

Git 使用筆記繁中版

盧永鈞

June 30, 2024

Contents

1	簡介	1
2	基本操作	2
2.1	設置作者	2
2.2	設置指令縮寫	2
2.3	初始化專案	2
2.4	察看目錄狀態	2
2.5	建立 commit 的標準流程	2
2.6	察看 commit 歷史記錄	2
2.7	在 Git 中刪除檔案	3
2.8	救回誤刪的檔案	3
2.9	把檔案還原到特定 commit	3
2.10	在 Git 中變更檔名	3
2.11	取消版控	3
3	修改 commit 與進階操作	3
3.1	修該最新 commit 訊息	3
3.2	新增檔案到最新的 commit	3
3.3	拆掉 commit	4
3.4	reset 的三種模式	4
3.5	將 reset 還原	4
4	分支 branch	4
4.1	基本操作	4
5	疑難雜症	4
5.1	Git 不保存空目錄	4
5.2	讓 Git 忽略特定檔案	5
6	與 GitHub 專案同步	5
6.1	push 和 pull	5
6.2	大型專案常用指令	5
7	圖表示例	5
7.1	如何正確命名分支	5
7.2	插入圖片	6
7.3	插入表格	6
7.4	並排圖表	6
8	代碼示例	6
9	結論	7

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

```
1 git config --global.user.name "<name>"
2 git config --global.user.email "<email>"
3 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`，可以為每個專案可以設置獨立的作者

2.2 設置指令縮寫

```
1 git config --global alias.<指令別名> <指令>
2 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

```
1 git init
```

2.4 察看目錄狀態

```
1 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`)

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

然後就可以 `commit`，即把檔案加入儲存庫 (Repository)。

```
1 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，也可以在暫存區為空的情況下 `commit`

```
1 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

```
1 git log
2 git log --oneline
3 git log --oneline --graph
```

也可以用特定條件篩選 `commit` 記錄

```
1 //用作者篩選
2 git log --author="作者"
3 //用commit內容篩選
4 git log --grep="關鍵字"
5 //用檔案內容篩選
6 git log -S "關鍵字"
7 //檢視特定檔案的commit記錄
8 git log <檔案>
9 //檢視特定檔案的修改記錄
10 git log -p <檔案>
11 //☑看特定檔案每一行/特定行數的作者
12 git blame <檔案>
13 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 `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：

```
1 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

```
1 git checkout <誤刪的檔案>
2 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

```
1 git checkout HEAD~<數字> <檔案>
```

2.10 在Git中變更檔名

```
1 git mv <舊檔名> <新檔名>
```

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

```
1 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

```
1 git commit --amend -m "新訊息"
```

3.2 新增檔案到最新的 commit

```
1 //先加到緩存區
2 git add <要新增的檔案>
3 //加到最後一次commit
4 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。

```

1 //拆掉最後一次commit，以前一次的檔案覆蓋現有檔案
2 git reset <commit 前6碼>^
3 git reset master^
4 git reset HEAD^
5 //指定要拆到第幾個commit
6 git reset <commit 前6碼>
7 //進入互動模式編輯commit
8 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

```

1 git reflog
2 //這個也可以做到
3 git log -g

```

4 分支 branch

第一個 commit 即建立在預設的分支 `master` 上

4.1 基本操作

```

1 //新增分支
2 git branch <分支>
3 //刪除分支
4 git branch -d <branch name> //刪除分支，但合併的不可刪除
5 git branch -D <branch name> //刪除分支，但強迫刪除未合併分支
6 //改名
7 git branch -m <old name> <new name> //幫分支改名
8 //切換分支
9 git checkout <branch name>
10 //合併分支
11 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。

```
1 # 忽略特定檔案
2 <檔名>
3
4 # 忽略特定副檔名的檔案
5 *.<副檔名>
6
7 # 忽略特定資料夾的檔案
8 <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：

```
1 git clean -fX
```

6 與 GitHub 專案同步

6.1 push 和 pull

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：

```
1 git remote add origin <*.git>
```

注意到，遠端節點不一定要命名成 origin，這只是一種慣例。

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

```
1 git push -u origin <your branch>
2 \\\
3 git push origin <本地端分支名> :<遠端分支名>
```

6.2 大型專案常用指令

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

```
1 //複製一份 repository 到本地
2 git clone <url>
3 如何同步 fork 過來的專案
4 git remote -v //☑看遠端節點有無原作的節點
5 git remote add <在地節點名> <原作節點url>
6 git fetch <在地節點名>
7 git merge <在地節點名> / 要合併的節點
8 //刪除遠端分支
9 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

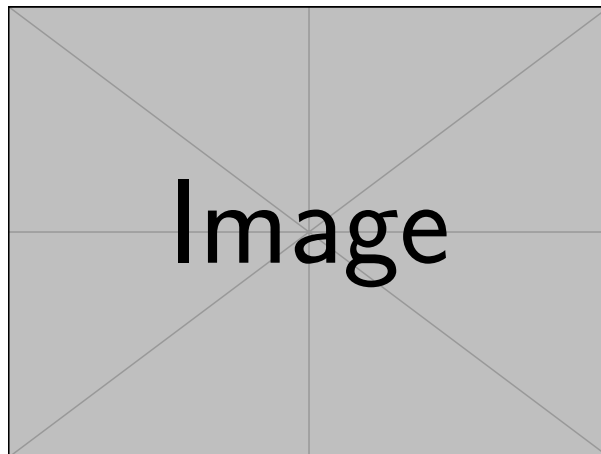


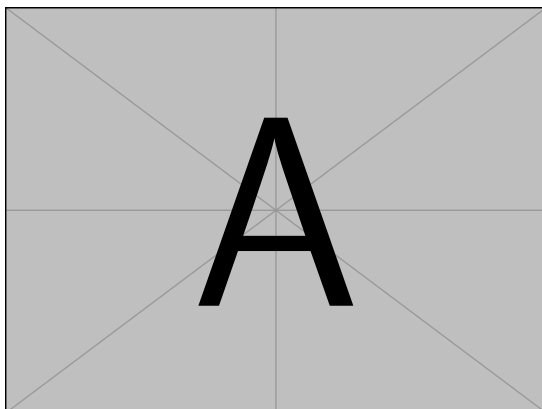
Figure 1: 這是一個示例圖片

7.3 插入表格

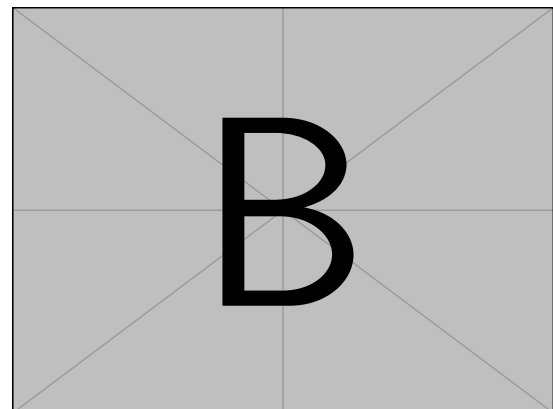
A	B	C
1	2	3
4	5	6
7	8	9

Table 3: 這是一個示例表格

7.4 並排圖表



(a) 子圖 A



(b) 子圖 B

Figure 2: 並排圖表示例

8 代碼示例

```
1 def hello_world():
2     print("Hello, world!")
3
4     hello_world()
```

Listing 1: Python 代碼示例

9 結論

結論打在這裏