安裝 GIT

- 首先安裝 GIT 本人
  - windows 用戶請參考此處。
  - mac 用戶使用終端機介面，輸入指令 **brew install git**，若還沒安裝 brew 請參考此處。
  - ~~可以的話換 mac 吧！~~
- 此手札範例全程皆以指令練習。
- 一般 IDE 或是相關 TOOL 會簡化 GIT 操作流程，但基本指令的概念有學習、練習的必要，可以的話配合練習服用效果更佳。
- 最常用的兩個指令
  - **確認狀態：$ git status 取得 GIT 狀態，或是合理的 GIT 流程提示。**
  - **內建文件：$ git help <command>，使用上下方向鍵查看文件，q 為離開。**
- 上緊發條，我們開始吧！

一般開發標準作業程序

依據環境以及程式語言的差異，步驟可能稍有不同。

請重新編排步驟，並堅定的按步驟執行。

大流程請參考如下步驟：

1. 在工單系統上建立新的工單，並給予工單編號。
2. 在本機儲存庫以 **工單編號 - 描述** 的格式建立新的分支（**branch**）。
3. 進行工單上的描述工作，而且是**只做描述的內容**。
4. 測試工作成果，確認工作完成且正確，並確保可以在開發環境上通過工單上描述的 QA 測試內容以及驗收標準。
5. 現在本機的工作目錄為 **dirty** 的狀態，可能包含了新增或是修改的內容。將檔案加入本機儲存的整備區（**staging area**）。
6. 將整備區的內容記錄（**commit**）至本機儲存庫。
7. 將記錄推送（**push**）至遠端的備份主機，一般來說也會是記錄工單的主機。ex：GitLab、Bitbucket、GitHub。
8. 工單應標記為**已解決（resloved）**，但還不是**已關閉（closed）**。
9. 一旦對成果完全滿意，將工單分支合併回主分支（可能是 master or 開發性分支），並將更新過的分支推送回代管系統。
10. 再次測試工作成果。
11. 更新工單系統，**關閉**該工單，刪除遠端以及本機的功能性分支。

GIT 專案初始化

取得遠端複本

1. 建立工作區資料夾。
2. **$ git clone <remote url>**。
3. **clone** 指令會拉回該遠端儲存庫所有的檔案、歷史記錄以及設定預設的遠端儲存庫連線資訊。
4. 教學用儲存庫連結請參考此處。

現有專案納入 GIT 控管

1. 專案目錄下 **$ git init**，此時專案目錄下的檔案並不會加入儲存庫。
2. 可輸入 **$ git status** 取得 GIT 狀態，或是合理的 GIT 流程提示。
3. 一次將所有的檔案納入 GIT 控管可輸入 **$ git add --all**。
4. 再一次輸入 **$ git status** 會看到被列入整備區的檔案清單。
5. 將整備區的狀態記錄至儲存庫 **$ git commit -m "message"**。
6. 若有需要遠端儲存庫，可進行**建立遠端連線**的步驟（參考後續章節）。

- 專案初始化若還有疑問，請參考此處。
- 可更改預設編輯器（ex：記錄訊息、合併工具），**Visual Studio Code** 設定參考此處。

建立遠端連線（remote & push）

1. 先至第三方資源代管網站建立帳號，並新開一個專案取得 url 資訊。
2. **$ git remote add <name> <url>** 新增遠端連線資訊。
3. **$ git remote remove <name>** 移除遠端連線資訊。
4. 呈上述，預設值為 **name = origin**，大部分的專案以及教學預設以此為命名，但是這個詞彙嚴格來說沒有多大意義，使用者應可依照喜好以及需求命名（但如果是透過 **$ git clone <url>** 則為預設值 **origin**）。
5. 設定好遠端資訊以後，使用 **$ git remote --verbose** 查看遠端資訊。
6. **$ git remote show <name>** 顯示儲存庫詳細資訊。
7. 第一次執行此指令可能會出現第三方代管的帳號密碼的請求，設定好之後就不會再出現了。
8. 第一次使用 **$ git push** 會出現如附圖錯誤。

```
KBs-MacBook:gitforteams-zip KB$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository us
ing

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

9. 呈上述，依照 GIT 提示使用 **$ git push <name>**，此範例使用 gitlab 當作儲存庫別名，接下來看到的 gitlab 就是代表 **<name>**。
10. 然後還是錯~~（不要揍我）~~，出現如附圖的 GIT 提示，意思是找不到上游分支（遠端追蹤分支）。

```
KBs-MacBook:gitforteams-zip KB$ git push gitlab
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream gitlab master
```

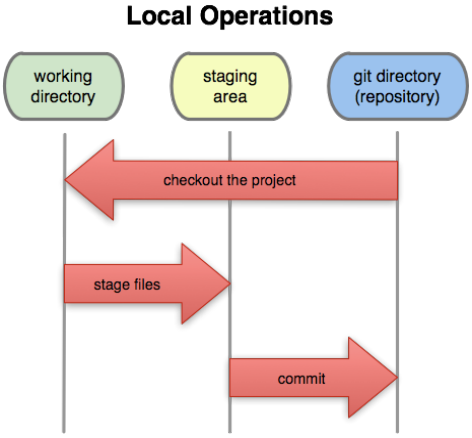
11. 呈上述，依照提示使用 **$ git push --set-upstream <name> <branch name>** ，此指令的意思是上傳分支至遠端儲存庫並設定追蹤該上游分支，且建立本機分支複本與遠端儲存庫間的連線關係。~~（因為上傳了... 所以就游上去惹 ，簡稱上游）~~
12. 參考附圖為推上去的結果。

```
KBs-MacBook:gitforteams-zip KB$ git push --set-upstream gitlab master
Counting objects: 607, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (583/583), done.
Writing objects: 100% (607/607), 9.02 MiB | 256.00 KiB/s, done.
Total 607 (delta 356), reused 0 (delta 0)
remote: Resolving deltas: 100% (356/356), done.
To https://gitlab.com/xul4m4d93/gitforteams.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from gitlab.
```

13. 因為已經設定好該 branch 的上游連線資訊，所以該 branch 之後只需要 **$ git push** 即可將本機記錄推送至遠端儲存庫。
14. 或是使用 **$ git push <name> <branch name>** 不設定追蹤的方式直接推送。
15. 若是想一次分享所有的分支內容，也可以使用 **$ git push --all <name>** 一次推送所有分支內容。

儲存庫一般操作

儲存庫操作流程



.gitIgnore

- 忽略特定檔案的配置檔，一般專案的設定檔檔名為 **.gitignore**，通常在檔案結構或是儲存庫的最上層。
- **$ git config --global core.excludesfile** 確認全域的忽略設定檔是否存在，如果沒有請建立一個 **.gitignore_global** 並設定檔案路徑在 **core.excludesfile** 底下。
- 忽略設定檔可參考此處產生實用的參考配置。
- 配置、設定完成後，即使使用 **$ git add —all** 也會過濾設定匹配的檔案。

add

- 將檔案加入暫存區 **$ git add <file>**。
- **$ git add --update** 將已被 GIT 追蹤的檔案一併加入暫存區。~~這個指令很好用哦哦哦~~
- **$ git add --all** 將工作區中所有的變動一併加入暫存區。（請先確認會被加入的檔案清單 **$ git status**）
- 若在 **add** 指令後檔案還有異動，該異動部分將不會進入暫存區，若異動過有追蹤必要則重複此命令，請參考附圖。

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   myTest.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myTest.txt
```

- 若要將檔案移出暫存區，可以使用 **$ git status** 提示的 **$ git reset HEAD <file>**。

commit

- 記錄當前的工作變更以及訊息，使用此指令將在工作分支上留下一個記錄點。
- **$ git commit** 跳出預設編輯器（ex：vim）輸入訊息，儲存離開編輯器後將記錄訊息至儲存庫。
- **$ git commit -m “message”** 將暫存區檔案記錄至儲存庫。
- **$ git commit --amend** 修改上一次記錄訊息。
- 良好的記錄物件應包含 …
  - 僅包含工單或是任務相關的程式碼。
  - 大小適中。
  - 良好的記錄訊息。
- 良好的記錄訊息應包含 …
  - 容易瀏覽的且清晰、簡短有力的敘述。
  - 若是更動範圍較大產生的較長敘述，應包含了目前程式的現況以及更動的主要原因。
  - 可以的話列出更動可能需要注意的細節或是可能淺在的風險。
  - 若有工單號碼請加入於訊息中提供後續歷史瀏覽參考。

stash

- 記錄當前工作區的變更並將工作區回到乾淨的狀態，記錄後可以切換到其他分支繼續工作（也許是緊急修正？）並可以在任何時候取回記錄前的工作狀態。
- 若發現在錯誤的分支上開發，可先記錄變更後切換分支，接著取回該變更。
- 此指令僅在本機可見且有效。
- **$ git stash save --include-untracked "<message>"** 記錄所有工作區的變更（包跨未被追蹤的檔案）並添加訊息方便辨識。
- **$ git stash list** 確認變更清單，最新的一筆記錄將在最上方顯示。
- **$ git stash (pop | apply) stash@{i}** 簽出記錄的變更。
- 呈上述，**pop** （建議）參數為簽出該筆變更後立即刪除，**apply** 則是單純簽出變更。
- **$ git stash drop stash@{i}** 刪除變更。

pull

- 拉取遠端儲存庫的最新記錄並合併本機儲存庫 **$ git pull <name> <branch name>**。
- **$ git pull = $ git fetch + $ git merge FETCH_HEAD**
- 呈上述，**pull** 預設為 **merge** 的方式更新遠端儲存庫分支的變更以及合併當前工作分支。
- 若是同一支線變更內容較少或是功能較單一的記錄，應考慮使用重設基準點的方式與 **pull** 一起使用，會使得歷史記錄更為清爽易讀。
- 呈上述，指令為 **$ git pull --rebase=preserve**，有可能需要解決衝突，若有疑問請使用 **$ git status** 尋求 GIT 提示。

push

- 推送本機的工作記錄至遠端儲存庫 **$ git push <name> <branch name>** ，若已設定遠端連線資訊 （**remote**），將分支以及變更推送至連線資訊指定的位置。
- 呈上述，透過帶入參數 **--set-upstream** 指定上游分支，下一次推送該分支僅需要 **$ git push** 即可推送至遠端儲存庫。
- 筆者偏好使用不帶參數的方式，因為~~頭腦不好~~需要明確的指定比較清楚啦！
- **$ git push --all <name>** 一次性的推送所有分支。
- **$ git push <name> <local branch>：<remote branch>** 將本機的分支推送到遠端新機的狀況可以使用此指令，此情境在適用多個 **remote** 的情況（**remote** 狀態可透過 **$ git remote -v or $ git remote show <name>** 確認）。

解決衝突

- GIT 只是版本、內容追蹤工具，當出現無法判斷的變更差異時 GIT 會將檔案合併主控權拋回給使用者。
- 可以使用 **$ git status** 尋求 **merge** 檔案的提示。
- **$ git mergetool <file>** 開啟合併工具解決衝突檔案。
- 可以設定自定義的 merge tool，筆者偏愛 **Visual Studio Code** 設定參考此處。

- vs code 合併示意圖請參考附圖。

- 解決衝突後存檔並關閉合併工具，GIT 就會判定解決衝突囉！

記錄查詢以及標籤

log

- 專案的歷史記錄可透過此指令查找。
- **$ git log** 檢視專案全部歷史記錄，使用上下方向鍵查看記錄，q 為離開。
- **$ git log <branch name>** 檢視指定的分支歷史記錄。
- **$ git log <target branch name>..<compare branch name>** 顯示目標分支與比較分支差異的歷史記錄（遠端分支也可以使用此指令）ex：**$ git log A..B** 列出在B分支上A沒有的記錄。
- 呈上述，也可以使用 **$ git difftool A..B** 使用差異工具查看。
- **$ git log --online** 查看專案全部簡短歷史記錄。
- **$ git log --graph** 查看專案歷史記錄，每行附加以 ASCII 畫出分支的分歧及合併的歷史圖形參考。
- **$ git log --grep <pattern>** 查找指定的 pattern 記錄。

reflog

- 顯示本機儲存庫的歷史記錄、操作記錄（ex：簽出分支）**$ git reflog**。
- 適用於尋找遺失的檔案~~（當你執行 **$ git reset --hard <commit>** 的時候發現搞錯記錄對象了…）~~

blame

- 確認檔案血統 **$ git blame <file>**，列出完整檔案以及記錄ID、作者、時間、行號以及行號對應的特定程式碼的對應。
- 呈上述，限制為若已經不在檔案內的程式碼就無法追蹤了，此指令僅能呈現當前檔案的程式碼記錄。
- ~~使用此指令的時候請心平氣和。~~

bisect

- 對於程式本身沒有提供錯誤進一步的追蹤訊息導致無法追蹤的情況下，可以使用此指令幫助除錯。
- **bisect** 會對指定的記錄區間使用二分法搜尋，啟動後將會在 **detached HEAD** 的狀態下漫遊在歷史記錄中，程式設計師即可在各個記錄點執行程式並將結果回報給 **bisect**，讓 **bisect** 繼續幫助您尋找問題點。
- **$ git bisect start** 啟用 **bisect** 程序。
- **$ git bisect bad <commit>** 不正常的記錄，若沒有帶 **commit** 則預設為當前記錄。
- **$ git bisect good <commit>** 正常的記錄。
- 一旦宣告好的記錄以及不好的記錄，即會進入 **bisect** 程序。
- 在每一個 **detached HEAD** 測試程式，並回報給 GIT **$ git bisect（good｜bad）**，直到 GIT 搜尋出有問題的記錄為止。
- 找到問題之後 **$ git bisect reset** 回到你開始執行 bisect 前的位置。
- 執行結果請參考附圖。

```
KBs-MacBook:ctbc KB$ git bisect start
KBs-MacBook:ctbc KB$ git bisect bad
KBs-MacBook:ctbc KB$ git bisect good 296673f
Bisecting: 1 revision left to test after this (roughly 1 step)
[525222b2abfb877b4175f2f211b530582aaa48c3] Add StringValidateUtils
KBs-MacBook:ctbc KB$ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[92261510494d3256022acd32488d1ca0f629d388] add schedule config
KBs-MacBook:ctbc KB$ git bisect bad
92261510494d3256022acd32488d1ca0f629d388 is the first bad commit
commit 92261510494d3256022acd32488d1ca0f629d388
Author: KB <kb19900709@gmail.com>
Date:   Fri Jun 30 00:07:22 2017 +0800

    add schedule config

:040000 040000 1d6975772ffe8b4399eb5972884b31c5e76255c5 720d4b5960e4450f8ce2e3ec8be855ea055ca298 M      ctbcCore
KBs-MacBook:ctbc KB$ git bisect reset
Previous HEAD position was 9226151... add schedule config
Switched to branch 'master'
```

tag

- 可視為可用文字識別的 **commit**。
- **$ git tag** 列出所有標籤。
- **$ git tag <tag name> <commit id>** 以 commit id 為參考建立新的 **tag**。
- **$ git show <tag name>** 查看該 **tag** 的相關訊息。
- **$ git tag --delete <tag name>** 移除本機的標籤。
- **$ git push <name> <tag name>** 推送標籤至遠端儲存庫。
- **$ git push --delete <name> <tag name>** 移除遠端儲存庫的標籤。
- **$ git checkout <tag name>** 簽出指定標籤的程式碼，並進入 **detached HEAD** 狀態。

使用/合併 分支

- GIT 的分支是羽量級的系統。
- 熟悉 GIT 分支的特性，改變開發模式以及思維。
- GIT 鼓勵在開發過程中頻繁的使用分支與合併，因為所有的操作（包含新建分支以及切換分支等）都能快速執行。
- 若有工單系統，依據工單系統上的情境以及問題描繪出分支圖或是 flow 範本（ex：git-flow）。
- 分支剛建立時會在分支的交叉點有相同的歷史記錄，使用 **log** 指令時會顯示其親代分支的歷史記錄。

可能的分支情境

- 嘗試實現一個可能沒有結果的概念。
- 可能與當前發佈版本不相關的的東西。
- 成果發佈前可能需要經過他人審核。
- 完成當前工作前可能需要處理其他工作。
- 上述任何一個條件若成立，則應切出新分支作業。

branch

- **$ git branch <branch name>** 建立新分支。
- **$ git branch --list** 列出本機分支。
- **$ git branch --remotes** 列出遠端分支。
- **$ git branch --all** 列出所有分支。
- * 標示表示為目前工作目錄中簽出的分支。
- remotes 表示不在本機的分支。
- 一般來說應會顯示 **origin** 儲存庫別名，表示為本機複本複製來源。

checkout

- **$ git checkout --track <remote branch>** 簽出指定的遠端分支（上游分支）並追蹤其工作目錄。
- **$ git checkout <local branch>** 簽出指定的本機分支。
- **$ git checkout -b <branch name> = $ git branch <branch name> + $ git checkout <branch name>**。
- 呈上述，此指令為在目前的分支建立一個新的分支並簽出該分支。
- **$ git checkout -b <new branch> <target branch>** 此指令為從指定分支新開一個分支。
- 可以觀察到每個分支的歷史記錄皆是獨立的。

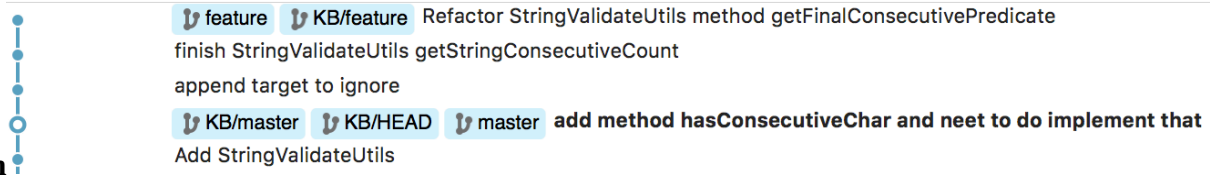
merge normal step

1. 本機需先與遠端主機歷史記錄同步（**pull**）需要進行 **merge** 的兩個分支，同步之後使用 **checkout** 指令切換到合併的主分支。
2. **$ git merge <branch name>** ，如果真的有合併的行為，而不是歷史快轉（**fast-forward**），就會顯示輸入記錄訊息的編輯器，通常也會保留原本的訊息。
3. **merge** 如果有立刻反悔的情況，可以使用 **$ git reset --merge ORIG_HEAD**。

4. 呈上述，或是使用 **$ git reset <commit>** 回到指定版本。
5. 合併完成後，使用 **$ git push** 將合併的成果推送回遠端儲存庫。
6. **$ git branch --delete <branch name>** 刪除本機分支複本。
7. 若刪除時 GIT 發現有還沒合併過的記錄，GIT 會囉嗦的提醒你所以不用太擔心記錄和分支消失的狀況。
8. **$ git push --delete <name> <branch name>** 刪除遠端儲存庫分支。

merge option

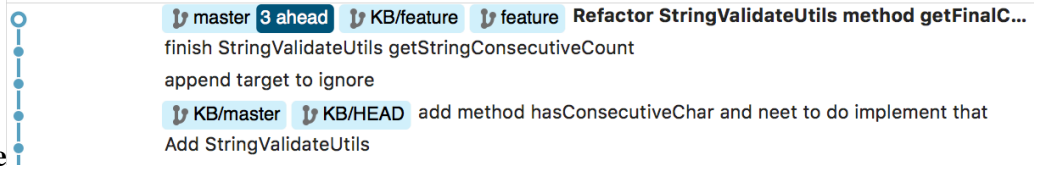
- 以下將會以 **master**、**feature** 作為合併分支範例，情境為 **feature** 合併回 **master**。
- **$ git checkout master**，目前工作區為 **master**。



origin

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

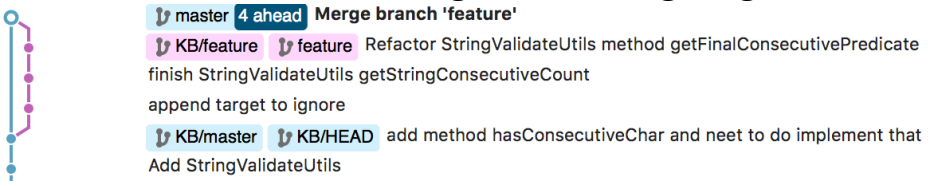
- 預設的 **merge** 策略為快進（**fast-forward**）。
- 呈上述，若需要合併的分支是主分支的直接上游，GIT 會使用快進模式且不會有新的 **commit** 點，請參考如下附圖。



$ git merge feature

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

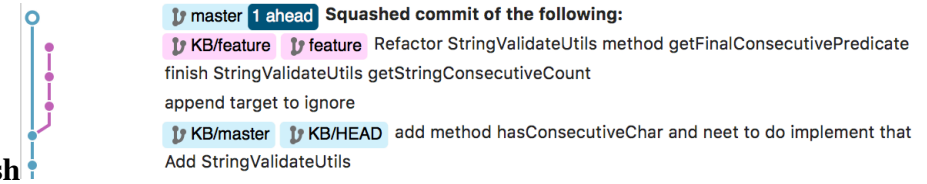
- 若希望分支圖可以長出小耳朵、**分清楚分支間的記錄關係**，可使用非快進策略（**No Fast-forward**）
- 使用非快進策略在還原（**revert**）合併記錄時會比較輕鬆。
- 非快進合併後會多出一個記錄點，預設訊息為 **Merge branch 'merge target name'**



$ git merge feature --no-ff

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

- 若分支在合併之後可以直接刪除，可以考慮使用歸併策略（**squash**）。
- 使用歸併合併後會還需 **commit** 一次，GIT 不會自動 **commit**。
- 呈上述，GIT 會自帶歸併的子分支歷史記錄訊息（on **feature** commit log）。

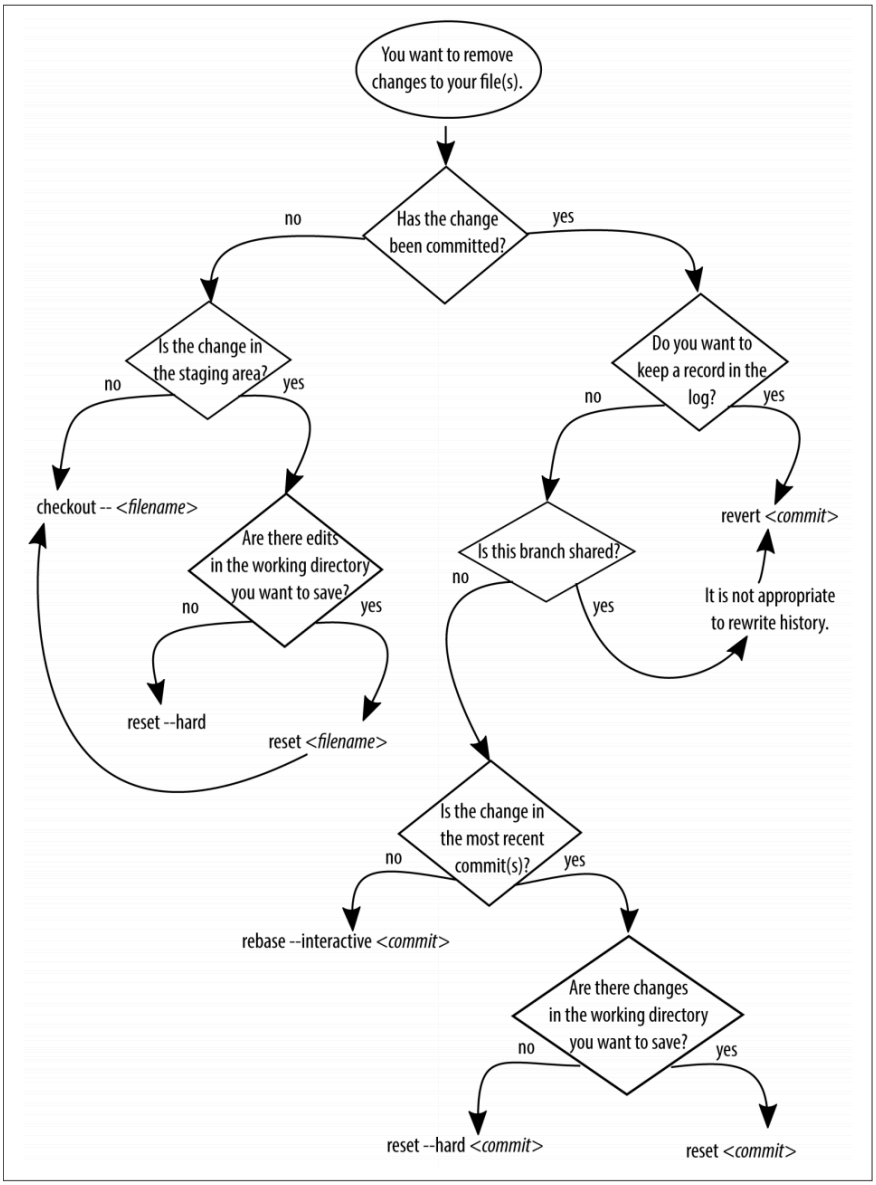


$ git merge feature --squash

更多使用分支相關內容請參考此處。

GIT 時光機（還原、重設檔案，優化歷史記錄）

- 在記錄分享至遠端儲存庫後，再修改共享的歷史一般認為是不好的行為。
- 若有修改歷史記錄請在尚未分享至遠端儲存庫的歷史記錄操作。
- 修改歷史記錄可能會帶來的影響以及其必要，在使用較強勢的指令前請先好好思考想~~不想準時下班~~。
- 時光機流程請參考附圖。



reset

- 將工作區還原到該記錄點且保留檔案變更 **$ git reset <commit>**。
- 呈上述，若使用 **--hard** 參數將會使指定記錄後的工作區檔案變更消失，請謹慎使用。
- 可以透過保留工作區檔案的還原方式移除歷史記錄，將零碎的歷史記錄整合成完整的記錄。
- 若想要將檔案移出暫存區，可以使用 **$ git reset -- <file>** or **$ git reset HEAD <file>**。

revert

- 還原（反轉）指定記錄的工作變更，並在分支上多一個還原記錄 **$ git revert <commit>**。
- 依據時光機流程圖的建議，**revert** 適合用在已經發佈的分支記錄。
- **$ git revert <commit> --mainline 1** 還原合併記錄並保留歷史線圖上左邊數過來第一個分支，此指令適用於兩個分支執行真正的合併，有時候必須處理還原衝突請使用 **$ git status** 取得 GIT 提示。
- 可使用 **$ git show <commit>** 確認 **commit** 記錄資訊，若有真正的合併行為將會提示兩個 **merge** 的 **commit id**。

```
KBs-MacBook:ctbc KB$ git show f229dc6
commit f229dc69e3ab47a0f2ad5a38f592cd01c69ca34c (HEAD -> master)
Merge: 9371daa 9ebc020
Author: KB <kb19900709@gmail.com>
Date:   Sat Jul 29 23:54:46 2017 +0800
```

- 呈上述，請參考附圖。 Merge branch 'feature'

rebase

- 重新定義分支的基準點。~~想像成把某個樹枝剪下來移到另外一個樹枝上。~~
- 效果不同於 **merge**。當分支很多時，妥善利用 **rebase** 可使得歷史記錄可讀性更高。
- GIT 執行 **rebase** 步驟
  - GIT 刪除子分支的歷史記錄。
  - 在子分支加上親代分支（**target branch**）的歷史記錄。
  - 在親代分支的末端往上增加在步驟一被刪除的歷史記錄的複本（不同的 **commit** 卻是相同的程式）。
- 因為會有不同的 **commit** 卻是相同程式的問題，依據時光機流程圖的建議若該分支已經推上遠端儲存庫則不適合此項操作。
- 執行過程為先 **checkout** 至需要重新定義基準點的分支，接著 **$ git rebase <traget branch>** 執行重新定義基準點。
- 以下開始 **rebase** 範例，情境為當前分支 1-bad_jokes 執行 **rebase** master 分支。
- 輸入指令的終端機輸出請參考附圖，GIT 會提示 **rebase** 的相關訊息。

```
KBs-MacBook:gitforteams KB$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Added information about additional people to be thanked.
Using index info to reconstruct a base tree...
M       README.md
Falling back to patching base and 3-way merge...
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: Failed to merge in the changes.
Patch failed at 0001 Added information about additional people to be thanked.
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".
```

- 執行過程中使用 **$ git status** 看看 GIT 會給什麼提示。

- 呈上述，附圖範例為需要解決衝突 README.md。

```
KBs-MacBook:gitforteams KB$ git status
rebase in progress; onto b4faee3
You are currently rebasing branch '1-bad_jokes' on 'b4faee3'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

        both modified:   README.md
```

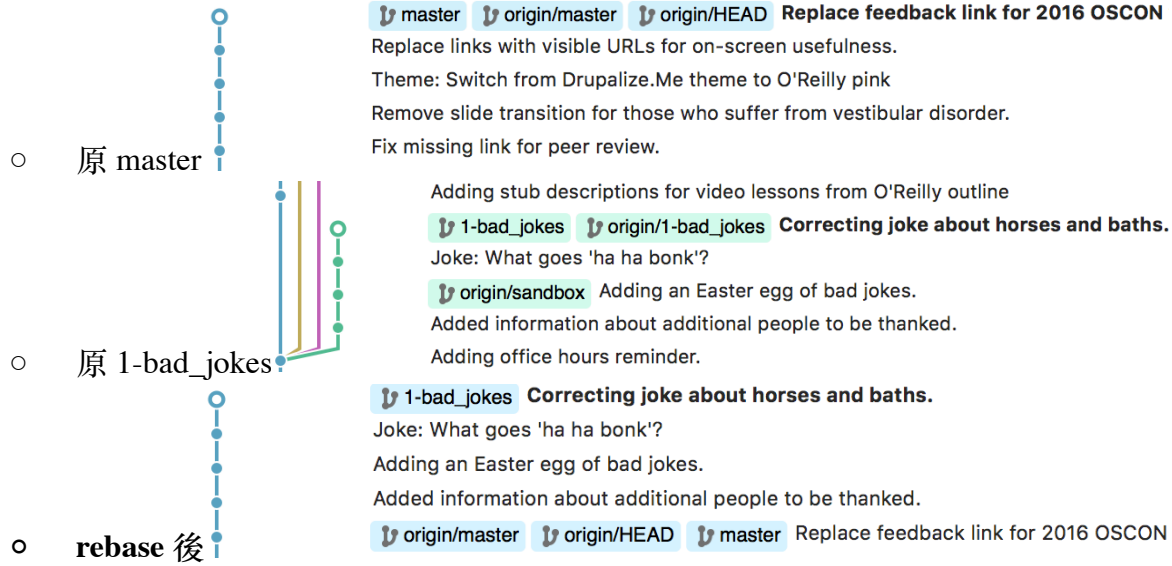
- 解決完衝突後的 **$ git status** 請參考附圖。

```
KBs-MacBook:gitforteams KB$ git status
rebase in progress; onto b4faee3
You are currently rebasing branch '1-bad_jokes' on 'b4faee3'.
  (all conflicts fixed: run "git rebase --continue")

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

- 解決衝突後依照提示 **$ git rebase --continue** 繼續 **rebase** 的流程。
- 衝突都解決完以後 GIT 會在親代分支的末端往上增加子分支的記錄的複本，子分支移動到與親代分支前面，看起來就像是親代分支最新的記錄點才加入的子分支（可快進）。
- 呈上述，請參考附圖。

```
KBs-MacBook:gitforteams KB$ git rebase --continue
Applying: Added information about additional people to be thanked.
Applying: Adding an Easter egg of bad jokes.
Applying: Joke: What goes 'ha ha bonk'?
Applying: Correcting joke about horses and baths.
```

- 任何一個時間點只要~~怕~~想中止操作，使用 **$ git rebase --abort** 讓子分支回到 **rebase** 前的狀況。
- 以下為範例中的分支截圖示意圖



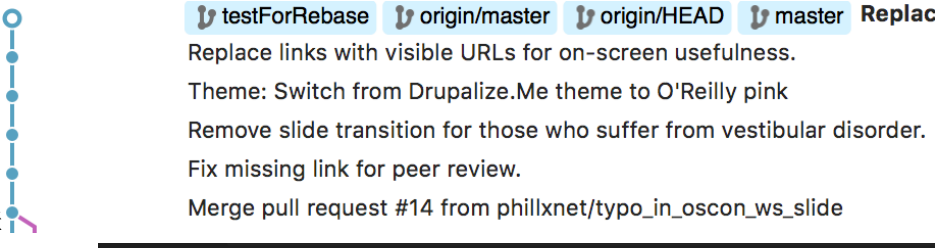
- 原 master
- 原 1-bad_jokes
- **rebase 後**

rebase interactive

- 互動式重整分支。
- 依據時光機流程圖的建議若該分支已經推上遠端儲存庫則不適合此項操作。
- 在需要重整的分支使用 **$ git rebase --interactive <commit>** 開啟編輯器進行互動式重整與調整。
- 呈上述，**<commit>** 必須選擇欲重整的記錄區間前一筆。
- 執行命令後請參考附圖。

```
pick 74dad73 Fix missing link for peer review.
pick 1c6ab90 Remove slide transition for those who suffer from vestibular disorder.
pick 376efa3 Theme: Switch from Drupalize.Me theme to O'Reilly pink
pick 1643448 Replace links with visible URLs for on-screen usefulness.
pick b4faee3 Replace feedback link for 2016 OSCON workshop.

# Rebase 27a3770..b4faee3 onto 27a3770 (5 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# 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.
#
# Note that empty commits are commented out
```

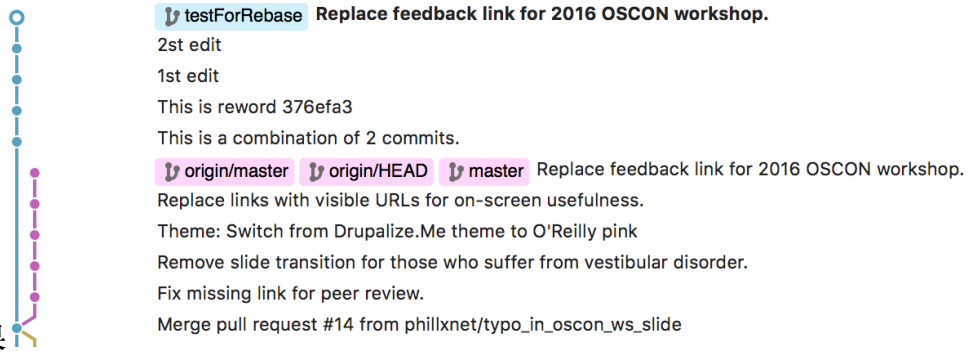
- GIT 註解提示為可調整選項。調整紅線部份命令以達到不同效果。
- GIT 會依時間順序依次執行對應的命令，較早的記錄先執行。
- 常用的為 **pick**（預設：正常使用該記錄）、**reword**（更改記錄訊息）、**edit**（**detached HEAD**）、**squash**（與前一筆記錄合併）。
- 執行 **edit** 時 GIT 會停留在該筆記錄並進入 **detached HEAD** 狀態，且暫存區不會有任何資料。
  - 配合 **$ git reset HEAD~1** 將版本還原至上一版並保留工作變更。
  - 進行基本的 **$ git add <file>** 將檔案加入暫存區、**$ git commit -m "message"** 將暫存區檔案記錄至儲存庫。
  - 呈上述兩點，此指令可做到**拆解記錄**，可將單筆記錄拆分多筆。
- 請參考附圖執行過程。
  - 原分支結構



  - 原互動式調整參數

```
pick 74dad73 Fix missing link for peer review.
squash 1c6ab90 Remove slide transition for those who suffer from vestibular disorder.
reword 376efa3 Theme: Switch from Drupalize.Me theme to O'Reilly pink
edit 1643448 Replace links with visible URLs for on-screen usefulness.
pick b4faee3 Replace feedback link for 2016 OSCON workshop.
```

- 執行結果



checkout

- 依據時光機流程圖的建議，在檔案尚未進入暫存區時可以透過 **$ git checkout -- <file>** 將檔案還原至上一個記錄點的狀態。
- 也可以使用 **$ git checkout <commit>**，將工作區狀態暫時還原至某個記錄點，從當前分支的連續記錄上暫時脫離，透過記錄留下的參考訊息我們可以回朔到任何一個記錄點尋找任何被 GIT 追蹤的檔案。
- 呈上述，指令後當前工作區狀態又可以稱為 **detached HEAD** 狀態。

```
KBs-MacBook:gitforteams KB$ git checkout eed5023
Note: checking out 'eed5023'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>
```

- 指令操作請參考附圖。`HEAD is now at eed5023... Joke: What goes 'ha ha bonk'?`

```
KBs-MacBook:gitforteams KB$ git status
HEAD detached at eed5023
nothing to commit, working tree clean
```

- 執行指令後狀態請參考附圖。
- 在 **detached HEAD** 狀態下可以使用 **$ git checkout -b <branch name>** 新開分支留下當前工作目錄狀態方便繼續追蹤歷史記錄。
- 若有任何想要加入的其他分支指定記錄，可以使用 **$ git cherry-pick -x <commit>** 將指定記錄下的所有變更加入到當前分支，當然有可能要解決衝突問題。
- 呈上述，遇到衝突時請依照 **$ git status** 的提示逐步處理。
- 呈上述，**-x** 參數會在指令後新的記錄上增加一行 **（cherry picked from commit <commit id>）**方便追蹤用。
- 將此暫時性的追蹤記錄分支整合後，合併回工作分支後記得刪除保持儲存庫的整潔。

刪除歷史記錄檔案

- 若有需要刪除歷史記錄的檔案如過肥的檔案或是極機密的資料，請參考此工具（BFG）。
- 呈上述，mac 用戶可以透過 **brew install bfg** 下載此命令檔~~早叫你換mac子~~。
- **bfg** 效能以及便利性比 GIT 內建的 **filter-branch** 好非常多，請參考此視頻。
- 執行 **bfg** 前請記得備份本機儲存庫以及通知其他團隊成員，執行完畢後會生成執行過程報告在 GIT 專案根目錄的平行資料夾（projectFileName.bfg-report）。
- 確認無誤後執行 **$ git reflog expire --expire=now --all && git gc --prune=now --aggressive** 抹去本機歷史記錄（包含操作記錄）並做垃圾回收。

- 執行完 **bfg** 後使用 **$ git push <name> --force --all --tags** 推送變更過的分支。
- 如果有其他團隊成員，請團隊成員重新 **clone** 儲存庫或是 **rebase** （**$ git pull --rebase=preserve**）工作分支繼續工作。

指令備忘錄

- **$ git shortlog -s** 【commit count / author】
- **$ git rev-list --all --count** 【total commit count】
- **$ git diff --name-only --diff-filter=U** 【need to merge file list】

- **$ git config --global mergetool.keepBackup false** 【original file with conflict markers can be saved as a file，default is true】

建立時間：**2017/07/04**
完成時間：**2017/07/30**
來源參考：網路資源、**Git 團隊使用手冊**、**Git 官方網站**

作者：**KB.Liao**
信箱：**kb19900709@gmail.com**