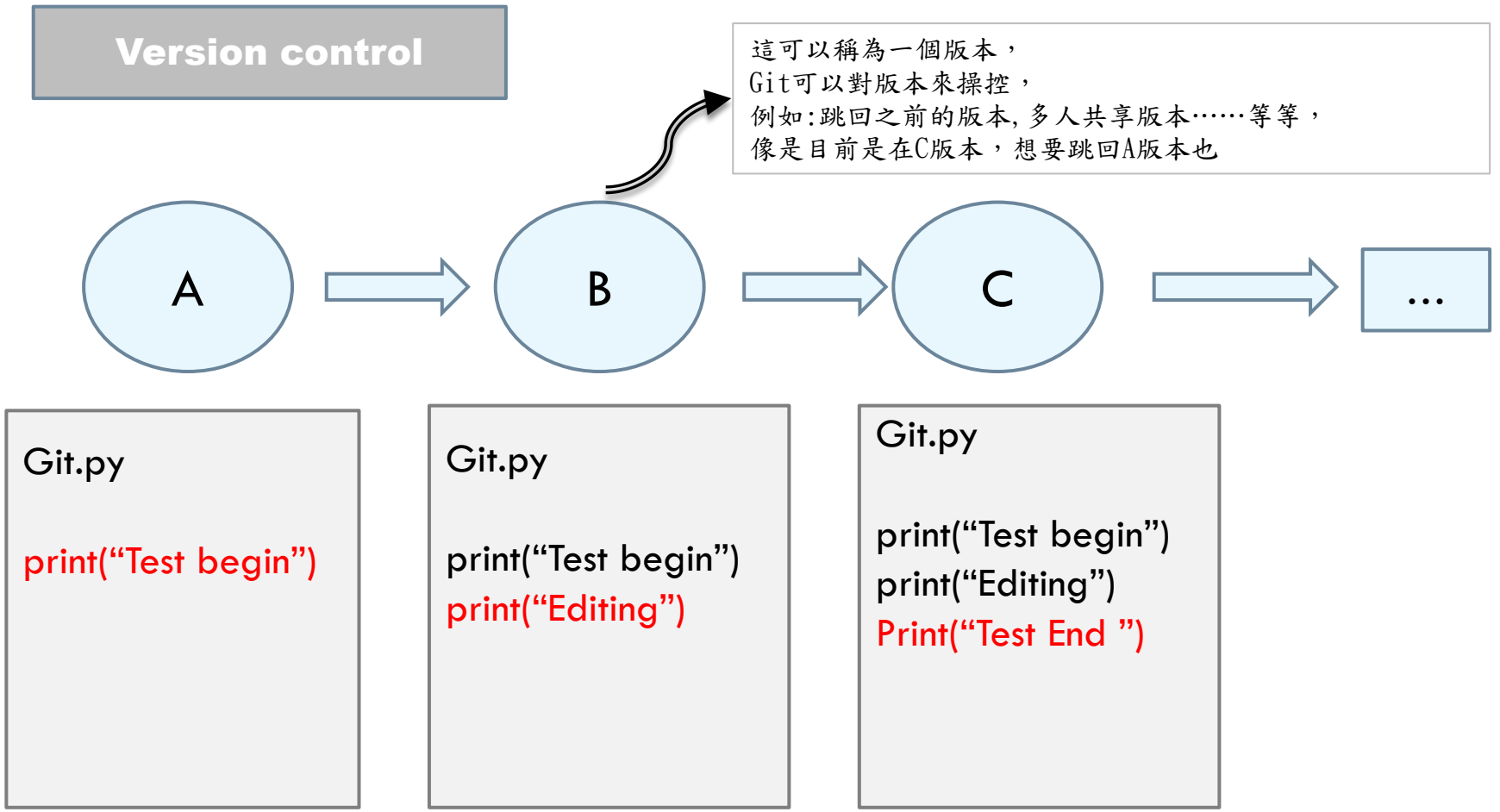
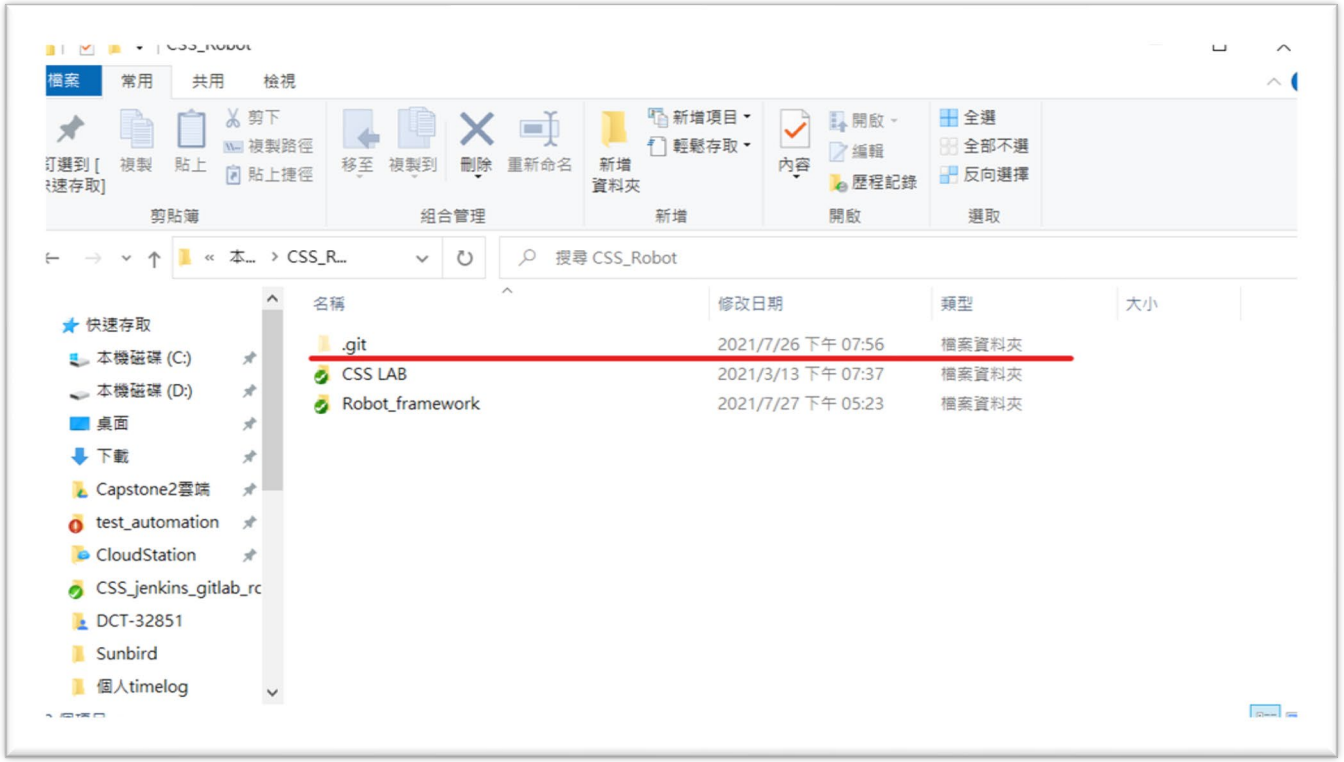
WHAT IS GIT

Why do we use GIT?

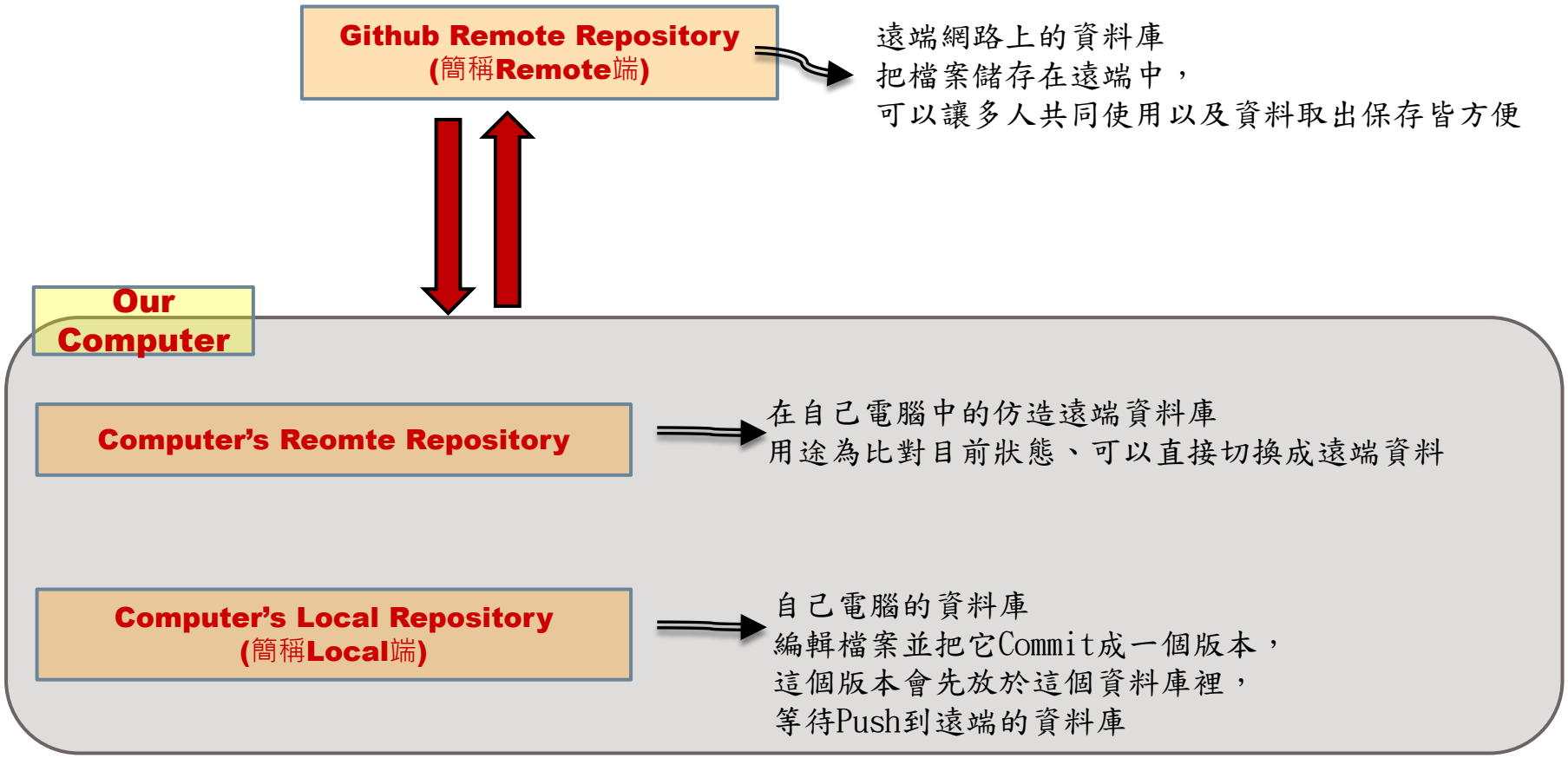


How to check repository

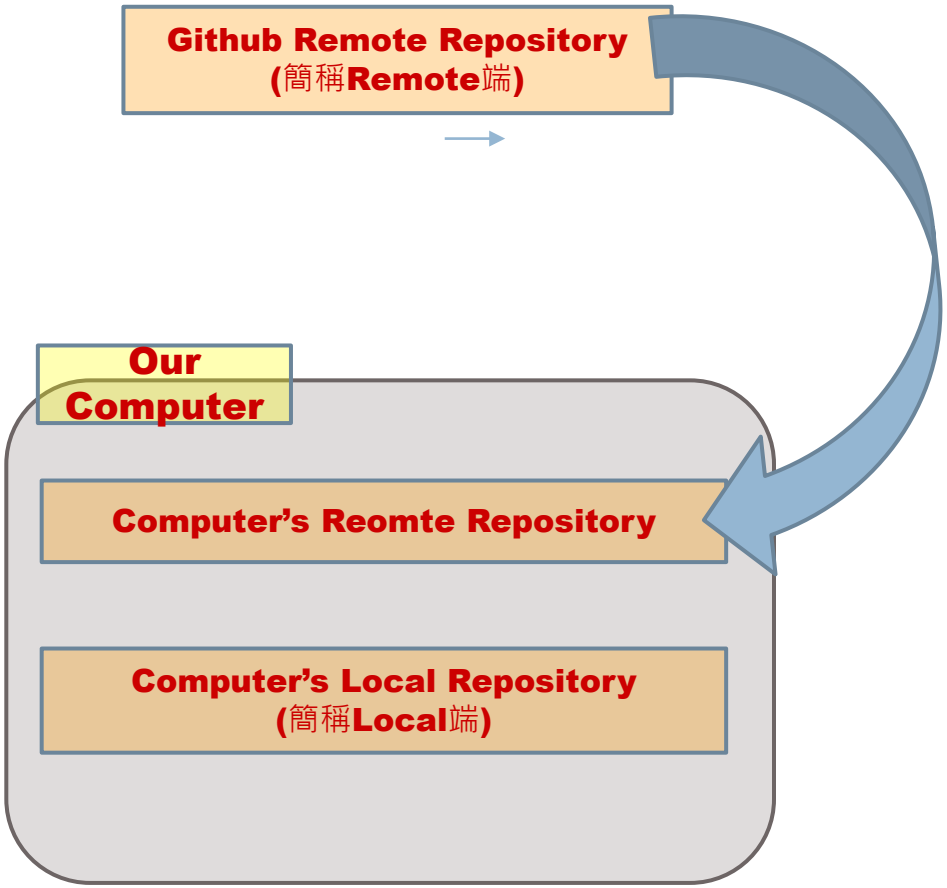
- 如果你有建立出repository，檔案夾上方會出現<.git>的隱藏項目資料夾，內部儲存有關這個repository的git資訊，
- 若你刪除之後，這個資料夾就不具有git repository的功能了



The principle of GIT



常用指令簡易介紹: Fetch



＊ 執行fetch指令，
在電腦裡面的程式或是檔案不會造成更改，
會更改會是在本機電腦的Reomte Repository，
將本機的Reomte Repository來做更新。

常用指令簡易介紹: Commit

Git主要的功能為版本控制，
你可以把變更的資料的儲存成一個版本，
而Commit就是要拿來儲存成一個版本的指令。

- 假設有一個text.py在local端的檔案為以下

```
print("Origin version")
```

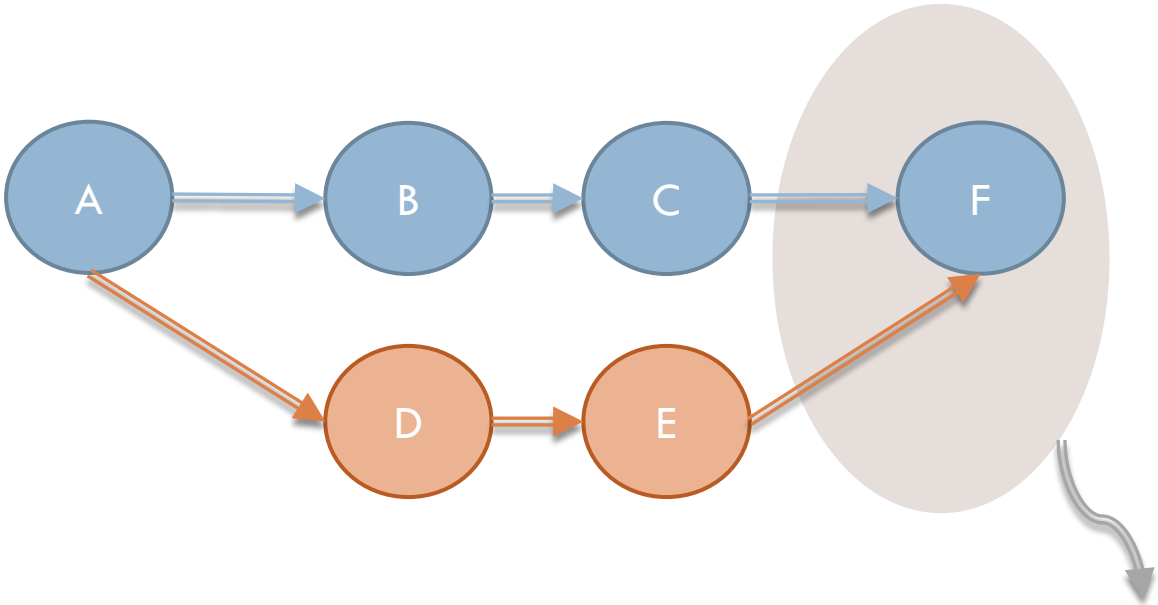
- 若你加了一些code在這個text.py，Git會去比較你加了什麼和之前不一樣的東西

```
print("Origin version")
print("Add something Different")
```

- 現在就可以把它存成一個版本，如果你之後想回來這個版本，就可以隨時回來這個版本，
  現在這樣代表的是在你的電腦建立了這個版本，但是別人的電腦仍然看不到此版本。
  它只存在在你的local端。

常用指令簡易介紹: Merge

在Git中，如果要講簡單一點，
Pull代表的是更新最新的資料，
但其實說清楚一點，它代表的是Fetch + Merge。
而Merge的意思代表的是把兩個東西合在一起
它是以目前的狀況來做比對，而不是利用Commit紀錄一個一個比對



把兩個的支流(可為不同也可為相同支流)合成同一個，稱為Merge

常用指令簡易介紹: Pull(Fetch + Merge)

先前介紹Fetch以及Merge，
則Pull即Fetch+Merge功能(更新遠端狀態，並且合併版本讓現在狀態和遠端的一樣)
Git雖然會版本控制，但它不知道什麼時候需要更新最新的資料
必須要使用者自行操作

假設有一個text.py在local端的檔案為以下

```
print("Origin version")
```

假設在一個text.py在remote端的檔案為以下

```
print("Origin version")
print("test remote")
print("remote version")
```

此時若使用者執行'Pull'指令，
簡單來說，Git會讓Local端的資料會Remote端的資料一樣，來做更新的動作

```
print("Origin version")
print("test remote")
print("remote version")
```

常用指令簡易介紹: Push

在先前已經做過Commit的動作，並且用Pull用成最新的狀態，
現在需要把先前Commit出來的版本傳給Remote端給大家使用

假設有一個text.py在local端的檔案為以下並已經Commit成一個版本了

```
print("Origin version")
print("test remote")
print("Remote version")
print("local version")
print("local version")
```

假設在一個text.py在remote端的檔案為以下

```
print("Origin version")
print("test remote")
print("Remote version")
```

執行pull的指令後，Remote端的資料會變成和local端的資料一致

```
print("Origin version")
print("test remote")
print("Remote version")
print("local version")
print("local version")
```

常用指令簡易介紹: Push

- Force Push ：

  功能: 強值把遠端的資訊改成和local端的資訊一樣

  用途: 在整理commit記錄或使用Rebase等等改變commit分支狀況

  版本:　1.（較不安全版）git push －force

  -> 若有人在你push前有增加commit記錄，

  則會直接被忽略並覆蓋掉她的Commit記錄

  2.（較安全版）　git push －force-with-lease

  -> 若有人在你push前有增加commit記錄，

  會跳出錯誤並不會用你Puhs上去，

  必須Fetch或Pull更新遠端狀態來做確認

常用指令簡易介紹: Switch/Checkout

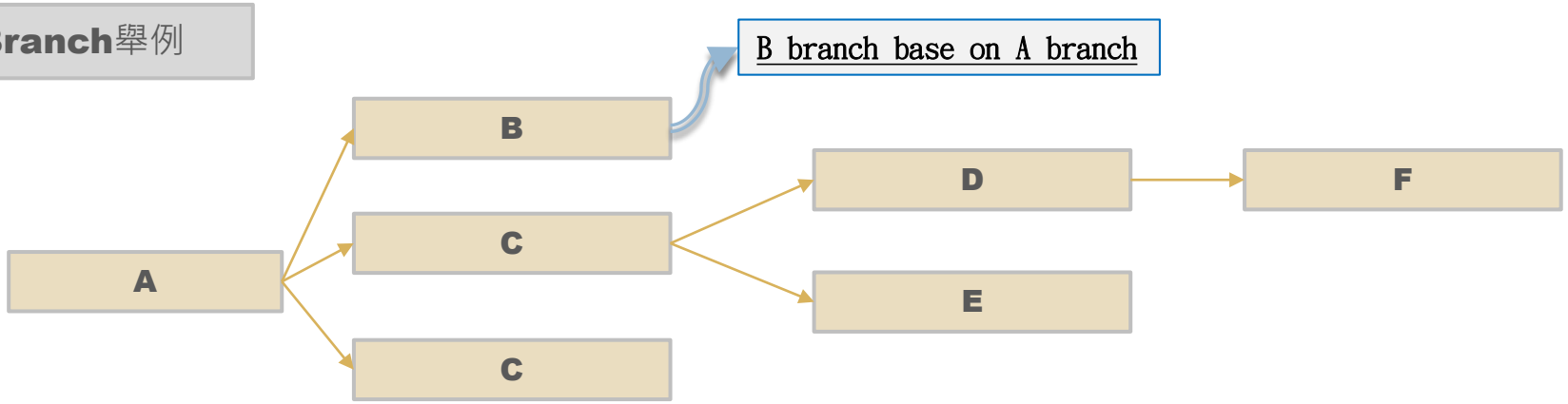
Branch

1. 一個獨立的路徑
2. 每條路徑都會基於某個指定的Branch來繼續延伸(Base On XXX Branch)
3. 在A Branch 撰寫你的Code，結束後Push;而B Branch撰寫相同檔案但不同的code，他們是獨立的，所以不會因為code 不同而有衝突要解決

Switch Branch

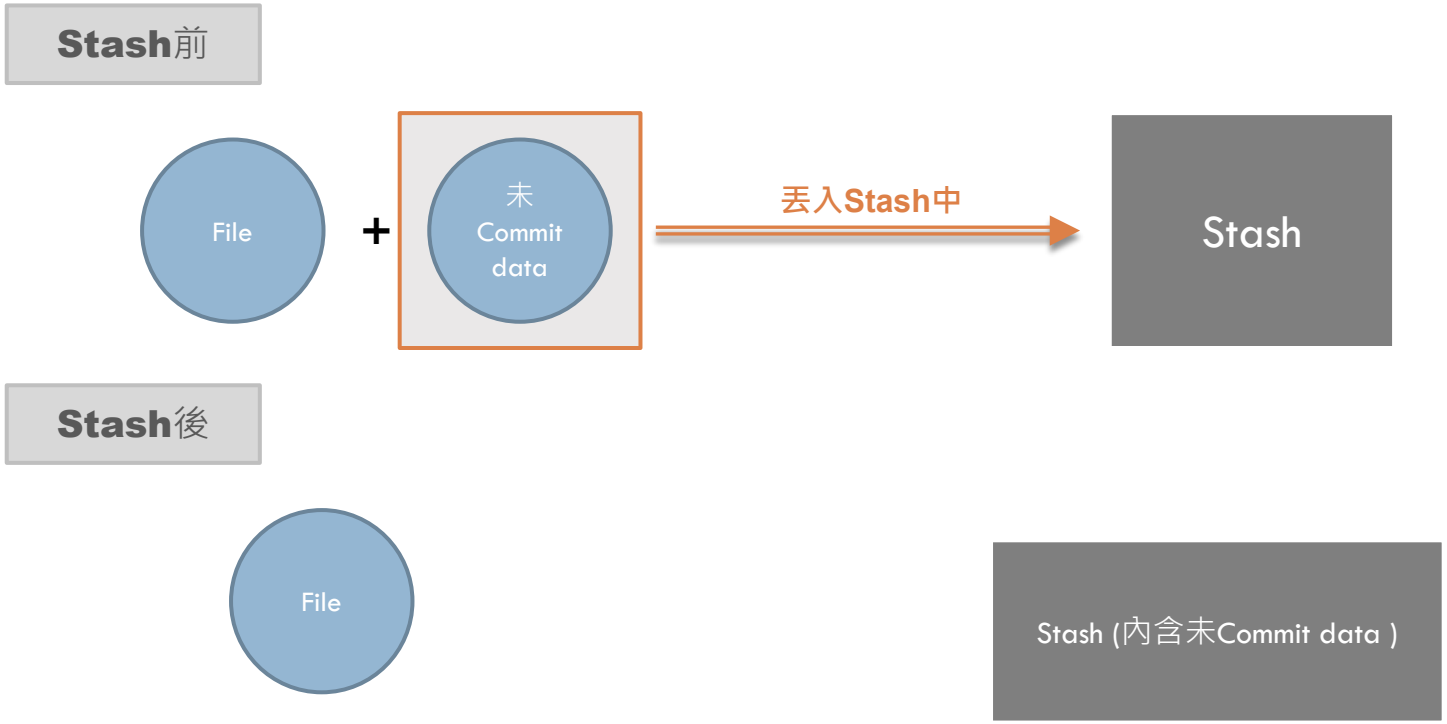
1. 可以切換到不同的branch來撰寫code
2. 在切換的時候，會分要切換到local的還是remote的
3. 若是切到local端的話，代表如果你之前有在local端的這個branch做事，檔案若有變更依然會留著，不會因為切換而不見(但較不建議這樣做)
4. 若是切到remote端的話，建議先執行‘Fetch‘確保為最新資料，再做切換

Branch舉例



常用指令簡易介紹: Stash

可以將Stash解釋成存放暫存器的概念，
如果你想要切換Branch，但你目前已經有變更，又怕這些變更沒有存起來會不小心用丟
你可以Stash起來，並且讓變更資料永久保存。



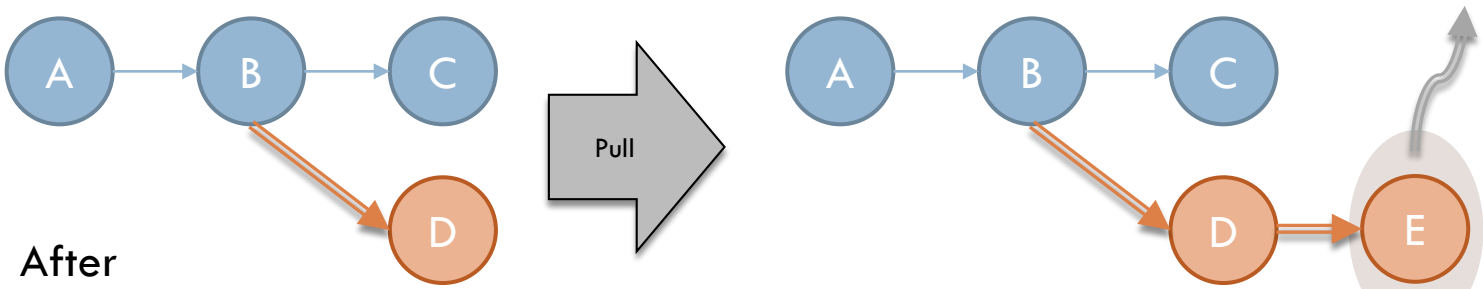
常用指令簡易介紹: Rebase

在Branch中有討論過，Branch需要Base On某個Branch，
但你若要改變Base On的Branch，就必須要‘Rebase‘
你若想要讓Log紀錄乾淨一點，也需要‘Rebase‘

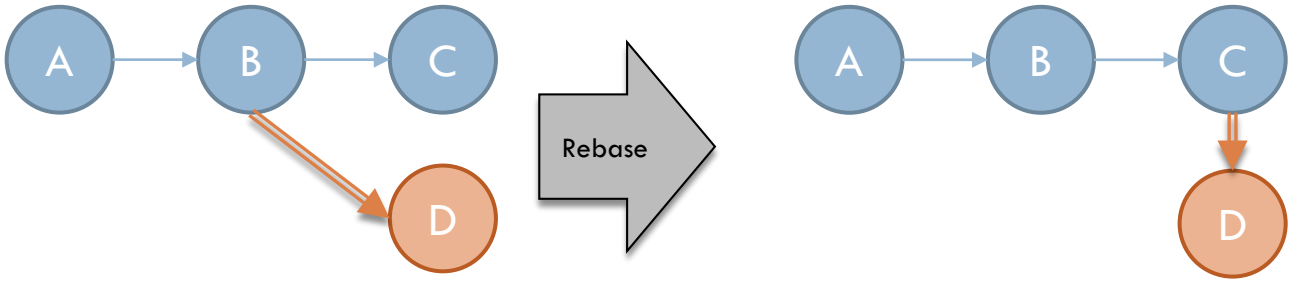
想要讓**Log**紀錄乾淨一點

Before

如果Pull藍色的Branch，就會多一個Merge的Commit紀錄
因為藍色多一個C的Commit，但橘色的Branch沒有這個變更

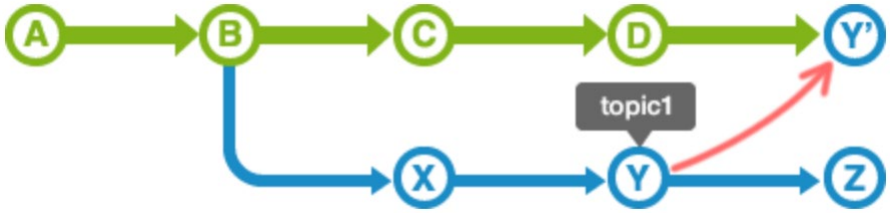


After

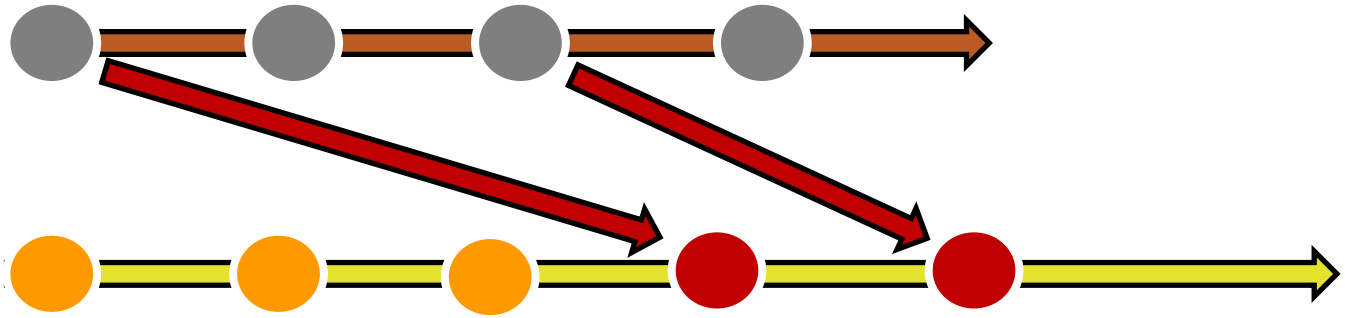


常用指令簡易介紹: Cherry-pick

★ 從其他的分支複製指定的提交，然後導入現在的分支。



★ 在創一個新的Branch時，可以指定需要的commit紀錄拉到此Branch來添加內容



cmd執行Git指令示範及觀念

Set Up Repository

★ 狀況2: 先建立Local的Repository，再讓它連上遠端Repository

- 初始化(檔案夾新增.git檔案夾來儲存GIT data): $ git init

------git init會只是單純創造出一個可以使用git但什麼都沒有設定的檔案夾------

- 加入遠端資料庫：$ git remote add <遠端資料庫簡稱(origin)> <url>

------加入後，若直接打git push則無法push到你想要的資料夾，因為git init 沒有幫你設定(很重要!!!)------

- 需要建立遠端與本地端的連結: $ git push -u <遠端資料庫簡稱(origin)> master

---- < -u >意思同為<--set-upstream>，只需要數定一次即可，下次可以直接使用git push就可以了(很重要!!!) ----

測試git是否可以執行

- 查看現在Git的狀態 :

  $ git status

- 把**指定**檔案加入staging area:

  $ git add <檔案名稱>

  把**全部**檔案加入staging area（會把一些不必要的檔案推上去):

  $ git add .

- Commit到Local Repository :

  $ git commit -m “Commit紀錄”

- Push到遠端資料庫

  $ git push -u <遠端資料庫(origin)> master（未連結Remote端資料庫)

  $ git push （已連結Remote端資料庫）

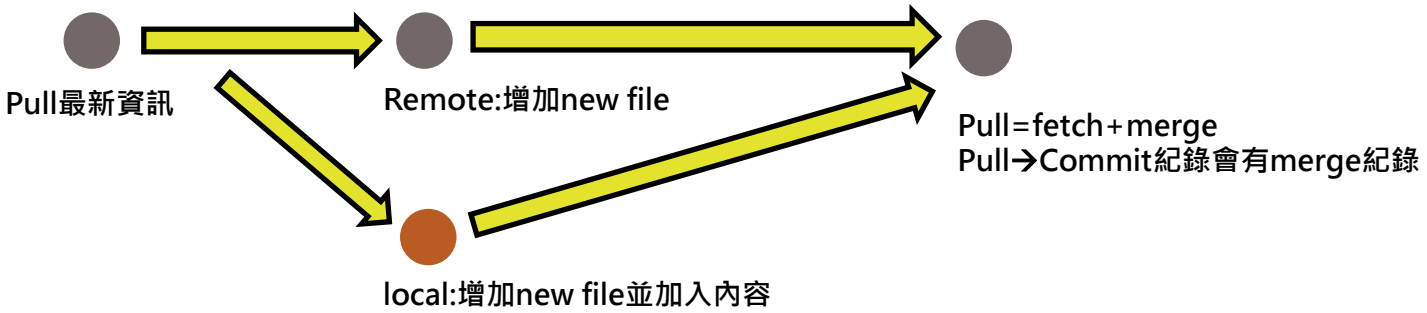
(範例圖片在下一頁)

測試git是否可以執行

```
D:\NTUT\CCS Git\gitclone>cd.>a.txt

D:\NTUT\CCS Git\gitclone>git status
On branch master
Your branch is up to date with 'origin/master'.

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

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

D:\NTUT\CCS Git\gitclone>git add a.txt

D:\NTUT\CCS Git\gitclone>git add .

D:\NTUT\CCS Git\gitclone>git status
On branch master
Your branch is up to date with 'origin/master'.

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


D:\NTUT\CCS Git\gitclone>git commit -m "add a.txt file"
[master 1ee637e] add a.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

D:\NTUT\CCS Git\gitclone>git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

D:\NTUT\CCS Git\gitclone>git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 236 bytes | 236.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone.git
   374000d..1ee637e  master -> master
```

（圖片僅示意，請依照當下狀況判斷）

測試Pull功能




```
D:\NTUT\CCS Git\gitclone>git log
commit 3f293f0097db46fcf0537de42791adfa3578ff8f (HEAD -> master)
Merge: 7f8d469 d60eca5
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 12 10:27:33 2020 +0800

    Merge branch 'master' of https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone into master

commit 7f8d469a219d26f8c3eed47e1d7e370323754f38
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 12 10:25:15 2020 +0800

    add DoingMaster file
```

（圖片僅示意，請依照當下狀況判斷）

利用Log查看目前狀態

★ 可以看到你的Commit紀錄以及你的分支是否為最新狀態

★ Head為一個指標，指向目前狀態

★ HEAD-> master ： 本地端分支master，並現在目前在Master

★ orgin/master ：　　遠端分支最新的狀態

★ orgin/HEAD ：　遠端HEAD指標

```
D:\NTUT\CCS Git\gitclone>git log
commit 1ee637eab2b93cf1fced90cec669885d176b6fa2 (HEAD -> master, origin/master, origin/HEAD)
Author: Jerry <fly990314@gmial.com>
Date:   Sat Oct 10 13:39:12 2020 +0800

    add a.txt file

commit 374000d1a93ab6b8ad92f617702f4467e47a4dfd
Author: <E5><BB><96><E6><98><B1><E7><BF><94> <t109598061@ntut.org.tw>
Date:   Sat Oct 10 13:36:54 2020 +0800

    Delete a.txt

commit 9436ae9ff1fe5370ef46c3c4001d405f14e0783c
Author: Jerry <fly990314@gmial.com>
Date:   Sat Oct 10 13:35:31 2020 +0800

    a.txt
```

（圖片僅示意，請依照當下狀況判斷）

執行Fetch觀察指標變化

★ 同步Remote分支變化(目前圖片狀態為Remot端有人更新資料)

```
D:\NTUT\CCS Git\gitclone>git log
commit 58a223f312a6dc036db12405d3f14463ced9f107 (HEAD -> master, origin/master, origin/HEAD)
Author: <E5><BB><96><E6><98><B1><E7><BF><94> <t109598061@ntut.org.tw>
Date:   Sat Oct 10 16:02:25 2020 +0800

    Delete b.txt
```

```
D:\NTUT\CCS Git\gitclone>git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 1 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), 229 bytes | 3.00 KiB/s, done.
From https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone
   58a223f..f745100  master     -> origin/master
```

```
D:\NTUT\CCS Git\gitclone>git log
commit 58a223f312a6dc036db12405d3f14463ced9f107 (HEAD -> master)
Author: <E5><BB><96><E6><98><B1><E7><BF><94> <t109598061@ntut.org.tw>
Date:   Sat Oct 10 16:02:25 2020 +0800

    Delete b.txt
```

(圖片僅示意，請依照當下狀況判斷)

Branch的操作

- 創造一個新的Branch ：

  $ git branch <Branch名稱>

- 檢查現在的Branch狀態 ：

  $ git branch

- 切換Branch ：

  $ git checkout <Branch名稱>

- 創造且切換到一個新的Branch ：

  $ git checkout -b <Branch名稱>

```
D:\NTUT\CCS Git\gitclone>git branch -d branch1
Deleted branch branch1 (was f745100).

D:\NTUT\CCS Git\gitclone>git branch
  banch2
* master

D:\NTUT\CCS Git\gitclone>git branch -d banch2
Deleted branch banch2 (was 58a223f).

D:\NTUT\CCS Git\gitclone>git branch
* master

D:\NTUT\CCS Git\gitclone>git branch branch1

D:\NTUT\CCS Git\gitclone>git branch
  branch1
* master

D:\NTUT\CCS Git\gitclone>git checkout branch1
Switched to branch 'branch1'

D:\NTUT\CCS Git\gitclone>git branch
* branch1
  master

D:\NTUT\CCS Git\gitclone>git checkout -b branch2
Switched to a new branch 'branch2'

D:\NTUT\CCS Git\gitclone>git branch
  branch1
* branch2
  master
```

（圖片僅示意，請依照當下狀況判斷）

Branch的操作

- 創造一個新的Branch 並綁定Remote端Push的分支:

  $ git branch <Branch名字>

  $ git push -u < remote名稱> <branch名字>

```
D:\NTUT\CCS Git\gitclone>git branch
* master

D:\NTUT\CCS Git\gitclone>git branch branch1

D:\NTUT\CCS Git\gitclone>git push -u origin branch1
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for branch1, visit:
remote:   https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone/-/merge_requests/new?merge_r
equest%5Bsource_branch%5D=branch1
remote:
To https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone.git
 * [new branch]      branch1 -> branch1
Branch 'branch1' set up to track remote branch 'branch1' from 'origin'.
```

(圖片僅示意，請依照當下狀況判斷)

Branch的操作

- 刪除Branch ：

  （方法1） $ git branch -d <Branch名稱>

  （方法2） $ git checkout <Master>

  $ git branch -d <Branch名稱>

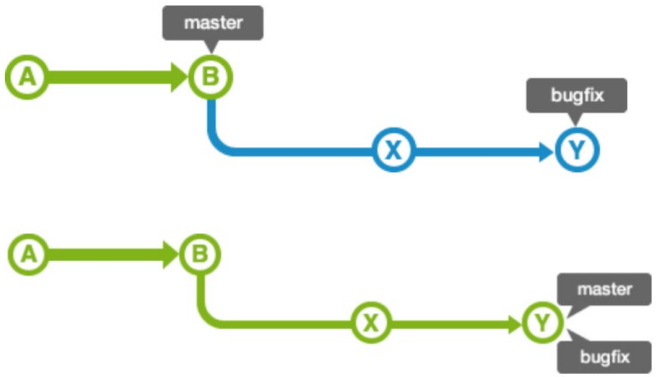
  $ git push < remote名稱> <空的>:<分支名字>

```
D:\NTUT\CCS Git\gitclone>git checkout master
Already on 'master'

D:\NTUT\CCS Git\gitclone>git branch -d branch1
Deleted branch branch1 (was eee5cad).

D:\NTUT\CCS Git\gitclone>git push origin :branch1
To https://css-gitlab.csie.ntut.edu.tw/109598061/gitclone.git
 - [deleted]         branch1
```

（圖片僅示意，請依照當下狀況判斷）

Merge操作變化

★ fast-forward ： 修改後Commit紀錄相同，但之後的Commit紀錄會變得更複雜。



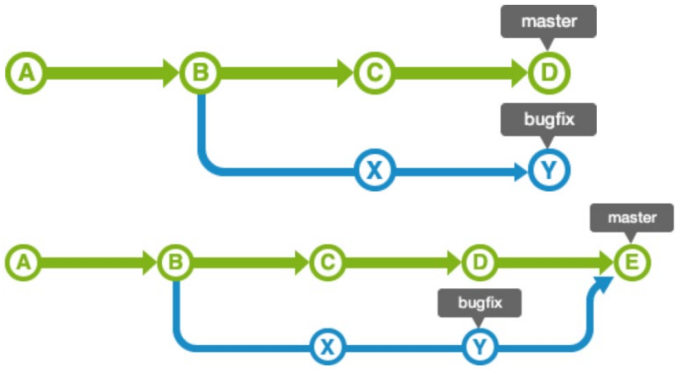
- STEP1:創一個新的Branch <branch1>
- STEP2:在Branch1中加入branch1.txt
- STEP3:在Branch1 中commit
- STEP4:再切到master發現沒有 branch1.txt
- STEP4:此時在master做merge
- STEP5:此時才會發現有branch1.txt

```
D:\NTUT\CCS Git\gitclone>git checkout -b branch1
Switched to a new branch 'branch1'

D:\NTUT\CCS Git\gitclone>git branch
* branch1
  master

D:\NTUT\CCS Git\gitclone>cd.>branch1.txt

D:\NTUT\CCS Git\gitclone>git add .

D:\NTUT\CCS Git\gitclone>git commit -m "add branch1.txt"
[branch1 a045139] add branch1.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 branch1.txt

D:\NTUT\CCS Git\gitclone>git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

D:\NTUT\CCS Git\gitclone>git merge branch1
Updating f745100..a045139
Fast-forward
 branch1.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 branch1.txt
```

(圖片僅示意，請依照當下狀況判斷)

Merge操作變化

★ non fast-forward



- STEP1 ： 在Master新增AddInMaster.txt並commit
- STEP2 ： 切到Branch1
- STEP3 ： 在Branch1新增AddInBranch1.txt並commit
- STEP4 ： 再切到Master，發現沒有AddInBranch1.txt
- STEP5 ： 此時在master做merge
- STEP6 ： 才會有AddInBranch1.txt

```
D:\NTUT\CCS Git\gitclone>cd.>AddInMaster.txt

D:\NTUT\CCS Git\gitclone>git add .

D:\NTUT\CCS Git\gitclone>git commit -m "add AddInMaster.txt file"
[master 05cda7d] add AddInMaster.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 AddInMaster.txt

D:\NTUT\CCS Git\gitclone>git checkout branch1
Switched to branch 'branch1'

D:\NTUT\CCS Git\gitclone>cd.>AddInBranch1.txt

D:\NTUT\CCS Git\gitclone>git add .

D:\NTUT\CCS Git\gitclone>git commit -m "add AddInBranch1.txt file"
[branch1 d9aa219] add AddInBranch1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 AddInBranch1.txt

D:\NTUT\CCS Git\gitclone>git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

D:\NTUT\CCS Git\gitclone>git merge branch1
Merge made by the 'recursive' strategy.
 AddInBranch1.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 AddInBranch1.txt

D:\NTUT\CCS Git\gitclone>git log
commit a4aa410ec12d1891da227f8e3dc610284a904455 (HEAD -> master)
Merge: 05cda7d d9aa219
Author: Jerry <fly990314@gmial.com>
Date:   Sun Oct 11 17:58:43 2020 +0800

    Merge branch 'branch1' into master

commit d9aa2192563d4f3632fc423cbc10293f0a18b7d1 (branch1)
Author: Jerry <fly990314@gmial.com>
Date:   Sun Oct 11 17:57:25 2020 +0800

    add AddInBranch1.txt file
```

(圖片僅示意，請依照當下狀況判斷)

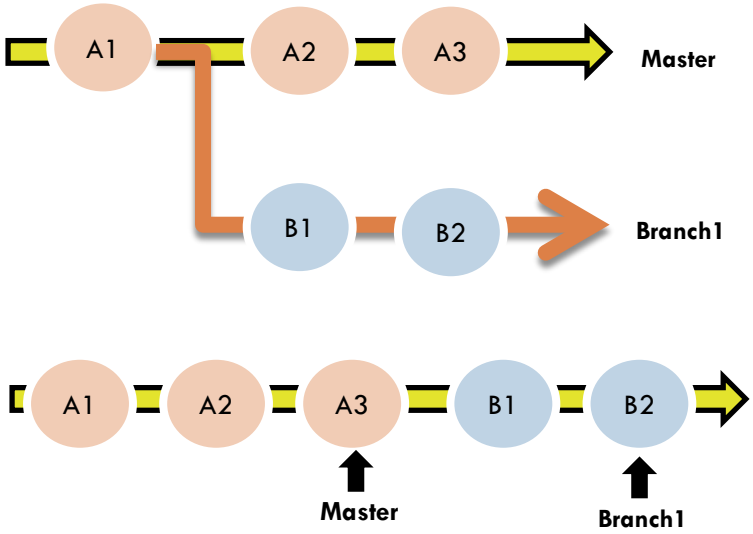
Stash指令集

★ 若無法立刻切換到其他分支，可以使用stash暫存功能，就不需要立即去Commit

★ Stash會將目前的Work Direction和Stage Area的檔案暫存起來；等需要再取出。

$ git stash list	->顯示目前Stash的清單
$ git stash save	->把已追蹤的檔案建立暫存版(save可省略)
$ git stash save -u	->把已追蹤以及未追蹤的檔案建立暫存版(save可省略)
$ git stash save -u “Message”	->建立暫存以及添加註解(save可省略)
$ git stash pop	->取回最近一筆的暫存
$ git stash apply	->取回最近一筆的暫存(但此暫存還會留在清單上)
$ git stash apply “stash@{1}”	->取回指定暫存
$ git stash drop “stash@{1}”	->刪除指定暫存
$ git stash clear	->刪除所有暫存檔0

Rebase未衝突情形



- STEP1 ： 切換到Branch1中

  $ git checkout branch1

- STEP2 ： 在Branch1中對Master 做rebase

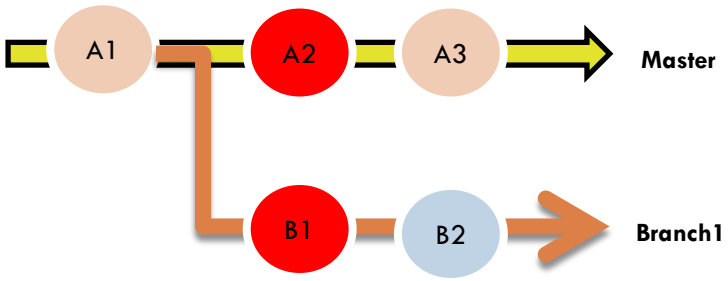
  $ git rebase master

- STEP3 ： git log 檢查是否成功

  $ git log

```
D:\NTUT\CCS Git\gitclone>git rebase master
Successfully rebased and updated refs/heads/branch1.

D:\NTUT\CCS Git\gitclone>git log
commit 7ba6d3ed8eaa3735534ceffa2304aebe7cc0c7ab (HEAD -> branch1)
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 16:11:53 2020 +0800

    add B2.txt

commit 828e34b501027321d7eacb63bb6142691b2d251c
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 16:10:37 2020 +0800

    add  B1.txt

commit 8e106b999e78b09c62a8b2d21b1d252157a6de18 (master)
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 15:56:00 2020 +0800

    add A3.txt

commit 0157d5f492f9c98d796ea6de9d1e49676783eadd
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 15:55:03 2020 +0800

    add A2.txt

commit 98f7d6fcf9966cf9705f187e40ef77fd04b838c4
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 15:53:40 2020 +0800

    add A1.txt
```

Rebase已衝突情形




```
D:\NTUT\CCS Git\gitclone>git rebase master
error: could not apply 1e747a6... add A2_change.txt
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 1e747a6... add A2_change.txt
CONFLICT (add/add): Merge conflict in A2_change.txt
Auto-merging A2_change.txt

D:\NTUT\CCS Git\gitclone>git add A2_change.txt

D:\NTUT\CCS Git\gitclone>git rebase --continue
```

- STEP1 ： 切換到Branch1中

  $ git checkout branch1

- STEP2 ： 在Branch1中對Master 做rebase

  $ git rebase master

- STEP3 ： 發現內容有Conflict並且解決後重新rebase

  $ git add <修改檔案名稱>

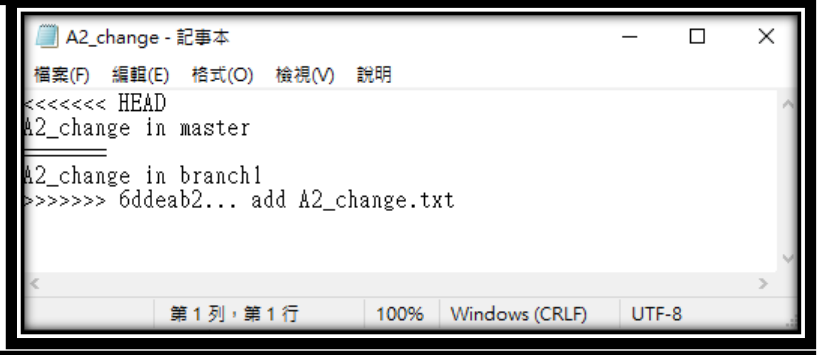
  $ git rebase –continue

  若想取消rebase的話，則可輸入

  $ git rebase --abort

- STEP4 ： 等解決後

  可以用VIM更改Commit名稱(branch1的)




```
add A2_change.txt and resolve the conflict

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# interactive rebase in progress; onto 479af9b
# Last command done (1 command done):
#    pick 1e747a6 add A2_change.txt
# No commands remaining.
# You are currently rebasing branch 'branch1' on '479af9b'.
#
# Changes to be committed:
#       modified:   A2_change.txt
#
```

Rebase –i 變更commit記錄

- STEP1 : 使用 $git log 查看現有的Commit紀錄，

  可以使用HEAD指標來標示想移到哪個Commit紀錄中或也可以使用commit id前六碼去做移動。

  (‘^‘後面跟一個數字代表第幾個父提交，EX: HEADˆ or HEADˆ1都是指HEAD的前一個，

  ~<n>代表連續n個ˆ，EX: HEAD~2or HEAD~~指HEAD的前兩個)

HEAD →

HEAD~1 →

HEAD~2 →

HEAD~3 →

```
D:\NTUT\CCS Git\gitclone>git log
commit 57032a9156df3daf83c1c1fc9eed9c08921c9fd3 (HEAD -> master)
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:26:20 2020 +0800

    add A3.txt

commit 6b6f920e494d2036cd7ba2887a7bc3e52a45dde5
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:47 2020 +0800

    add A2.txt

commit 98cb79c17972d46c74d7f304dc420595a5f6090c
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:19 2020 +0800

    add A1.txt

commit aea60bb0d19c96376e5034d58f5c428c427401e3
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 19:47:24 2020 +0800

    reset
```

Rebase –i 變更commit記錄

- STEP2 : 輸入 $git rebase -i HEAD~2 ，使HEAD移動到 add A1.txt 的Commit紀錄，在HEAD前面的Commit紀錄可以讓你用VIM介面編輯器讓你來做編輯的動作。

```
pick 6b6f920 add A2.txt
pick 57032a9 add A3.txt

# Rebase 98cb79c..57032a9 onto 98cb79c (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Rebase –i 合併commit記錄

- STEP3(合併)：VIM從一般指令模式切換到編輯模式→輸入a，ｉ，ｏ，ｒ 進入編輯模式，
  若把第二行的pick改成squash可把兩個紀錄做合併(在此用合併當範例)；
  按Esc則跳出編輯模式，並輸入 :wq　來做儲存並退出的動作

Squash →

```
pick 6b6f920 add A2.txt
pick 57032a9 add A3.txt

# Rebase 98cb79c..57032a9 onto 98cb79c (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Rebase –i 合併commit記錄

- STEP4（合併）：接下來對commit紀錄來做更改，
  可以把 ”add A2.txt”和 “add A3.txt”的Commit紀錄合併成
  ”add A2.txt and A3.txt ”，再將它儲存起來。

```
# This is a combination of 2 commits.
# This is the 1st commit message:

add A2.txt

# This is the commit message #2:

add A3.txt

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Oct 21 20:25:47 2020 +0800
#
# interactive rebase in progress; onto 98cb79c
# Last commands done (2 commands done):
#    pick 6b6f920 add A2.txt
#    squash 57032a9 add A3.txt
# No commands remaining.
# You are currently rebasing branch 'master' on '98cb79c'.
#
# Changes to be committed:
#       new file:   A2.txt
#       new file:   A3.txt
#
~
```

```
# This is a combination of 2 commits.
# This is the 1st commit message:

add A2.txt and A3.txt

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
```

Rebase –i 合併commit記錄

- STEP5(合併)：使用 $git log 查看目前的Commit紀錄，
  會發現有”add A2.txt and A3.txt”的變更紀錄。

之前的HEAD~2 →

```
commit 12e63643f3deb9e734adfa2b7182479d5ef153a4 (HEAD -> master)
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:47 2020 +0800

    add A2.txt and A3.txt

commit 98cb79c17972d46c74d7f304dc420595a5f6090c
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:19 2020 +0800

    add A1.txt

commit aea60bb0d19c96376e5034d58f5c428c427401e3
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 19:47:24 2020 +0800

    reset
```

Rebase –i commit記錄

- STEP3(編輯)：若把第一行的pick改成edit可把兩個紀錄做合併(在此用合併當範例)；
  按Esc則跳出編輯模式，並輸入 :wq 來做儲存並退出的動作。

edit →

```
pick 6b6f920 add A2.txt
pick 57032a9 add A3.txt

# Rebase 98cb79c..57032a9 onto 98cb79c (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

Rebase –i 編輯commit記錄

- STEP4（編輯）：跳出VIM編輯器後，它會 checkout 到欲修改之提Commit紀錄。

  輸入$ git commit --amend -m “(要變更之字串)“；

  若打$ git commit --amend ， 則會跳到VIM給你編輯，

  編輯完輸入 $ git rebase –continue，即可跳到log查看Commit紀錄

```
add A2.txt

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Oct 23 13:07:40 2020 +0800
#
# interactive rebase in progress; onto 937b239
# Last command done (1 command done):
#    edit 3d69acf add A2
# Next command to do (1 remaining command):
#    pick e6c03a7 add A3
# You are currently splitting a commit while rebasing branch 'master' on '937b239'.
#
# Changes to be committed:
#       new file:   A2.txt
#
~
~
```

切換到指定的版本

- 使用Reset指令使HEAD回到指定的Commit紀錄

  $ git reset <Commit紀錄位置>

  (--mixed → 暫存區的檔案丟掉；工作目錄的檔案不變。)

  (--soft → 工作目錄和暫存區檔案不變，只有HEAD移動而已。)

  (--hard → 工作目錄和暫存區的檔案都丟掉。)

```
D:\NTUT\CCS Git\gitclone>git log
commit f8c7ffa56c9662ba9a2a1d99164df35af99f3c59 (HEAD -> master)
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:47 2020 +0800

    add files

commit 98cb79c17972d46c74d7f304dc420595a5f6090c
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 20:25:19 2020 +0800

    add A1.txt

commit aea60bb0d19c96376e5034d58f5c428c427401e3
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 19:47:24 2020 +0800

    reset
```

```
D:\NTUT\CCS Git\gitclone>git reset HEAD~2

D:\NTUT\CCS Git\gitclone>git log
commit aea60bb0d19c96376e5034d58f5c428c427401e3 (HEAD -> master)
Author: Jerry <fly990314@gmial.com>
Date:   Wed Oct 21 19:47:24 2020 +0800

    reset

commit 479af9bbde64cd28199dae963f5b99bd1ec95142
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 18:34:00 2020 +0800

    add A2_change.txt

commit 9e7596e3fbbbf6ea3c77562dd64082f22055e9c1
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 18:02:51 2020 +0800

    add A1.txt

commit ad674dd28666ab365fdb5fba349fb53400f2a949
Author: Jerry <fly990314@gmial.com>
Date:   Mon Oct 19 16:34:47 2020 +0800

    reset
```

切換到之前的版本但仍存留變更

- 使用Revert指令不會使HEAD移動，
  只會把指定的Commit紀錄取消掉，
  並把之前的Commit資料復原，
  並且和現在的狀態再Commit一次。

  $ git revert HEAD~

  $ git revert HEAD~ --no-edit

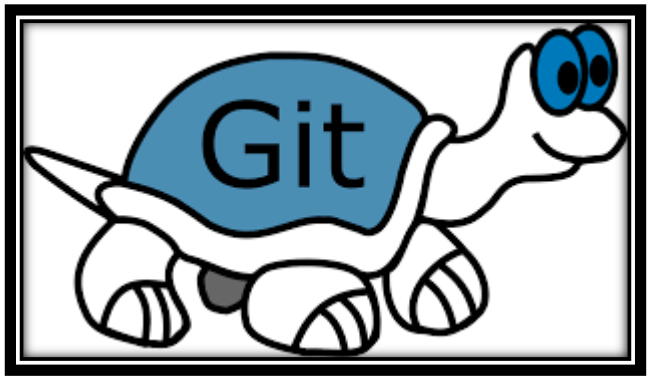
- Reset和Revert的差別在於:
  使HEAD往前和往後。

```
D:\NTUT\CCS Git\gitclone>git revert HEAD~~ --no-edit
Removing A1.txt
[master b61d3bd] Revert "add A1.txt"
 Date: Thu Oct 22 21:02:04 2020 +0800
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 A1.txt

D:\NTUT\CCS Git\gitclone>git log
commit b61d3bd03f049d29cf6f6ccc9b9fc394b7993bd4 (HEAD -> master)
Author: Jerry <fly990314@gmial.com>
Date:   Thu Oct 22 21:02:04 2020 +0800

    Revert "add A1.txt"

    This reverts commit 93f17cf5dc94e00323f6784f9df8897c4e13abbe.

commit f996833b128c0c64ebc027db678206b29337bb94
Author: Jerry <fly990314@gmial.com>
Date:   Thu Oct 22 21:00:59 2020 +0800

    add A3.txt

commit 33e332e0ee7b6c220a8a886e5f693d7183bec8a0
Author: Jerry <fly990314@gmial.com>
Date:   Thu Oct 22 21:00:22 2020 +0800

    add A2.txt

commit 93f17cf5dc94e00323f6784f9df8897c4e13abbe
Author: Jerry <fly990314@gmial.com>
Date:   Thu Oct 22 20:59:59 2020 +0800

    add A1.txt
```

Remarks:

Tortoised Git

常用Git工具



僅支援Windows 系統(以該軟體來介紹)

Git Tortoise

支援Windows 以及 MacOS系統

Atlassian SourceTree

SourceTree

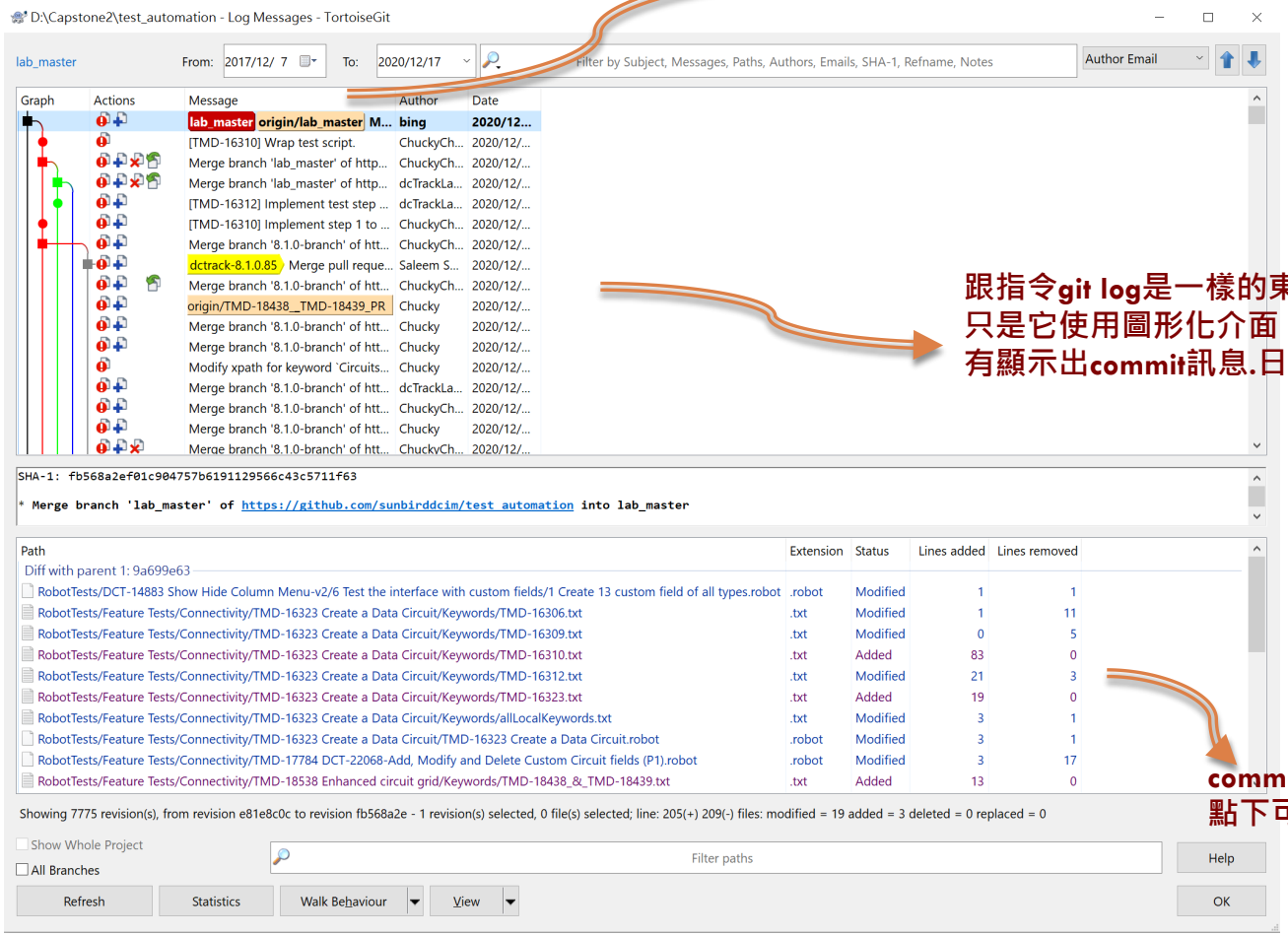
Remarks (Tortoised Git) :

- 在你的git repository，會發現你到任何一檔案夾按下右鍵，會多出現三個沒看過的選項。
- Git Synchronization : 較少去使用，可在這裡查看未Push的Commit的變更。
- Git Commit -> “master” … : 要Commit時，
  進入裡面去新增Commit Message和檢查變更內容。
- TortoiseGit : 裡面有許多Git的功能，都以圖形化了，
  只要選擇你想要的功能，
  Tortoised就會幫你執行一連串的指令。
  例如 : 基本的功能Pull, Push, …等等

Remarks (Tortoised Git) :

---***Show log***---

標示Head指標，告訴你目前的狀態



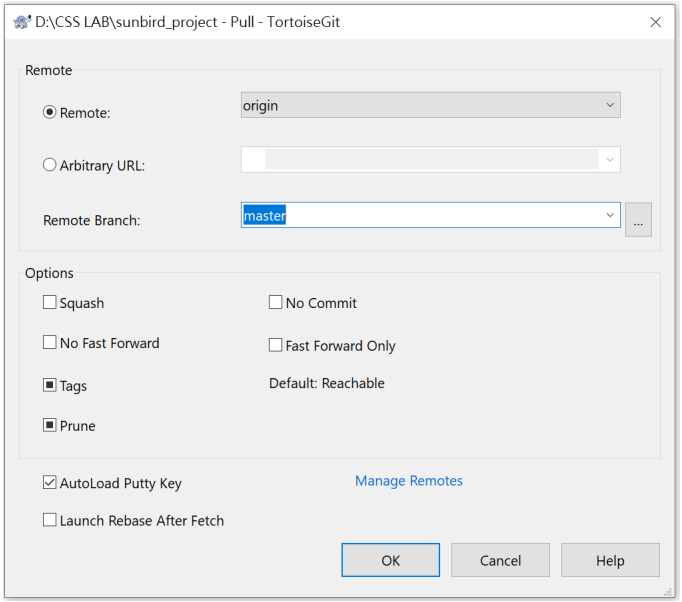
跟指令**git log**是一樣的東西，
只是它使用圖形化介面，變得更簡單更容易懂，
有顯示出**commit**訊息.日期.使用者.....等等

commit紀錄變更的檔案內容，
點下可以詳細看到檔案變更前後的差距

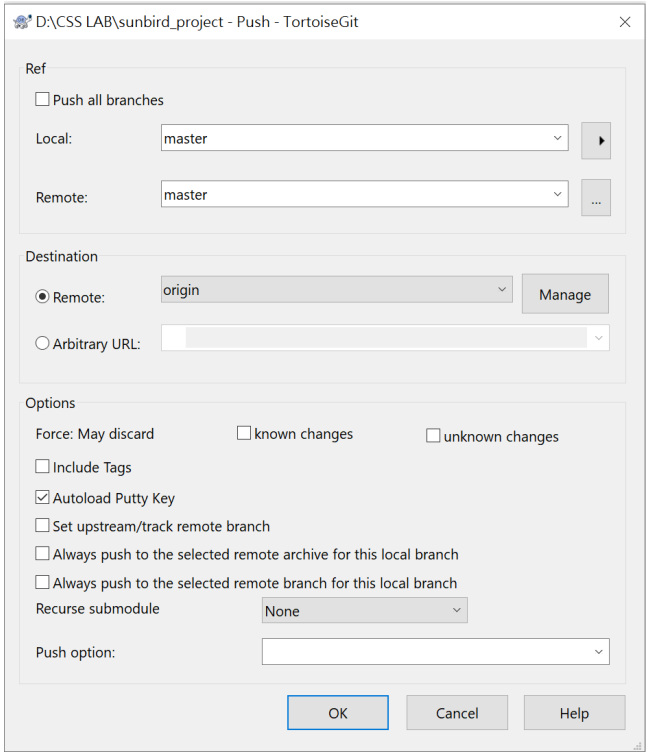
Remarks (Tortoised Git) :

- 基本的Pull Push介面

---***Pull***---

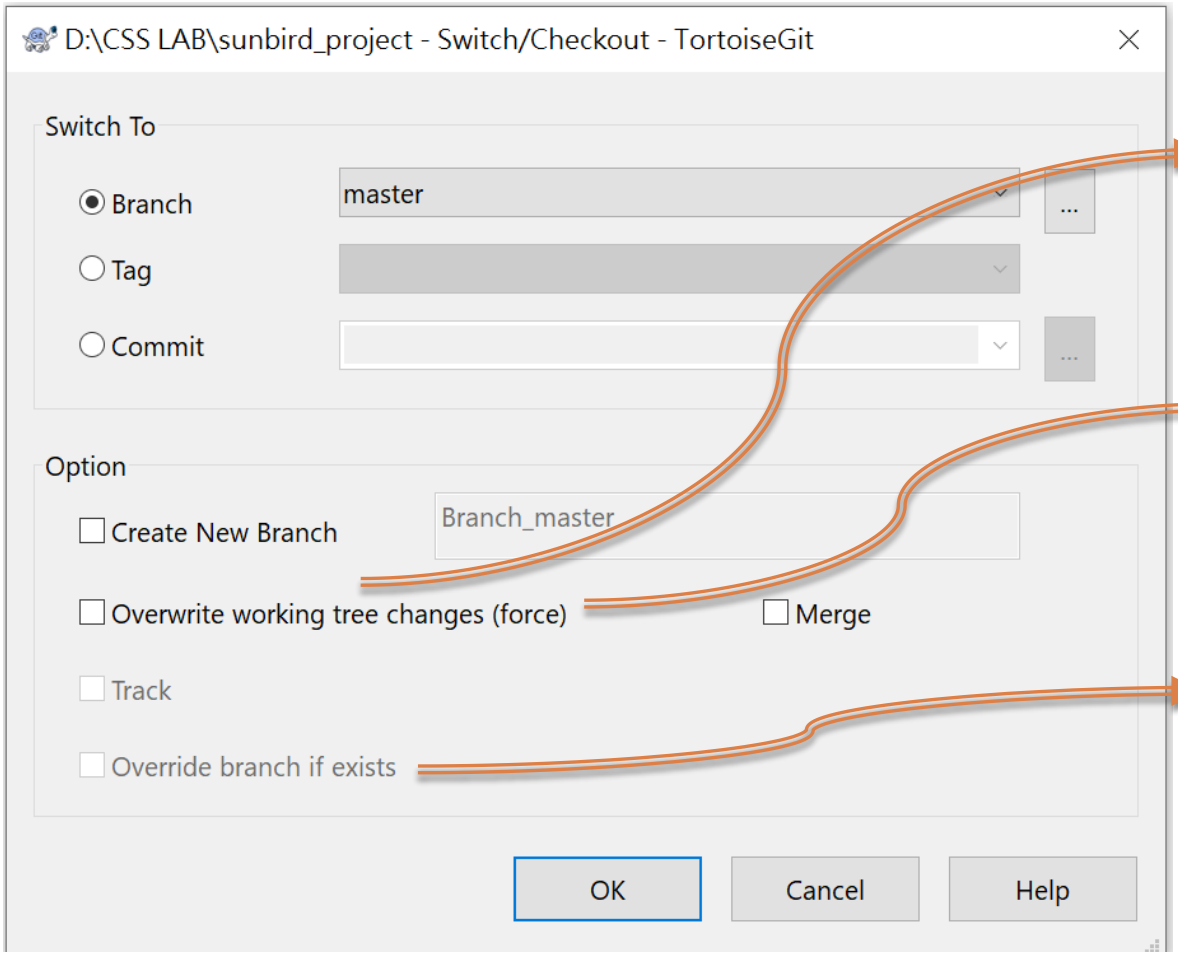


---***Push***---



Remarks (Tortoised Git) :

---***Switch branch***---



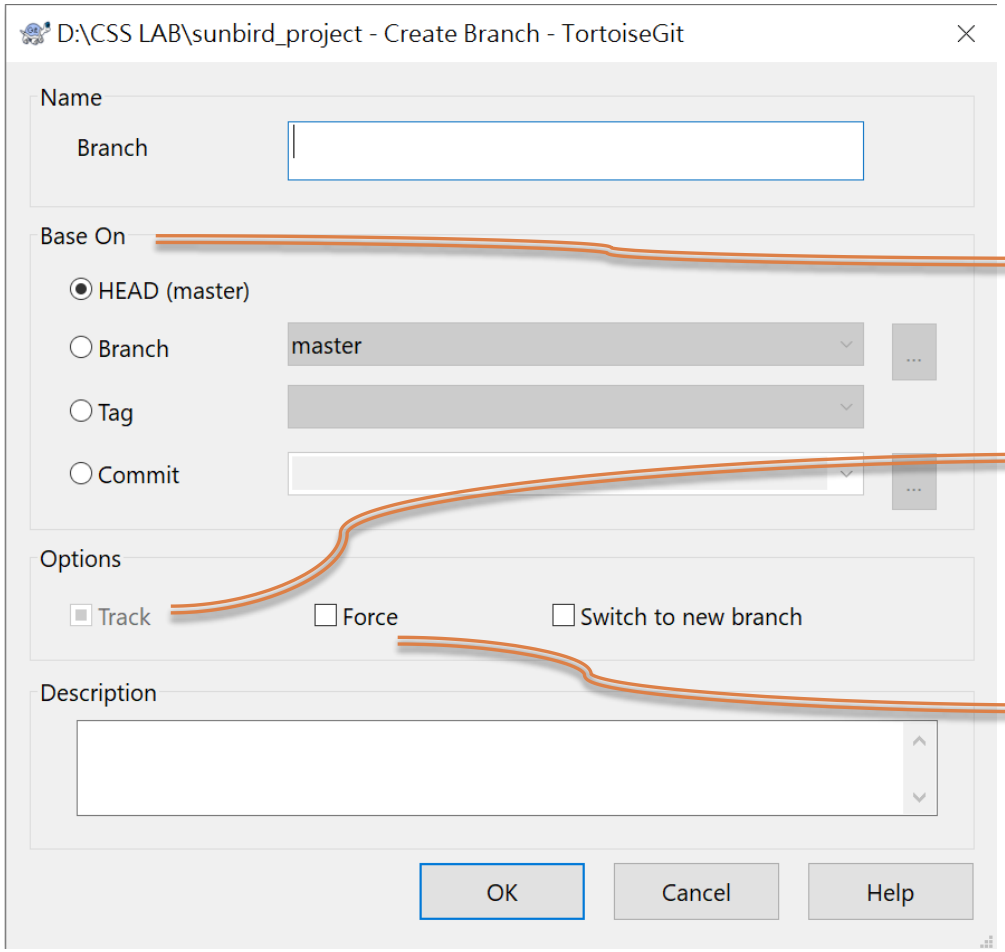
如果你切換成remote端的Branch時，
它需要先創出和remote端一樣的新的Branch
若果切換成local端的Branch的話，
就可以不用勾選這個

類似force push的效果，
對working tree(log)做force的動作，
建議不要使用，較為安全。

如果在Local Repository有一個一樣的banch，
則Override。
一般來說有Switch成remote端的Branch，
就需要勾選這個

Remarks (Tortoised Git) :

---*Create Branch***---**



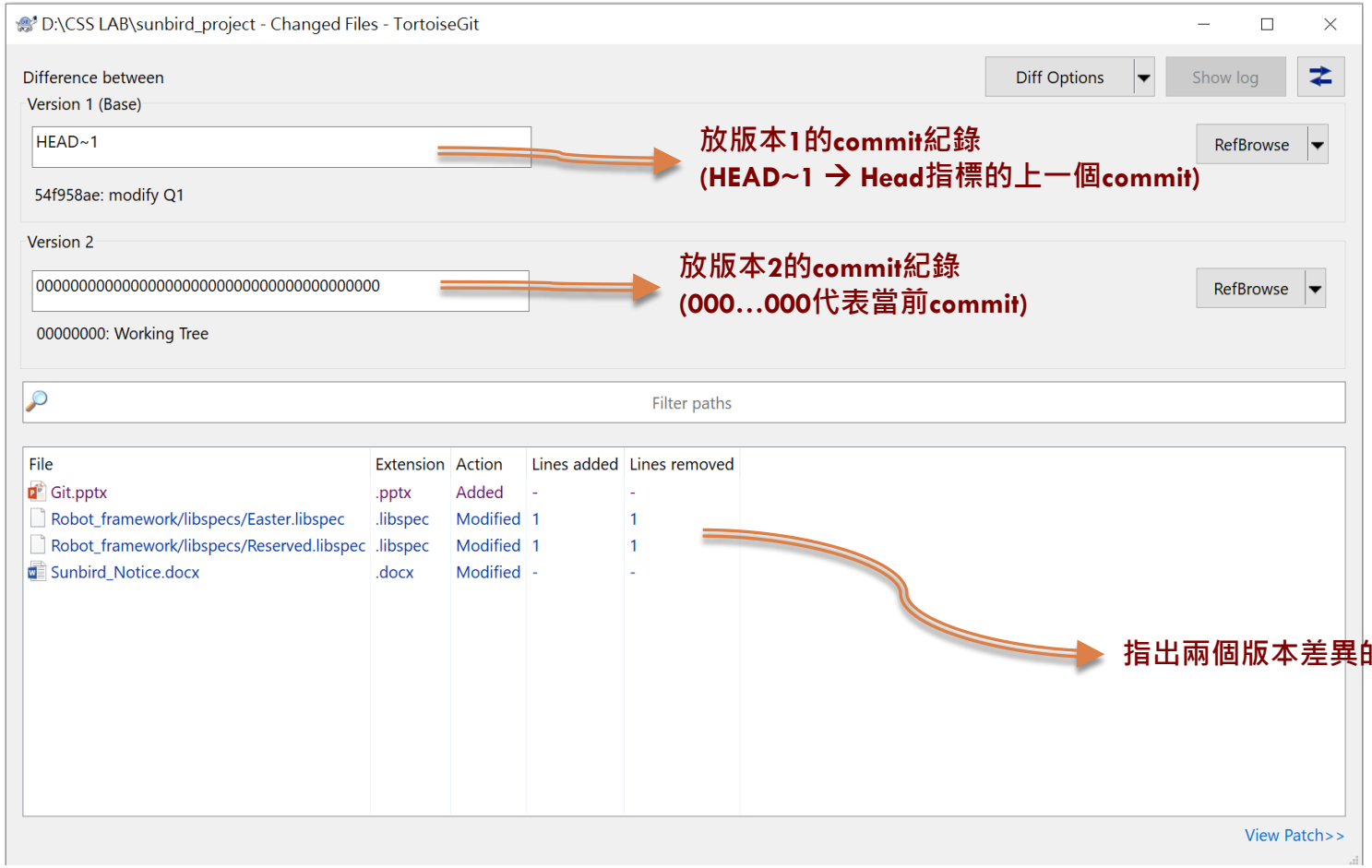
可以參考Rebase的觀念，
選擇你要base的branch。

若勾起來，
push的地方會預設成你Base的Branch，
通常是不會勾選。

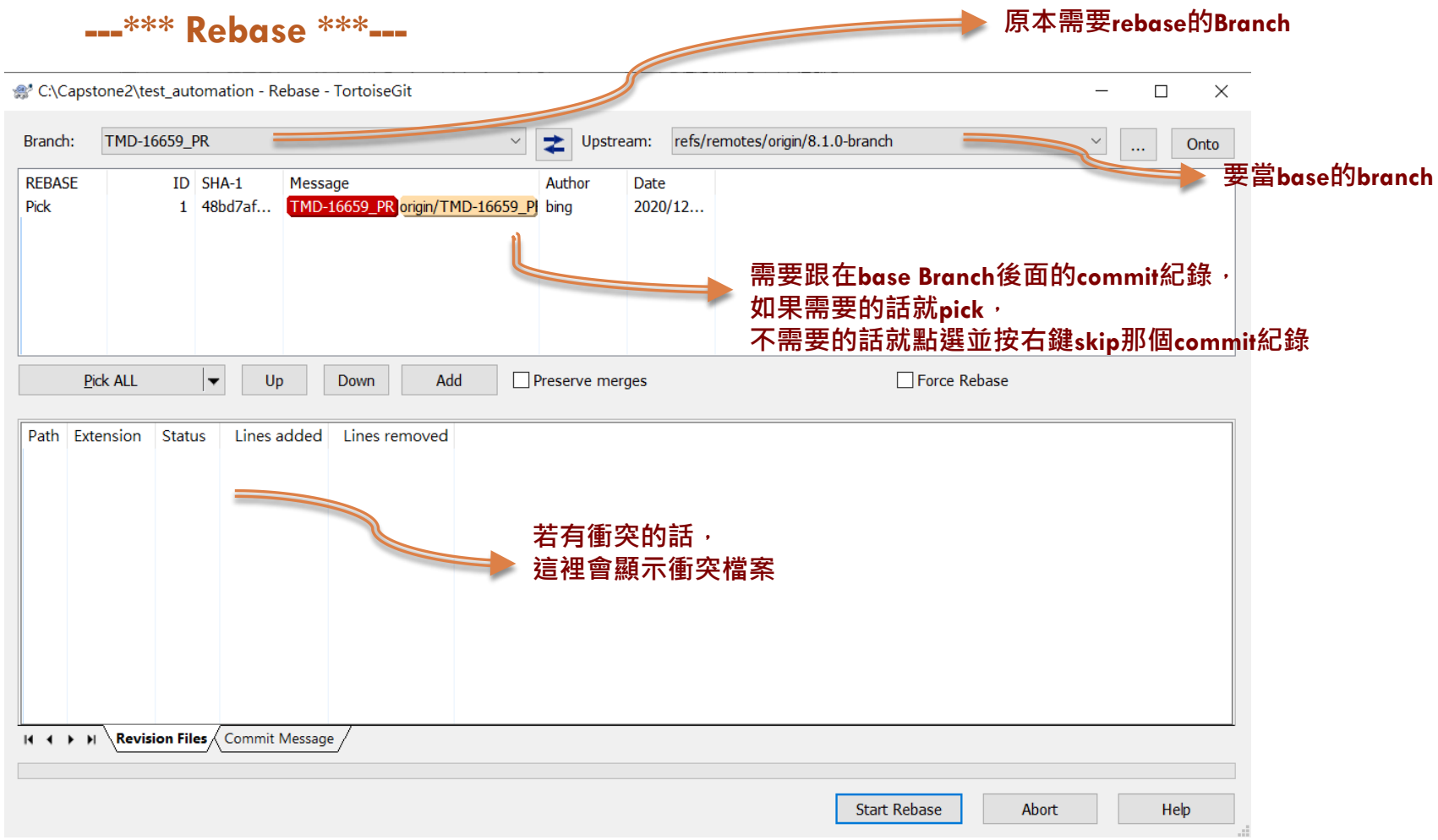
類似force push的效果，
如果你有同樣名稱的Branch的話，
就會被新的且空的Branch 取代掉。

Remarks (Tortoised Git) :

---***Diff with previous version***---

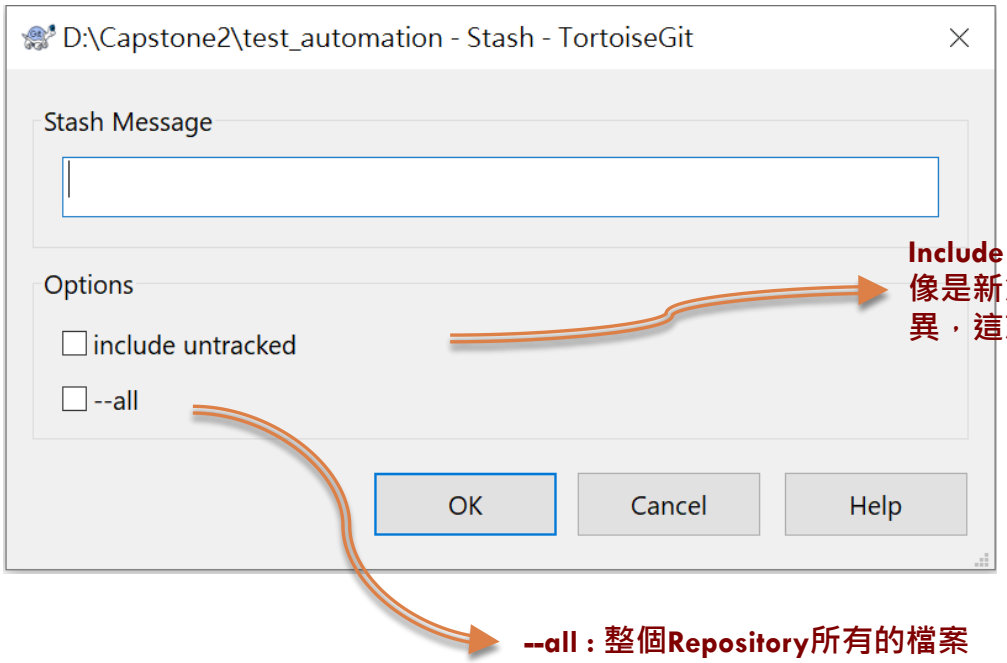


Remarks (Tortoised Git) :



Remarks (Tortoised Git) :

---*** Stash ***---



Include untracked :
像是新創的檔案，git沒有做抓取這個檔案的變更差異，這就叫untracked

--all : 整個Repository所有的檔案