Git 使用筆記繁中版

盧永鈞

July 3, 2024

Contents

1 簡介

這是簡介部分，簡要介紹文檔的背景和目的。Enslish font is ok

2 基本操作

2.1 設置作者

使用 config 指令設定作者和信箱

```
git config --global.user.name "<name>"
git config --global.user.email "<email>"
git config --list //檢視設定
```

也可以至家目錄的 git 設定檔 /.gitconfig 修改

將 global 替換成 local，可以爲每個專案可以設置獨立的作者

2.2 設置指令縮寫

```
git config --global alias.<指令別名> <指令>
git config --global alias.l "log --oneline --graph" //簡化命令
```

2.3 初始化專案

使用 init 指令使資料夾被 Git 控制

```
git init
```

2.4 察看目錄狀態

```
git status
```

2.5 建立 commit 的標準流程

先將想要追蹤的檔案加入暫存區 (Staging Area, index)

```
//將檔案加入暫存區
git add <檔案>
//支持萬用字符，僅加入特定類型檔案
git add *.檔案後綴
//將所有檔案加入暫存區
git add --all
//在資料夾根目錄使用，等效於--all
git add .
```

然後就可以 commit，即把檔案加入儲存庫 (Repository)。

```
git commit -m "說明"
```

注意到，commit 只會對加入暫存區的檔案作「備份」。然而，也可以在暫存區爲空的情況下 commit

```
git commit --allow-empty -m "<說明>"
```

2.6 察看 commit 歷史記錄

使用 log 指令察看 commit 記錄

```
git log
git log --oneline
git log --oneline --graph
```

也可以用特定條件篩選 commit 記錄

```
//用作者篩選
git log --auhtor="作者"
//用commit內容篩選
git log --grep="關鍵字"
//用檔案內容篩選
git log -S "關鍵字"
//檢視特定檔案的commit記錄
git log <檔案>
//檢視特定檔案的修改記錄
git log -p <檔案>
//☒看特定檔案每一行/特定行數的作者
git blame <檔案>
git blame -L <開始行數>,<結束行數> <檔案>
```

2.7 在 Git 中刪除檔案

若使用 linux 自帶的的 rm，還得手動加入暫存區，用 Git 自帶的 rm 指令可以結合上述兩個操作：

```
git rm <要刪除的檔案>
```

2.8 救回誤刪的檔案

若只是檔案誤刪，使用 checkout 指令，從暫存區取用資料復原

```
git checkout <誤刪的檔案>
git checkout . //救回所有刪除的檔案
```

2.9 把檔案還原到特定 commit

將<檔案>還原到<數字>個版本前的 commit

```
git checkout HEAD~<數字> <檔案>
```

2.10 在 Git 中變更檔名

```
        git mv <舊黨名> <新黨名>
```

2.11 取消版控

使用 Git 的 rm 指令，可以將檔案解除版控而不刪除

```
git rm <要刪的檔案> --cached
```

3 修改 commit 與進階操作

3.1 修該最新 commit 訊息

```
git commit --amend -m "新訊息"
```

3.2 新增檔案到最新的 commit

```
//先加到緩存區
git add <要新增的檔案>
//加到最後一次commit
git commit --amend --no-edit
```

3.3 拆掉 commit

符號^代表前一次，^^代表前兩次，以此類推。注意到，reset 指令只是回到某版本的狀態，其餘版本都還留存，並沒有刪除。

```
//拆掉最後一次commit, 以前一次的檔案覆蓋現有檔案
git reset <commit前6碼>^
git reset master^
git reset HEAD^
//指定要拆到第幾個commit
git reset <commit前6碼>
//進入互動模式編輯commit
git rebase -i <SHA-1前七碼>
```

3.4 reset 的三種模式

模式	mixed（預設）	soft	hard
工作目錄	不變	不變	丟掉
暫存區	丟掉	不變	丟掉
commit 拆出來的檔案	丟回工作目錄	丟回暫存區	完全丟掉

Table 1: reset 模式

3.5 將 reset 還原

已知 reset 不刪除檔案，因此只要記得 SHA-1 的前 7 碼，完全可以直接 reset 回去。若忘記可以使用 reflog 指令察看

```
git reflog
//這個也可以做到
git log -g
```

4 分支 branch

第一個 commiit 即建立在預設的分支 master 上

4.1 基本操作

```
//新增分支
git branch <分支>
//刪除分支
git branch -d <branch name> //刪除分支, 但合併的不可刪除
git branch -D <branch name> //刪除分支, 但強迫刪除未合併分支
//改名
git branch -m <old name> <new name> //幫分支改名
//切換分支
git checkout <branch name>
//合併分支
git merge <要合併的分支>
```

5 疑難雜症

5.1 Git 不保存空目錄

在該目錄中新增.gitkeep 檔案即可

5.2 讓 Git 忽略特定檔案

在與.git 同級的資料夾新增.gitignore 檔案，加入要忽略的檔案即可。

```
#忽略特定檔案
<檔名>

#忽略特定副檔名的檔案
*.<副檔名>

#忽略特定資料夾的檔案
<資料夾名稱>
```

注意到，在建立.gitignore 檔案前就存在的，但在忽略清單中的檔案不會被忽略！ 可以在建立.gitignore 後，用 clean 強制清除：

```
git clean -fX
```

6 與 GitHub 專案同步

6.1 push

首先至 GitHub 網站創建專案 (repository)，然後在本地要同步的，已經受 git 版控的資料夾加上遠端節點：

```
\\☒看有無遠端節點
git remote -v
\\檢視遠端節點的詳細資料
git remote show origin
\\新增節點
git remote add origin <URL>
\\更換現有節點
git remote set-url origin <新url>
```

注意到，遠端節點不一定要命名成 origin，這只是一種慣例。而 origin 即爲代表 URL 的變數

然後就可以把本地端的專案推送到 Github 了。使用 push 指令推送特定分支到 origin。注意到，若 origin 存在同名分支，即更新進度，不存在則創建分支

```
//將本地分支推送到同名的遠端分支
git push -u origin <本地分支>
//將本地分支推送到指定名稱的遠端分支
git push origin <本地分支>:<遠端分支>
```

上述命令的 u 代表 upstream，意思是設置遠端的 origin\master 爲上游節點，下次只需輸入 push 就會推送一樣的本地分支到遠端節點上。

6.2 pull

pull 即爲 fetch 和 merge 的結合

```
//從URL拉取
git pull <REPOSITORY的URL>

git pull <遠端別名> <遠端分支名>:<本地分支名>
git pull <遠端別名> <遠端分支名>
```

6.3 fetch

fetch 指令就是將遠端資料庫的更動抓回本地端，因此若有新的 commit，origin\master 和 origin\head 會變動

```
//抓取遠端資料庫信息
git fetch
//更新遠端數據庫所有更新
git fetch <遠端別名> <遠端分支名>
//更新遠端數據庫特定更新
git fetch <遠端別名>
```

更新完後，使用 merge 指令即可將進度更新

```
git merge origin/<遠端分支>
```

6.4 大型專案常用指令

沒有存取權時，先去 GitHubfork 一份到自己的 repository

```
//複製一份repository到本地
git clone <url>
如何同步fork過來的專案
git remote -v //□看遠端節點有無原作的節點
git remote add <在地節點名> <原作節點url>

git fetch <在地節點名>
git merge <在地節點名>/要合併的節點

//刪除遠端分支
git push origin :<要刪除的分支名>
```

7 圖表示例

7.1 如何正確命名分支

分支名稱	說明
Master	穩定，可以隨時使用的版本，不會直接 commit，版本標籤通常打在此分支上
Hotfix	緊急問題修復開的分支，修好記得 merge 到 Master 和 Develop。
Release	上線前的測試板，好了一樣要併到 Develop，避免日後開發需要
Feature	新增功能時，從 Feature 拿來的

Table 2: 分支命名範式

7.2 插入圖片

7.3 插入表格

A	B	C
1	2	3
4	5	6
7	8	9

Table 3: 這是一個示例表格

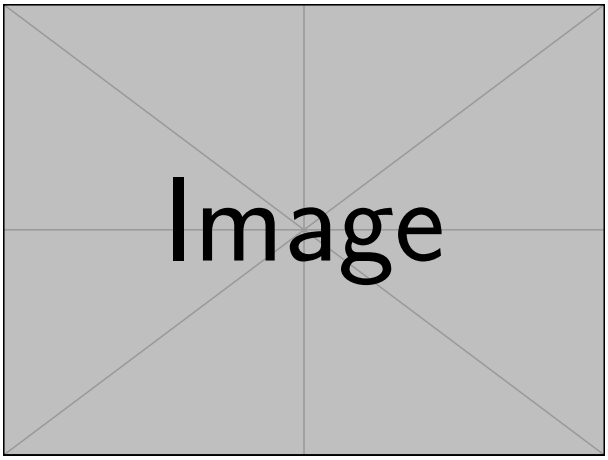
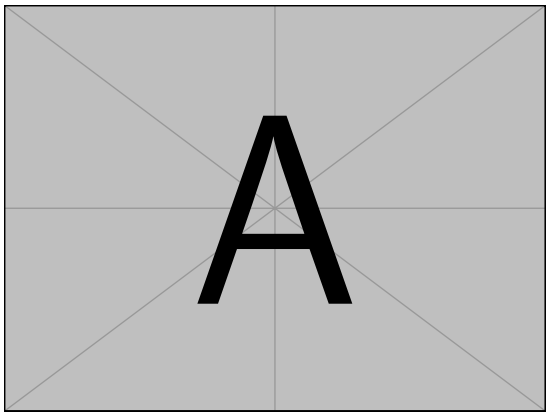
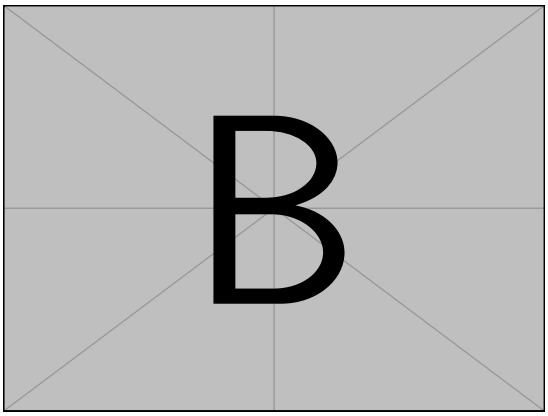


Figure 1: 這是一個示例圖片



(a) 子圖 A



(b) 子圖 B

Figure 2: 並排圖表示例

7.4 並排圖表

8 代碼示例

```
        def hello_world():
        print("Hello, world!")

        hello_world()
```

Listing 1: Python 代碼示例

9 結論

結論打在這裏