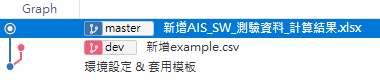
Local端git版本控制示範

1. 專案環境設定 & 套用模板
2. Git bash: cd /d/projects
3. Git bash: mkdir git\_try
4. Git bash: cd git\_try
5. Git bash: git init
6. Git bash: touch server.R
7. Git bash: touch ui.R
8. 去shiny gallery網頁，隨便抓個模板，複製貼上到server.R和ui.R裡面
9. Git bash: git add .
10. Git bash: git commit –m “環境設定 & 套用模板”
11. 打開source tree，把/d/projects/git\_try，丟進去，就可以在history觀察到第一筆commit記錄
12. 看圖 & 開始學畫圖
13. 建立分支 & 合併
14. Git bash: git branch dev
15. 去source tree重新整理，就看到圖形多了一個branch。而master在dev前面，表示我們現在的”head”是在master上
16. Git bash: git checkout dev。
17. 去source tree重新整理，就看到dev跑到master的前面了
18. 我新增SAS考卷的csv檔進來，然後commit後，註解寫”新增csv檔”
19. Git bash: git add . ; git commit –m “新增csv檔”
20. 去sorce tree看一下，現在的圖變成這樣

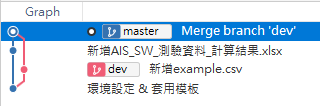
可以看到dev現在有2筆commit，但master現在只有一筆commit

1. 接著，我把分支換回master(Git bash: git checkout master)，然後看看資料夾中，還有沒有剛剛上傳的example.csv -> 沒有了!! 超酷的，因為在master的分支上，並沒有example.csv這個commit的存在。
2. 那繼續做，我在master這個分支上，也上傳一個檔，是” AIS\_SW\_測驗資料\_計算結果.xlsx”，做法是先確認現在在master上，然後手動copy這個檔到現在的資料夾中，再git add . ; git commit “新增AIS\_SW\_測驗資料\_計算結果.xlsx”
3. 回去source tree看，會看到下圖



可以看到master有兩筆commit，dev也是兩筆commit

1. 去資料夾上看，會看到有AIS\_SW\_測驗資料\_計算結果.xlsx這個檔，但沒有example.csv這個檔，因為現在的身分是master，不是dev
2. 再來，就是要把兩個分支合併。我現在想先站在master的分支上，然後把dev給合併進來。用的語法是(先確認自己在master上): git merge dev
3. 回到source tree上看，可以看到



從線圖可以直接看到，兩條線合併了。最上面是master，因為我剛剛是用master的身分去merge人家，所以圖示中可以看到master有四筆commit(要把紅色的也算進來)；而實際去資料夾上看，csv和xlsx檔，現在都看的到

1. 那…dev呢？現在看dev只有2個commit，所以我如果切換到dev，會不會還是只看的到.csv檔，看不到.xlsx檔？還真的是!!那我該如何把dev也更新到最上面？就一樣的招，先切換到dev，然後git merge master
2. 儘管上面的做法可以work，但這邊要介紹另一種做法，叫rebase，意思是換機底到誰那邊，然後把現在這個分支做過的事，在新分支上重做一次。所以，我如果寫 git rebase master，那意思是，我先把基底換到master上(所以此時我master有的，我都要有，那我現在以dev的身分，就同時擁有.xlsx和.csv兩個檔了)，然後，我要把我剛剛在dev上做過的事，以master為底重做一次，結果因為dev剛剛做過的事，master都做過了，所以這邊就不用再做了。所以結果就會變下圖：



可以看到dev也跑到最上面了，現在dev也同時具備4個commit。

1. 其實剛剛這樣講rebase有點亂，但記住精神是，”換基底到別人身上，再重做原本自己做過的事”。現在就重新示範rebase的例子。
2. 我在dev的身分上，新增一個dev\_try.txt檔 (touch dev\_try.txt; git add .; git commit –m “新增dev\_try.txt”)
3. 我換到master的身分上，新增一個master\_try.txt檔(git checkout master; touch master\_try.txt)
4. 此時的source tree變成這樣：



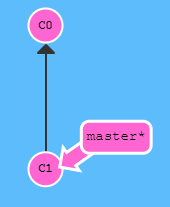
可以看到master做了新增master\_try.txt, dev做了新增dev\_try.txt。

1. 那我如果在master的身分，去做git rebase dev會怎樣？依照剛剛的精神，他就會先跳到dev的那條線上，然後，把master剛剛做過的”新增master\_try.txt”，再做一次。做完後，source tree變成這樣：



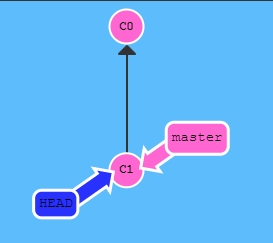
他直接把master和dev給merge了。所以，rebase的意思，大概就4這樣

1. Moving around in Git
2. 從剛剛的source tree，可以看到線圖中有很多commit記錄，這一章就是要教我們，如何自由的遊走於各個版本之間
3. 首先，看一下剛剛的線圖，要有一個概念，我們每個點都是一個commit，整個git都是以commit做基礎。然後，master或dev這種branch name，都是用來指向某個commit。例如下圖：



C0是最一開始的commit，再來是C1這個commit。而現在master這個branch name，他就指向C1

1. 注意一下master後面有個star sign，他的意思其實是HEAD指向master。所以目前三者的關係是 HEAD -> master -> C1
2. 還記得之前我們會用 git checkout master; git checkout dev，來切換身分，其實他就是在講我要把HEAD指向哪個branch name，你要指向master，就是git checkout master。
3. 那有趣了，HEAD可不可以不要指向master，而改成指向commit呢？例如指向C1？
4. Sure，當然可以，如果我們打 git checkout C1，那示意圖會變成：



1. 所以我們看到，剛剛master後面的星號不見了，然後多出一個叫HEAD的東西，指向C1
2. 同樣的道理，當然也可以打git checkout C0，把HEAD指到C0
3. 那…這樣做到底要幹麻？我們回到剛剛實作的例子看看。剛剛實作的例子，work tree長成這樣：



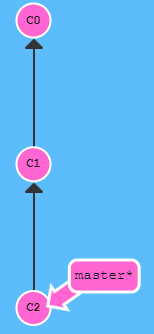
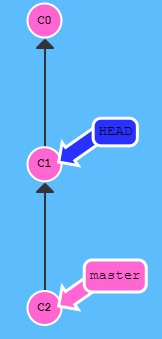
從圖中可以看出，master, dev這兩個branch name都指向最後一筆commit，然後HEAD是指向master(因為master排在dev前面)。

1. 那想像一下，我現在不管是站在master的位子，或dev的位子，我都看的到.xlsx, .csv, 和一開始模板的所有檔案
2. 那如果我今天想退回到”新增example.csv”這個commit的狀態呢？那484就可以使用現在學到的這招拉？我只要git checkout “新增example.csv”，484就可以退到那個commit的狀態，我從資料夾中，就指會觀察到example.csv這個檔，以及”環境設定 & 套用模板”中設好的server.R和ui.R？
3. 概念上，Yes!!但實際來說，commit的name不叫“新增example.csv”，而是一個hash值，要用 git log去看，所以打git log後，可以看到”新增example.csv”的這筆commit，他的hash值是f370dd5456c025d4160ee4c00f9393644590a722。那我要把HEAD指向這個commit，就要打成： git checkout f370dd5456c025d4160ee4c00f9393644590a722
4. 這時候，回去看一下source tree，就可以看到：

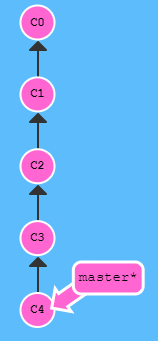
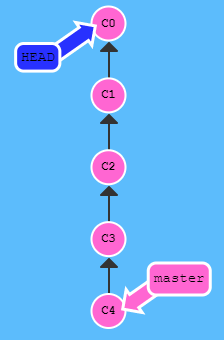


讚讚讚，HEAD真的跑到我指定的commit了。然後去資料夾上看，果然看到資料夾中的內容只剩下example.csv這個檔，以及”環境設定 & 套用模板”中設好的server.R和ui.R

1. 所以，我懂了，學這個HEAD的功能，就是幫助我在不同的commit間移動。那，回到實務面，如果我每次的hash值都要用git log去查，然後複製一大串hash值，這也太累了吧。一個簡單的應付方式，是只打前面幾碼，那如果前面幾碼就能唯一決定這個commit，那就可以work了。例如 git checkout f370dd5，其實就ok了，而f370dd5其實就是從source tree那邊看來的。
2. 但我們還是想，有沒有更簡單的方式？其實有的，叫”相對引用”，他的作法是先列出一個參考點(例如 master)，然後再加參數：^表示向上移動一個commit，~n表示向上移動n個commit
3. 例如：原本的work tree長成下圖，那我如果打git checkout master^，那就是以master當參考點，往上移動一步，所以就會指向C1

 git checkout master^ 

1. 除了以branch name為參考點(如剛剛的master)，也可以拿HEAD當參考點。以上圖為例，本來的HEAD是指向master，那他的上一層，就也會是C1，所以，剛剛的git checkout master^，效果等同於 git checkout HEAD^
2. 那如果要往上n層，就用~n

 git checkout HEAD~4 

1. 首先叫講HEAD。HEAD其實就是個reference，想像互動網頁的那個星號，他都會指向某個branch name，而且，都是指在這個branch的最新一筆commit上。例如，指在master這個分支的最新一筆commit上，或是指在dev這個分支的最新一筆commit。
2. OK，這邊的概念通了，我們現在回到實作的例子，現在的work tree長這樣：



1. 我現在想，先用git checkout dev，把HEAD指到dev (可看到dev排在master前面，表示HEAD指向dev了)

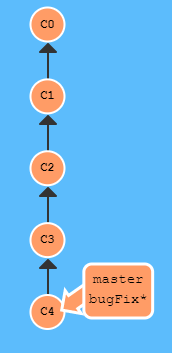


然後，再用git checkout dev~3，把HEAD指到最一開始的環境設定



可以看到，HEAD現在指到最一開始的環境設定了，此時，去資料夾看一下，可以發現裡面只剩下server.R和ui.R而已，剛剛的那些.csv檔和.xlsx檔都不見了。

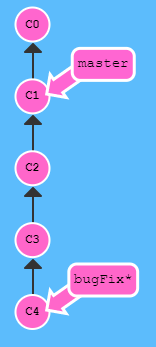
1. 這邊要提醒一下，為啥剛剛是用git checkout dev~3，不是git checkout dev~4？因為你看線圖中，master或dev，都是藍色底色，對照現圖，往前3個點才是HEAD，所以才會是3，不是4。
2. 目前為止，我們學會如何移動HEAD了，那…我如果想移動branch name呢？以下面的示意圖為例：



可以知道，現在有兩個branch，一個叫master，一個叫bugFix，這兩個branch name都指向C4。而現在的HEAD指向bugFix\*。

我如果想把master指向C1，我可以這樣做：

git branch –f master HEAD~3 (對比於剛剛用git checkout，是要對checkout的點，e.g. HEAD，做事情，現在這邊git branch，就是要對branch做事情。做啥事？就是強迫(force, -f) master這個branch，移動到HEAD往前3



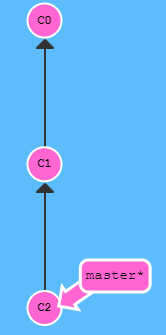
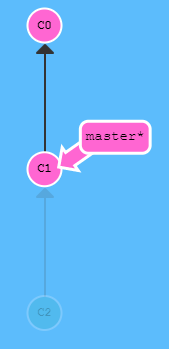
可以看到，master被移動到C1的位子了。

1. 回到實作的例子，我現在的work tree如果長成下面這張圖：



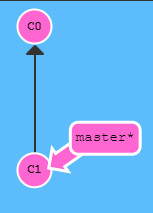
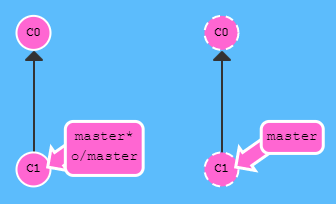
然後，我現在想把master，移動到”新增AIS\_SW\_測驗資料\_計算結果.xlsx”，可以嗎？git branch -f master HEAD~2，結果出現error，目前不知道原因為何。

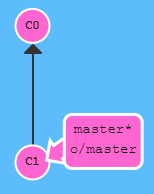
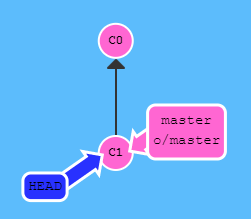
1. 復原
2. 剛剛學的是怎麼任意移動HEAD，但有另一種狀況是，我commit出去後，後悔了，我想退回我commit以前的版本，然後這筆錯誤的commit我想刪掉。那這個概念，就是ctrl + z 的復原概念，在git中，用git reset HEAD~n來處理，你要復原幾步，n就寫幾
3. 例如，本來的線圖如果長成：

 退回上一個操作，git reset HEAD~1 

1. 處理合併的衝突
2. 剛剛的練習中，我在dev的branch新增一個csv檔，在master的branch新增一個.xlsx檔，所以我merge的時候，不會有衝突。以master去merge dev為例，對master來說，他就多新增一個.xlsx檔就好。
3. 但如果，今天在dev的commit，是對server.R做了修改，在master的commit，也是對server.R做了修改，那merge的時候，衝突就發生了。
4. 把模板修改成自己想要的樣子，並練習各種狀態(untracked, modify, staged,…)
5. 如果想看之前的版本是寫些甚麼，該怎麼做？
6. 如果想直接變回之前的版本，把之後的版本全刪光，該怎麼做？
7. 建立分支

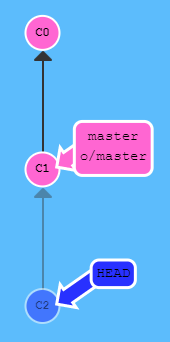
remote端

1. Remote repository (遠端數據庫) 其實就是 local repository的副本，他的好處是：
2. 備份用：local端我們已經會用git指令來做版本控制，但如果local端資料遺失，那就全毀了。但如果把他同步到remote repository，就可以進行救援備份
3. 共同開發：現在你把code放在remote端，大家都看的到，就可以一起編輯了！
4. 目前的主流都是用網站(例如github, gitlab, bitbucket)去對remote repository做視覺化。但背後的最大功臣還是remote repository，所以這邊就要先弄清楚remote repository
5. gitlab亂嘗試
   1. 在gitlab網站上，先隨便建立一個repository
      1. 我先在gitlab上，手動創建一個first\_try的repository
      2. 然後我在網站上修改了一下README.md，做第一次commit
   2. git clone
      1. 技術上來說，git clone就是要把gitlab上的remote repository，複製一份下來到local端
      2. 但真實意義，其實是和第一點相反的。git clone其實是把local repository複製一份到remote repository上
      3. 雖然很怪，但我們現在用第二點的方式，來講一下概念。
      4. 原本你在local repository的commit tree長這樣：
      5. git clone後，變成 
      6. 可以看到在remote reposity上，有兩個branch，一個是原本local端就有的master，另一個則是新的o/master
      7. o/master其實在我們實際使用時，會叫original/master，而這邊會寫o/master是他們做這個教學UI時的限制
      8. 那original/master是啥意思?他其實就是提醒你，這是一個remote branch(遠端分支)，命名原則為<remote repository name>/<branch name>，所以這邊就可以知道他的遠端數據庫被命名為original，然後這是original這個數據庫中的master branch
      9. 遠端分支跟我們以前學過的一般分支有點不一樣。簡單來說，如果你現在HEAD在master時，你做commit，會得到一筆commit。但如果你把HEAD移到o/master時，做commit，他會得到HEAD與branch分離的結果，如下圖：

 git checkout o/master 

可以看到，竟然不是把\*打到o/master上，而是直接拉HEAD出來

* + 1. 所以，當我繼續做 git commit後，得到的結果是：



這告訴我們，當我們加入新的commit時，o/master沒有更新。這是因為只有當remote更新的時候。o/master才會更新

* + 1. 回到實作，我去gitlab上剛剛創建好的first\_try，複製Clone裡面的url: <https://gitlab.com/ckshoupon/first_try.git>
    2. 然後在git bash上，打建立一個練習用的gitlab資料夾，並切到這裡面：Git shell: /d/projects; mkdir gitlab; cd gitlab
    3. 最後，用git clone，把gitlab上的first\_try，抓到這個資料夾中： git clone <https://gitlab.com/ckshoupon/first_try.git>
    4. 第一次做，會跳出視窗要你填gitlab的帳號密碼，因為他要知道你是用哪個gitlab帳號的身分，去clone人家，和別人協作這樣。輸入帳密後，就clone成功。
    5. 點進去這個資料夾後，裡面只有一個.git檔，沒有README.md這個檔
  1. git fetch
     1. 剛剛的git clone，是用在你把遠端數據庫，複製到本地端，讓本地端也開始有.git來記錄版本變化
     2. 但之後遠端數據庫也會一直更新，那我們要如何繼續抓新的資料呢？當然不會一直用clone，所以就會用這個新指令： git fetch
     3. 當我們做了git fetch後，原本本地端的remote branch，就會跟著更新
     4. git fetch具體的工作，是先把遠端數據庫的所有commit抓下來，然後比對一下本地remote branch上的commit，然後把本地的reomote branch上缺少的commit給補上
     5. 所以，git fetch並沒有影響本地的”非遠端branch”，所以重點在於，如果你想同步本地端和遠端的commit，你用git fetch其實是辦不到的，因為你只是同步了遠端分支而以
     6. 所以，我們應該把git fetch想成下載資料，而不是同步
  2. Git