

- 【Github】操作基礎步驟

一、在 Github 上建立 repository Owner 欄位的意思是我們的**專案資料夾**的意思 README.md 可以用來說明我們的 repository 的用途，他支援 markdown 的語法

緯育 TibaMe

2-2：創建 Repository

創建 repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Create a new repository

Owner * **Repository name**: uuboyscytest / myrepo

Great repository names are short and memorable. Need inspiration? How about [jubilant-chainsaw](#)?

Description (optional):

Public Anyone on the internet can see this repository. You choose who can commit.
 Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. [Learn more](#)
 Add .gitignore Choose which files not to track from a list of templates. [Learn more](#)
 Choose a license A license tells others what they can and can't do with your code. [Learn more](#).

This will set **main** as the default branch. Change the default name in your [settings](#).

Create repository

為 repository 命名

勾選新增 README
Mark

創建 repository

- 建立完成後，會出現這個頁面，可以了解一下

緯育 TibaMe

2-2：創建 Repository

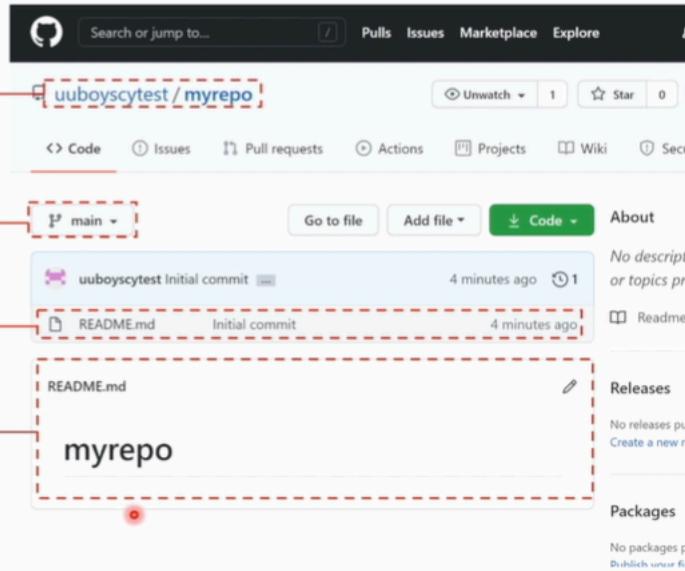
repository 相關操作

使用者帳號及 repository 名稱

預設分支，後續可在建立其他分支

repository 中的檔案

若 repository 中有 README.md，會自動顯示於此，此檔案通常用於解說 repository 的用途，以 markdown 語法撰寫



The screenshot shows a GitHub repository page for 'uuboyscystest/myrepo'. The top navigation bar includes links for 'Pulls', 'Issues', 'Marketplace', and 'Explore'. Below the header, there's a search bar and a red box highlights the 'Add file' button. The main content area shows a commit from 'uuboyscystest' titled 'Initial commit' made 4 minutes ago. A red box also highlights the 'Code' dropdown menu. On the right side, there are sections for 'About' (with a note about no description), 'Readme' (with a link to the file), 'Releases' (with a note about no releases published), and 'Packages'.

**複製別人的專案到自己的 repository
也可用於參與別人的專案**

```
[//]: # (column_list is not supported)
```

```
[//]: # (column is not supported)
```

[//]: # (column is not supported)

二、創建檔案

點選 Add file，建立一個 python 程式

[//]: # (column_list is not supported)

[//]: # (column is not supported)

[//]: # (column is not supported)

>在最上面的Create test.py 中輸入「在這個版本中做了什麼事情，讓以後可以了解，也可以使用系統預設的 commit message」

>勾選 Commit directly 可以將我們的修改，直接 Commit 到我們的 main 分支

勾選 Create a new branch 就會長出新的分支

>點選 Commit new file 就會跳回我們的專案資料夾頁面

可以看到紅色圈圈的位置，是我們目前版本的版本狀態，藍色圈圈則顯示，那個資料最近被異動後的版本狀態

除了自己創建，還可以複製別人的 repository

- 【Git】基礎 Merge main 步驟

- ## 1. Update the dev branch

Check out the dev branch >> Add all programs >> Fill in commit line >> Push >>

- ## 2. Update the main branch

Check out to the main branch >> Fetch and pull

- ### 3. Merge the main to the dev branch

Check out the dev branch >> Select the main branch and merge main to the current branch >> Fill in the commit line >> Push>>

- #### 4. Merge back the dev to the main branch

Check out the main branch >> Select the dev branch and merge the dev to the main branch >> Fill in the commit line >> Push>>

A branch merge to main branch flow:

1. Check out to the main branch 「git checkout main」
 2. Pull the main branch file 「git pull origin main」
 3. Merge A to the main branch 「git merge A」
 4. Fix conflict and combine 「git add .」 + 「git commit -m "Fix conflict"」
 5. Push to the main branch 「git push origin main」

- 【Github SSH Key 前置建構】



schleswigerstrasse 1000-Quellenkataloge@unibamberg.de

輸入完後會出現以下訊息內容，表示 Git Bash 幫我們自製作一把鑰匙以及預設路徑，名子在紅色處，預設路徑為藍色 會訊問我們要把鑰匙存在哪個路徑，我們可以使用預設的即可 按下 `Enter`

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
$ ssh-keygen -t ed25519 -C 'sam.huang.veda@gmail.com'
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Tibame_EX14/.ssh/id_ed25519):
```

一樣不用輸入，按下 Enter 即可，就會出現講義上的畫面，表示鑰匙製做完成了

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
$ ssh-keygen -t ed25519 -C 'sam.huang.veda@gmail.com'
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Tibame_EX14/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
```

在Git Bash 中輸入「ls ~/.ssh」就可以在我們隱藏的ssh目錄中，看到兩把鑰匙

第一把是私鑰(解鎖)，第二把是公鑰(上鎖)

```
|o.+ . B B ..o =
|.=.....@o. ....oo|
O OO.O.+=OO.. .
+---[SHA256]---+
```

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~
$ ls ~/.ssh
id_ed25519 id_ed25519.pub
```

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~
$
```

在Git Bash 中輸入「eval ssh-agent -s」

這個命令的用途是啟動 ssh-agent，並將 ssh-agent 的環境變數加入到當前 shell 的環境變數中

ssh-agent 是一個用來管理 ssh 私鑰的工具，它可以幫助您在使用 Git 和其他 SSH 相關命令時，不需要每次都輸入密碼或身份驗證

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
$ ls ~/.ssh
id_ed25519 id_ed25519.pub known_hosts
```

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
$ eval `ssh-agent -s`
Agent pid 709
```

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
```

在Git Bash 中輸入「ssh-add ~/.ssh/id_ed25519」

這個命令的用途是將我們的鑰匙加入到管理 ssh 私鑰的工具

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 ~
$ ssh-add ~/.ssh/id_ed25519
Identity added: /c/Users/Tibame_EX14/.ssh/id_ed25519 (sam.huang.veda@gmail.com)
```

最後將我們的 github 上鎖

在Git Bash 中輸入「cat ~/.ssh/id_ed25519.pub」

先將公鑰 cat 出來，如下方選取處，然後複製起來

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBuFHroew5gr+GTxt/0+ZepygINV5FB6ueQh2ejo4Vy5 aegi
s123214@gmail.com
```

回到 Github 網頁，進到 setting 中，接著依據下面截圖的位置，貼上我們的公鑰，就可以



Search or jump to...

Pull requests Issues Marketplace Explore

uuboyscytest / myrepo

Unwatch 1

Signed in as
uuboyscyte

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Set stat

main

1 branch

0 tags

Go to file

Add file

Code

uuboyscytest Create test.py

b57c4f6 7 days ago 2 commits

README.md

Initial commit

7 days ago

test.py

Create test.py

7 days ago

README.md

Help
Windows
Settings

Security & analysis

Emails

Notifications

SSH and GPG keys

Repositories

Packages

Organizations

Saved searches

and uni

Bio

Tell i

You can

URL

Twitte



uuboyscytest

Your personal account

Go to your personal profile

Account settings

Profile

Account

SSH keys

New SSH key

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot common SSH problems.

uuboyscytest

Your personal account

Go to your personal profile

Account settings

Profile

Account

Appearance

Account security

Billing & plans

Security log

Security & analysis

SSH keys / Add new

Title

uboytest

Key

ssh-ed25519 AAAAC3NzaC1I2DI1NTESAAAAI~~BuFHroew5qr+GTXt/0+ZepyglNV5FB6ueQh2ejo4Vv5~~
aegis123214@gmail.com啟用 V
移至頂部

Title 可以為鑰匙取名子 · Key 輸入我們的公鑰的資料



Confirm access

Password

Forgot password?

Confirm password

Tip: You are entering sudo mode. We won't ask for
your password again for a few hours.

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

 uuboystest	SHA256:TV94thevc923oIeyjsqdgz20meHhE4Loz40jFKLHTbE Added on 13 Jun 2021 Never used — Read/write	<button>Delete</button>
--	---	-------------------------

Check out our guide to generating [SSH keys](#) or troubleshoot common SSH problems.

在Git Bash 中輸入「`ssh -T git@github.com`」確認是否綁定成功，看到下面的文字表示成功

```
MINGW64:/c/Users/UUBOY
UUBOY@DESKTOP-ED4V07M MINGW64 ~
$ ssh -T git@github.com
The authenticity of host 'github.com (52.192.72.89)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (RSA) to the list of known hosts.
Hi uuboyscytest! You've successfully authenticated, but GitHub does not provide shell
access.

UUBOY@DESKTOP-ED4V07M MINGW64 ~
$
```

啟用 Windows
移至 [設定] 以啟用 Window

- 【Git 版本控制】

- 【Git Bash 指令】

基本上 Git Bash 與 Linux 的語法相通，所以會使用這些指令進行操作

常用終端機指令



指令	用途	範例
cd path	移動到 path 目錄	cd /tmp (移動到 /tmp)
echo "xxx" > file	新建一檔案名為 file，並將 xxx 寫入該檔案	echo "test123" > test.txt
mkdir tmpdir	創建 tmpdir 目錄	mkdir /tmp/test
mv path1 path2/	將 path1 檔案或目錄移至 path2 裡	mv /tmp/file /tmp/test
mv old new	將 old 改名為 new 若 new 非資料夾	mv /tmp/file /tmp/test
rm file	將 file 檔案刪除，無法刪除資料夾	rm /tmp/file (將 /tmp/file 刪除)
rmdir folder	將 folder 目錄刪除，folder 須為空目錄	rmdir /tmp/test
rm -r folder	將 folder 目錄連同裡面的目錄及檔案一起刪除	rm -r /tmp/test (將 /tmp/test 及其子目錄刪除)
pwd	查看所在目錄	
ls -la	列出當前目錄檔案資訊，包括隱藏檔案	

預設通常會是 C 槽的 User，如果想要進到 D 槽，可以輸入「cd D:」

【創建檔案】

在Git Bash 中輸入「echo “隨便打意思文字，會是創建檔案的內容” > 檔名.副檔名」>> 「echo “123” > test.txt」

目錄下就會出現一個檔案



【創建資料夾】

在Git Bash 中輸入「mkdir 資料夾名稱」>> 「mkdir testdir」

【檔案移動】

將我們建好的 test.txt 檔案移動到 testdir 資料夾中

在Git Bash 中輸入「mv test.txt testdir/」

本機 > Data (D:) > git_package > testdir



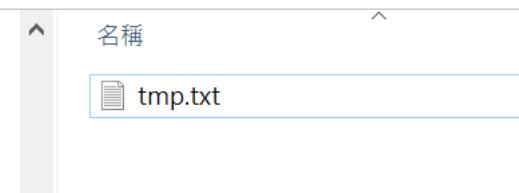
【查看所在目錄下的檔案】

在Git Bash 中輸入「ls」

【更改檔案名稱】

在Git Bash 中輸入「mv test.txt 想更改的名稱.txt」>>「mv test.txt tmp.txt」

Data (D) > git_package > testdir



【刪除檔案】

在Git Bash 中輸入「rm tmp.txt」

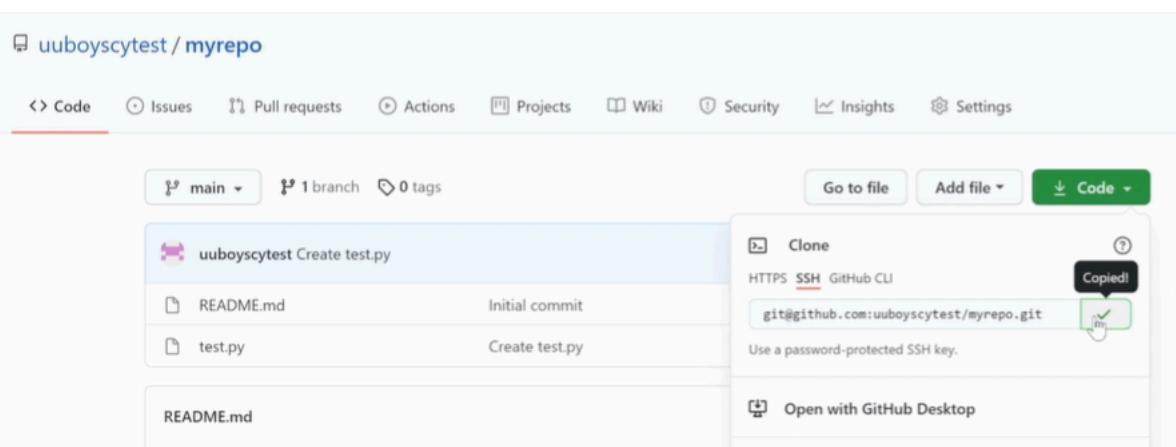
【刪除目錄(資料夾)】

需要先回到上一層，>在Git Bash 中輸入「cd ..」

在Git Bash 中輸入「rmdir testdir」

- 【Github repository 下載】

步驟一、複製SSH

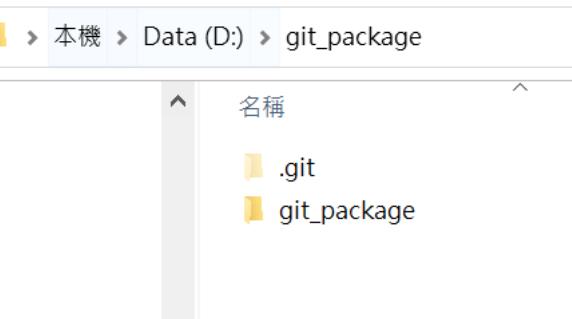


The screenshot shows a GitHub repository page for 'uuboyscytest / myrepo'. The 'Code' dropdown menu is open, showing options for 'Clone' via HTTPS or SSH. The SSH option is selected, and a message 'Copied!' with a clipboard icon is displayed next to it.

步驟二、貼到 Git Bash 中

```
$ cd ~/Desktop/  
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop  
$ git clone git@github.com:uuboyscytest/myrepo.git  
V:\Desktop\myrepo 然後把我們剛剛複製的段文字貼上來
```

輸入後就會出現我們的資料夾了



進入路徑，就可以發現我們進入到 main 的分支了，會出現的原因是，Git Bash 會自動偵測我們是否有受到 git 的版本控制

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop  
$ cd myrepo/  
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)  
$
```

這個就是我們現在所在的分支

我們可以在Git Bash 中輸入「ls -la」，可以看到隱藏的資料夾，所有的版本、分支的資訊都會被記錄在這個資料夾中

```
Tibame_EX14@LAPTOP-3D1KNQNT8 MINGW64 /d/git_package/git_packing
$ ls -la
total 5
drwxr-xr-x 1 Tibame_EX14 197121 0 Mar 16 11:19 .
drwxr-xr-x 1 Tibame_EX14 197121 0 Mar 16 11:19 ..
drwxr-xr-x 1 Tibame_EX14 197121 0 Mar 16 11:19 .git/
-rw-r--r-- 1 Tibame_EX14 197121 13 Mar 16 11:19 README.md
```

接下來需要進行 **人員基礎設定**，才可以開始進行版本控制

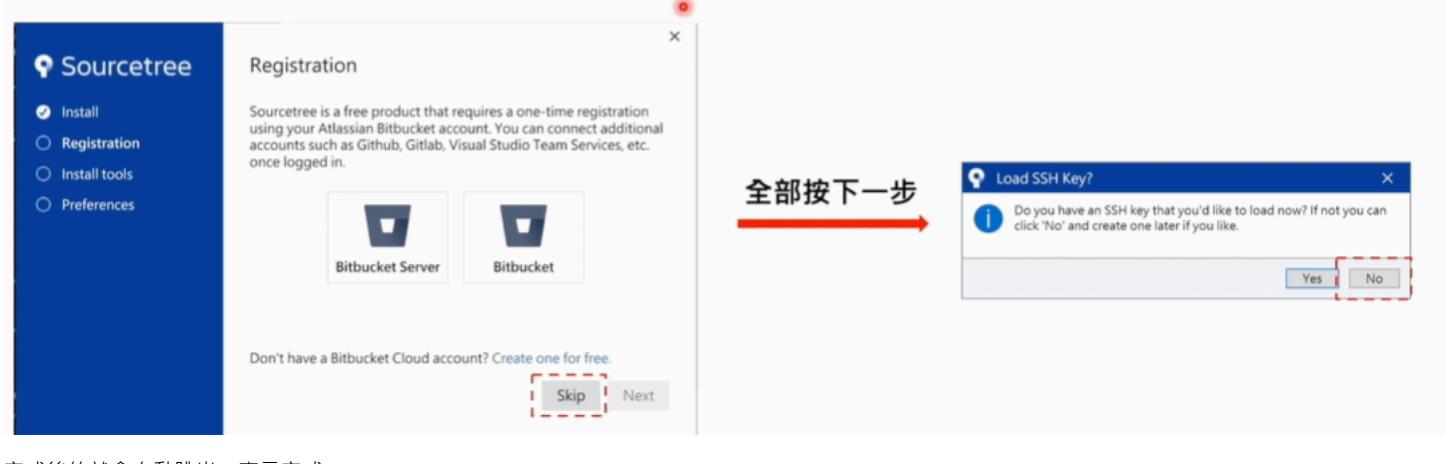
- 【設定開發人員基本資訊】

第一步、在Git Bash 中輸入兩段使用者資訊，這只是為了回溯、聯繫開發者用途，我會輸入 Github 上的使用者資訊：「git config --global user.name "FIRST_NAME LAST_NAME"」>>

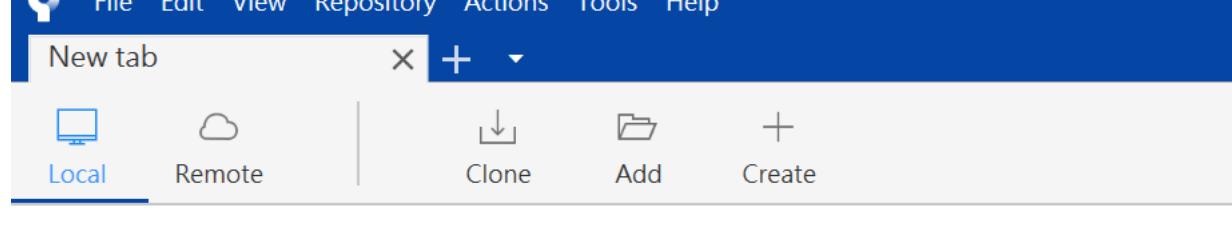
「git config --global user.name "samhuang95"」 「git config --global user.email "sam.huang.veda@gmail.com"」

第二步、安裝「Sourcetree」

主要有兩個需要注意



完成後的就會自動跳出，表示完成



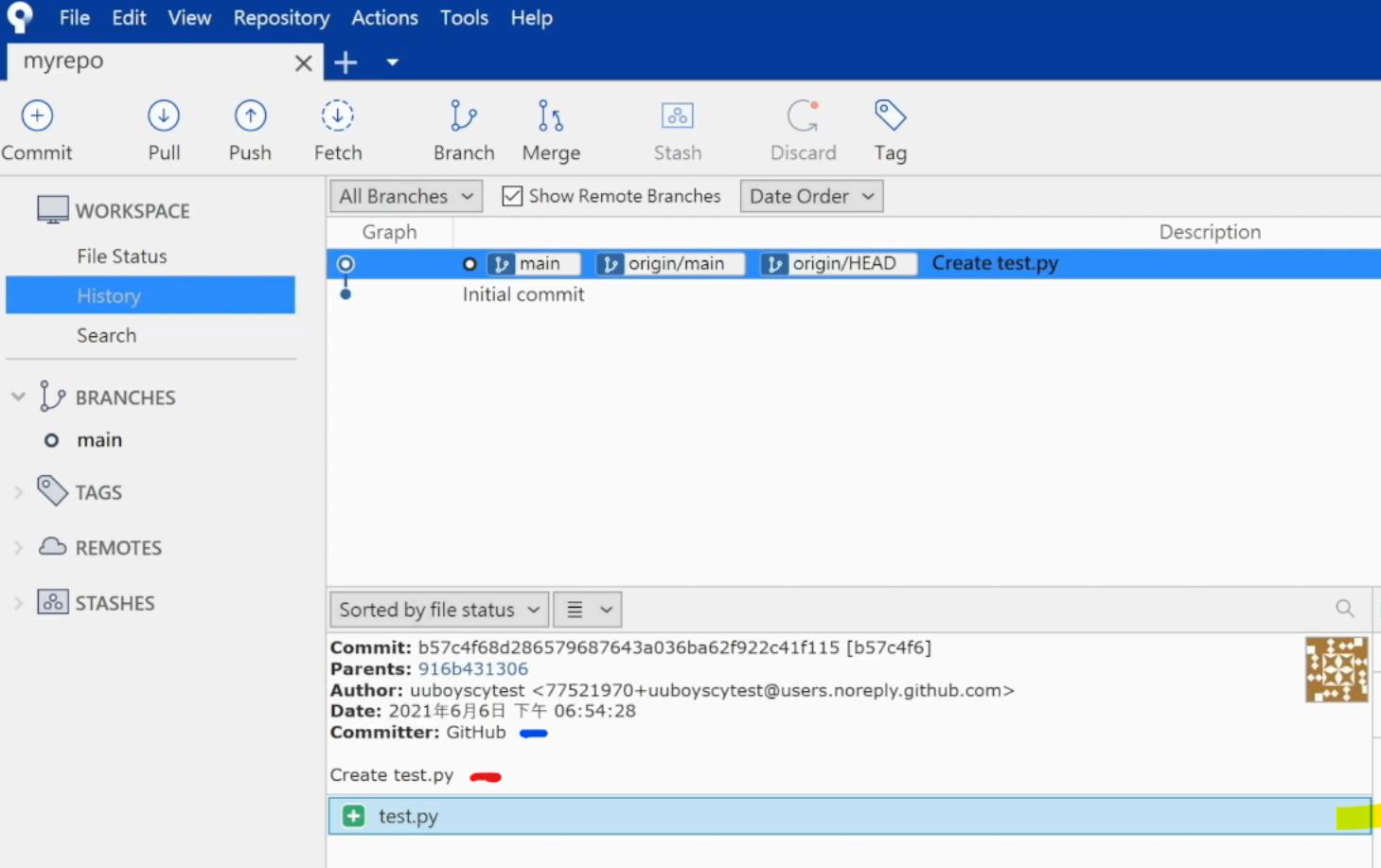
Local repositories

Search

Add a new bookmark or drag & drop repository folders into this area to begin

- 【開始版本控制】

>步驟一、開啟 Sourcetree 點選 Add (Add repository)，選擇我們專案資料夾，加入後，選擇 History 介面，可以看到版本的頁面 下圖說明備註：「藍色」表示用什麼編輯或是新增「紅色」表示編輯了什麼，我是我們在 Title 那邊寫的內容「黃色」編輯的檔案是修改了什麼樣的內容



>第二步、建立新的版本

先讓 Gitbash 移動到我們專案目錄下(main 分支)

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
```

了解版本狀態，在Git Bash 中輸入「git log」

會獲得 commit 節點，目前有兩個，第一個(紅色)表示 HEAD 現在所處於的版本狀態，HEAD=我們目前所在的地方 第二個(藍色)表示這是上一個版本的版本號

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git log
commit b69510f7f4a2904d7b3470a3ac5d90cb00270d3b (HEAD -> main, origin/main,
in/HEAD)
Author: samhuang95 <124756719+samhuang95@users.noreply.github.com>
Date:   Thu Mar 16 11:53:38 2023 +0800

    Create test.py

commit 36081b45d7e7e01b2187fcba60dc06e8922f4bcd
Author: samhuang95 <124756719+samhuang95@users.noreply.github.com>
Date:   Thu Mar 16 09:45:02 2023 +0800

    Initial commit

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$
```

了解版本狀態，在Git Bash 中輸入「git status」

如果要編輯 README 這個檔案，在Git Bash 中輸入「vi README.md」，就會進到 vim 的文字編輯畫面

接著在Git Bash 畫面中輸入「i」，不用按 ENTER 就可以進到 INSERT 模式，可以在裡面

新增一些文字，接著如果要離開，可以在Git Bash 畫面中按下「`esc`」然後在Git Bash 中輸入「`:wq`」，就可以進行儲存，並且關閉檔案，如此一來，我們已經完成檔案修改了

接著可以確認一下版本狀態，在Git Bash 中輸入「`git status`」，就會看到紅色的文字，顯示，`README.md` 已被修改，但是系統會提醒記得要將目前 `README.md` 檔案，加入到staging area(暫存區)

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)
$
```

readme.md這個檔案已經被修改了

步驟三、加入暫存區的方法

在Git Bash 中輸入「`git add README.md`」

這時候就會呈現綠色，表示這個檔案已被加入 staging area 了。

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)
$ |
```

現在會看到這個被修改過的檔案

步驟四、建立新版本的方法

接下來把這個狀態加為一個新的版本

在Git Bash 中輸入「git commit -m "雙引號中，會加入一段 commit message，用來表示我們這次的操作做了什麼事情"」>>「git commit -m "Modified README.md"」

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git commit -m "Modified README.md"
[main b4350e8] Modified README.md
 1 file changed, 9 insertions(+), 1 deletion(-)
```

這時候確認狀態「`git status`」，就會回到我們什麼都還沒有做的狀態。

這時候在Git Bash 中輸入「`git log`」，就會多出一個版本號，也就是我們最新的版本號(commit number)

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git log
commit b4350e88a3c6095c264170e4e0b93ca45038b73c (HEAD -> main) ✓ NEW
Author: samhuang95 <sam.huang.veda@gmail.com>
Date:   Thu Mar 16 14:31:28 2023 +0800

    Modified README.md

commit b69510f7f4a2904d7b3470a3ac5d90cb00270d3b (origin/main, origin/HEAD)
Author: samhuang95 <124756719+samhuang95@users.noreply.github.com>
Date:   Thu Mar 16 11:53:38 2023 +0800

    Create test.py

commit 36081b45d7e7e01b2187fcba60dc06e8922f4bcd
Author: samhuang95 <124756719+samhuang95@users.noreply.github.com>
Date:   Thu Mar 16 09:45:02 2023 +0800
```

我們重新整理 Sourcetree 就會出現我們的版本節點了

步驟五、批次加到 staging area

如果我們想要將很多編輯過的檔案，一次放到 staging area

模擬狀況:新增文件 > 檢視狀態 > 加入 > 檢視狀態 > 其中一個檔案不想要加入

新增文件、檢視狀態

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ echo '123' >> test1.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ echo '456' >> test2.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test1.txt
    test2.txt

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

加入 > 檢視狀態

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git add .
warning: in the working copy of 'test1.txt', LF will be replaced by CRLF th
t time Git touches it
warning: in the working copy of 'test2.txt', LF will be replaced by CRLF th
t time Git touches it

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  test1.txt
    new file:  test2.txt
```

其中一個檔案不想要加入> 檢視狀態

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git reset test2.txt

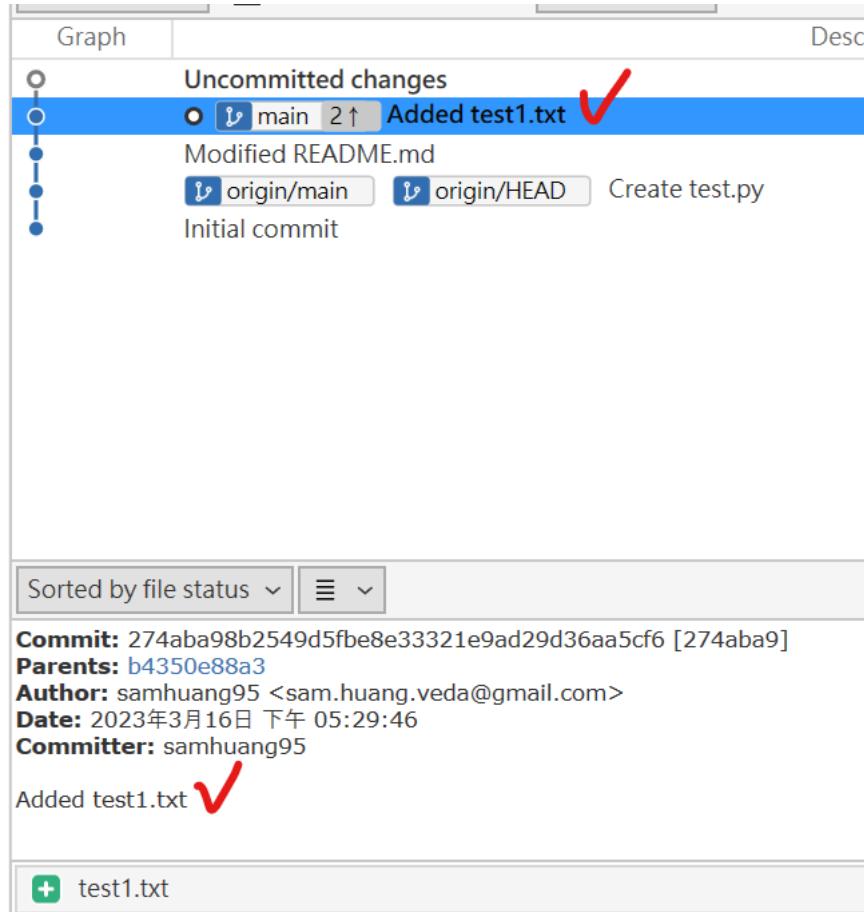
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  test1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test2.txt
```

確認新增

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git commit -m 'Added test1.txt'
[main 274aba9] Added test1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test1.txt
```



```
Tibame_EX14@LAPTOP-3D1KNQT8 MIN
$ git status
On branch main
Your branch is ahead of 'origin/main'
  (use "git push" to publish your changes)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test2.txt

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

Tibame_EX14@LAPTOP-3D1KNQT8 MIN
$
```

解決方法為將某檔案不要被 git 偵測到 在Git Bash 中輸入「vi .gitignore」>「i」，開始編輯文件 ignore 文件，輸入我們不想要被偵測的檔案名稱 >>「test2.txt」，之後就不會再次顯示出來，也不會將這個檔案加入到新的版本

如果我們想要某資料夾，不要被 git 偵測到一樣在 Git Bash 中輸入「`vi .gitignore`」>「`i`」，開始編輯文件 `ignore` 文件，輸入我們不想要被偵測的資料夾名稱+「`/`」>>「`target/`」也可以將一系列結尾相同的檔案，不要被 git 偵測到，在 `ignore` 文件中，輸入「`XXX`」>>「`.log`」這個範例就是將所有「`.log`」為結尾的檔案，設備不要被 git 偵測到

```
MINGW64:/d/git_package/gi  
test2.txt  
target/  
*.log|  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
gitignore[+] [unix] (07:59  
-- INSERT --
```

結束之後按下「**esc鍵**」退出編輯，輸入「**:wq**」儲存並退出檔案，這時候在Git Bash中輸入「**git status**」

最後只要在Git Bash 中輸入「git add .」加入所有檔案

並在Git Bash 中輸入「`git commit -m 'Add .gitignore'`」，就算是完成一次的版本新增了

。【版本切換】方法 1

範例教材製作：

```
echo '123' >> test1.txt
```

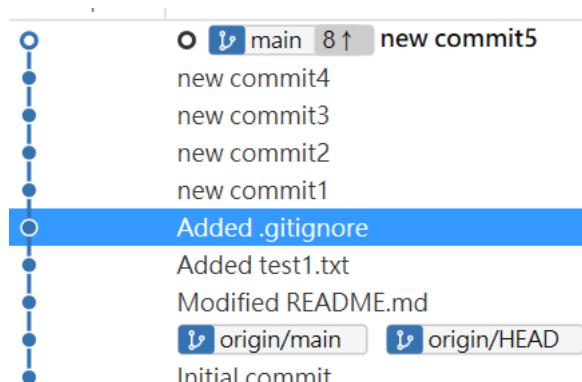
git add .

```
git commit -m "new commit1" echo '123' >> test1.txt
```

git add .

```
git commit -m "new commit2"
```

這幾段來創造 5 個版本



如果想要進行切換，就會需要先知道 commit number 是什麼，只要在 Git Bash 中輸入「`git log`」，就可以看到所有的 commit number。

```
MINGW64:/d/git_package/git_package

commit 9c94a7acc9f13fb59e3c44ac8515cb668a392b00 (HEAD -> main)
Author: samhuang95 <sam.huang.veda@gmail.com>
Date:   Thu Mar 16 19:06:54 2023 +0800

    new commit5

commit 037591b95c2915c386e2628f26f52d1f9aa07b87
Author: samhuang95 <sam.huang.veda@gmail.com>
Date:   Thu Mar 16 19:06:47 2023 +0800

    new commit4

commit 8d4c085fa2d18a11f7d33fa1acb64c829d028b49
Author: samhuang95 <sam.huang.veda@gmail.com>
Date:   Thu Mar 16 19:06:39 2023 +0800

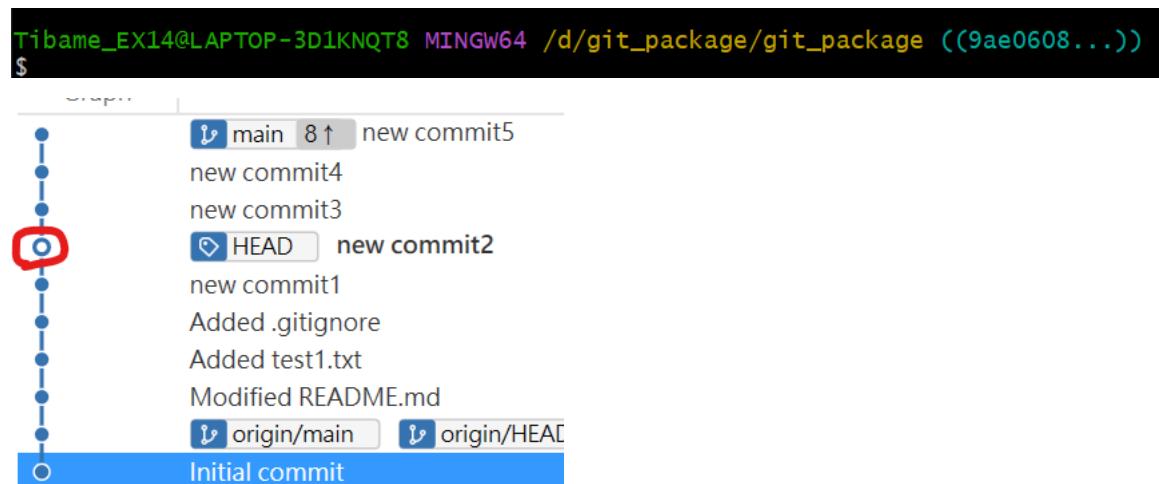
    new commit3

commit 9ae06082808cfecb774425e8ce1e2ce9ea02031cc
Author: samhuang95 <sam.huang.veda@gmail.com>
Date:   Thu Mar 16 19:06:29 2023 +0800

    new commit2
:
```

步驟二、切換方法1

複製想要切換的 commit number，在Git Bash 中輸入「q」離開畫面，在Git Bash 中輸入「git checkout **查詢出來的commit number**」>>「git checkout 9ae0608.....」這時候就會出現我們的版本號在旁邊了



補充:

其實不需要這麼完整的輸入全部的 commit number，可以使用比較短的版本，

在Git Bash 中輸入「`git log --oneline`」，就可以看到簡短的版本號。使用上與上一步一樣「`git checkout 9ae0608`」。

步驟三、切換回 main(最新狀態的版本)

這一步就比較簡單了，只需要在Git Bash 中輸入「`git checkout main`」，就完成了。

- ### ◦ 【版本切換】方法 2

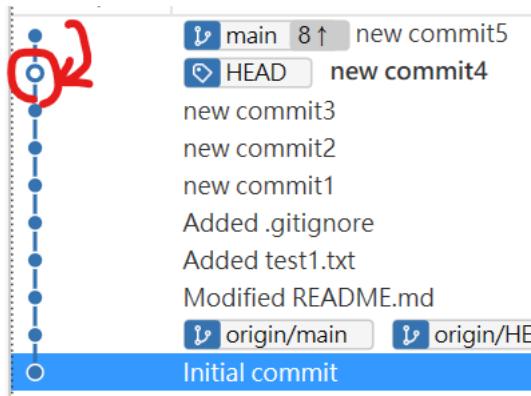
步驟一、先確定目前所在位置

在Git Bash 中輸入「`git log --oneline`」，了解目前的位置

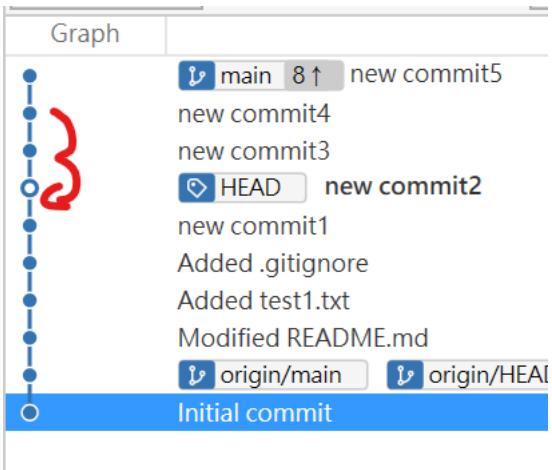
```
Tibame_EX14@LAPTOP-3D1KNQT8 M
$ git log --oneline
9c94a7a (HEAD -> main) new commit4
037591b new commit4
8d4c085 new commit3
9ae0608 new commit2
434e447 new commit1
4c048fe Added .gitignore
274aba9 Added test1.txt
b4350e8 Modified README.md
b69510f (origin/main, origin/
36081b4 Initial commit
```

步驟二、切換版本

在Git Bash 中輸入「`git checkout HEAD~`」，可以切換到 HEAD 所在的前一個版本



如果想要切換到 HEAD 所在的前兩個版本，在Git Bash 中輸入「`git checkout HEAD~2`」



◦ 【分歧】

如果我們退回去的版本，是尚未有修改到 working directory (加入 .gitignore 文件)，就會出現分支

\$ git checkout HEAD~2
Previous HEAD position was 81456f8 new commit 4
HEAD is now at 196debd new commit 2

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo ((196debd...))
\$ git checkout HEAD~3
Previous HEAD position was 196debd new commit 2
HEAD is now at cb92491 Added test1.txt

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo ((cb92491...))
\$ git status
HEAD detached at cb92491
Untracked files:
 (use "git add <file>..." to include in what will be committed)
 test2.txt

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

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo ((cb92491...))
\$ |

Date	Commit	Description
19 六月 2021	main	new commit 5
19 六月 2021	new commit 4	
19 六月 2021	new commit 3	
19 六月 2021	new commit 2	
19 六月 2021	new commit 1	
19 六月 2021	Added .gitignore	
19 六月 2021	Modified README.md	
19 六月 2021	origin/main	origin/HEAD
6 六月 2021	Initial commit	

- 【Git 上傳到 Github】

- 【單一分支】

本地 Git 上傳到 Github

在Git Bash 中輸入「`git push origin main`」 「`origin main`」表示會指定將目前本地端的 repository(存儲庫，我為:D:\git_package\git_package 資料夾) · 並指定分支的所有版本狀態上傳到 Github

```
use git push to publish your local commits

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git push origin main
Enumerating objects: 26, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 16 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (24/24), 1.84 KiB | 1.84 MiB/s, done.
Total 24 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), done.
To github.com:samhuang95/git_package.git
  b69510f..9c94a7a main -> main
```

- 【多個分支】新增分支

第一步、呈現分支

在Git Bash 中輸入「`git branch`」會呈現出我們所有的分支(main)

```
MINGW64:/c/Users/UUBOY/Desktop/myrepo
UUBOY@DESKTOP-ED4VO7M M
$ git branch
* main
UUBOY@DESKTOP-ED4VO7M M
$ |
```

第二步、新增分支

新增一個分支在Git Bash 中輸入「`git branch` 我們的分支名稱」>>

「`git branch new-branch`」 · 可以再次輸入「`git branch`」查詢我們所有的分支 · `main`前面的「*」表示我們現在所處的分支位置

```
Tibame_EX14@LAPTO
$ git branch
* main
  new-branch
```

第三步、切換分支

在Git Bash 中輸入「`git checkout new-branch`」 · 再次輸入「`git branch`」查詢我們所處的分支

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (new-branch)
```

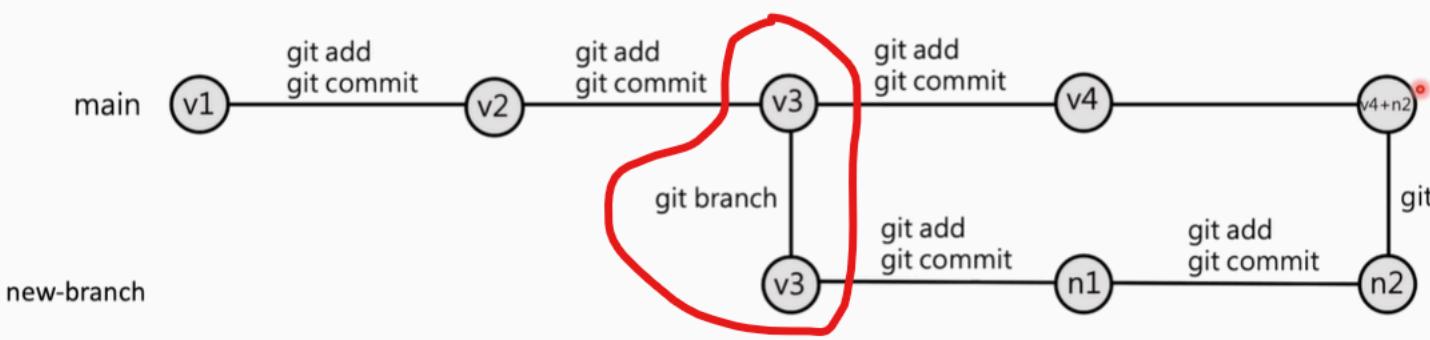
在 Sourcetree 顯示 · main & new-branch 的節點都會在同一個位置

The screenshot shows the Sourcetree interface. On the left, there's a sidebar with 'WORKSPACE', 'File Status', 'History' (which is selected and highlighted in blue), 'Search', and sections for 'BRANCHES', 'TAGS', 'REMOTES', and 'STASHES'. The 'BRANCHES' section lists 'main' and 'new-branch'. The 'new-branch' entry is circled in red. The main pane displays a 'Graph' view of the commit history. At the top of the graph, there are buttons for 'All Branches', 'Show Remote Branches', and 'Date Order'. Below the graph, the commits for the 'new-branch' are listed: 'new commit4', 'new commit3', 'new commit2', 'new commit1', 'Added .gitignore', 'Added test1.txt', 'Modified README.md', and 'Create test.py'. A blue bar highlights the first commit, 'Initial commit'. The 'main' branch is also shown with its commits.

步驟

- 【多個分支】分支操作

目標是創造紅色圈起的部分



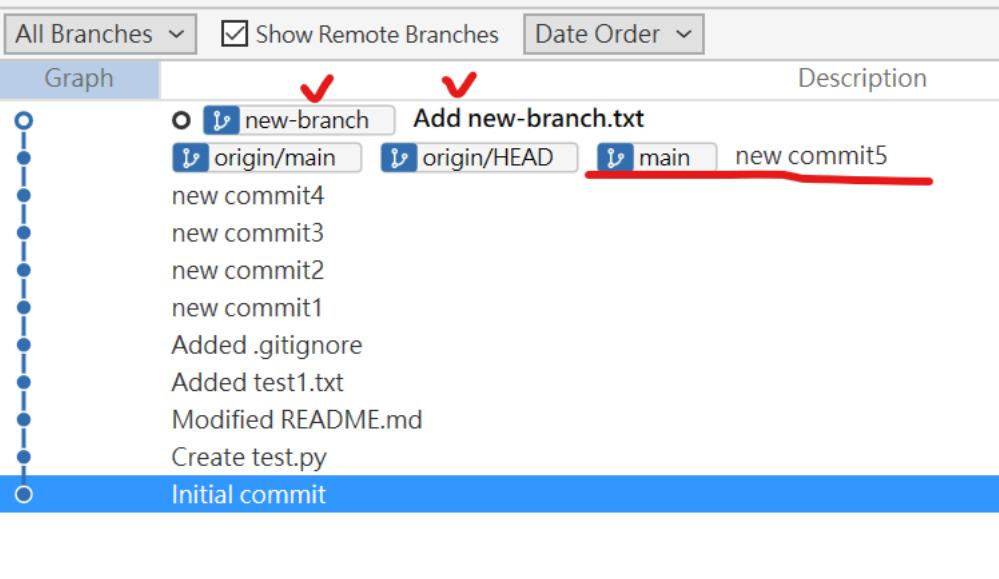
在分支中心新增文字檔 · 範例教材製作:

```
echo '123' >>new-branch.txt
```

```
git add .
```

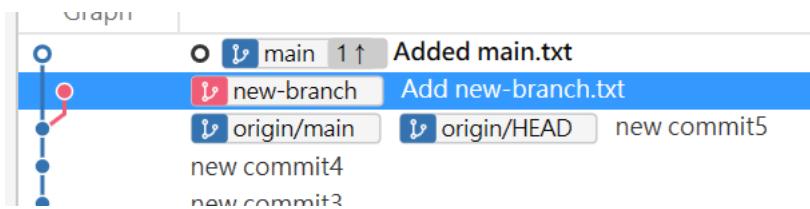
```
git commit -m "Add new-branch.txt"
```

這時候就會在 sourcetree 中看到我們的新增以及目前版本位置



這時候先切換回 main 然後做一次新增檔案 > 加入 > commit

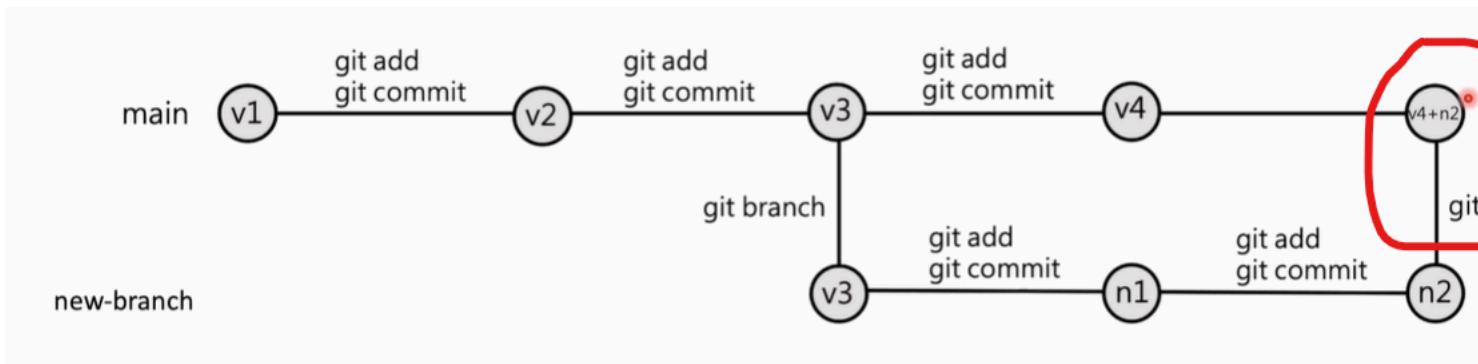
就會在 Sourcetree 中發現 · main 與 new-branch 的走向出現分歧



未來我們要製作專案或是與其他人共識的時候，這樣的操作就很重要！

◦ 【合併分支 Merge】

目標是創造紅色圈起的部分



假設今天我們 new-branch 的開發已經完成，並且需要 Merge 回我們的 main 分支

步驟一、切回 main 分支

在Git Bash 中輸入「git checkout main」回到 main 分支，然後可以在Git Bash 中輸入「ls」確定一下資料夾中的所有資料

步驟二、合併

在Git Bash 中輸入「git merge 分支名稱」>>「git merge new-branch」

這時候會出現一個可以輸入 commit message 的地方

 MINGW64:/d/git package/git package

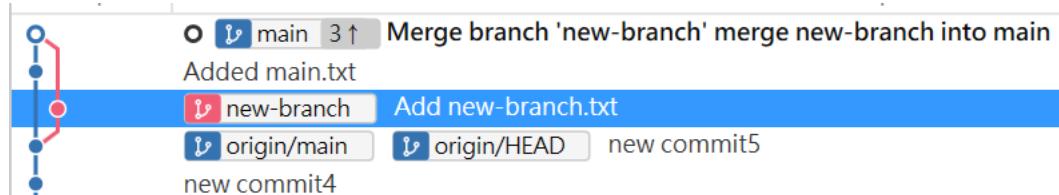
輸入的方式為：移動到最後一行，然後輸入我們修改的內容，然後按下按鍵「esc」，然後輸入「:wq」儲存、退出。

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git merge new-branch
Merge made by the 'ort' strategy.
 new-branch.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 new-branch.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ |
```

只要出現這個畫面，就表示成功 merge 了步驟三、確認

在Git Bash 中輸入「`ls`」確認一下資料夾的資料狀態，並確認 Sourcetree 上的狀態是否合併成功。



。【合併分支衝突解決】conflict

■ 【教學前置】

為了示範我們會先創造一個衝突，以下操作為快速複習用

conflict

- 若兩個不同的分支同時修改到同一個檔案，則可能產生衝突
 - 先試著建立兩個分支，test1 及 test2，並同時修改 README.md
 - 分別將兩分支都建立新版本

```
# myrepo
add something123
~  
~  
~  
~  
~  
README.md [+] [unix] (22:18 01/02/2021)
-- INSERT --
```

```
MINGW64:/c/Users/uuboyscy/Desktop/myrepo
# myrepo
add something456
~
~
~
README.md [+] [dos] (00:28 02/02/2021)
-- INSERT --
```

在Git Bash 中輸入「git branch」，查看目前有哪些分支

在Git Bash 中輸入「`git branch test1`」，創建新的分支

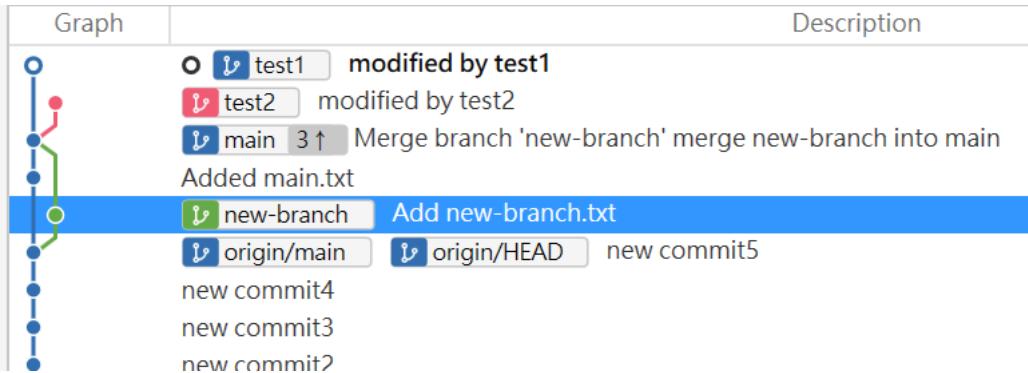
方法二、在Git Bash 中輸入「`git checkout -b test2`」，可以在創建的同時，進入到新創的分支中

對 test2 中的 README.md 進行修改，在Git Bash 中輸入「`vi README.md`」，然後隨便在一個地方修改，但因為我們是要來測試分支衝突圖的，所以需要記住修改的位置，因為會需要將一樣的修改送到 test1 中。

然後一樣 add .、commit 後，test2 就會是新的版本

完成後我們在 Sourcetree 中就會看到以下畫面

接下來移動到 test1 的分支，並且重複上方的步驟



接下來我們會依序將 test1、test2 合併回 main

先切回 main 分支，然後使用 merge 將 test1 合併，到了這一步會發先與之前 new-branch 不同，沒有出現需要輸入 commit message 的畫面，因為 test1 與 main 的節點分支的走向其實一樣，所以不需要輸入(這部分感覺是需要一段時間的經驗才會了解的)

接下來繼續 merge test2

這時候會出現下面的文字內容表示這是衝突，而我們的分支，也會變成暫時的節點

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)
$ git merge test2
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the
result.

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main|MERG
TNG)
$
```

【解決衝突】

先在Git Bash 中輸入「`git status`」確認一下檔案狀態，會出現一個兩個人同時修改 `README.md` 這個檔案

```
UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main|  
ING)  
$ git status  
On branch main  
Your branch is ahead of 'origin/main' by 4 commits.  
(use "git push" to publish your local commits)  
  
You have unmerged paths.  
(fix conflicts and run "git commit")  
(use "git merge --abort" to abort the merge)  
  
Unmerged paths:  
(use "git add <file>..." to mark resolution)  
 both modified: README.md
```

這時候就會需要協調「哪一個檔案要修改」或是「哪一個留下來」或是「兩個同時留下來」

以下操作會是將**特定分支留下來**

步驟一、

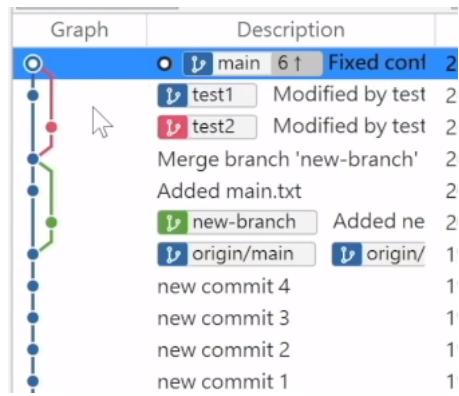
在Git Bash中輸入「`vi README.md`」，會看到一些特殊的分隔符號，會呈現出修改的內容以及哪一個分支產生的資訊，這時候只要將有衝突的資料刪除或是修改，就可以按下「`esc`」、「`:wq`」結束並儲存編輯

[//]: # (column is not supported)

這時候在Git Bash 中輸入「git add .」、commit，狀態就會回到我們的 main

UUBOY@DESKTOP-ED4VO7M MINGW64 ~/Desktop/myrepo (main)

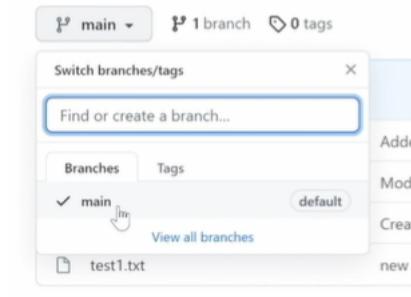
在 Git Bash 中輸入「`cat README.md`」將檔案中的文字 show 出來，會出現我們修改完成的結果，這時候 Sourcetree 的狀態也會回到正常的狀態了。



■ 【上傳分支】

目標是刪除我們地端以及雲端 Github 上的分支

先到 Github 專案的 repository，點開 main 會發現我們剛剛建立的所有 branch 都沒有在上面那些分支，都還沒有 push 到 Github 上，所以要先做 push 的動作



【push 的方法】先上傳版本狀態

在Git Bash 中輸入「`git push origin 分支名稱`」>>「`git push origin main`」，就會成功把 `main` 目前的版本狀態 push 到 Github 上，會出現我們目前所有的檔案

samhuang95 Merge branch 'test2'		...
	.gitignore	Added .gitignore
	READMD.md	modified by test1
	README.md	modified by test2
	gitignore	Added .gitignore
	main.txt	Added main.txt
	new-branch.txt	Add new-branch.txt
	test.py	Create test.py
	test1.txt	new commit5

先上傳分支

「git push origin 分支名稱」>>「git push origin new-branch」· 把 new-branch 這個分支 push 到 Github 上

The screenshot shows the GitHub repository interface. At the top, there are three status indicators: 'main' (selected), '2 branches', and '0 tags'. Below this is a search bar with the placeholder 'Find or create a branch...'. Underneath the search bar are two tabs: 'Branches' (selected) and 'Tags'. The 'Branches' tab shows two entries: 'main' (marked with a checkmark) and 'new-branch'. The 'new-branch' entry is highlighted with a red oval. At the bottom of the list is a blue link 'View all branches'.

接下來用一樣的方式將 test1、test2 上傳

■ 【刪除分支】

在Git Bash 中輸入「git branch -d 想刪除的分支名稱」>>「git branch -d new-branch」

完成後，可以在Git Bash 中輸入「git branch」，會發現我們的 new-branch 分支已經被刪除了，但 Github 上還沒有被刪除

刪除 Github 上的 branch

方法一、網頁上操作 · 請依照截圖步驟，就可以完成

main ▾ 4 branches 0 tags

Your main branch isn't protected
Protect this branch from force pushing or deletion, or require

samhuang95 Merge branch 'test2' ...

	.gitignore	Added .gitign
	README.md	modified by t
	README.md	modified by t

test1 Updated 41 minutes ago by samhuang95

new-branch Updated yesterday by samhuang95

方法二、git 指令

在Git Bash 中輸入「git push origin :想要刪除的分支名稱」>>「git push origin :test1」，

原理是「`git push origin :test1`」這段指令等於「`git push origin test1:test1`」，我將本地端的 `test1` 分支上傳到 Github 的 repository 內，並且更新我們的 `test1` 分支，可以嘗試輸入「`git push origin test1:test1`」看看，這時候 `test1` 又會回來，

所以如果我們是輸入「`git push origin :test1`」，就表示我們用空的分支來更新遠端的 `test1` 分支

。【分支多人開發】

■ 【分支名稱、說明】

1. **develop:** 盡量不會作為 commit、branch，狀態也會在已經可以提供做為測試了而所有開發人員也會在 develop 下，創建一個 feature 分支來開發自己的新功能，確定沒有問題後，才會 merge 回 develop，所以 develop 通常是開發程式的基礎，所以 develop 分支，通常不會包含開發中或是不完整的程式。
 2. **feature:** 為了建立新功能而建立的分支，建立好後，會 merge 回 develop，通常命名方式為「feature/function_name or function description」。
 3. **master/main:** 比較早期的專案會是 master，但因為有黑奴問題，所以現今的都改為 main 分支，main 版本通常是穩定、可上線給 user 使用的版本，開發者也不會直接對 main 進行 commit，而是只能從 develop 或 release merge。
 4. **release:** develop 開發到夠穩定、符合客戶要求後，會先 merge 到 release，通常 release 是上線最後測試的分支版本，沒有問題會 merge 到 main。如果上線過程中出現 bug 或是需求調整，然後專案也比較緊急的話，會先在 release 分支，開一個 hotfix 分支來進行修改，修復完成後會先 merge 到 develop，然後同時合併到 main。

5. **hotfix**: 穩定版本出現緊急問題，才會使用從 main 或是 release 開一個 hotfix 分支來來處理，解決完成後會同時 merge 到 develop、main 需要 merge 回 develop 的原因是，這個 bug 在開發階段不會出現，只有在上線後才可能出現，所以下次版本更新的時候，還是可能會出現

▪ 【Git Flow 開發流程】

什麼是 Git Flow



為解決以下問題而產生的開發流程

- 團隊內每個人的 commit 習慣不同
- 修正過的 bug 在 merge 後又出現
- 分支過於雜亂



參考文獻

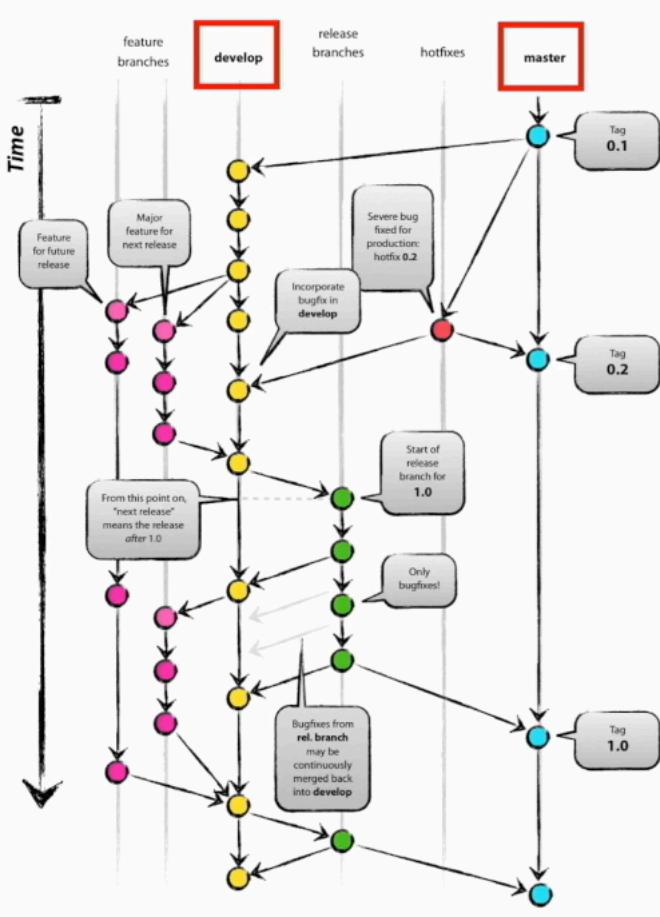
<https://nvie.com/posts/a-successful-git-branching-model/>



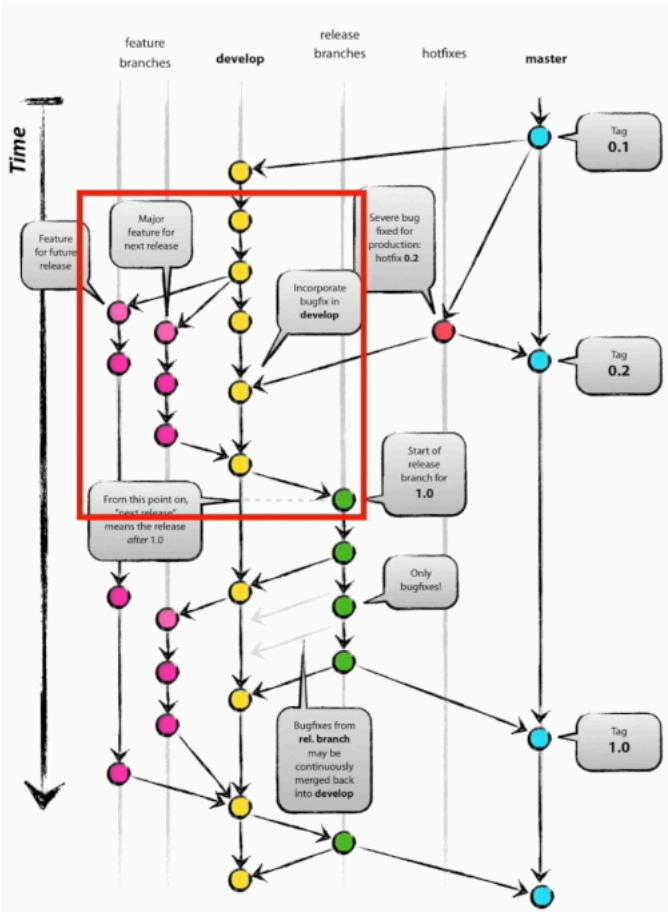
除了解決雜亂問題，也可以清楚知道**功能如何被建立**出來

【Git Flow 開發原則、流程】

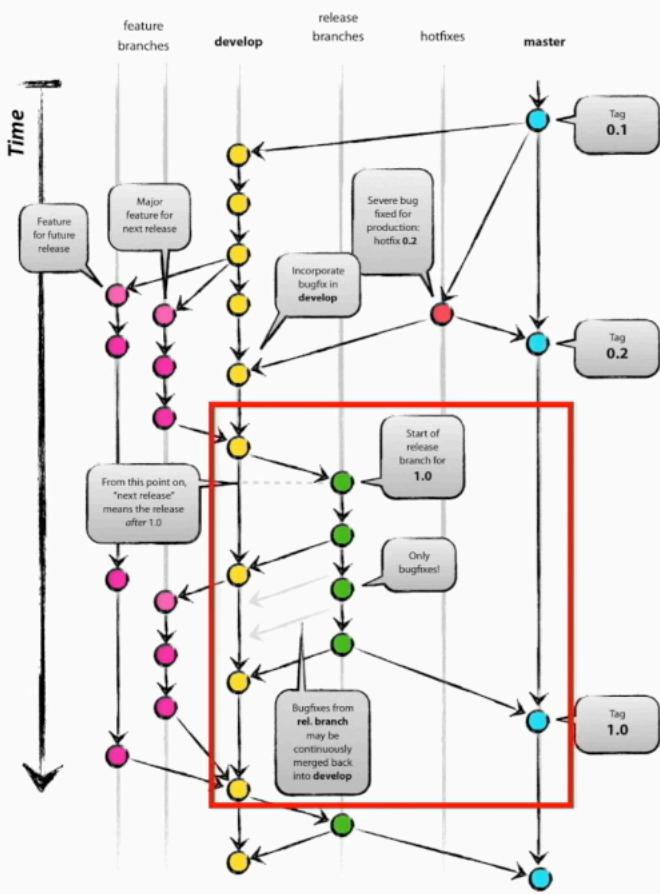
1. 除了 develop、main 分支屬於**常駐性質**，其他分支在任務開發或 debug 完成後，就會刪除



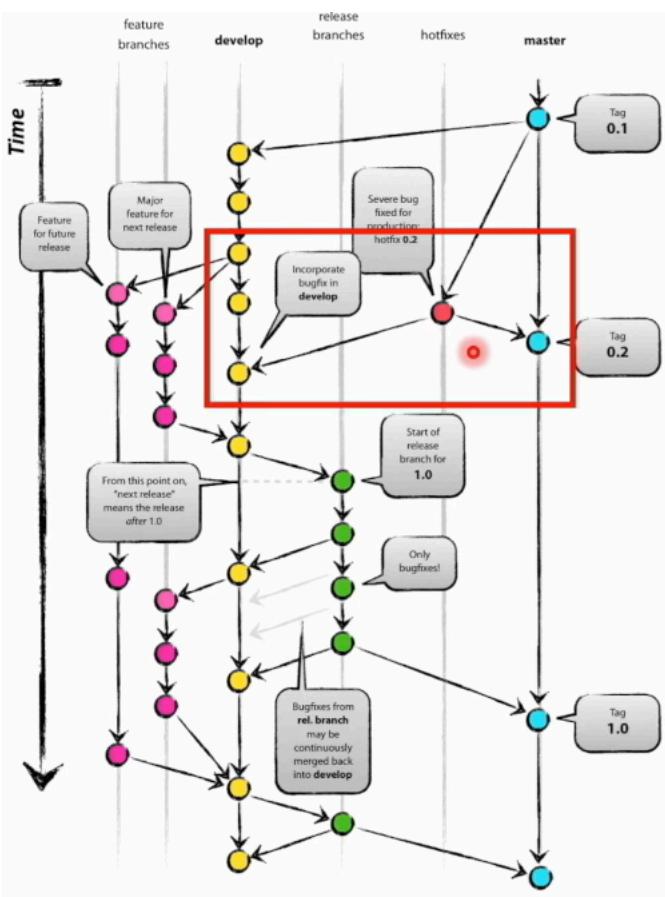
2. 如果專案業開發新的功能，都必須從 develop 分支創建 feature 分支來進行開發功能，開發完成後 merge 回 develop，feature 任務結束就刪除
通常命名方式為「feature/function_name or function description」



3. 產生測試、上線，通常會將 develop merge 到 release 做最後測試 release 只能修復 bug 不能添加 function，如果發現 bug，必須要創建 hotfix 修復 bug，然後 merge 回 release，最後 release 測試沒有問題後，會 merge 回 main、develop 分支，然後 release 也會刪除



4. 上線期間，緊急修復 bug，會在 main 分支創建 hotfix，然後修復完成後會 merge 回 main、develop



其他流程:GitHub Flow、GitLab Flow

▪ 【GitHub 多人開發】push

教材呈現，右邊要參與左邊的開發，所以會使用**右邊的參與者**進行操作

Fork

- 利用 fork 參與其他專案

The screenshot shows a GitHub repository page for 'uuboyscy/testfork'. A red arrow points from the top right towards the 'Fork' button in the header. The 'Fork' button is highlighted with a red dashed border. The repository name 'uuboyscy/testfork' is at the top left. The header also includes 'Pulls', 'Issues', 'Marketplace', 'Explore', and other navigation links. Below the header, there are sections for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', and 'About'. The 'About' section notes 'No description, website, or topics provided.' The 'Code' section shows a file named 'README.md' was created 6 minutes ago. The 'Releases' and 'Packages' sections both state 'No releases published' and 'No packages published' respectively.

參加者進到專案持有者的頁面，按下「Fork」後，會轉跳回自己的帳號頁面，並同時複製整個 repository(clone)，接下來操作流程如下：

- 將 fork 過來的專案 clone 到本地端
- 開新分支
- 開始開發新功能

開新分支與之前的操作一樣，只是名稱會參考上面的主流 [分支名稱](#)

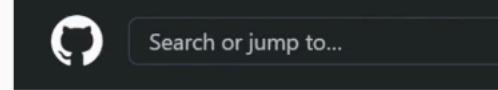
【開發完成後】在地端 push 回自己的 GitHub 然後在 [GitHub 上 pull request](#) 提出合併請求

- 新功能開發完畢
- 先將新分支 push 至自己的 GitHub
- 使用 pull request 對原專案擁有者提出合併請求

【push 完成後】會看到下方頁面，會有新分支的名稱以及 pull request 按鈕

Pull request

.....



uuboyscytest / testfork

forked from uuboyscy/testfork

Code

Pull requests

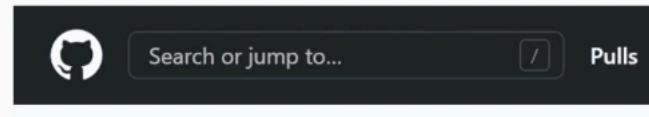
Act

新分支上傳後出現該選項
可對原專案擁有者提出合併請求

A screenshot of a GitHub repository interface. At the top, there's a yellow notification box containing a message: "feature/add-something had recent p 1 minute ago". Below this is a dropdown menu labeled "main" with a "Go to" button. A search bar says "Find or create a branch...". Underneath, there are tabs for "Branches" and "Tags", with "main" selected. A red dashed box highlights the "feature/add-something" branch, which is currently checked. A red arrow points from the text "新分支" (New Branch) to this highlighted branch. Another red arrow points from the text "【點擊 pull request 後】" to the "feature/add-something" branch in the list.

【點擊 pull request 後】可以提出要合併的分支以及目標，也可以備註讓專案持有者知道修改的內容

Pull request



 [uuboyscy / testfork](#)

<> Code ! Issue

es Pull requ

▶ Activities

Open a pull request

Create a new pull request by comparing changes across two

選擇欲合併的分支來源
及合併至哪個分支

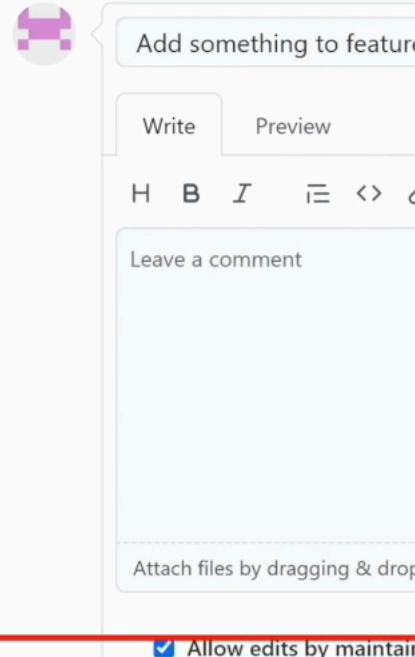
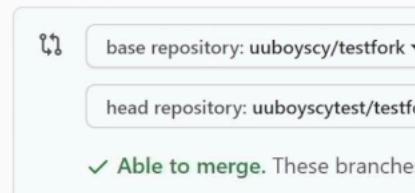
The screenshot shows a GitHub merge pull request interface. At the top, there are dropdown menus for 'base repository' set to 'uuboyscy/testfork' and 'base branch' set to 'main'. Below that, another dropdown for 'head repository' is set to 'uuboyscytest/testfork'. To the right of these dropdowns is a button labeled 'compare: fe...'. A large green checkmark icon is prominently displayed at the bottom left of the interface. To its right, the text 'Able to merge. These branches can be automatically merged' is written in a green font.

Add something to feature/add-something

Write Preview

H B I E <> ☰ E E ✓

Pull request



選擇完畢後點選此按鈕
即可向原專案擁有者提出合併請求

【專案持有人的批准畫面】

Pull request

此時原專案擁有者的畫面會出現合併請求

The screenshot shows a GitHub repository page for 'uuboyscy/testfork'. The 'Pull requests' tab is selected, indicated by a red underline. There is one open pull request titled 'Add something to feature/add-something' with a status of '1 Open' and '0 Closed'. A search bar at the top has the filter 'is:pr is:open'. To the right, there's a sidebar with options like 'Label issues' and 'Add something to feature/add-something'.

按下此按

- 【進階操作】Revert/ Rest / Rebase !!! 常用!!!

- 【Git Revert】將最後一次的 commit 取消

可以在Git Bash 中輸入「ls」，先確認一下我們有哪一些檔案

然後新增一個檔案，並 commit，作為教材

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ echo 123 >> testtest.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git add .
warning: in the working copy of 'testtest.txt', LF will be replaced by CRL
next time Git touches it

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git commit -m 'test'
[main ff1bbf8] test
 1 file changed, 1 insertion(+)
 create mode 100644 testtest.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$
```

Graph main 1 ↑ test

origin/main origin/HEAD

test1 modified by test1

origin/test2 test2 modified

Merge branch 'new-branch' merge r

Added main.txt

origin/new-branch Add new-b

new commit5

new commit4

new commit3

new commit2

new commit1

Added .gitignore

Sorted by file status ▾

☰ ▾

Commit: ff1bbf86cce132799f598a1299e3b15774b9

Parents: 2df5580839

Author: samhuang95 <sam.huang.veda@gmail.com>

Date: 2023年3月21日下午 03:11:50

Committer: samhuang95

test

+ testtest.txt

如果我們今天要取消這個 commit (取消剛剛建立的 testtest.txt 檔案)

>在Git Bash 中輸入「git revert HEAD」後，會出現下面的畫面，這是一個可以輸入 commit message 的地方，但通常會幫我們寫好，如下圖顯示的「deleted...」

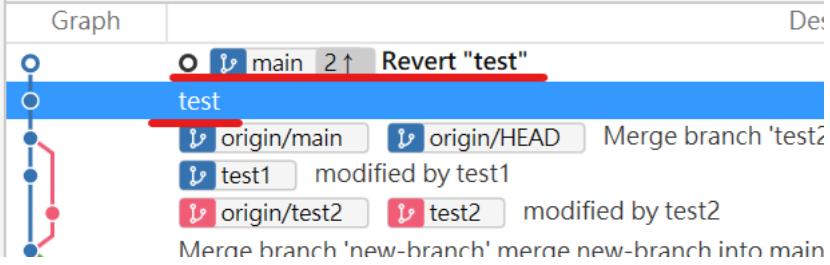
接下來只要輸入「`i`」按下「`esc`」輸入「`:wq`」儲存並退出，就算是完成「Revert」

可以輸入「ls」再次確認一下檔案是否成功刪除

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ ls
README.md  gitignore  new-branch.txt  test1.txt
README.md  main.txt   test.py        test2.txt

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
```

但是 sourcetree 上面更新後，會出現一個 訊息，Revert 上一個 commit 的訊息



所以 Revert 的概念，會比較像是新增一個 commit 來還原當時尚未新增 testtest.txt 這個檔案時的分支

All Branches Show Remote Branches Date Order

Graph

main 2 ↑ Revert "test"

test

origin/main origin/HEAD Merge
modified by test1
origin/test2 test2 modified by te
Merge branch 'new-branch' merge new-bran
Added main.txt
origin/new-branch Add new-branch.txt
new commit5
new commit4
new commit3
new commit2
new commit1

Sorted by file status

Commit: 2df55808392b0aaa0c6582cf582575144d620503
Parents: 59dac25e13, c9e81092c1
Author: samhuang95 <sam.huang.veda@gmail.com>
Date: 2023年3月17日 下午 08:04:54
Committer: samhuang95

Merge branch 'test2'
this is merge test2 process

README.md

所以 Revert 是一個不會修改到歷史紀錄，相對安全的方法

- 【Git Rest 基礎操作】可以將過去做過的 commit，包含歷史紀錄都直接刪除(複雜、危險)

先建立兩個 commit 作為教材

```

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ echo 'a=1' >> r1.py

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ echo 'a=2' >> r2.py

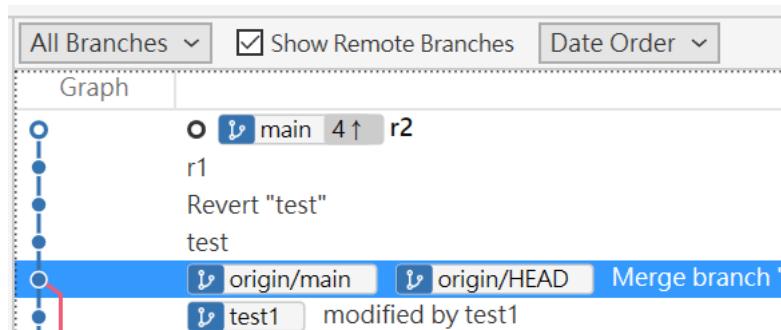
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ git add r1.py
warning: in the working copy of 'r1.py', file 'r1.py' would have been
  added by the command
    me Git touches it

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ git commit -m 'r1'
[main 2225901] r1
 1 file changed, 1 insertion(+)
 create mode 100644 r1.py

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ git add r2.py
warning: in the working copy of 'r2.py', file 'r2.py' would have been
  added by the command
    me Git touches it

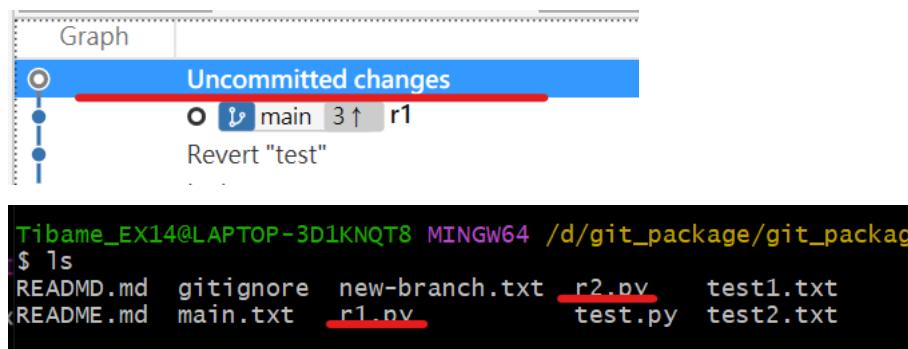
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package
$ git commit -m 'r2'
[main 3903a8c] r2
 1 file changed, 1 insertion(+)
 create mode 100644 r2.py

```



接下來要執行刪除「r2.py」這個 commit

在Git Bash 中輸入「git reset HEAD~」# 這邊寫的「~」表示回到上一節點(HEAD~)也就是「r1 的狀態」，與 checkout 的用法很像
這時候 sourcetree 中會出現一個暫時的節點，而且其實兩個檔案都還存在



這時候在Git Bash 中輸入「git status」看一下目前的狀態，可以看到它顯示的確有一個被新增或修改的檔案「r2.py」在 working directory 內，只是這個檔案尚未被新增到 *staging area(暫存區) 中。

*註解:可以參考「【版本控制】中的【開始版本控制】筆記」

```

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    r2.py

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

```

如果我想要將剛剛 Reset 的commit 檔案重新加回來

方法一、重新建立 commit

直接將 `r2.py` 檔案 add 進去 staging area，然後重新建立一個新的 commit

方法二、找回來檔案

將原本 rest 的檔案找回來，在Git Bash 中輸入「git reflog」

可以讓我們看到我們變更的歷史紀錄，輸入後會出現下面的畫面，這裡會記錄一些 HEAD 節點移動的歷史紀錄

然後我們要使用 `rest` 回到過去的狀態，目前想要回去「HEAD@{1}」：`commit: r2`

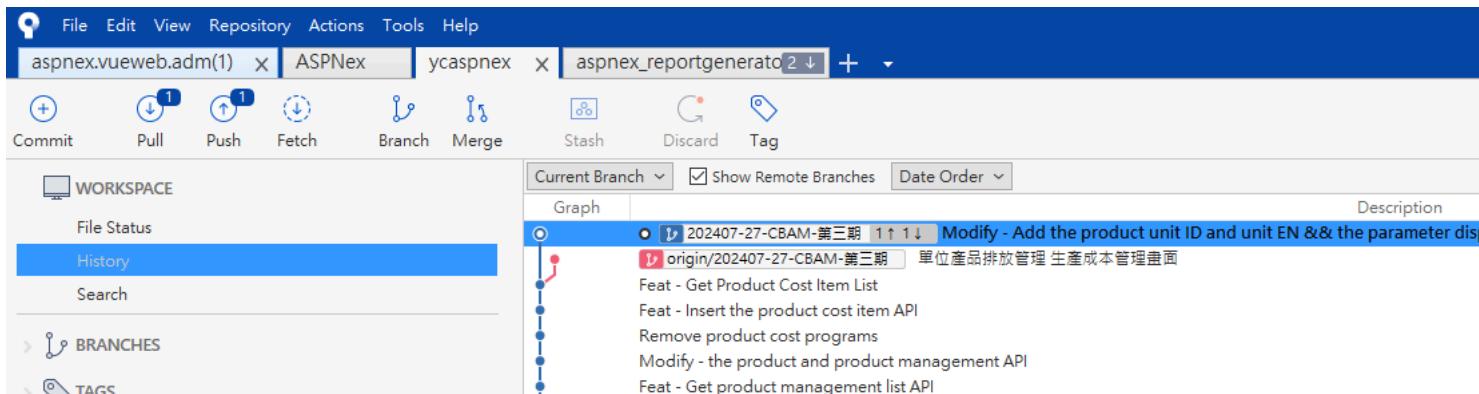
所以先複製「HEAD@{1}」，然後按下「q」離開畫面

然後輸入「`git reset HEAD@{1}`」

這時候就會在 sourcetree 中發現，我們的 r2.py commit 回來了

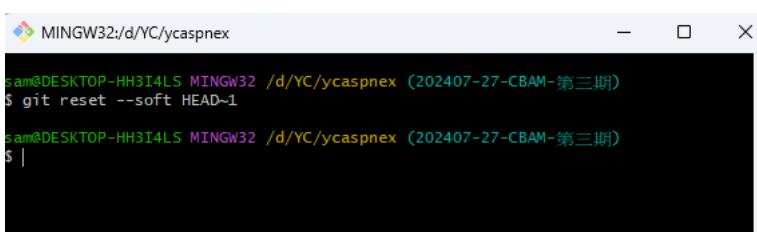
○ 【Git Rest 使用情境】 !!! 常用!!!

當與其他人在同一個分支的時候，因為時間差的關係，會造成我們已經 commit 後，無法執行 push，因為系統會顯示需要先 pull，但這時因為與我們 commit 的程式碼衝突，因此也無法順利 pull 的兩難狀況。

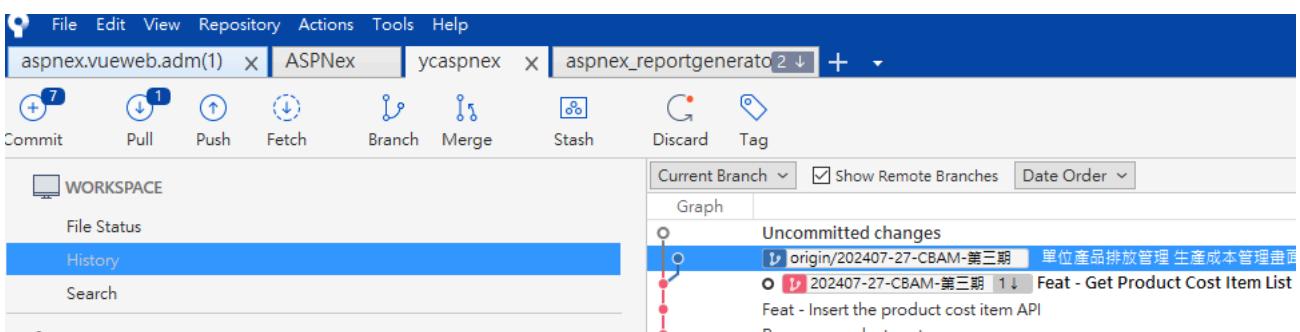


因此為了解決這樣的問題，可以依據下方步驟進行

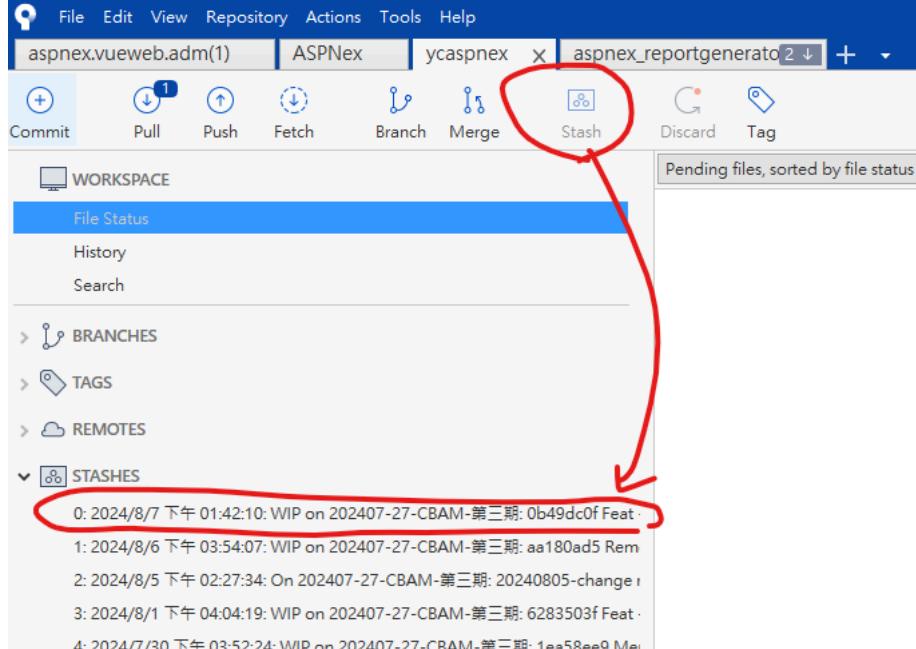
- 一、首先打開 git shell，輸入 `git reset --soft HEAD~1` 表示退 1 個 commit



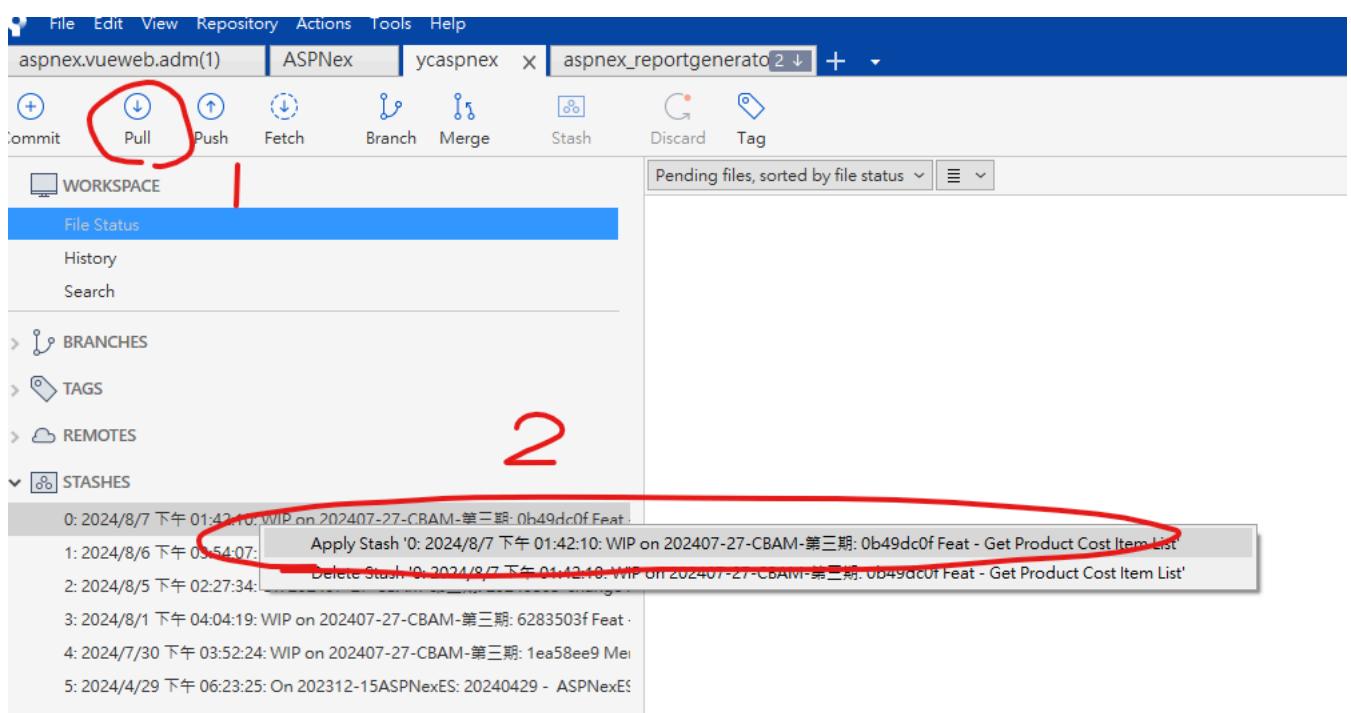
完成後就會出現下方樣式



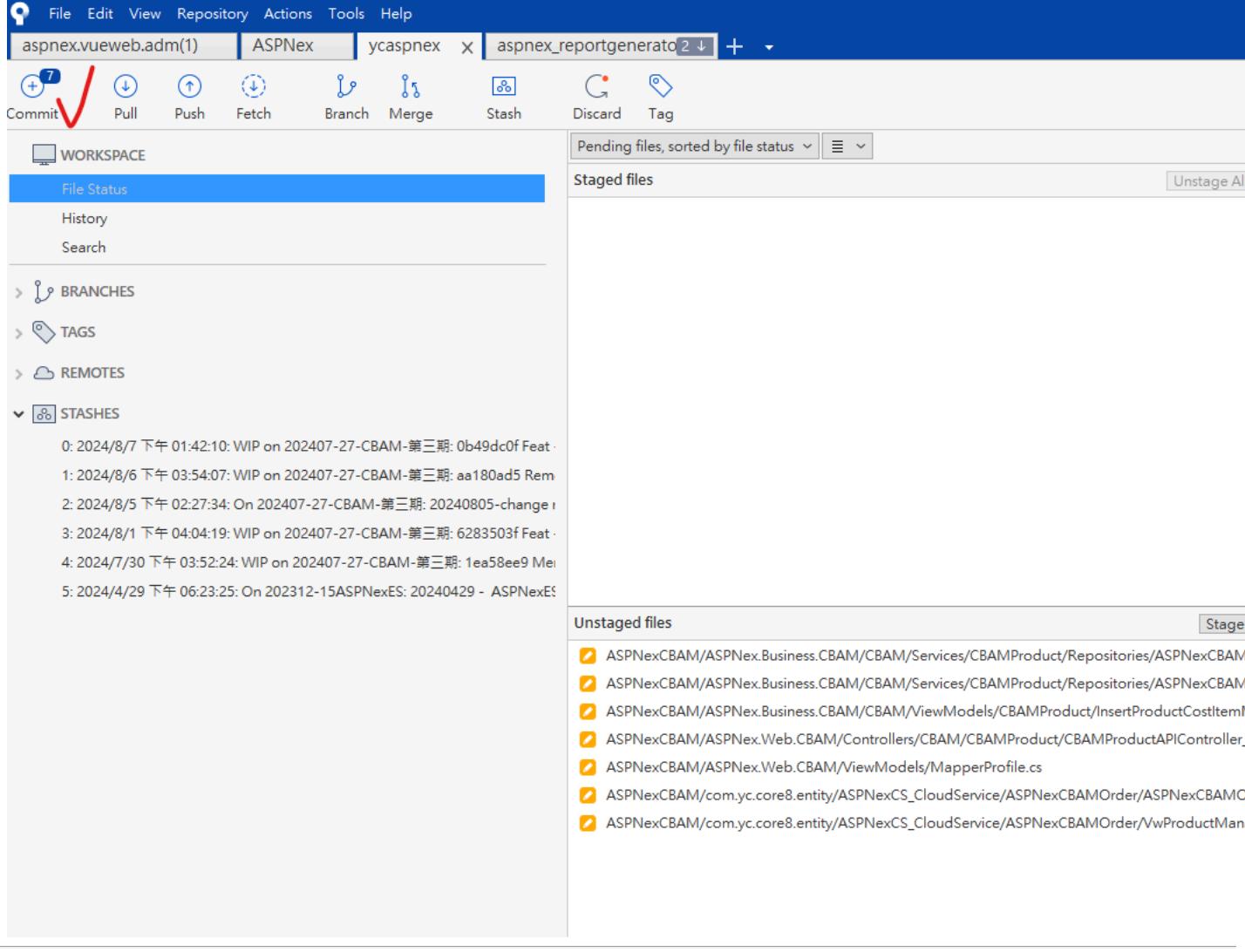
- #### ■ 一、將希望 commit 的程式，先儲存起來



三、然後 pull 下雲端的程式碼



四、完成 commit and push



◦ 【Git Rest 三種模式】mixed、soft、hard

▪ 【模式概述】

1. mixed:預設模式，拆除 commit 後，會將commit 曾經新增或修改的檔案保留在 working directory 中
2. soft:拆除 commit 後，會將檔案保留在 staging area
3. hard:拆除 commit 後，還會將因為 commit 而受到影響的檔案，直接復原

【注意】如果在任何已移除的狀況下，輸入相同的 commit message

▪ 【mixed】

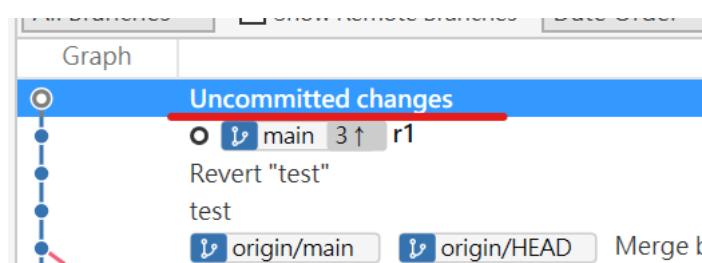
預設模式，拆除 commit 後，會將commit 曾經新增或修改的檔案保留在 working directory 中

▪ 【soft】

拆除 commit 後，會將檔案保留在 staging area

將 r2.py 使用 soft 的方式拆除，在Git Bash 中輸入「git reset HEAD~ --soft」，

會發現 r2.py 已成功拆除，並回到 r1 的節點，然後也出現一個暫時的節點



在Git Bash 中輸入「git status」，會發現與使用 mixed 的結果不同他會將檔案加入到 staging area 中

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_
$ git status
On branch main
Your branch is ahead of 'origin/main' by 3
  (use "git push" to publish your local co
Changes to be committed:
  (use "git restore --staged <file>..." to
    new file:  r2.py

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_
$
```

■ 【hard】

拆除 commit 後，還會將因為 commit 而受到影響的檔案，直接復原

先在Git Bash 中輸入「relog」查詢想要回復 r2.py 的位置然後，在Git Bash 中輸入「git reset HEAD@{3}」回復
在Git Bash 中輸入「git reset HEAD~ --hard」

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW6
$ git reset HEAD~ --hard
HEAD is now at 2225901 r1
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW6
$
```

這時候 sourcetree 的畫面會發現 r2.py 的 commit 被拆掉了，並直接回到 r1.py 的節點，而且檔案也的確完全被刪除了

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ ls
README.md  gitignore  new-branch.txt  test.py    test2.txt
README.md  main.txt   r1.py          test1.txt
```

但如果我們想要復原 r2.py，其實也可以使用 reflog 可以回復到 r2 的狀態

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ ls
README.md  gitignore  new-branch.txt  test.py    test2.txt
README.md  main.txt   r1.py          test1.txt
```

雖然恢復了，但出現了暫時的節點，而且檔案也沒有搶救回來，但如果我們使用

「git reset HEAD@{5} --hard」應該就會連同暫時節點都會消失

這時候 git status 的狀態會是這個檔案被刪掉了

```

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

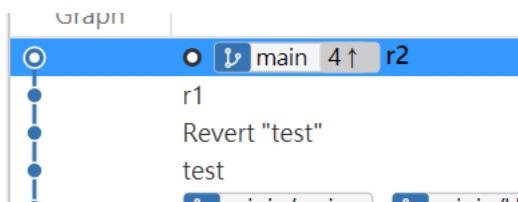
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      deleted:    r2.py

no changes added to commit (use "git add" to track changes)

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ |

```

為了要救回檔案，我們在Git Bash 中輸入「`git restore .`」，這時候救成功解決，**消除暫時的節點、搶救檔案**的任務了



```

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git restore .

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ ls
README.md  gitignore  new-branch.txt  r2.py    test1.txt
README.md  main.txt   r1.py        test.py  test2.txt

```

■ 【不可回復狀況】

我們先在Git Bash 中輸入「`git reset HEAD~~--soft`」，使用 soft 的方式進行 reset，並查看狀況

```

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git reset HEAD~~--soft

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   r2.py

Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (main)
$ 

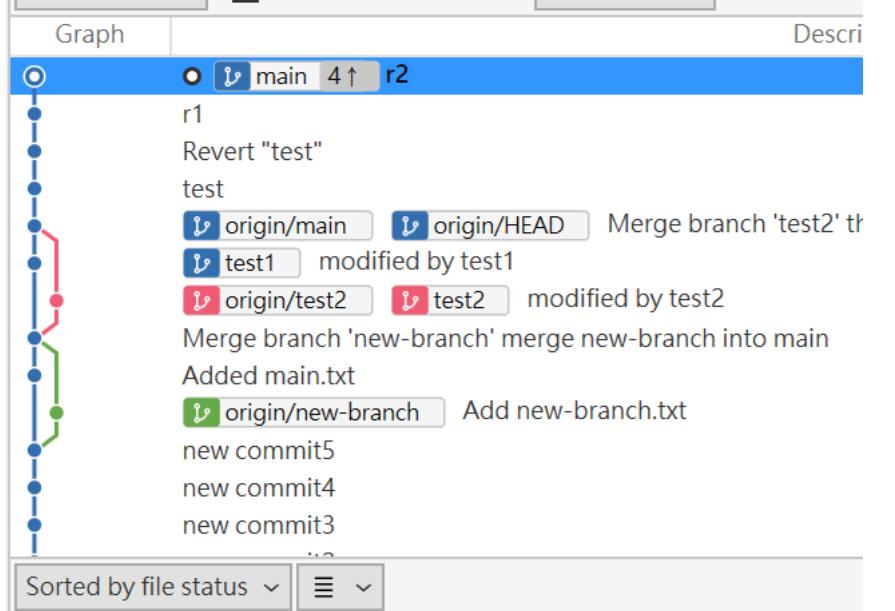
```

這時候使用同一個 commit message 來進行 commit

在Git Bash 中輸入「`git commit -m 'r2'`」，這時候可以觀察，原本的 commit ID，與commit 後的 commit ID 會變不同，這樣子就是將歷史做了修改，過去的 commit 紀錄將被新的**複寫**，這時候如果我們 push 回 github 可能會造成環境衝突的問題

但如果在所有的 commit 都還沒有 push 到 github 之間，將歷史做了修改是沒有關係的

commit 前：

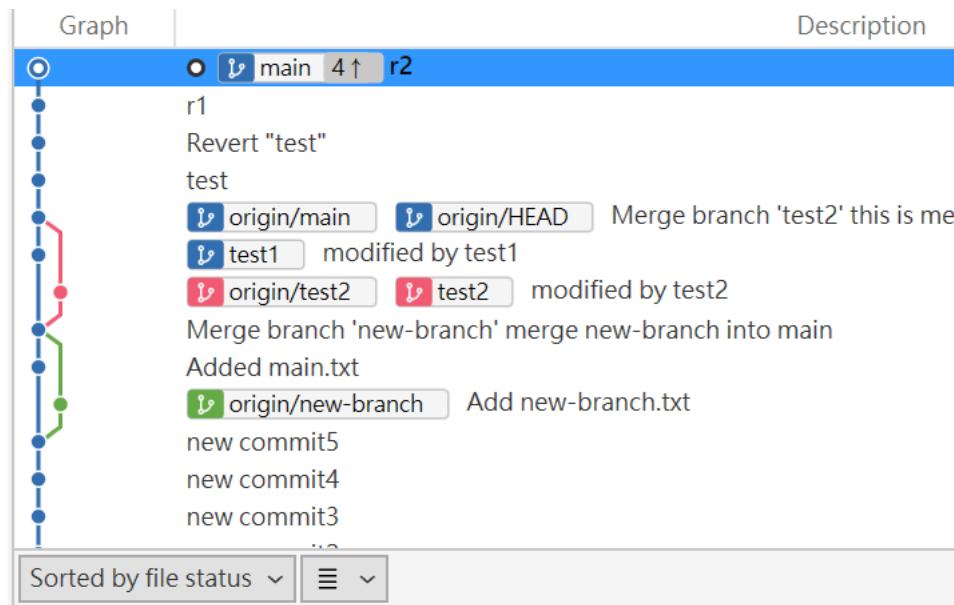


Commit: 3903a8ccfc8b2c86c52bdd0d040c889312c43179 [3903a8c]

Parents: 22259019c7

Author: samhuang95 <sam.huang.veda@gmail.com>

commit 後:



Commit: c821a8943529373e346980b5ecfc13caf9da096f [c821a89]

Parents: 22259019c7

Author: samhuang95 <sam.huang.veda@gmail.com>

Date: 2023年3月22日 下午 08:05:57

Committer: samhuang95

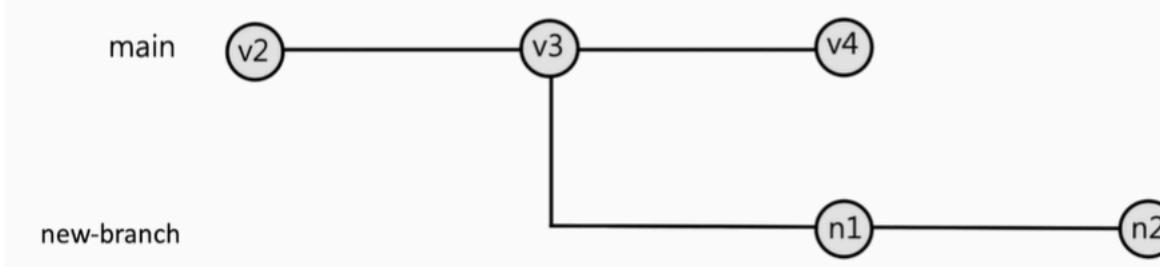
r2

◦ 【Git Rebase】上傳到 github 前整理分支

分支的分水嶺位置，稱為「**基底**」，**Git Rebase** 可以重新定義某一個分支，是從哪個基底長出來的。

假設今天我們要將 **new-branch** 合併到 **main**，可以使用 **rebase** 的方式，操作如下：

先切換到 **new-branch** 分支，然後在**Git Bash** 中輸入「**git rebase main**」，就可以重新定義基底到 **main** 分支



切換到 new-branch 使用 git rebase main

結果如下圖:



教學操作如下:

我們先在 main 建立一個 new-branch .

然後 new-branch 創立1個名為 newbranch.txt 的文件，然後新增兩次文字在其中，等於往前走兩步

先切回 main，然後新增一次文字在其中，等於往前走一步

Graph Description

main-commit

- new-branch-commit2
- new-branch-commit1
- r2
- r1
- Revert "test"
- test
- Merge branch 'test2' this is merge test2 process
- modified by test1
- modified by test2
- Merge branch 'new-branch' merge new-branch into main
- Added main.txt
- Add new-branch.txt

Sorted by file status ▾ main.txt

Commit: 6ea45fd0d107a4ff7469f0497758faee408f6a71 [6ea45fd]
Parents: c821a89435
Author: samhuang95 <sam.huang.veda@gmail.com>
Date: 2023年3月22日 下午 08:27:21
Committer: samhuang95

main-commit

main.txt

然後 main 往前走一步，new-branch 往前走兩步，變成如上圖一樣的示意結果

接下來會將 new-branch 的基底節點，改成 main 最新狀態的節點

以下操作:

先切換到 new-branch 分支，然後在Git Bash 中輸入「git rebase main」，這時候在 sourcetree 上可以看到如下的變化

```

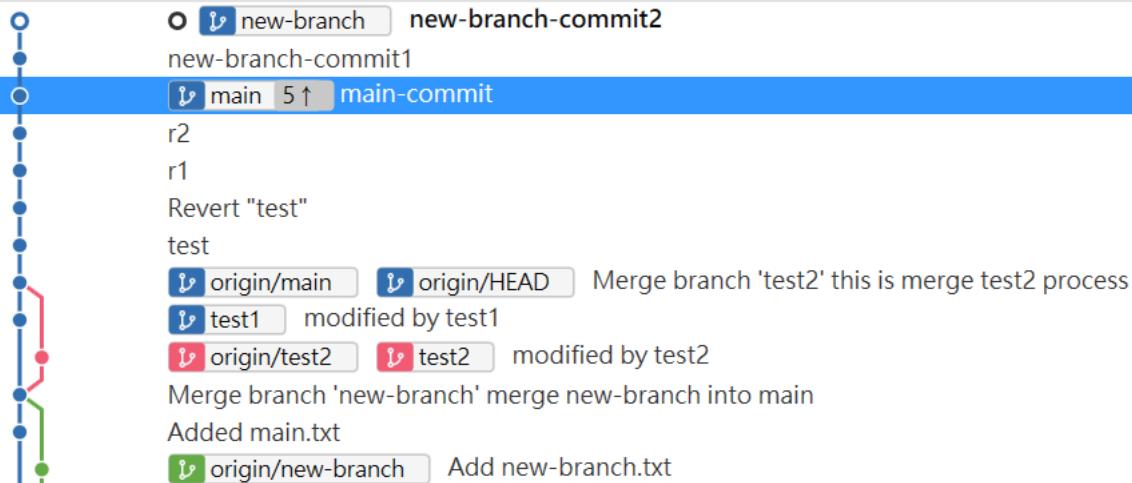
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (new-branch)
$ git rebase main
Successfully rebased and updated refs/heads/new-branch.

```

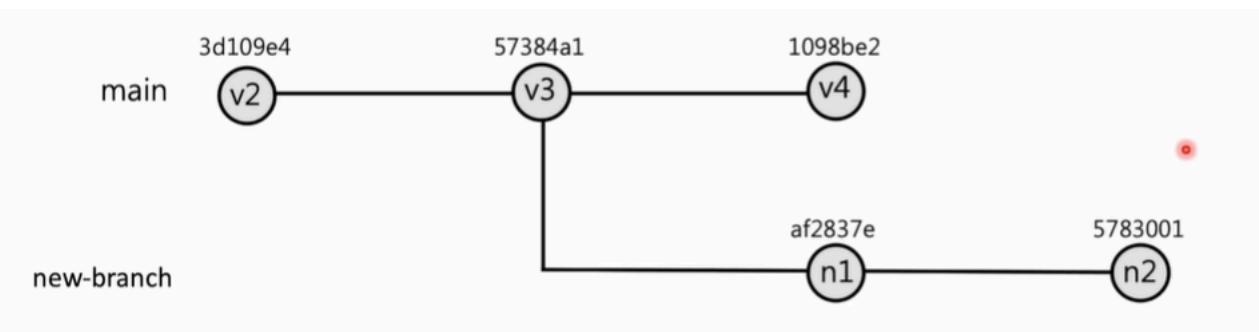
All Branches Show Remote Branches Date Order

Graph

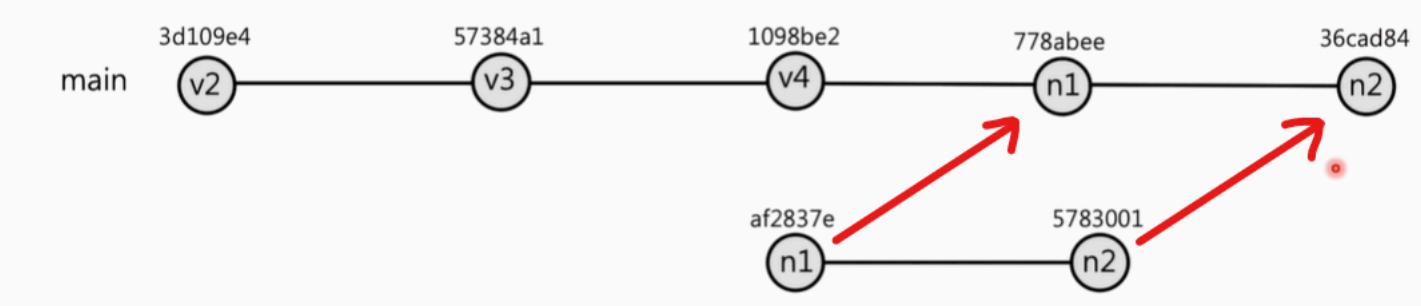
Description



但實際上 rebase 變化邏輯如下:



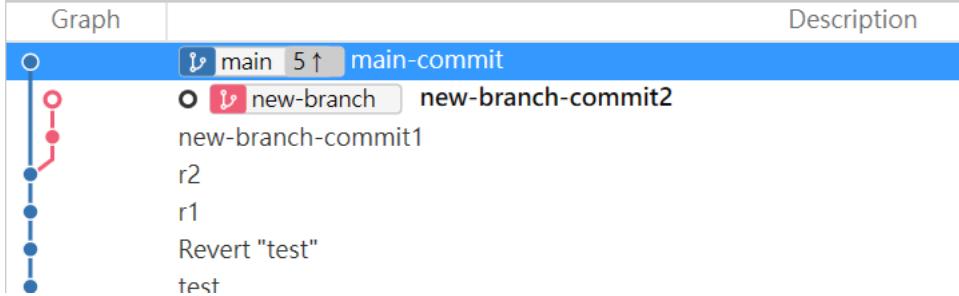
可以發現其實 rebase 是幫我們將 new-branch 上所有的動作，重新執行過一次，所以會有新的 commit ID



如果想要還原到合併之前，可以使用「reflog」的方式達到效果，而我們用來觀察合併的變化，就會發現 rebase 就是上面的操作

```
MINGW64:/d/git_package/git_package
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/git_package/git_package (new-branch)
$ git reflog
f40055e (HEAD -> new-branch) HEAD@{0}: rebase (finish): returning to refs/heads/
new-branch
f40055e (HEAD -> new-branch) HEAD@{1}: rebase (pick): new-branch-commit2
0871f5c HEAD@{2}: rebase (pick): new-branch-commit1
6ea45fd (main) HEAD@{3}: rebase (start): checkout main
add7842 HEAD@{4}: checkout: moving from main to new-branch
```

在Git Bash 中輸入「git reset HEAD@{4} --hard」



- git add 多個文件，但想忽略某一個文件

```
// 先用以下命令添加所有
git add -u
// 再以下指令排除掉你不想加入進去的某幾個文件(路徑可以用 git status 檢視)
git reset -- README.md
```

- 檢視 Git user Information

```
# 查看方法
git config --global user.name
git config --global user.email
# 設定方法
git config --global user.name "your name"
git config --global user.email "your email"
```

• 【補充紀錄】

commit number 是由一個 SHA-1 加密方式產生出來的 16 進位值，有機會重複，但幾乎不可能

• 【可以研究】

• 【補充】

先在資料夾中建立 .gitignore 文件

進入 vim .gitignore 中，並輸入不要上傳的文件(檔名副檔名)，之後

```
echo "123" > test.txt
mkdir 資料夾名稱
mv test.txt testdir/
git add .
git commit -m ""
git push
git log
```

• 【錯誤 error 排除】

如果在 push 的時候出現錯誤訊息「error: failed to push some refs to ...」

可能是因為電腦裡的內容是比較舊的，所以你應該先拉一份線上版本的回來更新，然後再推一次

我之前的狀況是，在地端修改檔案位置，並修改檔案名稱，因次出錯

解法：在 git bash 中輸入「git pull --rebase」呈現如下後，在 push 就可以了

```
Tibame_EX14@LAPTOP-3D1KNQT8 MINGW64 /d/robo_advisor/robo_advisor (main)
$ git pull --rebase
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 1.04 KiB | 212.00 KiB/s, done.
From https://github.com/samhuang95/robo_advisor
  090dd1b..ef10c37  main      -> origin/main
Successfully rebased and updated refs/heads/main.
```

- 【Git實習老師內容】

「git init」

「git status」

VS code 加入套件「**Git Graph**」

Github

```
git remote -v  
git remote add origin {github_ip}  
# git remote add origin https://github.com/samhuang95/test_git.git  
# 如果有推上去過，但是失敗(頁面可以看到，但是沒有過去，那就用下面的進行調整)  
# git remote set-url origin https://github.com/samhuang95/test_git.git  
git remote -v  
git push --set-upstream origin master
```

- 【操作步驟】

步驟一、初始化專案資料夾

在專案資料夾中，使用 cmd，輸入「git init」進行初始化

步驟二、追蹤所有檔案

接著在 cmd 中輸入「git status」查看狀態 Untracked files : 未追蹤 git add . —> . (all) 加入所有檔案 add —> 準備存入(檢查檔案) git commit -m 'new' -m '' —> 訊息 '' commit —> 存檔點 git log 出現訊息 —> commit的訊息內容 git remote -v 本地程式碼與遠端程式庫連結 origin <https://github.com/yen850515/yen0515.git> (fetch) origin <https://github.com/yen850515/yen0515.git> (push) origin —> 遠端name master —> 分支name

