Git 是分散式的版本控制系統, 從架設、簡易操作、設定, 此篇主要是整理 基本操作、遠端操作 等.

註: Git 的範圍太廣了, 把這篇當作是初學入門就好了.

注意事項

由 project/.git/config 可知: (若有更多, 亦可由此得知)

* origin(remote) 是 Repository 的版本
* master(branch) 是 local 端, 正在修改的版本

平常沒事不要去動到 origin, 如果動到, 可用 git reset --hard 回覆到沒修改的狀態.

Git 新增檔案

* git add . # 將資料先暫存到 staging area, add 之後再新增的資料, 於此次 commit 不會含在裡面.
* git add filename
* git add modify-file # 修改過的檔案, 也要 add. (不然 commit 要加上 -a 的參數)
* git add -u # 只加修改過的檔案, 新增的檔案不加入.
* git add -i # 進入互動模式

Git 刪除檔案

* git rm filename

Git 修改檔名、搬移目錄

* git mv filename new-filename

Git status 看目前的狀態

* git status # 看目前檔案的狀態

Git Commit

* git commit
* git commit -m 'commit message'
* git commit -a -m 'commit -message' # 將所有修改過得檔案都 commit, 但是 新增的檔案 還是得要先 add.
* git commit -a -v # -v 可以看到檔案哪些內容有被更改, -a 把所有修改的檔案都 commit

Git 產生新的 branch

* git branch # 列出目前有多少 branch
* git branch new-branch # 產生新的 branch (名稱: new-branch), 若沒有特別指定, 會由目前所在的 branch / master 直接複製一份.
* git branch new-branch master # 由 master 產生新的 branch(new-branch)
* git branch new-branch v1 # 由 tag(v1) 產生新的 branch(new-branch)
* git branch -d new-branch # 刪除 new-branch
* git branch -D new-branch # 強制刪除 new-branch
* git checkout -b new-branch test # 產生新的 branch, 並同時切換過去 new-branch
* # 與 remote repository 有關
* git branch -r # 列出所有 Repository branch
* git branch -a # 列出所有 branch

Git checkout 切換 branch

* git checkout branch-name # 切換到 branch-name
* git checkout master # 切換到 master
* git checkout -b new-branch master # 從 master 建立新的 new-branch, 並同時切換過去 new-branch
* git checkout -b newbranch # 由現在的環境為基礎, 建立新的 branch
* git checkout -b newbranch origin # 於 origin 的基礎, 建立新的 branch
* git checkout filename # 還原檔案到 Repository 狀態
* git checkout HEAD . # 將所有檔案都 checkout 出來(最後一次 commit 的版本), 注意, 若有修改的檔案都會被還原到上一版. (git checkout -f 亦可)
* git checkout xxxx . # 將所有檔案都 checkout 出來(xxxx commit 的版本, xxxx 是 commit 的編號前四碼), 注意, 若有修改的檔案都會被還原到上一版.
* git checkout -- \* # 恢復到上一次 Commit 的狀態(\* 改成檔名, 就可以只恢復那個檔案)

Git diff

* git diff master # 與 Master 有哪些資料不同
* git diff --cached # 比較 staging area 跟本來的 Repository
* git diff tag1 tag2 # tag1, 與 tag2 的 diff
* git diff tag1:file1 tag2:file2 # tag1, 與 tag2 的 file1, file2 的 diff
* git diff # 比較 目前位置 與 staging area
* git diff --cached # 比較 staging area 與 Repository 差異
* git diff HEAD # 比較目前位置 與 Repository 差別
* git diff new-branch # 比較目前位置 與 branch(new-branch) 的差別
* git diff --stat

Git Tag

* git tag v1 ebff # log 是 commit ebff810c461ad1924fc422fd1d01db23d858773b 的內容, 設定簡短好記得 Tag: v1
* git tag 中文 ebff # tag 也可以下中文, 任何文字都可以
* git tag -d 中文 # 把 tag=中文 刪掉

Git log

* git log # 將所有 log 秀出
* git log --all # 秀出所有的 log (含 branch)
* git log -p # 將所有 log 和修改過得檔案內容列出
* git log -p filename # 將此檔案的 commit log 和 修改檔案內容差異部份列出
* git log --name-only # 列出此次 log 有哪些檔案被修改
* git log --stat --summary # 查每個版本間的更動檔案和行數
* git log filename # 這個檔案的所有 log
* git log directory # 這個目錄的所有 log
* git log -S'foo()' # log 裡面有 foo() 這字串的.
* git log --no-merges # 不要秀出 merge 的 log
* git log --since="2 weeks ago" # 最後這 2週的 log
* git log --pretty=oneline # 秀 log 的方式
* git log --pretty=short # 秀 log 的方式
* git log --pretty=format:'%h was %an, %ar, message: %s'
* git log --pretty=format:'%h : %s' --graph # 會有簡單的文字圖形化, 分支等.
* git log --pretty=format:'%h : %s' --topo-order --graph # 依照主分支排序
* git log --pretty=format:'%h : %s' --date-order --graph # 依照時間排序

Git show

* git show ebff # 查 log 是 commit ebff810c461ad1924fc422fd1d01db23d858773b 的內容
* git show v1 # 查 tag:v1 的修改內容
* git show v1:test.txt # 查 tag:v1 的 test.txt 檔案修改內容
* git show HEAD # 此版本修改的資料
* git show HEAD^ # 前一版修改的資料
* git show HEAD^^ # 前前一版修改的資料
* git show HEAD~4 # 前前前前一版修改的資料

Git reset 還原

* git reset --hard HEAD # 還原到最前面
* git reset --hard HEAD~3
* git reset --soft HEAD~3
* git reset HEAD filename # 從 staging area 狀態回到 unstaging 或 untracked (檔案內容並不會改變)

Git grep

* git grep "te" v1 # 查 v1 是否有 "te" 的字串
* git grep "te" # 查現在版本是否有 "te" 的字串

Git stash 暫存

* git stash # 丟進暫存區
* git stash list # 列出所有暫存區的資料
* git stash pop # 取出最新的一筆, 並移除.
* git stash apply # 取出最新的一筆 stash 暫存資料. 但是 stash 資料不移除
* git stash clear # 把 stash 都清掉

Git merge 合併

* git merge
* git merge master
* git merge new-branch
* 下述轉載自: ihower 的 [Git 版本控制系統(2) 開 branch 分支和操作遠端 repo.](http://ihower.idv.tw/blog/archives/2620)x
* *Straight merge 預設的合併模式，會有全部的被合併的 branch commits 記錄加上一個 merge-commit，看線圖會有兩條 Parents 線，並保留所有 commit log。*
* *Squashed commit 壓縮成只有一個 merge-commit，不會有被合併的 log。SVN 的 merge 即是如此。*
* *cherry-pick 只合併指定的 commit*
* *rebase 變更 branch 的分支點：找到要合併的兩個 branch 的共同的祖先，然後先只用要被 merge 的 branch 來 commit 一遍，然後再用目前 branch 再 commit 上去。這方式僅適合還沒分享給別人的 local branch，因為等於砍掉重練 commit log。*

*指令操作*

* *git merge <branch\_name> # 合併另一個 branch，若沒有 conflict 衝突會直接 commit。若需要解決衝突則會再多一個 commit。*
* *git merge --squash <branch\_name> # 將另一個 branch 的 commit 合併為一筆，特別適合需要做實驗的 fixes bug 或 new feature，最後只留結果。合併完不會幫你先 commit。*
* *git cherry-pick 321d76f # 只合併特定其中一個 commit。如果要合併多個，可以加上 -n 指令就不會先幫你 commit，這樣可以多 pick幾個要合併的 commit，最後再 git commit 即可。*

Git blame

* git blame filename # 關於此檔案的所有 commit 紀錄

Git 還原已被刪除的檔案

* git ls-files -d # 查看已刪除的檔案
* git ls-files -d | xargs git checkout -- # 將已刪除的檔案還原

Git 維護

* git gc # 整理前和整理後的差異, 可由: git count-objects 看到.
* git gc --prune
* git fsck --full

Git revert 資料還原

* git revert HEAD # 回到前一次 commit 的狀態
* ~~git revert HEAD^ # 回到前前一次 commit 的狀態~~
* git reset HEAD filename # 從 staging area 狀態回到 unstaging 或 untracked (檔案內容並不會改變)
* git checkout filename # 從 unstaging 狀態回到最初 Repository 的檔案(檔案內容變回修改前)

Git Rollback 還原到上一版

* git reset --soft HEAD^
* 編輯 + git add filename
* git commit -m 'rollback'

以下與 遠端 Repository 相關

Git remote 維護遠端檔案

* git remote
* git remote add new-branch http://git.example.com.tw/project.git # 增加遠端 Repository 的 branch(origin -> project)
* git remote show # 秀出現在有多少 Repository
* git remote rm new-branch # 刪掉
* git remote update # 更新所有 Repository branch
* git branch -r # 列出所有 Repository branch

抓取 / 切換 Repository 的 branch

* git fetch origin
* git checkout --track -b reps-branch origin/reps-branch # 抓取 reps-branch, 並將此 branch 建立於 local 的 reps-branch

刪除 Repository 的 branch

* git push origin :heads/reps-branch
* git push origin --delete reps-branch