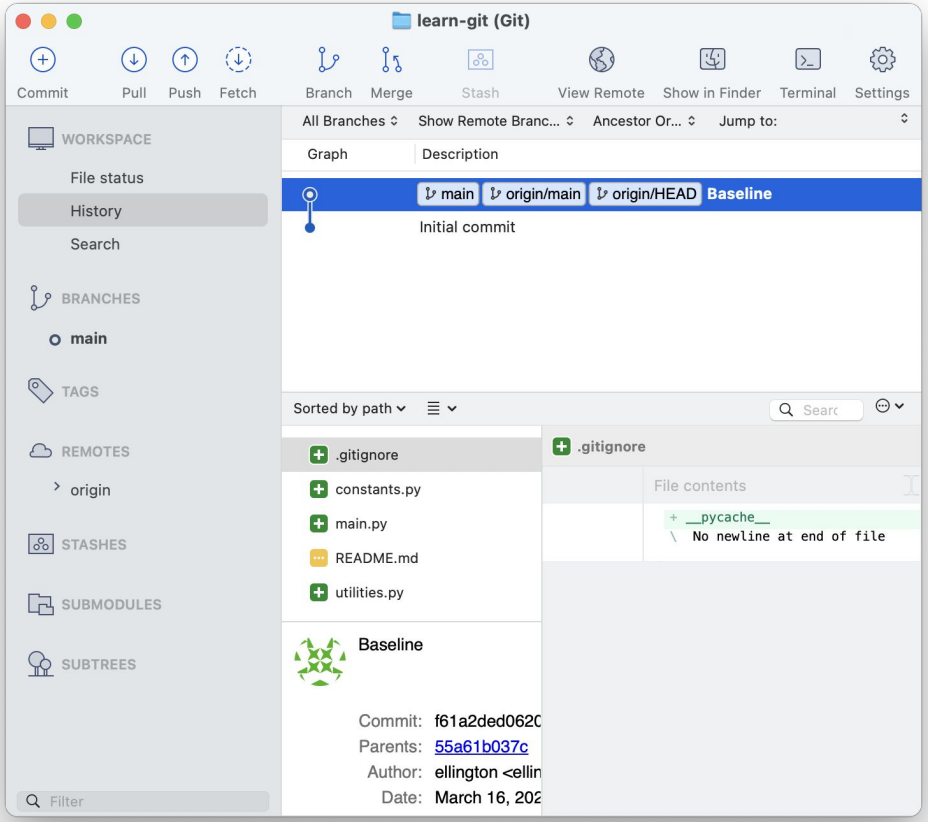
使用視覺化工具來查看版本紀錄會更直覺。例如

- gitk (**Windows**)
  `[repo]$ gitk`
- Sourcetre
- GitKraken
- TortoiseGit
- ...



視覺化工具 (續)

Sourcetree除了可以視覺化Git Repo的歷史版本資訊之外，也可以直接執行大部分常用的Git指令。



動手練習

Git檔案處理

請在下載的Git Repo執行以下動作

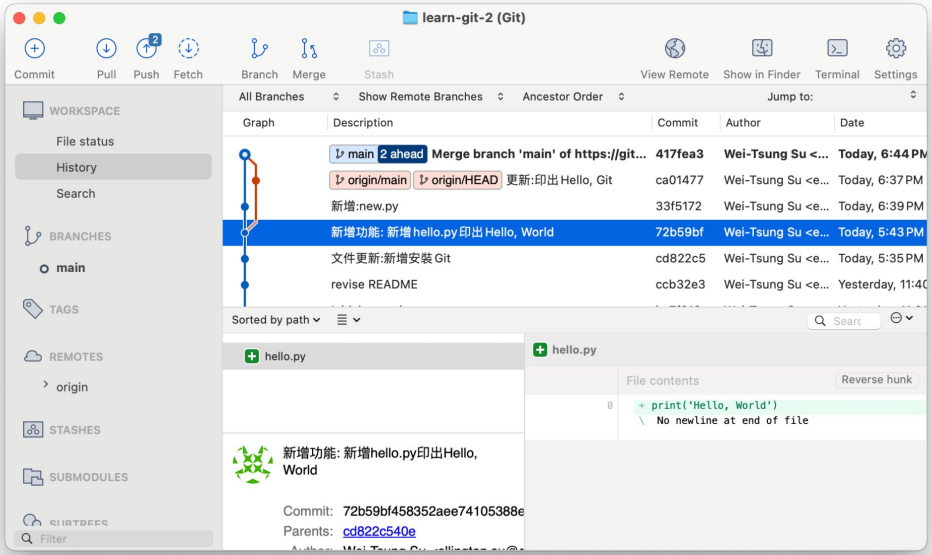
1. 新增 `new.py` 並更新專案版本
   (執行每個動作前檢 查檔案狀態)
2. 修改`new.py`並更新專案版本
   (執行每個動作前檢 查檔案狀態)
3. 刪除`new.py`並更新專案版本
   (執行每個動作前檢 查檔案狀態)

切換版本(git checkout)

如何切換版本?

1. 查詢版本的雜湊值
2. 以版本雜湊值切換版本

```
[repo]$ git checkout [雜湊值]
```



動手練習

切換Git專案版本

請試著利用`git checkout`切換版本

1. 切換任何一個舊版本
2. 切換至最新的本地端版本

回溯版本(git reset)

如果新增版本後發現有錯誤，所有回溯到之前的版本

1. 查詢版本的雜湊值
2. 以版本雜湊值回溯版本

   [repo]$ git reset --hard [雜湊值]
3. 在視覺化工具上可以發現之後的版本已消失

注意！通常只會回溯在本地端 產生但尚未上傳到Git伺服器上的版本！

動手練習

回溯Git專案版本

請試著利用`git reset`回溯版本

1. 新增一個版本
2. 回溯至至前一個版本

更新本地專案(git pull)

- 因為可能(且機率很高)有多個開發者同時更新專案到遠端伺服器上，所以可以透過指令先將遠端伺服器上的新版本同步到本地端。
- 不過，這個動作可能會產生分支而需要進行合併，因此需要指定合併的方式。(**合併是進階議題，這邊先不詳細討論。**)

```
[repo]$ git config --global pull.rebase false
```

接著，就可以更新本地專案

```
[repo]$ git pull
```

如果有發生分支會產生一個新版本來進行合併。

動手練習

更新本地專案

請透過`git pull`指令更新本地專案

將本地端新增版本更新到伺服器 (git push)

- 為了與團隊成員分享對專案進行的修改，必須透過`git push`指令將本機端新增版本更新到遠端伺服器上。

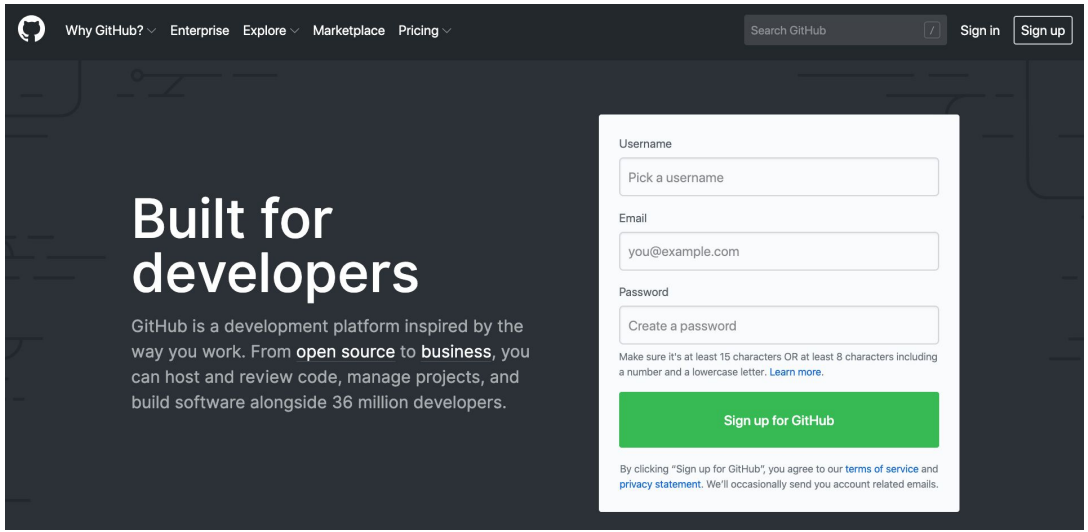
```
[repo]$ git push
```

注意！這個指令需要有專案的寫入權限才可以執行。

實際操作

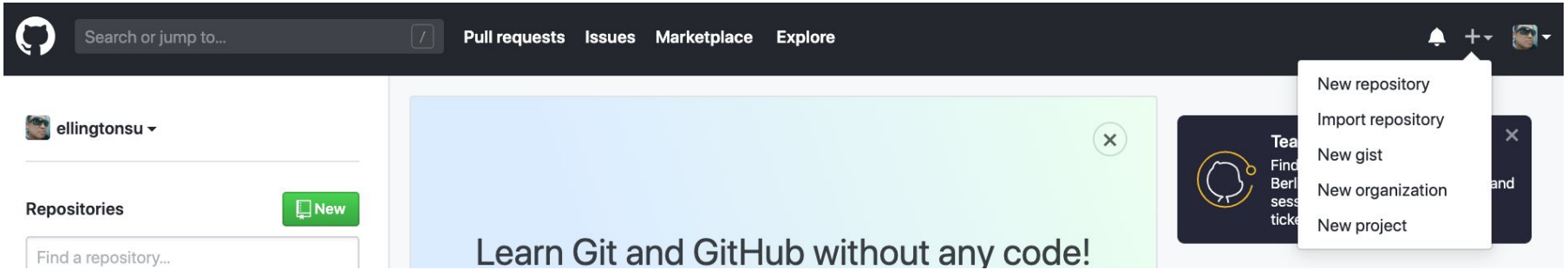
申請GitHub帳號

到GitHub官方網站(https://github.com/)申請帳號



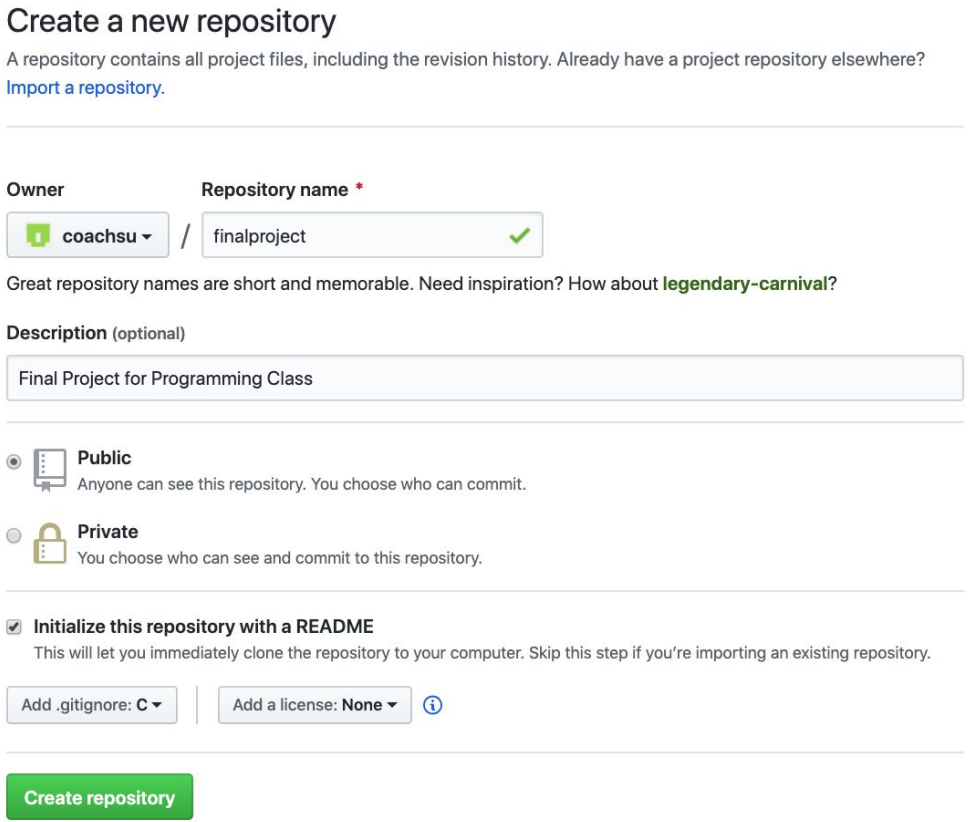
新增Git Repo

選擇New repository來新增一個Git Repo



新增Git Repo (續)

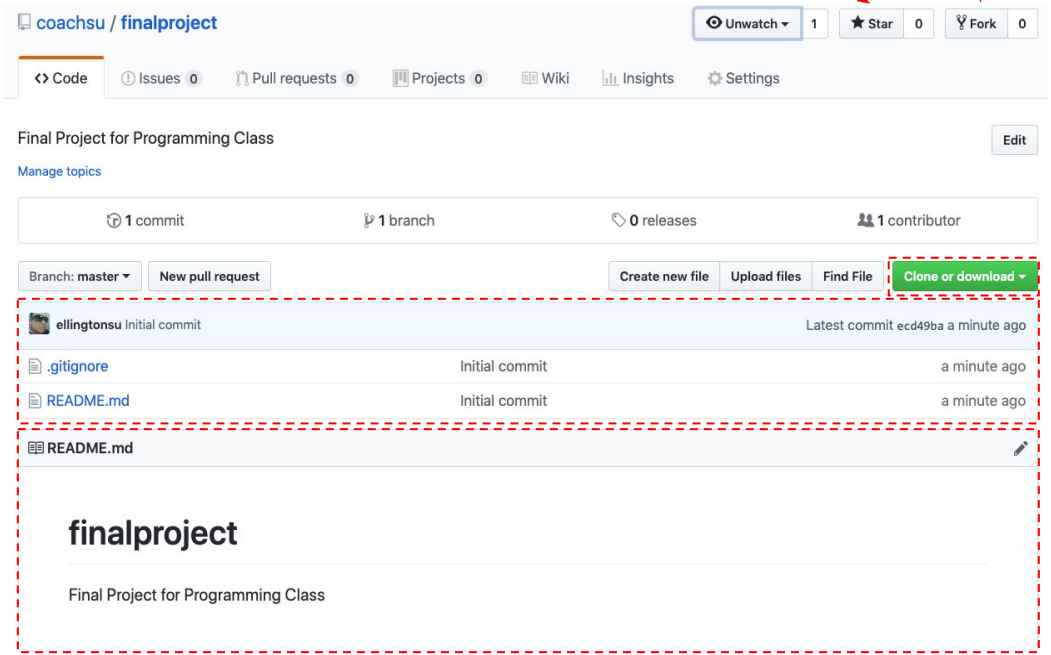
- Owner (專案擁有人)
- Repository name (專案名稱)
- Descrption (專案描述)
- Public / Private (公開 / 私人專案)
- 另外，可選擇是否新增
  - 專案說明(README)
  - 排除規則(.gitignore)
  - 專案授權(license)



Git Repo主頁

多少人喜歡這個專案?

多少人想貢獻這個專案?



取得專案URI

最後更新資訊

專案說明文件

Git Repo開發

- 下載Git Repo

  $ `git clone [Git Repo URI]`
- 新增檔案(例如, `main.c`)並追蹤

  [repo]$ `git add *`
- 建立新版本

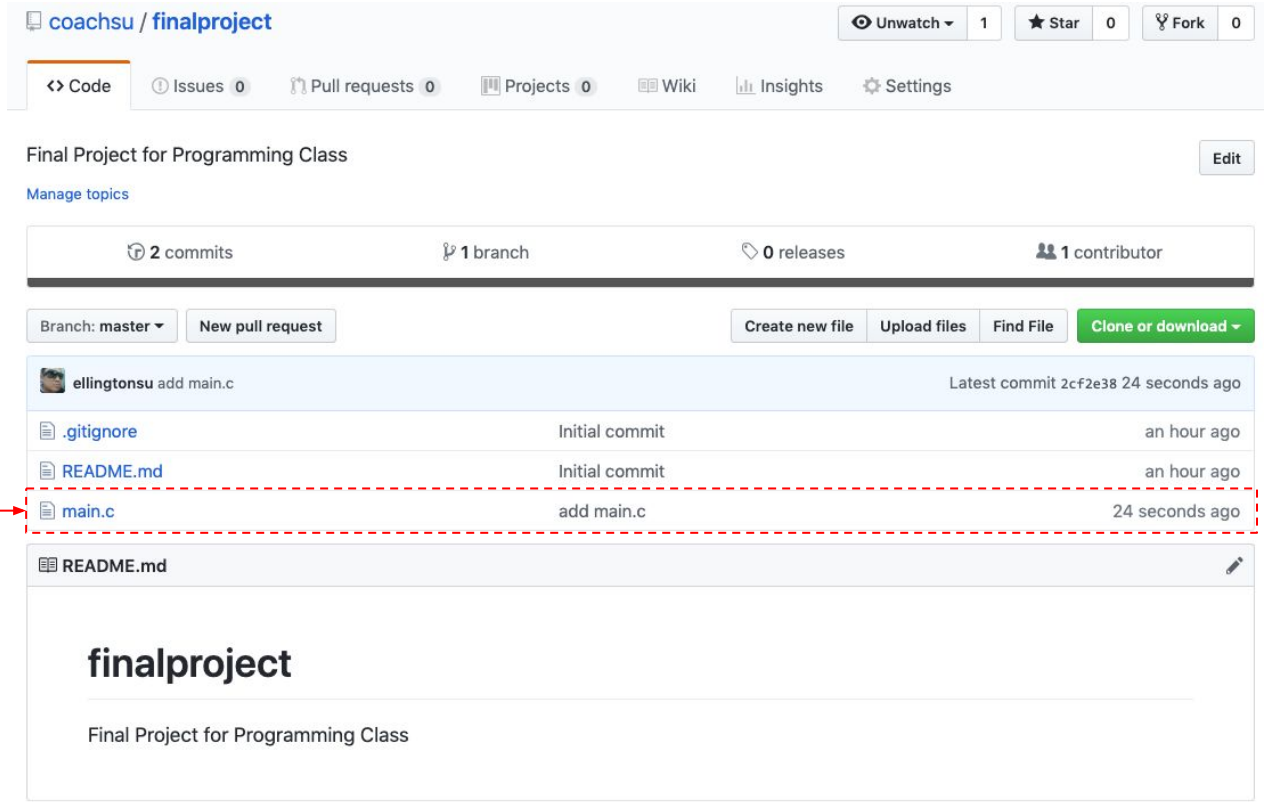
  [repo]$ `git commit -m 'add main.c'`
- 查看 Git Repo 內檔案的狀態

  [repo]$ `git status`
- 查詢目前的版本更新紀錄

  [repo]$ `git log`
- 更新遠端版本

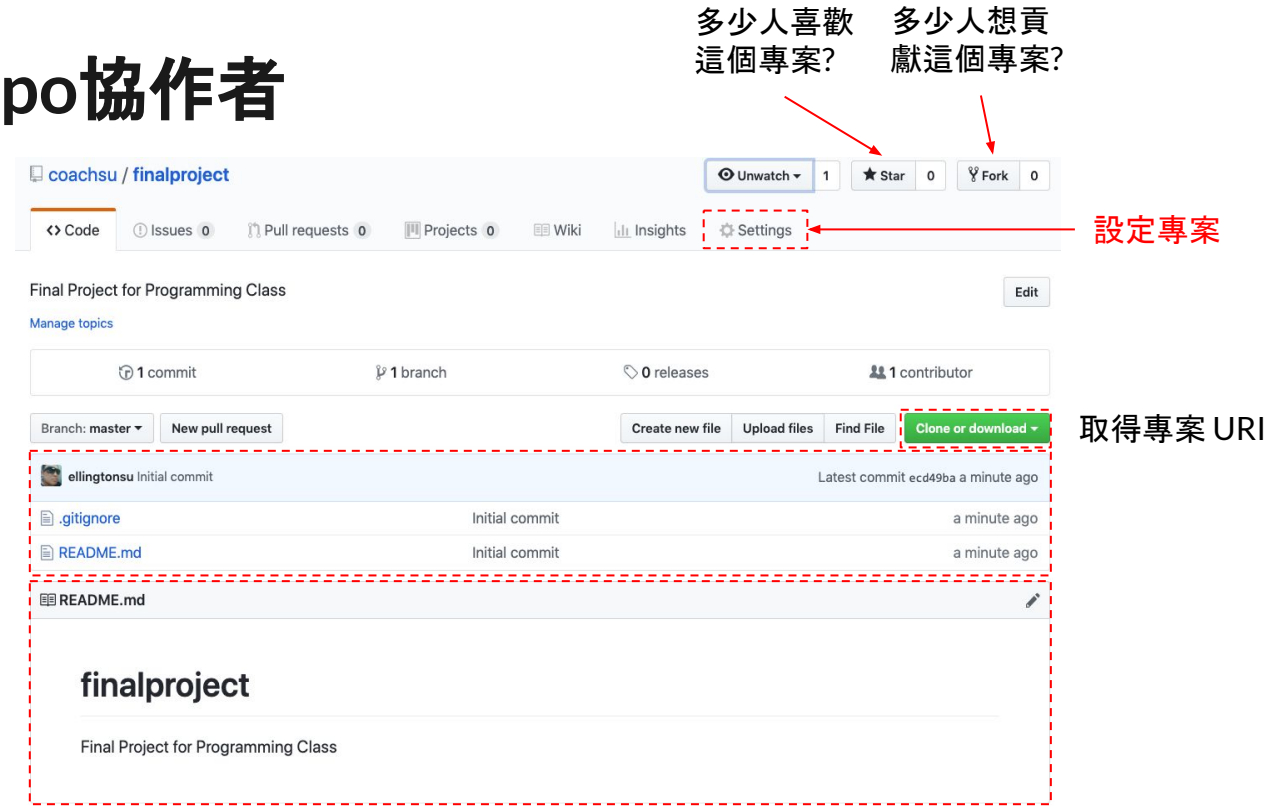
  [repo]$ `git push`

Git Repo主頁

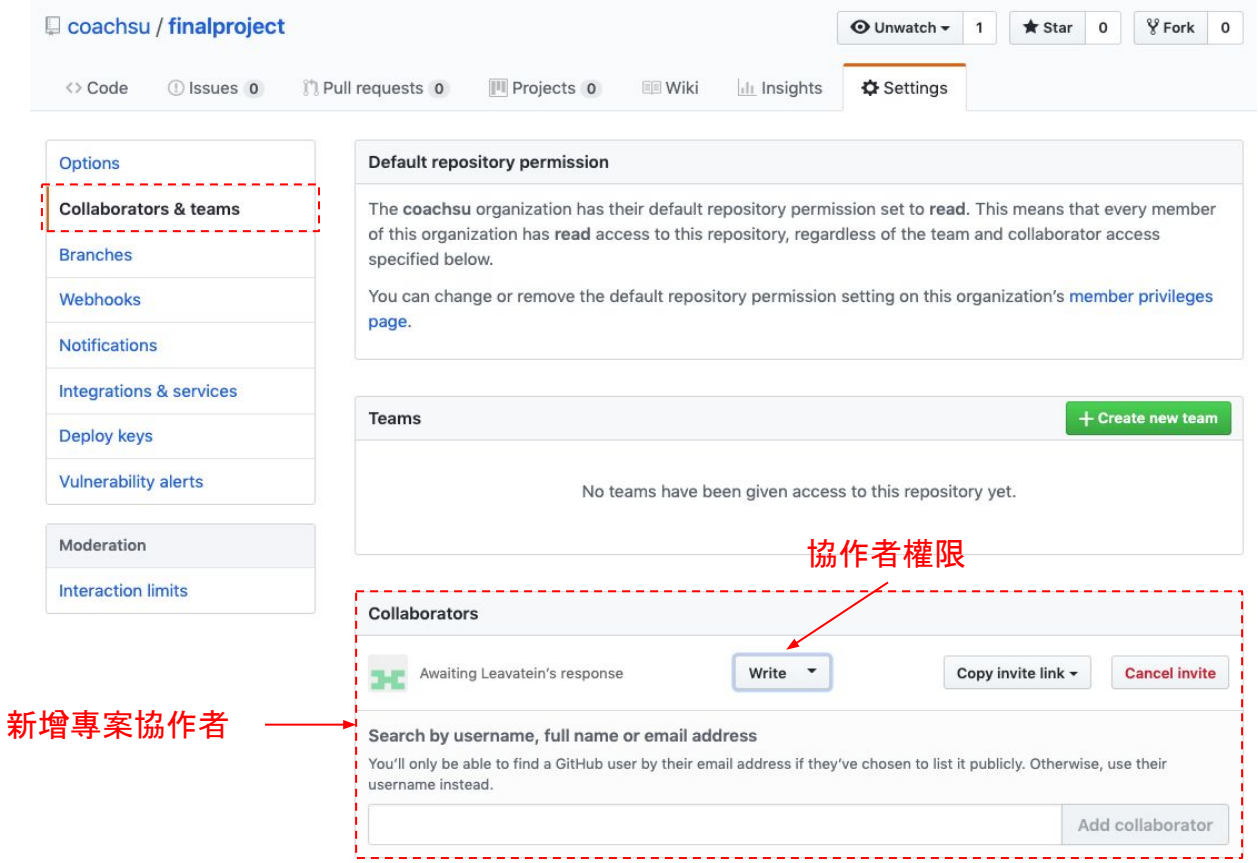


剛剛更新的檔案

新增Git Repo協作者



協作者權限


coachsu / finalproject
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Options
Collaborators & teams
Branches
Webhooks
Notifications
Integrations & services
Deploy keys
Vulnerability alerts
Moderation
Interaction limits
Default repository permission
The coachsu organization has their default repository permission set to read. This means that every member of this organization has read access to this repository, regardless of the team and collaborator access specified below.
You can change or remove the default repository permission setting on this organization's member privileges page.
Teams
+ Create new team
No teams have been given access to this repository yet.
協作者權限
Collaborators
Awaiting Leavatein's response
Write
Copy invite link
Cancel invite
新增專案協作者
Search by username, full name or email address
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.
Add collaborator

Q & A

Computer History Museum, Mt. View, CA