Git

Git 是什麼？版本控制是什麼？

Git 是一個分散式版本控制軟體。

版本控制能夠記錄檔案的內容變化，並且能夠查詢每個版本所更動的內容。這裡我們就以一位網頁設計師的開發過程當例子吧。

傳統要做版本控制，絕大部分都會像是下圖，用資料夾來去版本控制，但這樣的壞處是，我們根本不知道裡面到底哪些檔案有更動過。


20190308_v2 修改 BUG
20190307_v1 上線版
20190306_加入標題樣式
20190305_插入 CSS
20190304_新增網頁

但如果使用版本控制的話，以上問題都可以解決。在 Git 的世界裡，上圖的每個資料夾都可以視為一個版本，且能夠記錄每個版本修改了哪些檔案，如下圖。

v3 2019/3/6

加入標題樣式

modified index.html

modified all.css

2019/3/5

v2

插入 CSS

modified index.html

new all.css

2019/3/4

v1

新增網頁

new index.html

你可以很輕易看出來，每個版本新增 (new) 、編輯(modified)了哪些檔案。

像是 v1 的版本，版本說明是「新增網頁」，同時也會記錄他在專案上新增了一個 HTML 檔。而在 v3 版本上，可以看到他沒新增檔案，但卻針對 index.html、all.css 去做了編輯動作，有可能他在 HTML 插入了一個 h1 標籤，並且在 CSS 裡新增了 h1 的樣式。

這就是 Git 的好處，它甚至能看到開發者改了哪幾行 CODE 都一清二楚，同時也會顯示該版本是哪位開發者更新的，在團隊開發與溝通上，藉由觀察 Git 上的更新就順暢多了。

安裝git

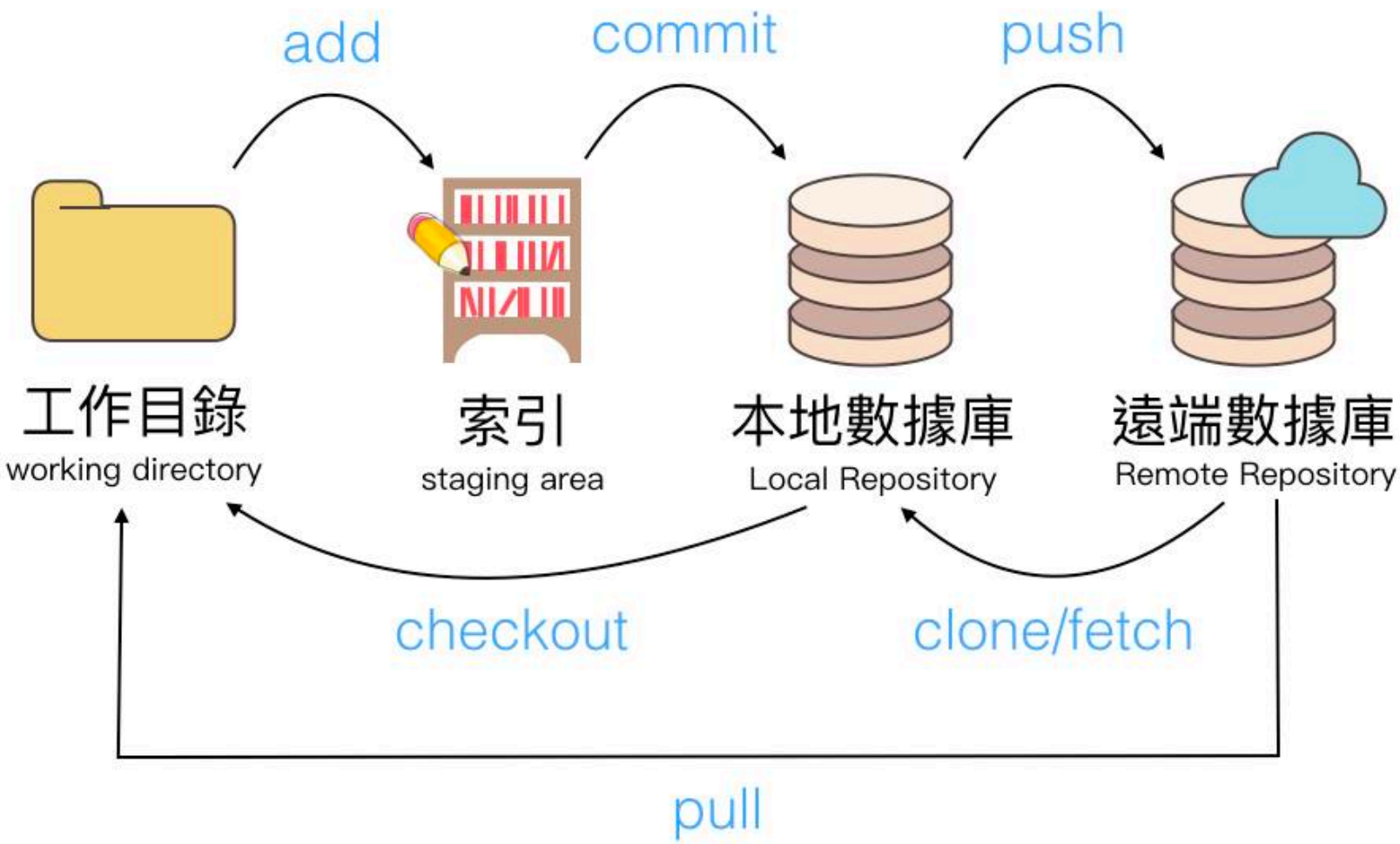
1. Git
2. 安裝Git之後，請設定用戶名稱和電子郵件地址，在 Git 每次的檔案版本發佈時，都會記錄該版本是哪位開發者做的。

   ```
   git config --global user.name "使用者名字"
   ```

   ```
   git config --global user.email "電子信箱"
   ```

3. 設定完成後可查詢設定

   ```
   git config --list
   ```


add
commit
push
工作目錄
working directory
索引
staging area
本地數據庫
Local Repository
遠端數據庫
Remote Repository
checkout
clone/fetch
pull

數據庫 (Repository)

數據庫 (Repository) 是記錄檔案或目錄狀態的地方，儲存內容的修改歷史記錄。在數據庫的管理下除了儲存修改歷史記錄外，還可以追蹤內容的狀態和版本喔。

Git的數據庫分為遠端數據庫和本地端數據庫。

- 遠端數據庫: 配有專用的伺服器，為了讓多人共享而建立的數據庫。
- 本地端數據庫：為了方便用戶個人使用，在自己的機器上配置的數據庫。

您可以在本地端數據庫上使用像是還原更改、跟踪更改等所有Git版本控制功能。

不過，如果想要公開在本地端數據庫的修改內容，就需要將內容上傳到遠端數據庫了。另外，透過遠端數據庫還可以取得其他人修改的內容。

工作目錄（Working Tree）是保存您目前正在處理檔案的目錄，Git 相關的操作都會在這個目錄下完成。

索引(Index) 位於工作目錄和數據庫之間，是為了向數據庫提交作準備的暫存區域。

所以在工作目錄上做的任何變更並不會直接提交到數據庫的。Git在執行提交的時候，不是直接將工作目錄的狀態儲存到數據庫，而是將索引的狀態儲存到數據庫。因此，要提交變更，首先必需要把變更內容加入到索引中。

索引的存在可以排除工作目錄裡不必要的檔案提交，還可以只將檔案變更內容的一部分加入索引並提交。

建立本地數據庫

1. 新增一個資料夾
2. 前往該資料夾
3. 執行 git init 指令
4. git status查看

git add、git commit - 提交版本

Untracked files 的意思是，Git 偵測到你有新增這筆檔案，但尚未是 Git 追蹤的對象。所以必須先將它加入到 staging area(索引)裡，就可以將檔案加入到追蹤對象。

將檔案加入到索引：git add <檔案名稱>

檔案狀態本來是 Untracked files，變成是Changes to be committed，這樣就表示有成功加入到索引。

Changes to be committed的意思是，放在索引的檔案即將會被提交成一個新版本(commit)。

提交一個新版本：git commit -m "<填寫版本資訊>"

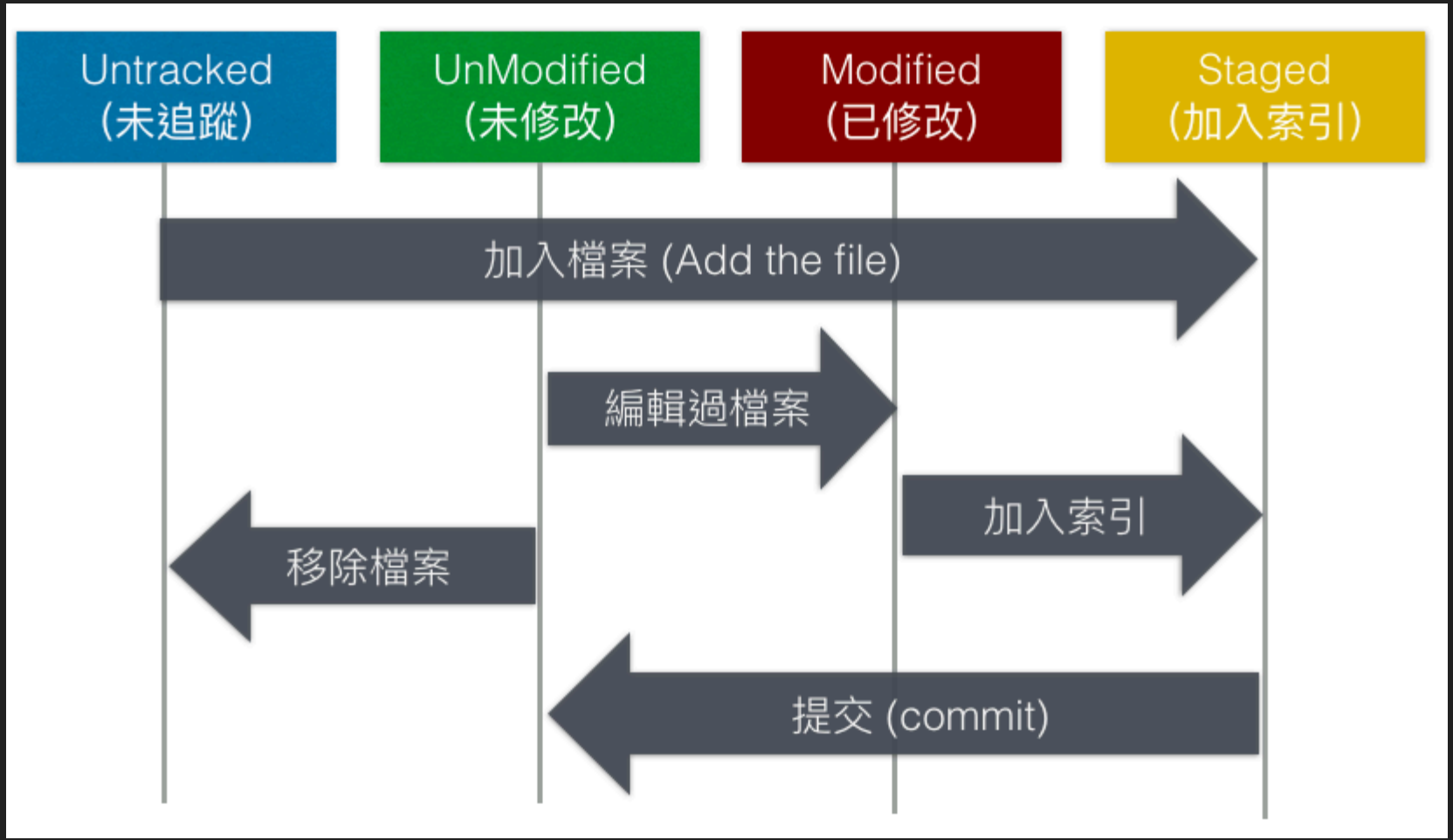
單一檔案加入索引：git add <檔案名稱>

所有檔案加入索引：git add .

提交版本：git commit -m "填寫版本資訊"

觀看當前狀態：git status

瀏覽歷史紀錄：git log


Untracked
(未追蹤)
UnModified
(未修改)
Modified
(已修改)
Staged
(加入索引)
加入檔案 (Add the file)
編輯過檔案
加入索引
移除檔案
提交 (commit)

當我們建立數據庫後，立即新增一個檔案時，用 git status 查詢，會發現它是在 Untracked 狀態。

這就表示此檔案還沒進到版本控制，藉由 git add . 將檔案加入到索引 (Staged) 後，準備提交成一個 commit 版本。

藉由 git commit -m <提交訊息> 後，該檔案就會開始被追蹤，檔案狀態就會變成 UnModified 狀態。

為了區分提交，系統會產生一組識別碼來給提交命名，識別碼會根據提交修改的內容計算出不重複的40位英文數字。指定識別碼，就可以在數據庫中找到對應的提交。

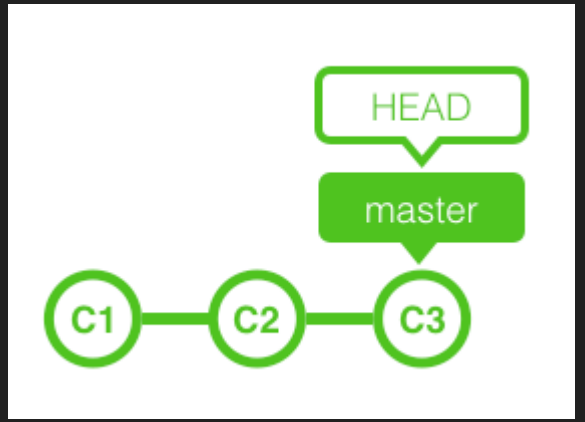
像錯誤修復或功能添加之類不同含義的更改，要盡量分開來提交。這樣可以方便事後從歷史記錄裡找出特定的修改內容。

分支 (branch)

HEAD 就是你目前指向的版本狀態，而 HEAD 可以選擇它指向到：

- 分支 (branch)
- commit 版本

在預設狀態時，master 就是我們的預設分支名稱，你可想像有條分支串起好幾個 commit。而此時的 HEAD，他是跟著 master 向前推進的。



在每個 commit 旁邊的亂數就是 SHA-1 值，可以取前幾碼來準備切換。

git checkout <sha number>

切換後，你就可以看到自己的工作目錄，彷彿用了時光機，回到以前你指定的開發歷史狀態。

讓 HEAD 再綁定回 master 的分支：git checkout master

檔案還原

只要你的 Git 有將檔案加入過 commit 裡，程式碼都是有辦法被還原的。儘管你刪掉整個分支，也仍然找得到資料。

新增檔案時，檔案還沒加入追蹤，想清空工作目錄：

- 顯示要被清除的檔案：git clean -n
- 強制清除檔案：git clean -f

檔案已加入追蹤，想還原工作目錄：

- 單一檔案指令 ：git checkout -- <file>
- 全部檔案指令：git checkout .

檔案加入到索引，想退到工作目錄：

- 指令：git reset HEAD

版本還原

1. 不想要最新前兩個 commit，想還原到第一個版本。但檔案都想保留：

```
git reset HEAD^^
```

2. 除了還原兩個版本外，連檔案都不想保留：

```
git reset HEAD^^ --hard
```

HEAD 也就是目前指標位置，^就是向前一個版本，，兩個 ^^ 就是向前推兩個版本，除此之外，用 ~ 也是可以的。例如 git reset HEAD~~，如果向前推進的版本很多，你也可以用數字取代，例如git reset HEAD^2。

所以 git reset HEAD^2 等於 git reset HEAD~~。

而 HEAD 較偏向是相對位置移動，如果你有 commit 的 SHA-1 編號時，使用 git reset <SHA-1編號> 也可以。

參考資料

連猴子都能懂的Git入門指南| 貝格樂（Backlog）

Git & GitHub 教學手冊- Git 環境教學