Git

Chapter 1

基礎概念

關於版本控制

- 即為Version Control System (VCS) (版本控制)
- 紀錄修改歷程
- 簡單地徹修對檔案的編輯
- 無法更改他人的變更
- 為什麼要版本控制

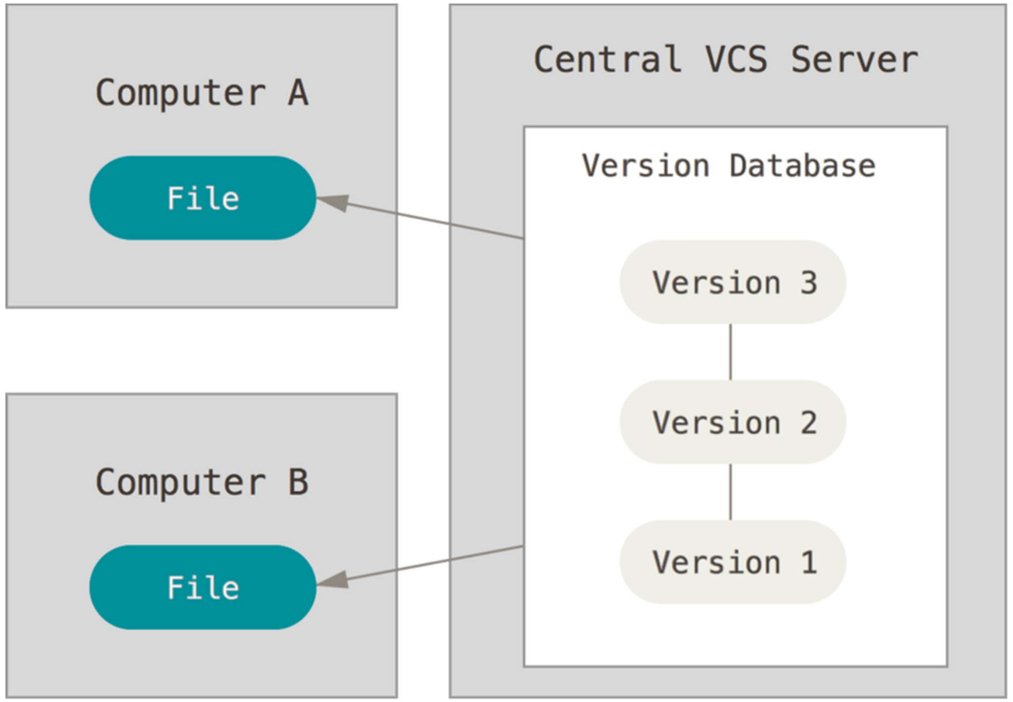
版本控制基本概念

- 版本控制軟體：
  - 追蹤、控管程式原始碼的更改，讓這些更動可以互相比對、相互合併，或復原為特定版本
  - 中央式(centralized)版本控制：將所有文件修改繼魯儲存於1台中央server上，團隊成員可以從中央server上取得應用程式最近時間點的快照紀錄(snapshot)
  - 分散式(distributed)版本控制系統：團隊成員不只是取得應用程式最近時間點的快照紀錄(snapshot)，而是取得中央server上文件修改紀錄資料庫的鏡像(mirror)

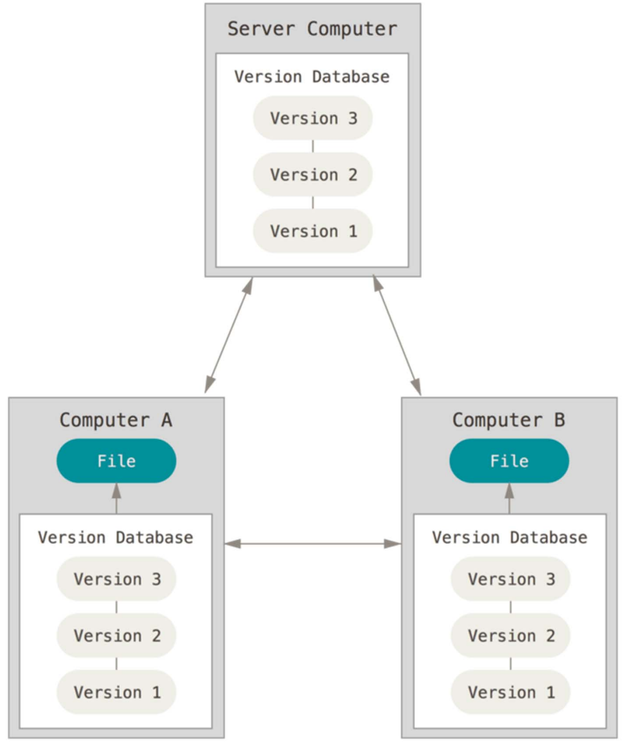
版本控制基本概念

- 中央式與分散式版本控制軟體
- 參考資料：https://git-scm.com/book/zh-tw/v2

中央式控制軟體



分散式控制軟體



Git版本控制軟體

- Git SCM：https://git-scm.com/
- Git是一個分散式版本控制軟體，由Linux作業系統發明人Linus Torvalds創作，目的是為了更好地管理Linux Kernel的開發
- SCM是Source Code Management的縮寫
- 官方網站提供一本免費教材：Pro Git (作者是Scott Chacon and Ben Straub) (繁體中文版) ： https://git-scm.com/book/zh-tw/v2

誰創造了Git

- Linux kernel 的開發者，Linus Torvalds
  在2005年以十天的時間開發了Git的第一個版本
- Git是用C語言開發的
  用於Linux核心開發的版本控制工具
- 它採用了分散式版本庫的作法，不需要伺服器端軟體，就可以運作版本控制

Git開發者: Linus Torvalds

Git 軟體安裝

- 為了讓大家可以實作，需要安裝下列軟體：
  - 版本控管軟體：Git for Windows
  - 程式開發工具：Visual Studio Code
  - 注意1：2項工具只要按照預設值安裝即可，不需額外多做設定
  - 注意2：Visual Studio Code提供的版本控制功能需要依賴Git主程式才能運作，安裝Git for Windows後務必完成設定

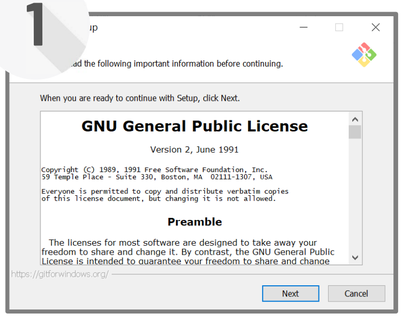
安裝Git for Windows

- 下載Git for Windows： https://git-scm.com/download/win

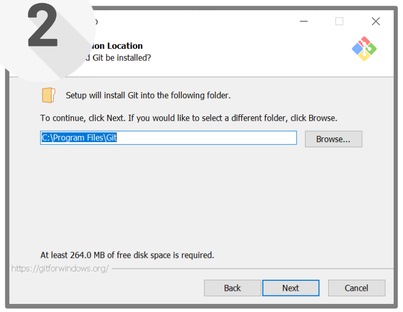
- 請務必將<Git安裝目錄>\bin 加入 PATH 環境變數
- PATH=<Git安裝目錄>\bin;其他PATH設定

Git for Windows安裝流程

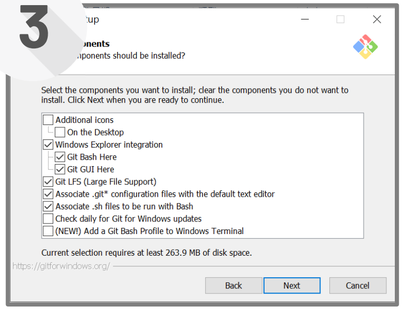
1



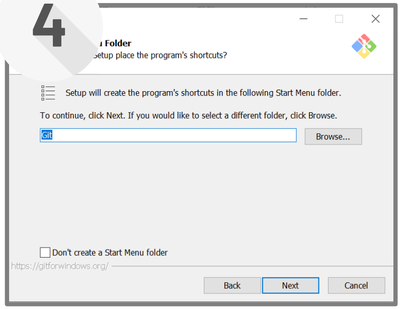
2



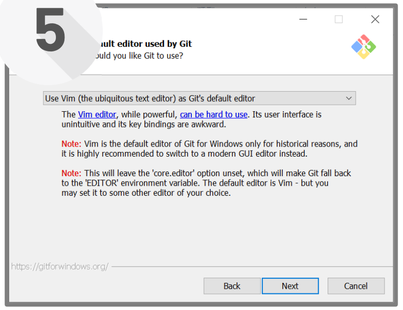
3



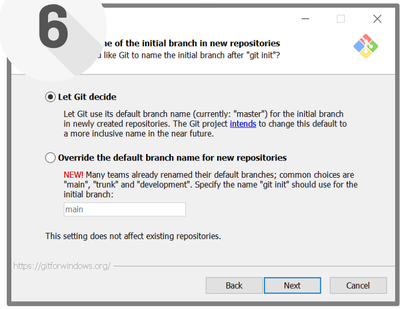
4



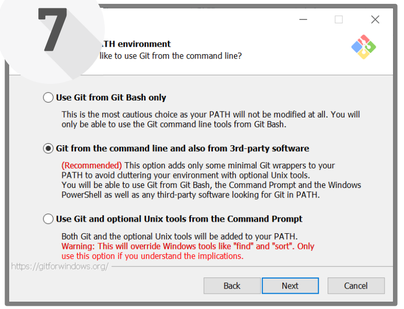
5



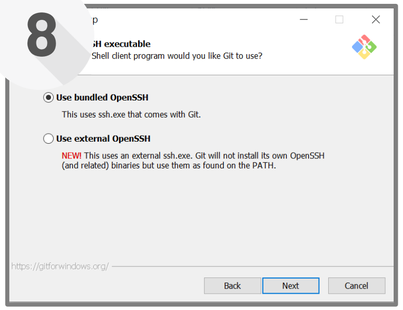
6



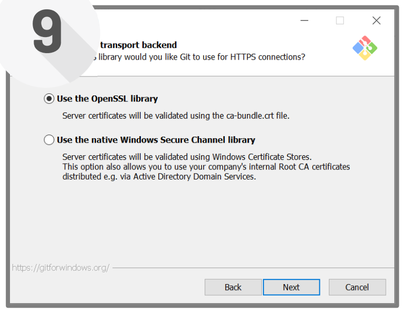
7



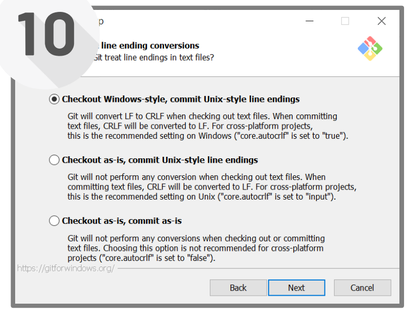
8



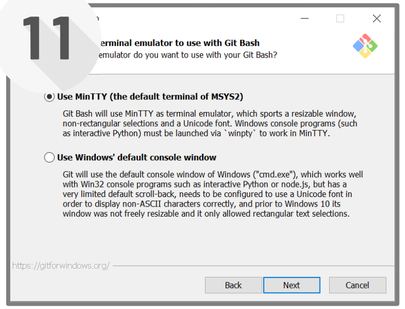
9



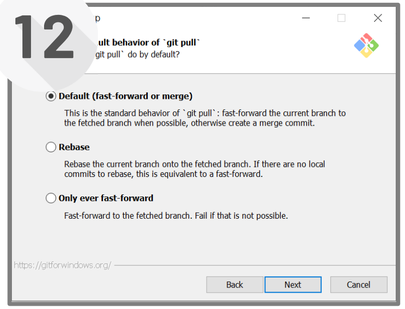
10



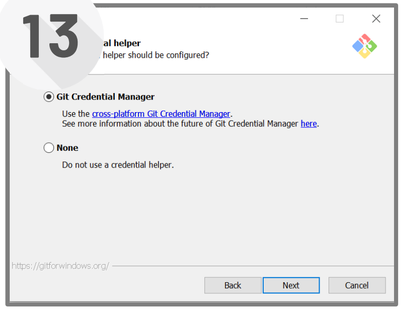
11



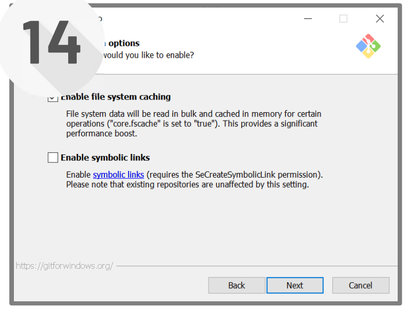
12



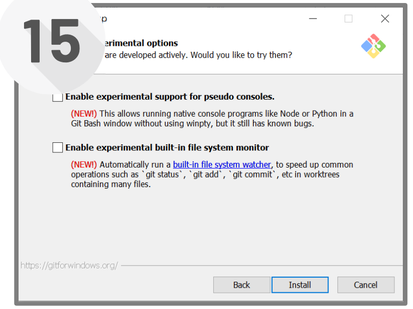
13



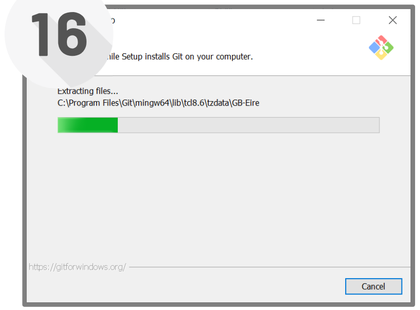
14



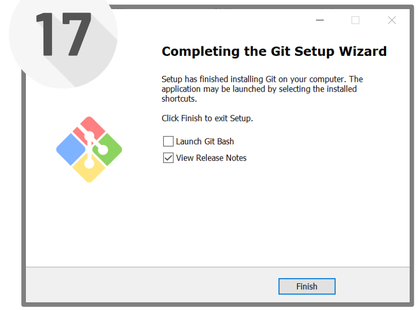
15



16



17



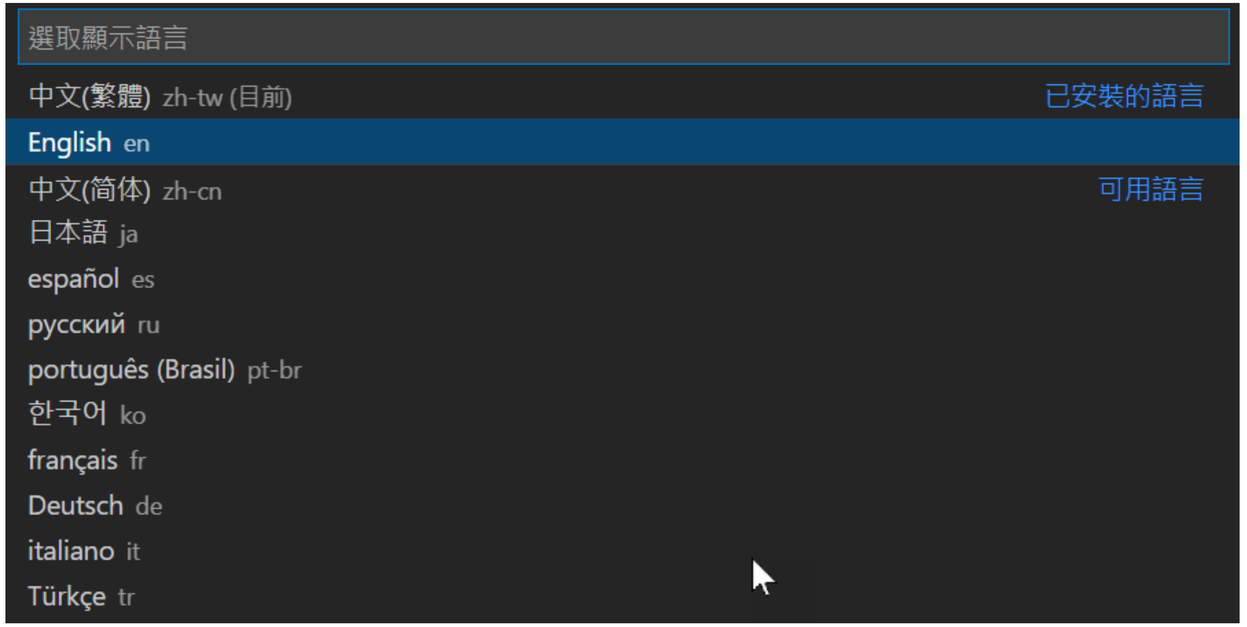
安裝Visual Studio Code

- 下載Visual Studio Code：https://code.visualstudio.com/

- Visual Studio Code環境設定 (Ctrl+,)：
  - 英文使用者介面(幫助熟悉Git指令)："locale":en
  - 字體大小： "editor.fontSize":18
  - Git自動接收資料："git.autofetch":false

Visual Studio Code 環境設定

- Visual Studio Code英文使用者介面：
  - Ctrl+Shift+P
  - 輸入 Configure Display Language
  - 選擇 English
  - 重啟 Visual Studio Code 完成設定



Git環境設定

- git config：客製化Git環境
  - git config會將Git環境參數儲存在3個不同位置的設定檔，低層級設定覆蓋高層級設定
  - 所有使用者共用設定：<Git安裝目錄>/mingw64/etc/gitconfig 檔，使用 --system 選項讀取、寫入 gitconfig 參數
  - 單一使用者專屬設定：<使用者目錄>/.gitconfig 檔，使用 --global 選項讀取、寫入 .gitconfig 檔參數
  - Repository個別設定： <repository目錄>/.git/config 檔，使用 --local 選項讀取、寫入 config 檔參數

Git環境設定

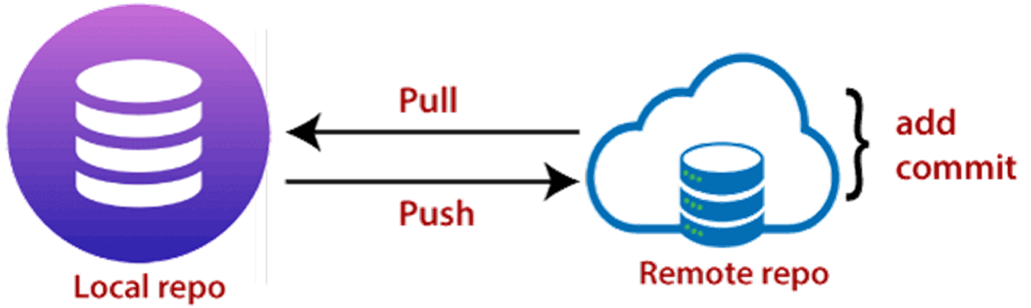
- git config：客製化Git環境
- 設定姓名、email：使用者身分會隨著commit寫入repository
  - C:\> git config --global user.name "John Doe"
  - C:\> git config --global user.email johndoe@example.com
- 檢查環境設定的參數值，由於Git從3個不同種類的位置讀取參數值，可能相同key，但不同數值，使用最細部的數值為主
  - C:\> git config --list
- 檢查特定key的參數值：
  - C:\> git config user.name

在VS Code設定Git環境參數

- 由於Visual Studio Code沒有提供Git環境設定相關功能，因此所有環境設定都必須依賴Command Line指令
- 若要使用Visual Studio Code內建的Terminal執行Git指令，可使用快速鍵：Ctrl + j，或是 Ctrl + `

Git基礎概念

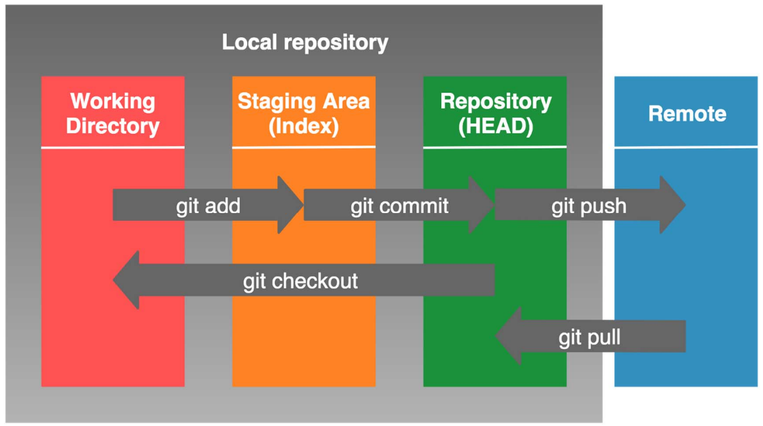
- Repository：儲存文件修改紀錄的資料庫
- Git的repository可區分為local repository與remote repository
  - Local repository：安裝在自己電腦上，方便個人使用，具備Git所有功能
  - Remote repository：安裝在遠端server上，目的是讓整個程式開發團隊共同開發某專案
  - Push：將local repository的內容上傳到remote repository
  - Pull：將remote repository的內容下載到local repository



https://w3cschoool.com/git-push

Git基礎概念

- Git local repository可以建立在作業系統任意目錄內，在該目錄內的檔案與子目錄就是要交給Git管理的資料
- Git local repository的主要組成：
  - Git directory：接受Git管理的作業系統目錄下的.git子目錄，儲存Git local repository所有資料，包含修改紀錄



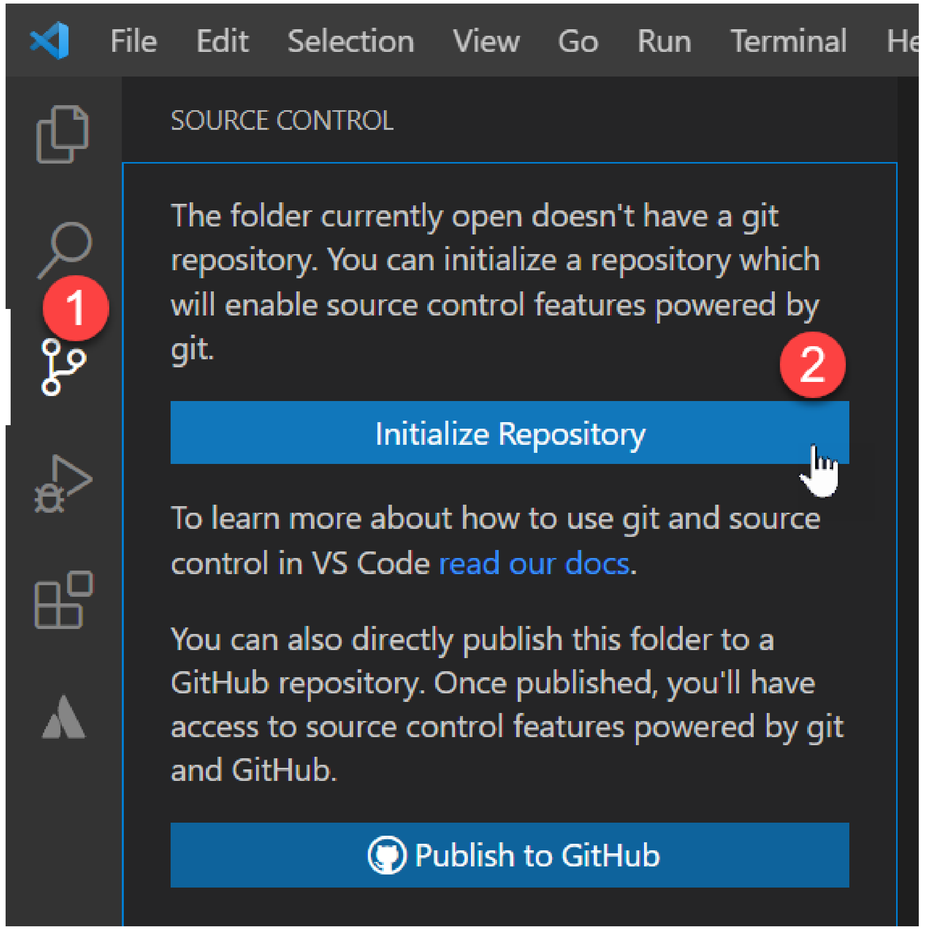
  - Working tree：接受Git管理的作業系統目錄內.git子目錄之外的所有檔案與目錄，放置目前正在編輯的版本的程式檔案，所有Git的相關操作都作用於working tree目錄
  - Staging area：接受Git管理的作業系統目錄的.git子目錄之下的index檔案，紀錄需要儲存進local repository的檔案變更，沒有記錄在staging area的檔案將不會存入repository

產生repository

- git init：將普通作業系統目錄變成git repository
- 假設要將尚未在版本控制軟體管理下的作業系統目錄 <repository目錄> 納入版本控制：
  - C:\> cd <repository目錄>
  - C:\repository目錄> git init
- git init指令會在<repository目錄>內建立一個新的子目錄.git，用來儲存所有的Git repository的相關檔案
- 在Windows作業系統下，.git目錄屬於隱藏目錄，需要修改預設顯示狀態才能看的到此資料夾

產生repository

- 使用Visual Studio Code初始化產生repository

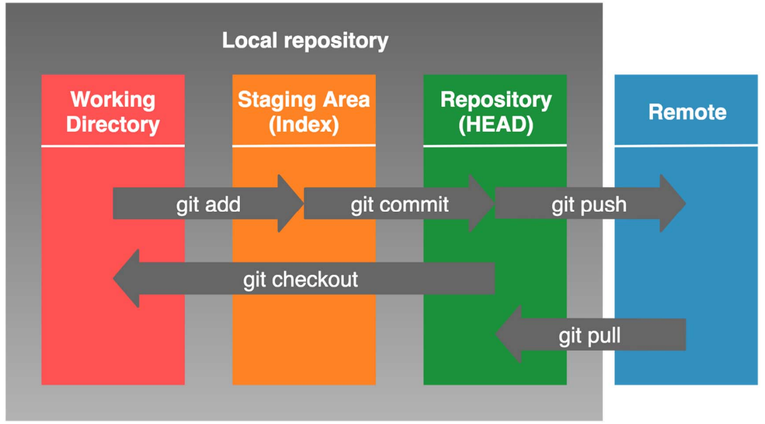


Chapter 2

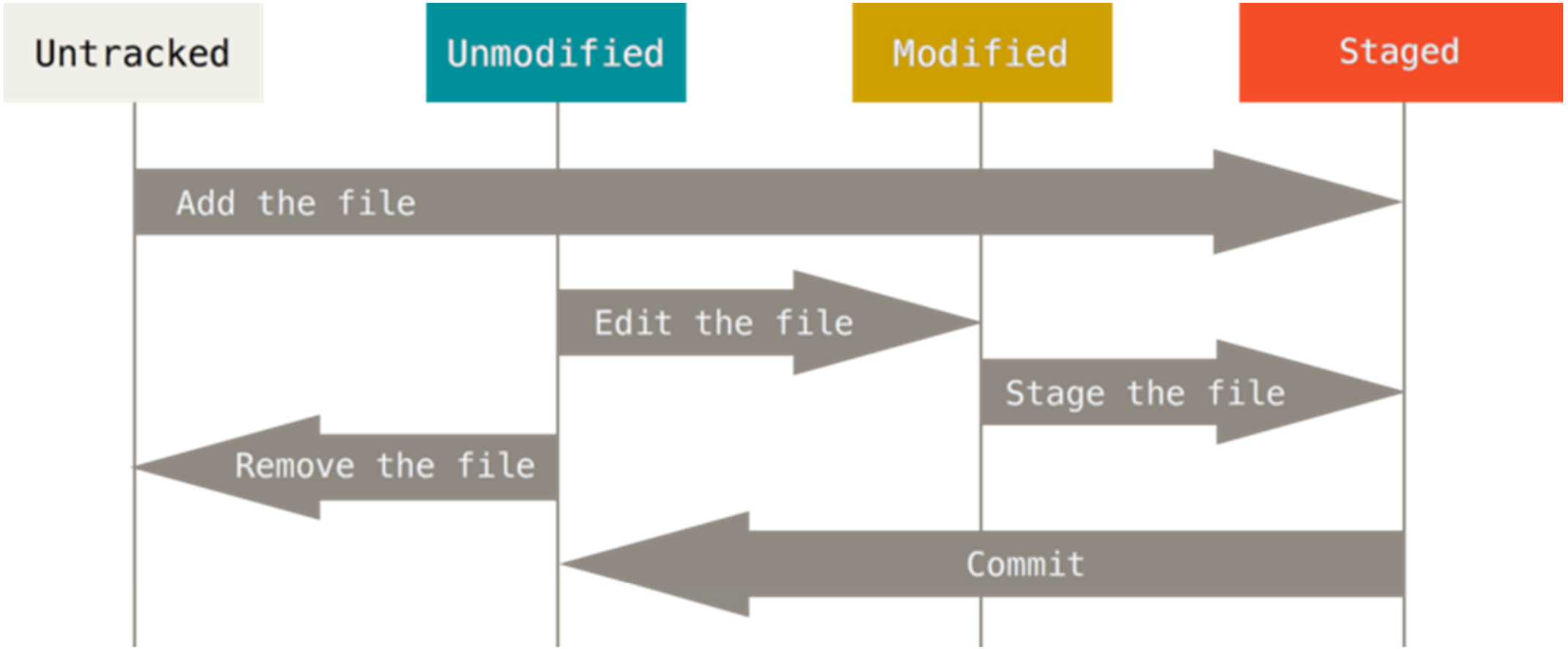
Git常用指令

Git基礎概念

- Working tree的檔案狀態：
  - Working tree目錄的檔案可以是tracked或是untracked，兩種狀態其一
  - Tracked狀態的檔案：被Git納入版本控制系統管理下的檔案，已被記錄在Git repository
  - Untracked狀態的檔案：沒有被Git納入版本控制系統管理下的檔案，沒有被記錄在Git repository
  - Tracked狀態又分成：committed/unmodified、modified、staged等3種狀態



Working tree的檔案狀態



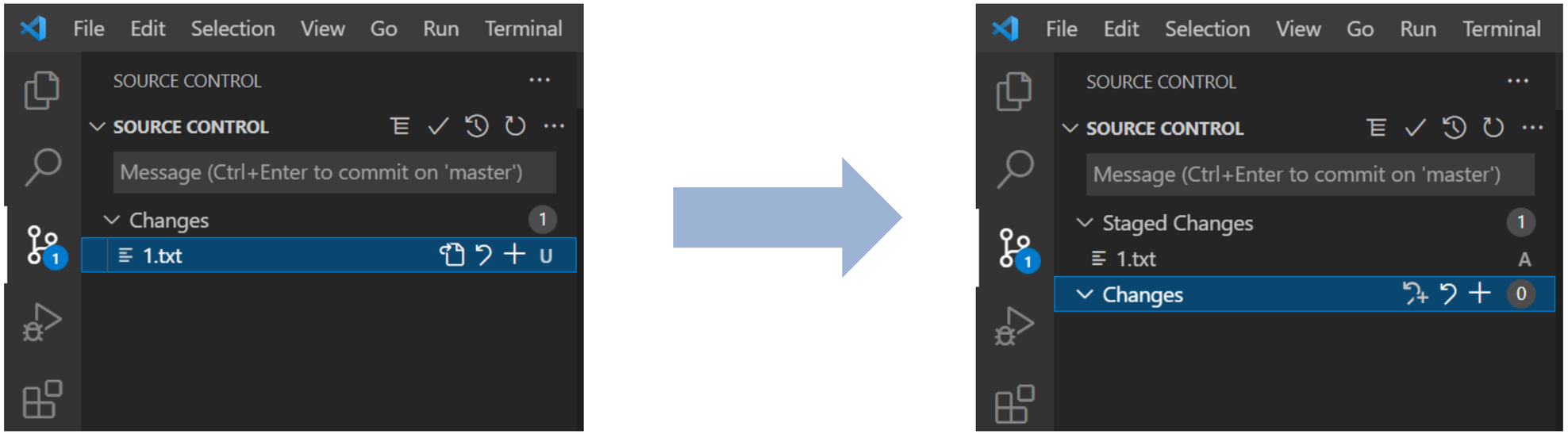
https://git-scm.com/book/zh-tw/v2

Working tree的檔案狀態

- Committed/unmodified狀態：由前次commit儲存到repository、還沒有經過再次修改的檔案
- Modified狀態：前次commit之後有經過修改，但還沒放入staging area，還沒執行commit的檔案
- Staged狀態：前次commit之後經過修改，並放入staging area，但還沒執行commit的檔案
- Untracked狀態：working tree內不屬於上述3種狀態(committed、modified、staged)的檔案

改變Working Tree檔案的狀態

- 使用Visual Studio Code改變working tree檔案的狀態
- 檔案所在位置，以及檔案右邊最後位置的字母：代表狀態
- STAGED CHANGES：代表已加入staged area
- CHANGES：代表尚未加入staged area
- A：added、U：untracked、M：modified、D：deleted，類似git status功能



改變Working Tree檔案的狀態

- 改變working tree檔案的狀態：丟棄尚未commit的更動
- 注意：丟棄尚未commit的更動屬於危險動作，執行下列指令時，不要隨便附加其他選項，以免造成嚴重後果

- 丟棄stage area的更動：c:\> git reset <檔案名稱>
(將staged狀態的檔案移出stage area)
- 丟棄working tree的更動：c:\> git checkout -- <檔案名稱>
(將這個檔案用儲存在repository的最新版覆蓋)

改變Working Tree檔案的狀態

- 將檔案由staged狀態變成committed狀態：git commit
- git commit指令影響stage狀態的檔案，新增或是修改但是尚未執行git add指令的檔案不會被git commit指令影響
- git commit的 --amend 選項：會將前次git commit指令的執行結果從repository移除，以這次的執行結果取代
- 注意，雖然amend的英文意思是修改，但是 --amend 選項的作用是使這次的commit紀錄取代舊的commit紀錄

Git基礎概念

- commit：將文件修改紀錄儲存到repository
- commit是將Git working tree的檔案修改紀錄(放在staging area的檔案修改紀錄)儲存到repository
- Commit訊息建議使用下列格式：
  第一行　：commit時修改內容的摘要
  第二行　：空行
  第三行以後：修改的理由
- 類似修復錯誤、功能新增等不同工作任務的修改，建議盡量分開來commit，這樣可以方便日後從歷史紀錄中找到特定的修改內容

Git基礎概念

- 執行commit之前：Git會要求輸入commit訊息，由於commit訊息是日後查看檔案修改紀錄的重要依據，請盡量詳細填寫
- 執行commit之後：Git比對上次commit的狀態與目前狀態產生差異紀錄，利用差異紀錄產生一組識別碼，將識別碼與差異紀錄按照時間序存入repository
- Git儲存的commit紀錄包含：commit識別碼、commit時間、使用者資訊(name與email)、commit訊息、差異紀錄

Git基礎概念

- 基本Git的使用流程
  - 修改working tree的文件檔案
  - 將修改過的檔案加入staging area (加入index)
  - 執行commit將staging area的修改紀錄存入repository
  - 注意：在working tree內做的任何變更並不會在commit時直接儲存到repository，Git commit只是將staging area的狀態儲存到repository

改變Working Tree檔案的狀態

- git status：顯示所有檔案目前的狀態
- git add：將檔案untracked變成tracked，或modified變成staged
  - 語法：git add 目錄名稱或是檔案名稱
- git diff：顯示modified狀態與staged狀態的詳細檔案資料 (顯示哪個檔案? 哪個部分? 做了那些修改?)
  - 語法：
  - git diff　　(顯示修改過，但是尚未stage的詳細資料)
  - git diff --staged　　(顯示修改過，並且已經stage的詳細資料)

改變Working Tree檔案的狀態

- git commit：將檔案由stage狀態變成committed狀態
- git commit指令影響stage狀態的檔案，新增或修改但尚未執行git add指令的檔案不會被git commit指令影響
- git commit的 --amend 選項，會將前次git commit指令的執行結果從repository移除，以這次的執行結果取代
- 注意：雖然amend的英文意思是修改，但是 --amend 選項的作用是使這次的commit紀錄取代舊的commit紀錄

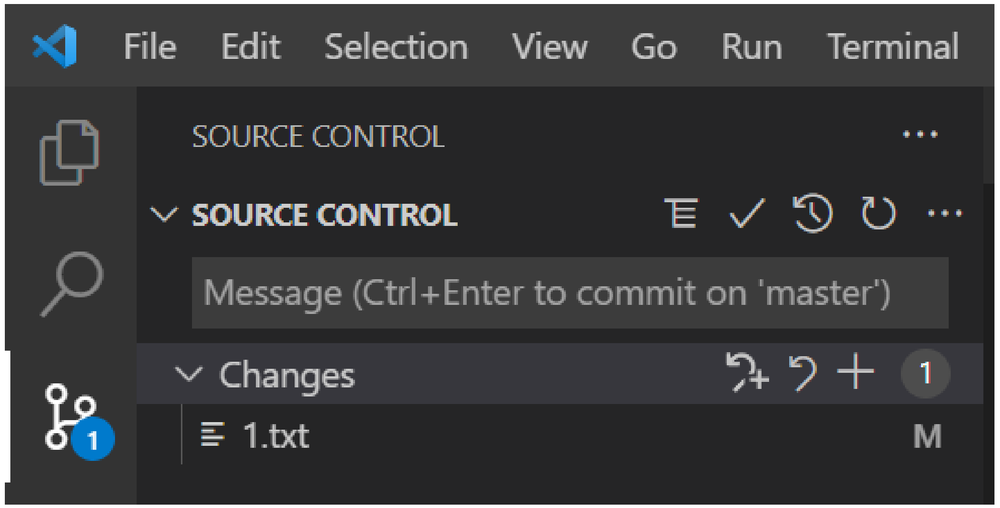
改變Working Tree檔案的狀態

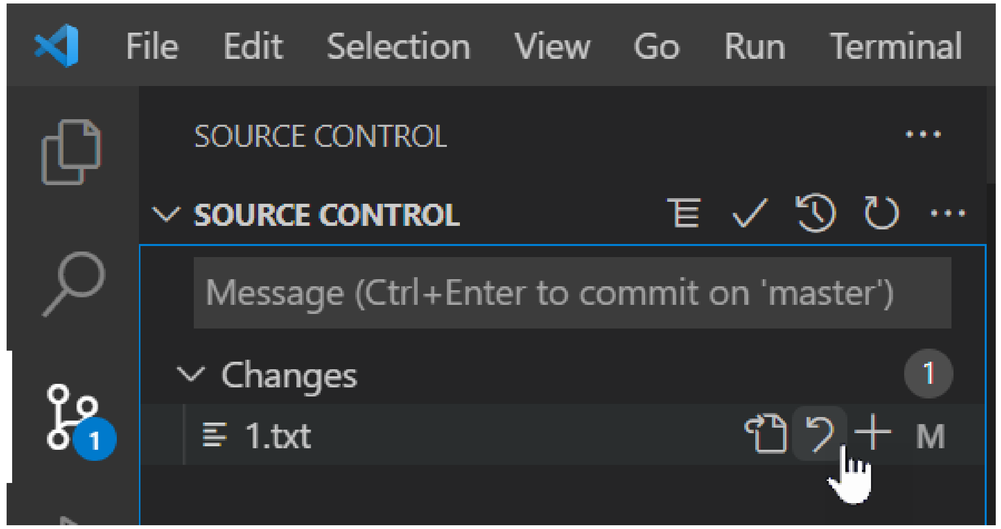
- 丟棄尚未commit的更動：注意! 丟棄尚未commit的更動屬於危險動作，執行以下指令時，不要隨便附加其他選項以免造成嚴重後果

- git reset 檔案名稱：丟棄stage area的更動
  (將staged的檔案移出stage area)
- git checkout -- 檔案名稱：丟棄working tree的更動
  (將這個檔案用儲存在repository的最新版覆蓋)

用Visual Studio Code丟棄尚未commit的更動

- 檔案所在位置CHANGES右邊的 符號：丟棄所有檔案的修改
- 檔案右邊的 符號：丟棄1個檔案的修改
- git checkout -- 檔案名稱：同樣功能





Lab

Git ignore file

- Git ignore file：不讓Git版本管理的檔案
- 檔案名稱是 .ignore，紀錄不想交給Git管理的檔案，避免在後續的commit過程中意外被存入repository
- .ignore的檔案應包含：
  - 由其他檔案產生出來的資料，如：log檔，編譯後的執行檔
  - 會隨著環境不同而變的設定檔，如：開發工具的設定參數
  - 包含密碼的檔案，如：資料庫密碼
- Github提供十多個不同程式語言的.ignore範例檔，在Github新建repository時可以選擇

Git ignore file

- 1個repository可以只使用1個 <repository目錄>/.ignore 檔案，運用遞迴作用到整個repository
- 在更為複雜的情況下，每個子目錄都可以擁有 .ignore 檔，這些 ignore file 只作用於檔案所在的子目錄

Git ignore file

- Git ignore file語法規則：
  - 空白行或是以 # 開頭的內容，會被Git忽略
  - 接受標準glob語法，並且遞迴作用到所有子目錄
  - 使用開頭 / 符號避免遞迴發生
  - 使用結尾 / 符號表示目錄
  - 在pattern語法之前使用！符號表示否定

  - Glob — Unix style pathname pattern expansion (UNIX風格路徑名模式擴展)
  - Glob語法詳細參考：https://en.wikipedia.org/wiki/Glob_(programming)

Git ignore file

- Glog語法簡介：
  - * 符號：對應 0~多個字元
  - [abc]：對應 a、b 或 c 的任何字元
  - ? 符號：對應0或1個字元
  - [0-9]：對應 0~9 的任何字元
  - dir1/**/dir2：對應以dir1開頭，以dir2結尾的任何目錄，如：dir1/xxx/dir2、dir1/xxx/kkk/dir2、 dir1/dir2

Lab

檢查Git commit的歷史紀錄

- git log：檢查git commit的歷史紀錄
- 依照時間的順序，列出repository內所有commit的相關資料
- 包含：SHA-1 checksum、使用者姓名email、commit時間、commit訊息
- git log -數字：限制顯示commit的數量
  - git log -2：只顯示最後兩次commit的相關資料
- git log的 --pretty 選項：改變預設commit相關資料的格式，內建四個選項：oneline、short、full、fuller、format
  - git log --pretty=oneline、short、full、fuller (四選一)
  - git log --pretty=format:"自訂格式"

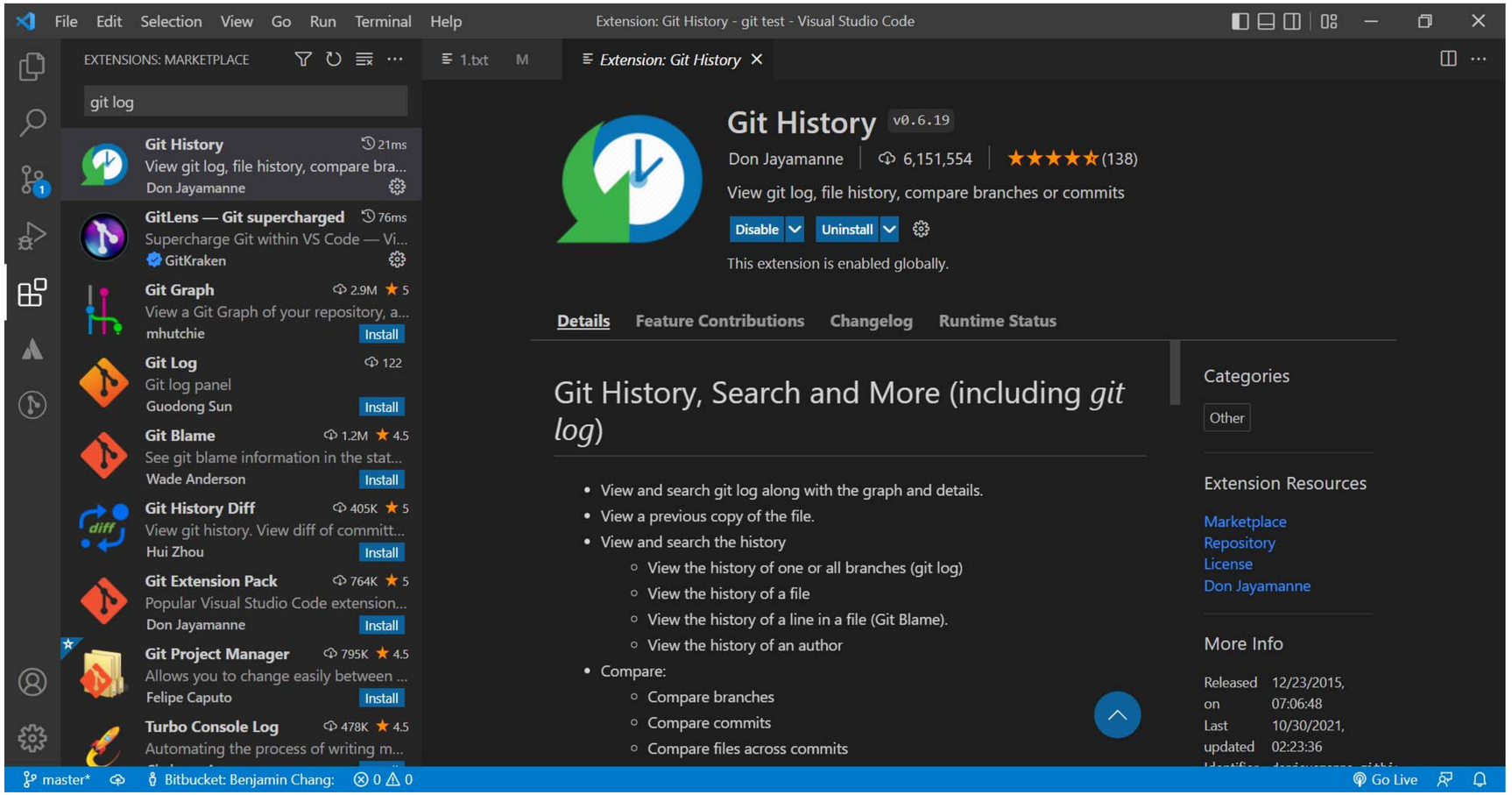
檢查Git commit的歷史紀錄

- git log --pretty=format:"自訂格式"
- 自訂格式常用語法

語法	意義	語法	意義
%H	SHA-1 cheaksum	%s	Commit訊息主題(第一行)
%h	簡化版SHA-1 cheaksum	%b	Commit訊息內容(主題之外)
%an	檔案產生者姓名	%cn	檔案提交者姓名
%ae	檔案產生者email	%ce	檔案提交者email
%ad	檔案產生者修改時間	%cd	檔案提交者修改時間
%ar	檔案產生者相對於目前的時間	%cr	檔案提交者相對於目前的時間

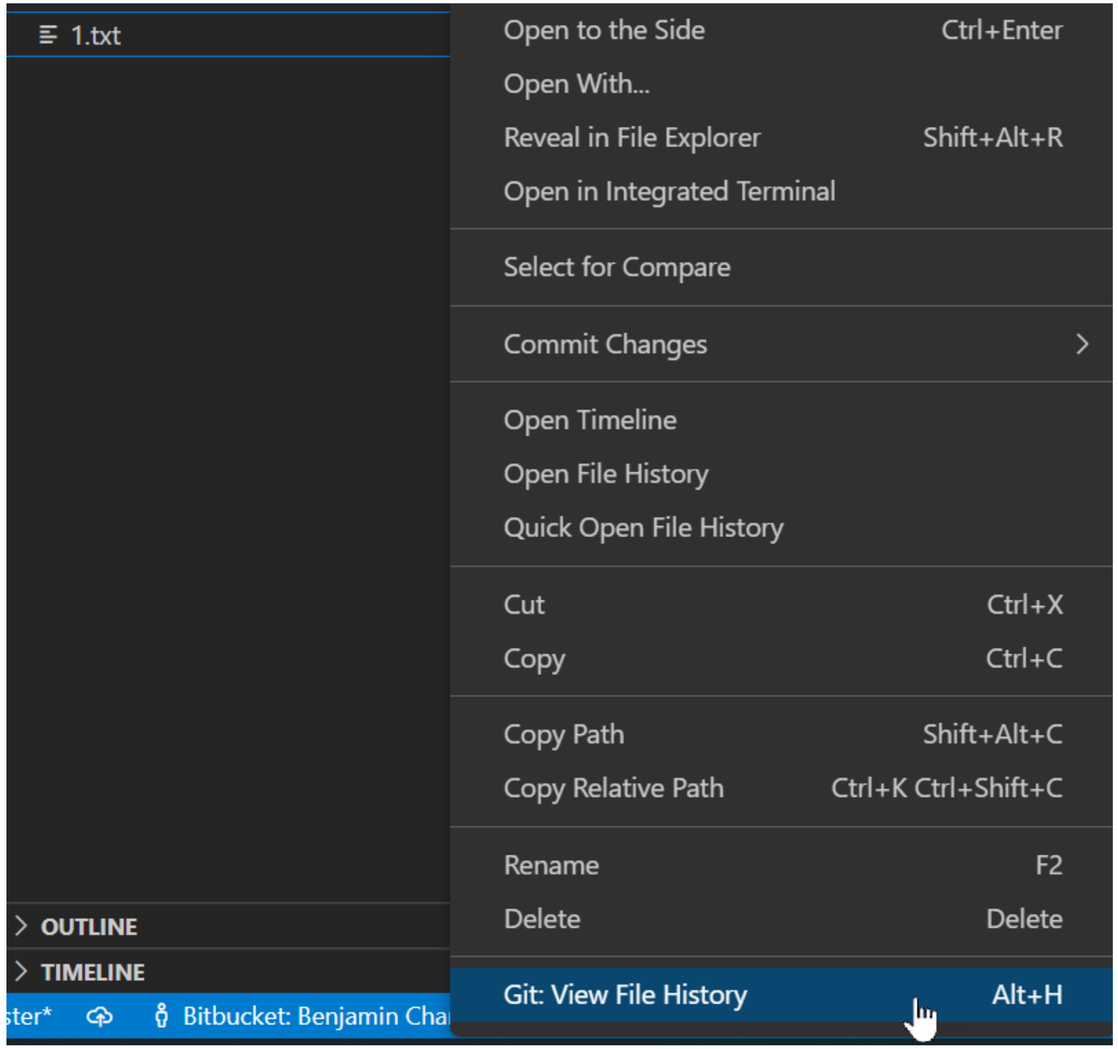
檢查Git commit的歷史紀錄

- Visual Studio Code沒有內建Git commit歷史紀錄的檢視功能，但是可以安裝其他extension達成
- 可以使用Git History (git log)



檢查Git commit的歷史紀錄

- Git history extension使用方式：
  - 在working tree的目錄或檔案上
  - 按下滑鼠右鍵
  - 選擇「Git: View File History」



Chapter 3

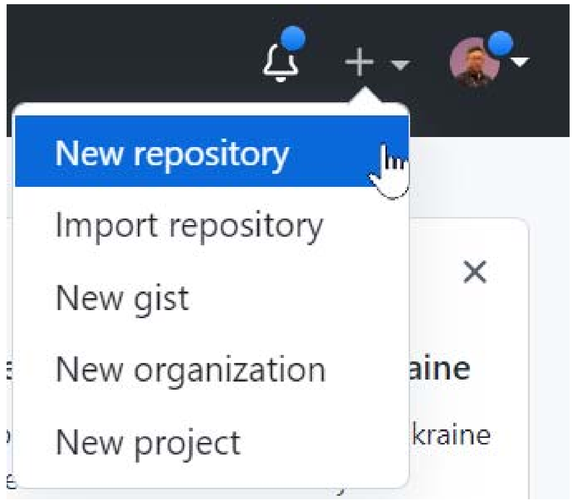
Git remote repository

Git remote repository基礎概念

- Git repository分為local與remote repository
- Remote repository主要目的是讓程式開發團隊的成員分享各自的local repository資料而建立
- 雖然git remote repository也可放在畚箕電腦，但是通常選擇放在遠端伺服器上，伺服器可以是自行建置的或是選擇市面上的git repository託管服務
- Git repository託管服務包括：Bitbucket, Github… 等
- 我們示範的是將remote repository放在Github上

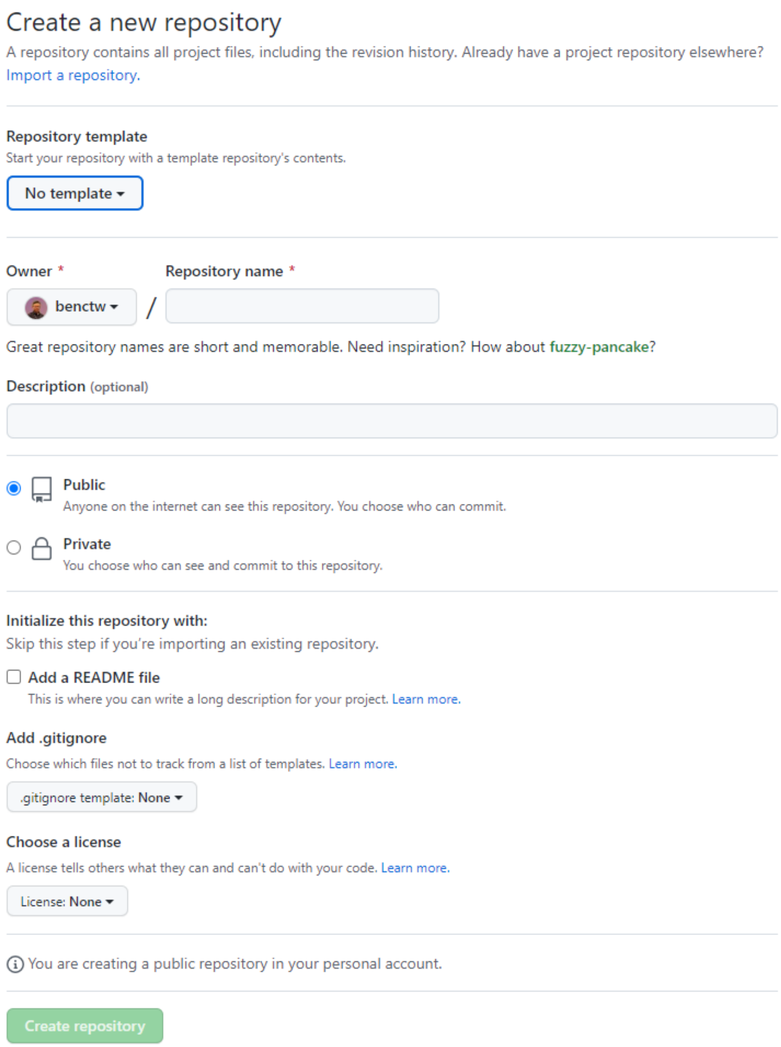
在Github建立Git remote repository

- 登入Github，畫面右上方 +號下拉選單，選擇「New repository」
- 輸入自訂義的repository名稱，create repository
- Github提供HTTPS、SSH兩種方式讓client git從本機連線到遠端github repository。我們使用HTTPS方式

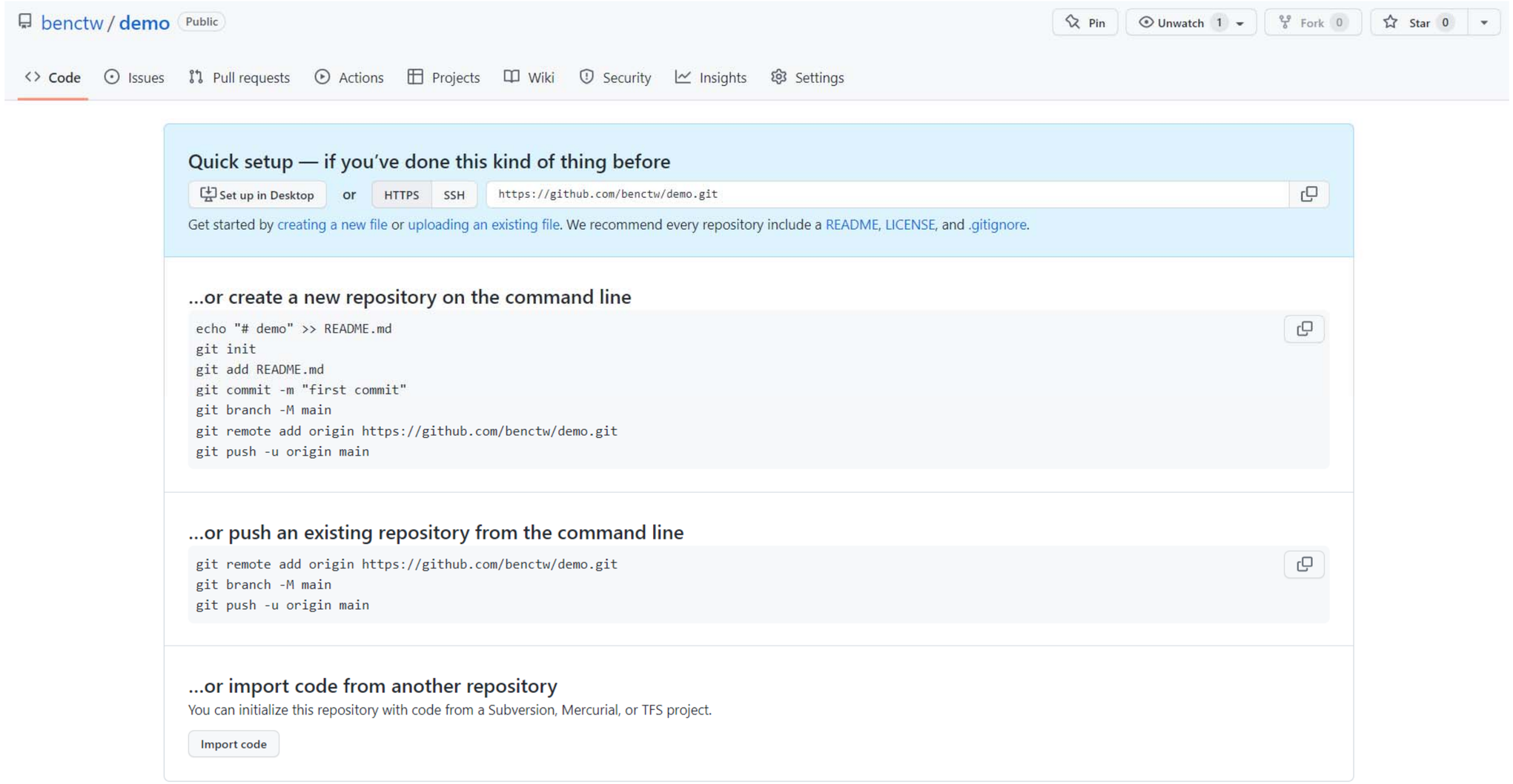


- URL格式：
  https://github.com/<Github帳號>/<repository名稱>.git

在Github建立Git remote repository



Github的說明指引



Git remote repository基本指令

- git remote add：在Git環境中加入remote repository
  語法：git remote add remote名稱 remote-url
- git remote -v：顯示目前設定的所有remote repository
- git remote show 名稱：顯示目前設定的某個remote repository
- git remote rename：變更remote repository名稱
  語法：git remote rename 舊名稱 新名稱
- git remote remove：刪除某個remote repository設定
  語法：git remote remove 名稱

操作示範

- git remote add demo-remote https://github.com/.../....git
- git remote
- git remote -v

```
C:\Users\Benjamin\Downloads\git test>git remote add demo-remote https://github.com/benctw/demo.git

C:\Users\Benjamin\Downloads\git test>git remote
demo-remote

C:\Users\Benjamin\Downloads\git test>git remote -v
demo-remote     https://github.com/benctw/demo.git (fetch)
demo-remote     https://github.com/benctw/demo.git (push)
```

操作示範

- git remote remove demo-remote
- git remote -v

```
C:\Users\Benjamin\Downloads\git test>git remote remove demo-remote

C:\Users\Benjamin\Downloads\git test>git remote -v
```

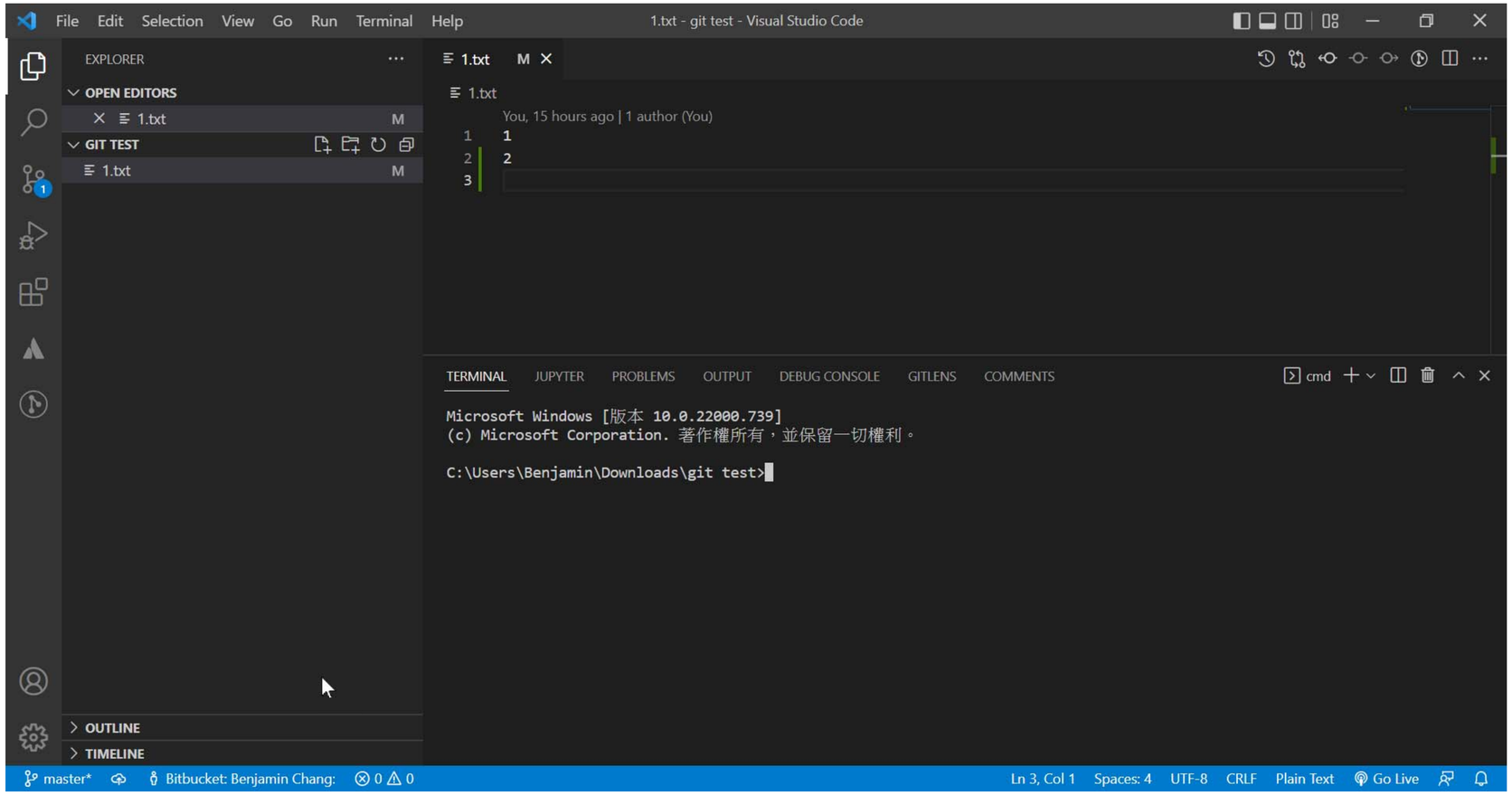
操作示範

- git remote add orogon https://github.com/benctw/demo.git
- git remote rename orogon origin
- git remote

```
C:\Users\Benjamin\Downloads\git test>git remote add orogon https://github.com/benctw/demo.git

C:\Users\Benjamin\Downloads\git test>git remote
orogon

C:\Users\Benjamin\Downloads\git test>git remote rename orogon origin

C:\Users\Benjamin\Downloads\git test>git remote
origin
```

Git remote repository in Visual Studio Code

- Visual Studio Code 沒有提供Git remote repository基本指令功能，新增、查詢、修改、刪除remote都要依賴command line指令
- 可以使用快速鍵Ctrl+j，快速在Visual Studio Code內呼叫Command line



關於Remote Repository的名稱

- 雖然remote名稱只是 https://github.com 網址的邏輯名稱(如電話簿的名稱)，可以隨意指定，但是 origin 是最常見的 remote 名稱，他是git clone指令的附帶產物，沒有特殊能力
- git clone指令的功能是運用remote repository紀錄產生，並且初始化local repository，執行過程中順便產生remote設定，所使用的remote名稱是origin，這造成了到處都看到origin名稱
- 所以我們在這裡使用origin作為remote repository的名稱

Git remote repository in Visual Studio Code

- git remote add origin https://github.com/benctw/demo.git
- git remote -v

```
C:\Users\Benjamin\Downloads\git test>git remote add origin https://github.com/benctw/demo.git

C:\Users\Benjamin\Downloads\git test>git remote -v
origin  https://github.com/benctw/demo.git (fetch)
origin  https://github.com/benctw/demo.git (push)
```

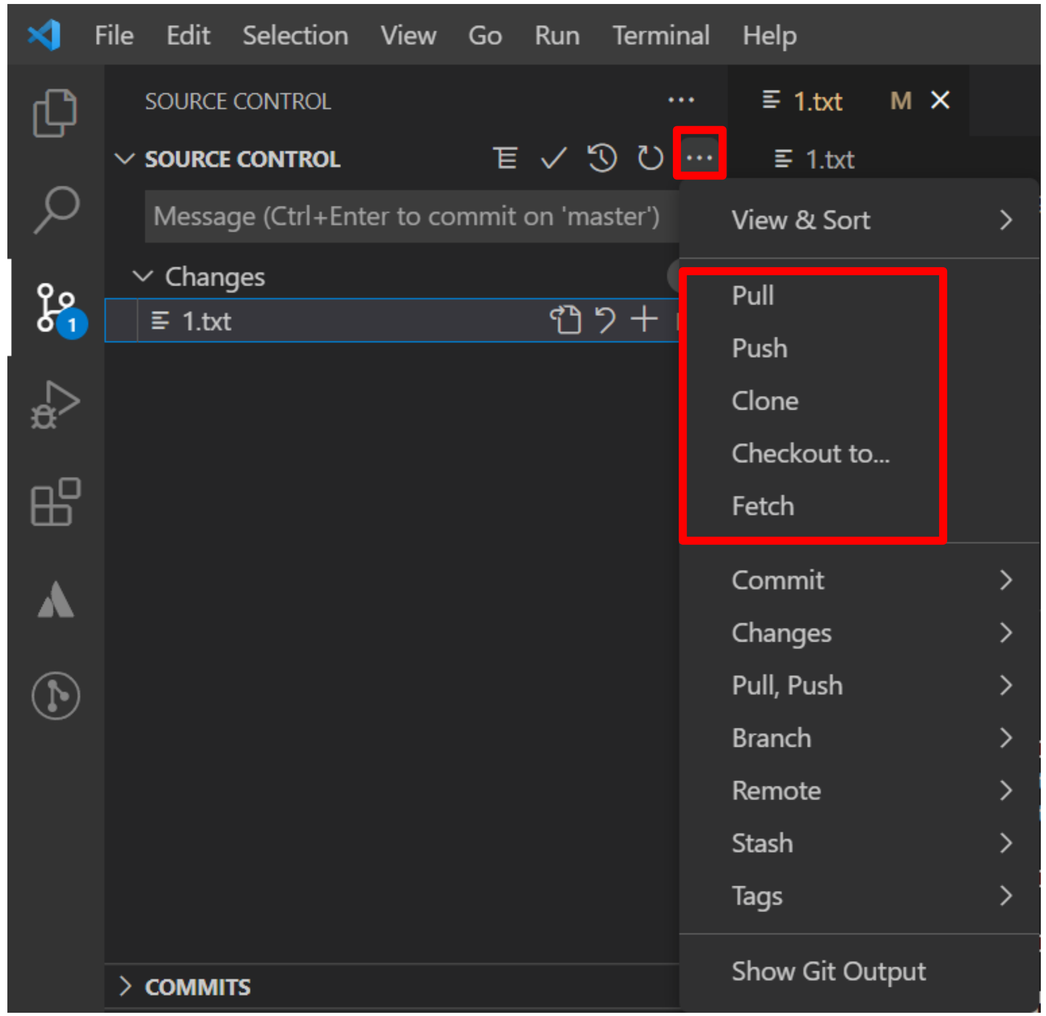
下載、上傳 Git remote repository 資料

- git push：將local repository資料上傳到remote repository
  語法：git push remote名稱 local-branch名稱
- git pull：將remote repository資料下載到local repository
  語法：git pull remote名稱 local-branch名稱

- 注意：雖然沒有硬性規定，但是習慣上，執行git push、git pull指令之前，請先執行git commit

在 VS Code下載、上傳 Git remote repository 資料

- Visual Studio Code中，下載、上傳到Git remote repository的功能依賴Source Control的 ... 功能
- 上傳：Push選項或是Push to...選項
- 下載：Pull選項或是Pull from...選項



Git Credential Manager for Windows

- 使用HTTPS連線上傳資料到remote repository時，Git預設行為是每次連線都要輸入帳號密碼，Git不會幫忙儲存登入資訊
- Git Credential Manager for Windows會負責儲存登入資訊，並且每次上傳資料時會自動提供登入資訊，免除每次輸入的麻煩
- Git Credential Manager for Windows不需安裝，不須啟用，只要確定Git環境設定有 credential.helper=manager-core 即可

Git Credential Manager for Windows

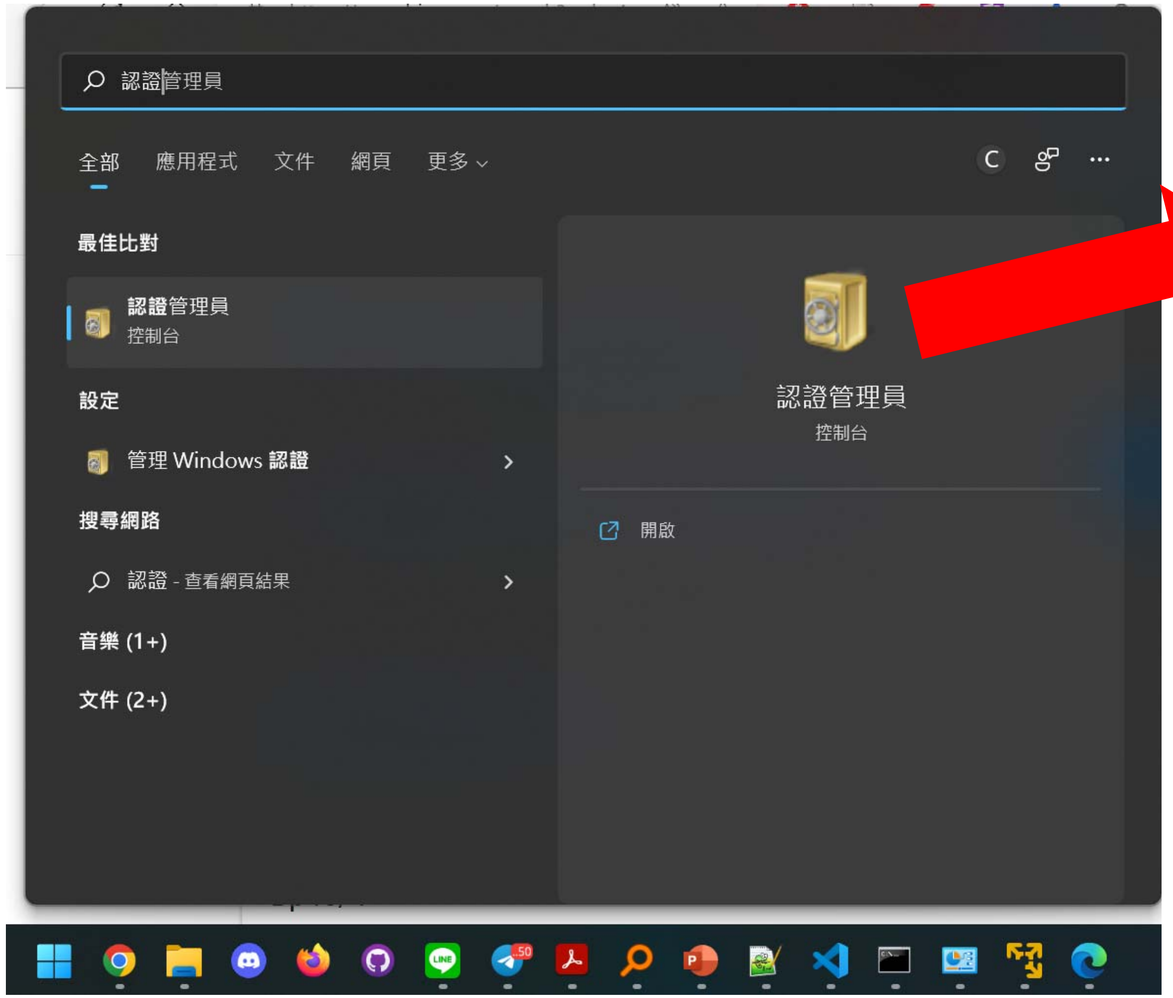
- git config --list

```
C:\Users\Benjamin>git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager-core
pull.rebase=false
credential.https://dev.azure.com.usehttppath=true
user.name=Benjamin
user.email=benctw@gmail.com
difftool.sourcetree.cmd='' "$LOCAL" "$REMOTE"
mergetool.sourcetree.cmd=''
mergetool.sourcetree.trustexitcode=true
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
```

Git Credential Manager for Windows

- Git Credential Manager for Windows將帳號密碼資訊儲存在Windows作業系統內建的Windows Credential Store儲存區
- 若要移除帳號密碼資訊需透過Windows認證管理員：
- 工作列的搜尋方塊中輸入「認證管理員」，然後選取[認證管理員控制台]
- 選取「Windows認證」，尋找「git:https://github.com」，並「移除」
- 下次在Visual Studio Code操作Github就會跳出Github帳號認證視窗

刪除既有的Git Credential

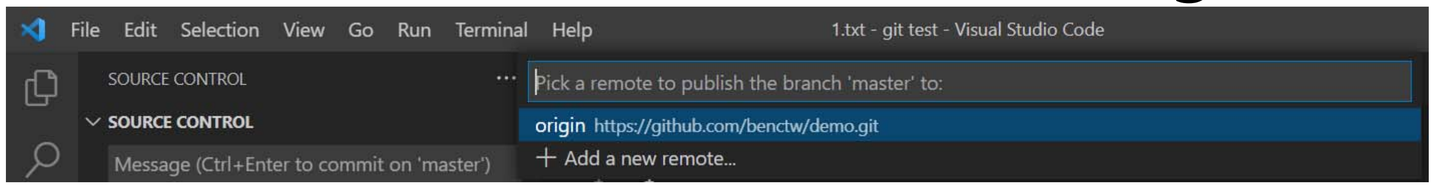


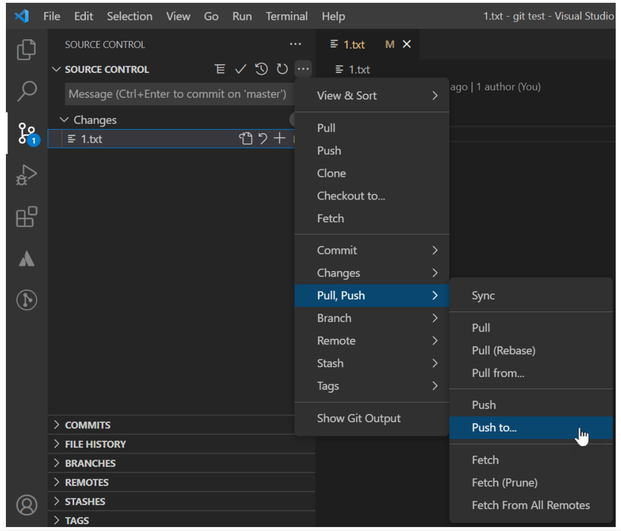
Lab

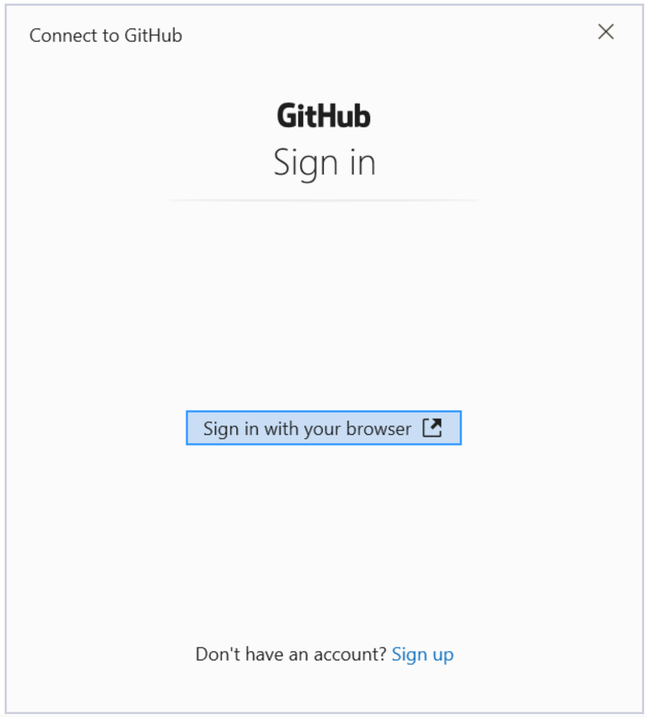
- 上傳Git remote repository：
  - 確認目前位置是master branch、remote已經是設定
  - 將master branch資料上傳到origin remote
  - 檢查上傳成果

在 VS Code 上傳 Git remote repository 資料

- 將 master branch 資料上傳到 origin remote
- git push remote名稱 local-branch名稱
- Github登入由Git Credential Manager for Windows提供







下載Git remote repository

- 使用Visual Studio Code打開某目錄
- 建立local repository
- 設定remote：
  git remote add origin https://github.com/.../....git
- 下載Git remote repository
  git pull origin master
- 檢查下載結果

```
C:\Users\Benjamin\Downloads\git pull test>git init
Initialized empty Git repository in C:/Users/Benjamin/Downloads/git pull test/.git/

C:\Users\Benjamin\Downloads\git pull test>git remote add origin https://github.com/benctw/demo.git

C:\Users\Benjamin\Downloads\git pull test>git remote -v
origin  https://github.com/benctw/demo.git (fetch)
origin  https://github.com/benctw/demo.git (push)

C:\Users\Benjamin\Downloads\git pull test>git pull origin master
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 179 bytes | 7.00 KiB/s, done.
From https://github.com/benctw/demo
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
```

團隊開發的Git使用流程

- 假設課程分組進行專題製作，全組同學開發相同名稱的專案，使用相同Github帳戶，使用相同名稱的remote repository
- 在執行git push功能是否出現問題? 尤其是多位組員修改相同位置、相同名稱的檔案
- 由於某位組員前次push到現在再次push之間，可能有其他組員已完成push流程，而造成自己push失敗，所以正確的流程應該是先執行pull，緊接著再執行push

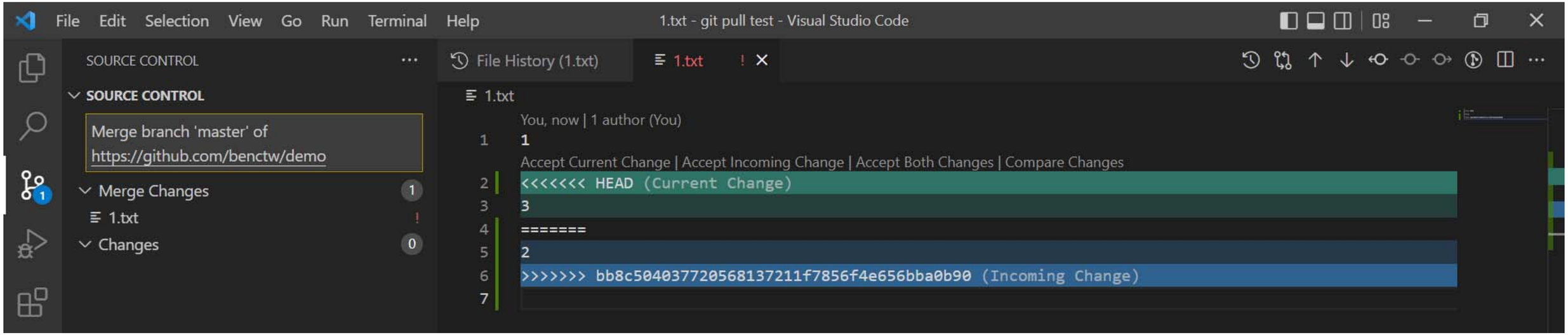
團隊開發的Git使用流程

1. 撰寫程式、測試程式、累績一定量後(例如一個完整的程式功能)，將修改內容commit儲存到本機local repository
2. 完成一個段落(例如一個完整程式流程)，並且commit之後，準備將local repository的修改紀錄push到remote repository
3. 由於不知道是否有其他組員已經完成push流程，為了避免push失敗，必須先執行pull
4. 如果pull流程發生衝突(conflict)，必須手動解決並且commit
5. 完成pull流程之後，才能執行push流程

如果你在辛苦pull以及解決衝突的過程中，又有其他組員push，那就...

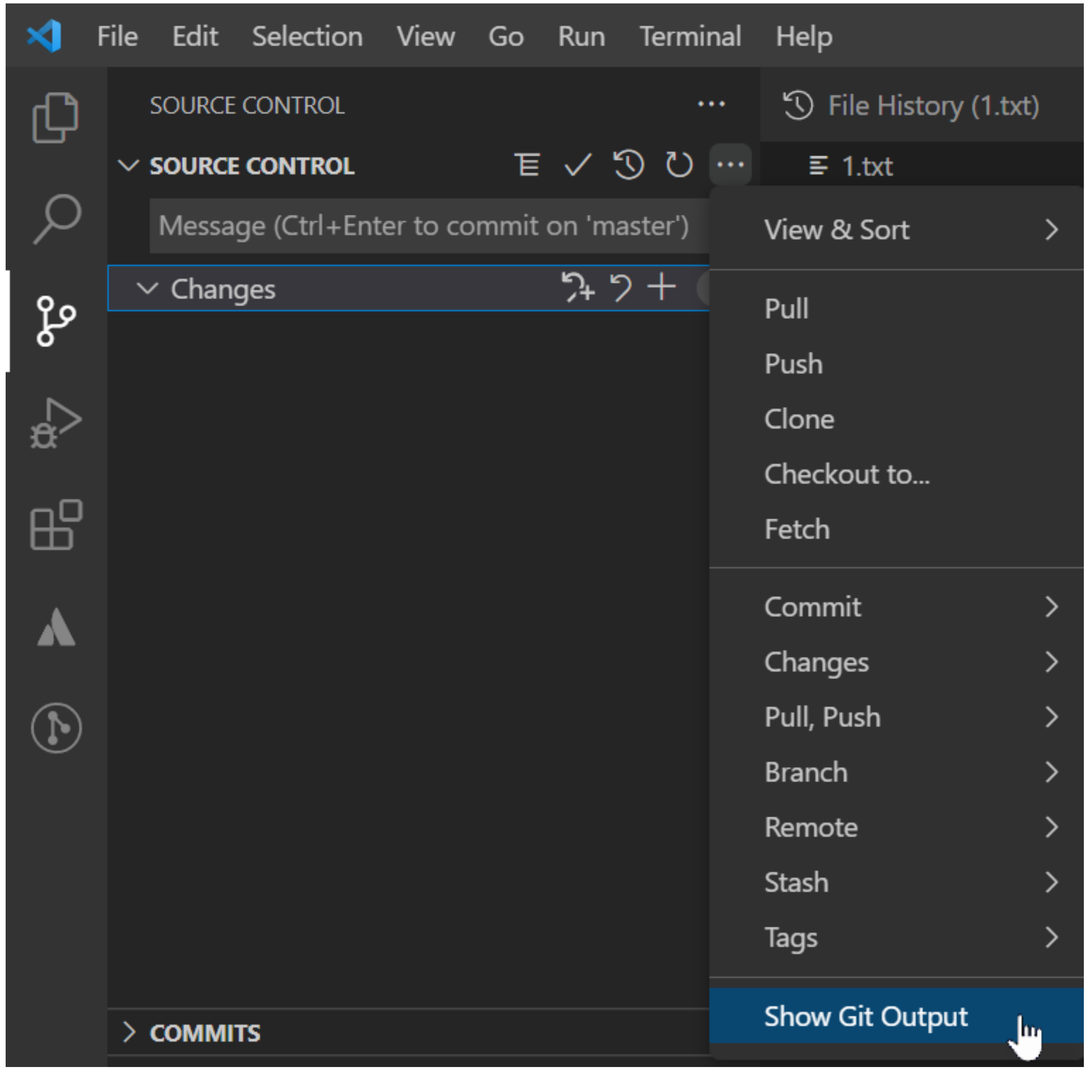
團隊開發的Git使用流程

- 衝突(conflict)：相同目錄、相同名稱的檔案有不同內容，而且Git無法自動合併檔案內容的情況
- Git會在發生衝突的檔案內容中加入特殊符號標示衝突內容的位置，方便使用者人工決定要如何合併檔案內容



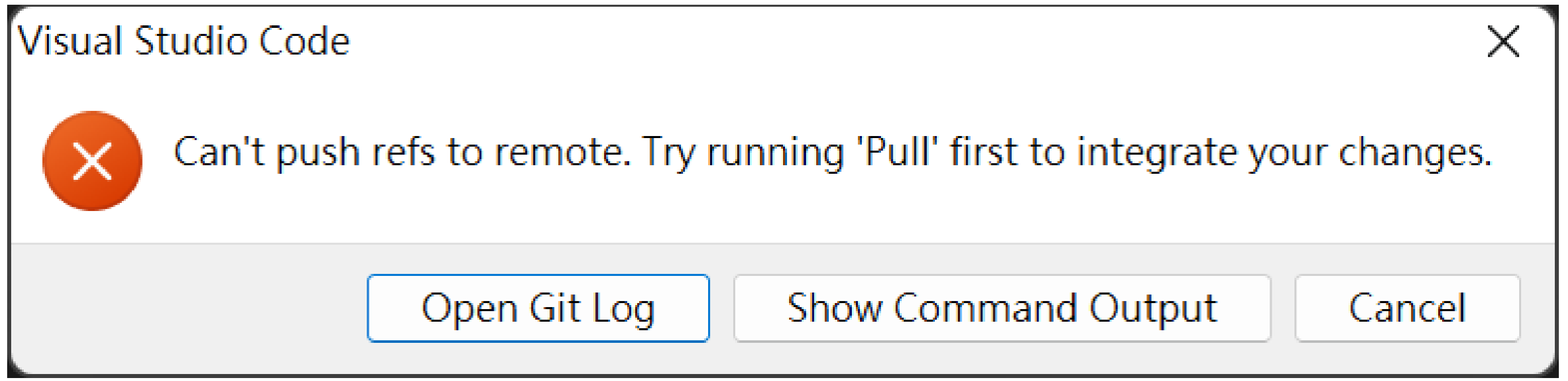
團隊開發的Git使用流程

- Visual Studio Code在某些push失敗的情況中不會顯示錯誤訊息，必須使用Shoe git output功能來檢視



團隊開發的Git使用流程

- Show Git Output視窗顯示的push失敗訊息




```
[2022-06-21T07:23:17.574Z] > git push origin master [1696ms]
[2022-06-21T07:23:17.574Z] To https://github.com/benctw/demo.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'https://github.com/benctw/demo.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
[2022-06-21T07:23:50.010Z] > git fetch [879ms]
```

Lab：團隊開發的Git使用流程

- 執行push流程失敗，改為執行pull
- 假設pull流程發現有檔案衝突(conflict)→手動解決衝突→commit
- 再次執行push流程
- 檢查push執行結果

Chapter 4

Git Branch

Git branch基本概念

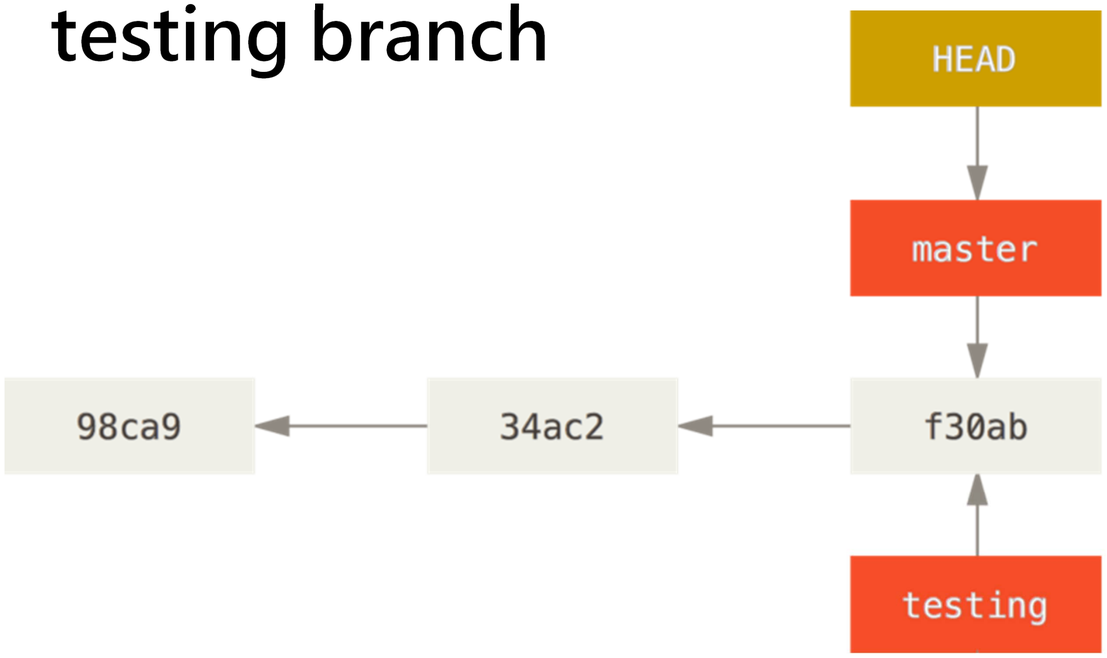
- Branch：同1組檔案的2個複製品
- Branching通常意味著檔案修改偏離主要流程，並且在不影響主要開發流程的情況下修改檔案
- 可以想像成將所有檔案複製1份儲存在其他位置，連同檔案修改紀錄也複製1份使用另外名稱儲存，讓原始檔案與複製檔案可以用不同方式、不同進度進行開發修改並各自自紀錄開發過程，而且不會互相影響
- 實際上Git branch是指向某次commit紀錄，可以移動的pointer。沒有真實複製檔案與commit紀錄，所以運作速度超快 (Git本身就是紀錄差異，而不是複製檔案)

Git branch基本概念

- git init指令預設產生名為master的branch作為預設branch，隨後對檔案修改的commit都會儲存在master branch之下，形成檔案修改的主要流程
- Master branch只是一個普通的branch，不具有特殊權限或能力
- 如果想在不影像主要流程的情況下嘗試新想法，可以在master branch之外建立新的branch進行獨立開發
- 新的branch在起始時，與master branch指向同一個commit紀錄，隨著後續commit，漸漸與master branch偏離形成獨立的開發流程
- 新的branch使用完畢後可以視狀況選擇保持獨立存在，或是與master branch合併(merge)

Git branch基本概念

- 除了名稱為master的branch之外，Git還會產生1個特殊的pointer，稱為HEAD，指向使用者目前正在編輯檔案所在的branch
- 以下圖為例，左圖HEAD指向master branch，修改及commit會存入master branch；右圖HEAD指向testing branch，修改與commit會存入testing branch



Git基本branch管理指令

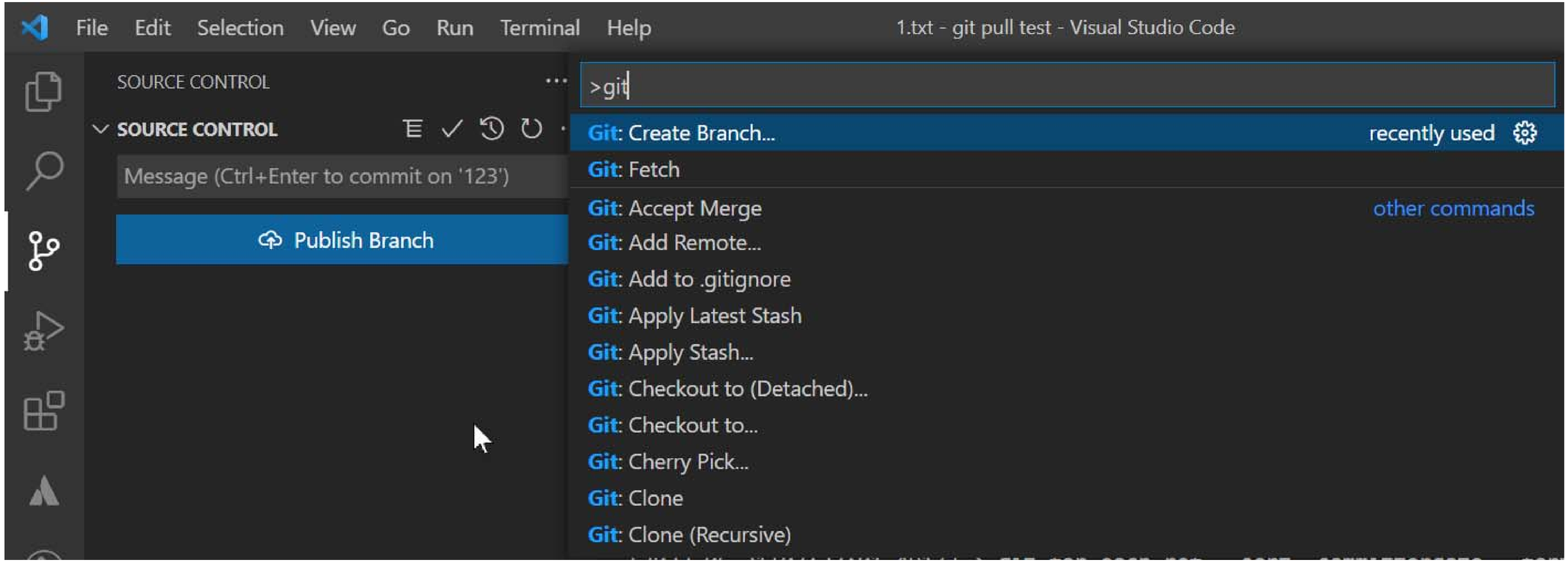
- git branch branch名稱：產生branch
- git branch --list：列出所有local branch
- git checkout branch名稱：切換到不同的branch
  (git branch指令只會產生新的branch，不會自動切換到新的branch)
- git checkout -b branch名稱：相當於連續執行git branch指令與git checkout指令
- git branch -d branch名稱：刪除branch
  為避免困擾，先切換到不需要被刪除的branch再執行刪除branch指令

在Visual Studio Code進行Git branch管理

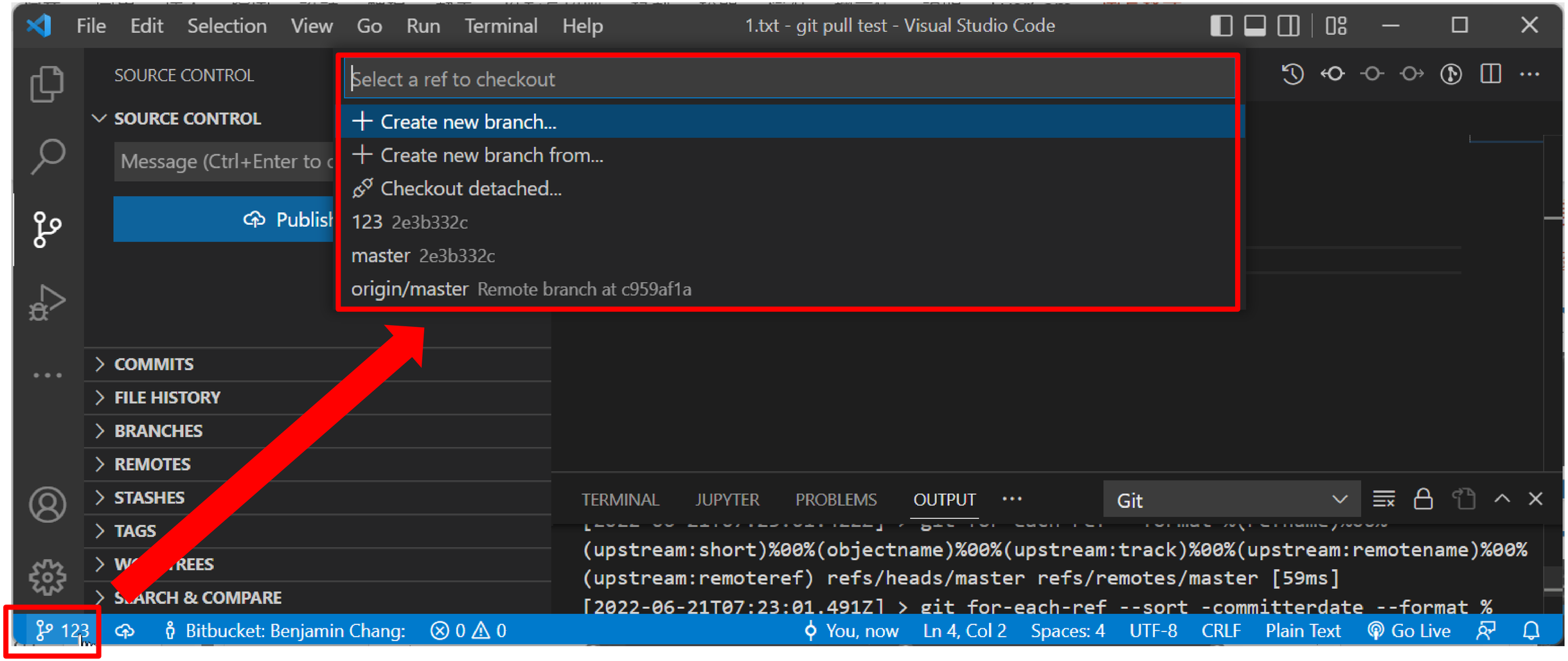
- Visual Studio Code的Git branch功能依賴command palette與畫面左下方的Git Status Bar
- Command Palette：使用快速鍵 Ctrl + Shift + P 顯示
- 操作：
  - 建立branch：在Command Palette輸入git create branch
  - 刪除branch：在Command Palette輸入git delete branch
  - 列出branch：點選Git Status Bar的branch名稱
  - 切換branch：點選Git Status Bar的branch名稱，接著在畫面上方選取目標branch名稱

Command Palette

- 使用快速鍵 Ctrl + Shift + P 顯示Command Palette



Git Status Bar

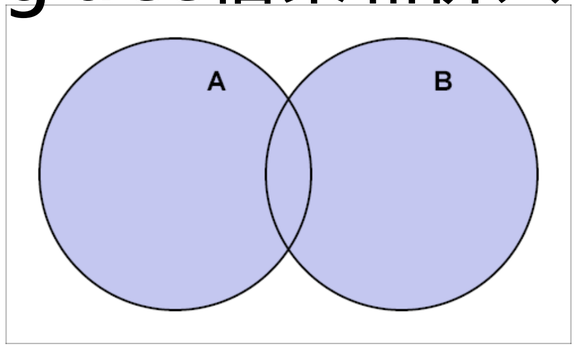


Lab：Visual Studio Code的Git branch管理

- 產生，並切換到demo branch
- 切換回master branch
- 刪除demo branch
  確認回到master branch，不是在demo branch，再做刪除demo branch

Git branch基礎概念

- 合併branch：將1個branch的repository紀錄與working tree檔案和併入另1個branch



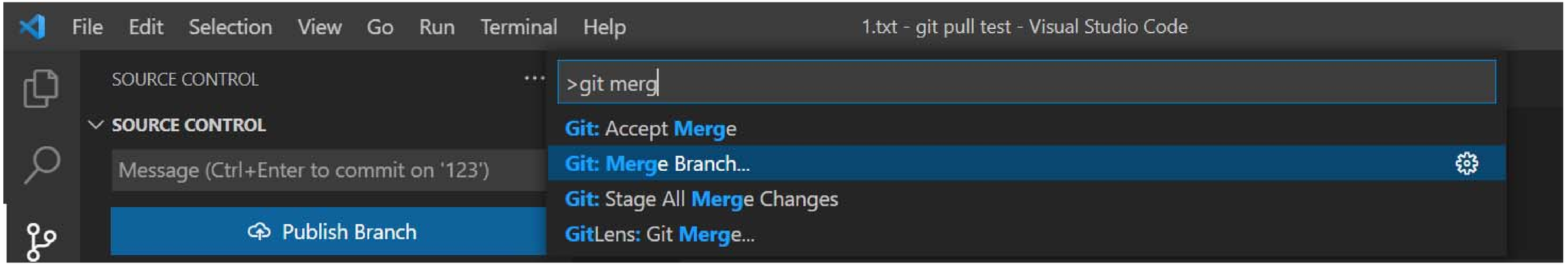
- 類似聯集的概念，其中，A branch擁有但是B branch沒有，反之亦然的檔案只要加入最終集合即可
- A branch與B branch共有的檔案(交集處)，Git會先嘗試將檔案內容合併，如果發生Git無法自動合併的情況(稱為衝突conflict)，就必須由手動解決
- 衝突conflict：2個branch中相同目錄，相同名稱的檔案擁有不同內容，曾且Git無法自動合併檔案內容的情況

Git基本branch管理指令

- git merge branch名稱：合併branch
- 指令意義：將某branch合併到目前操作的branch
- 雖然不是硬性規定，但是建議將其他branch合併到master branch

在Visual Studio Code操作合併branch

- 合併branch：在Command Palette輸入git merge branch



在Visual Studio Code操作合併branch

1. 為了實作新功能建立featureX branch，進行開發
2. 新功能尚未完成，為了緊急處理錯誤建立bugfix branch
3. 錯誤處理完畢，將master branch與bugfix branch合併
4. 回到featureX branch繼續開發
5. 將master branch與featureX branch合併
6. 視情況需要，可以考慮刪除featureX branch、bugfix branch

Lab：Git merge

- 建立featureX branch開發新功能
  - 開發新功能後，別忘了commit
- 建立bugfix branch修正錯誤
  - 修復錯誤後，別忘了commit
- 將master branch與bugfix branch合併
- 檢查log
- 將master branch與featureX branch合併
- 檢查log

恭喜

你應該可以掌握Git的用法了