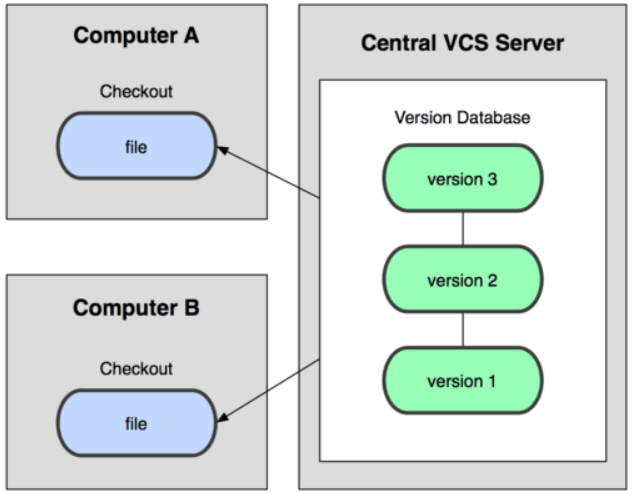
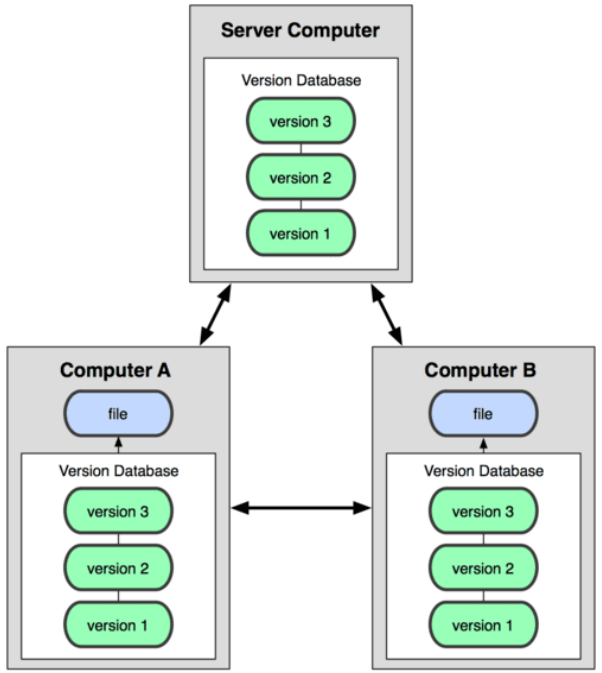
Git 基本

版本控制

- Local 式
  - 在自己電腦中建立一個版本資料庫(如: RCS)
  - 問題：協作有問題
- 中央式
  - 在一台Server 上儲存版本紀錄(如: Subversion(SVN))
  - 問題： Server 會掛、開發太亂
- 分散式
  - 所有開發者都有資料庫、獨立工作
  - Server爛了，不要緊，大家都有備份
  - 問題： 亂成一團
  - 解法： 使用好的Branch機制解決





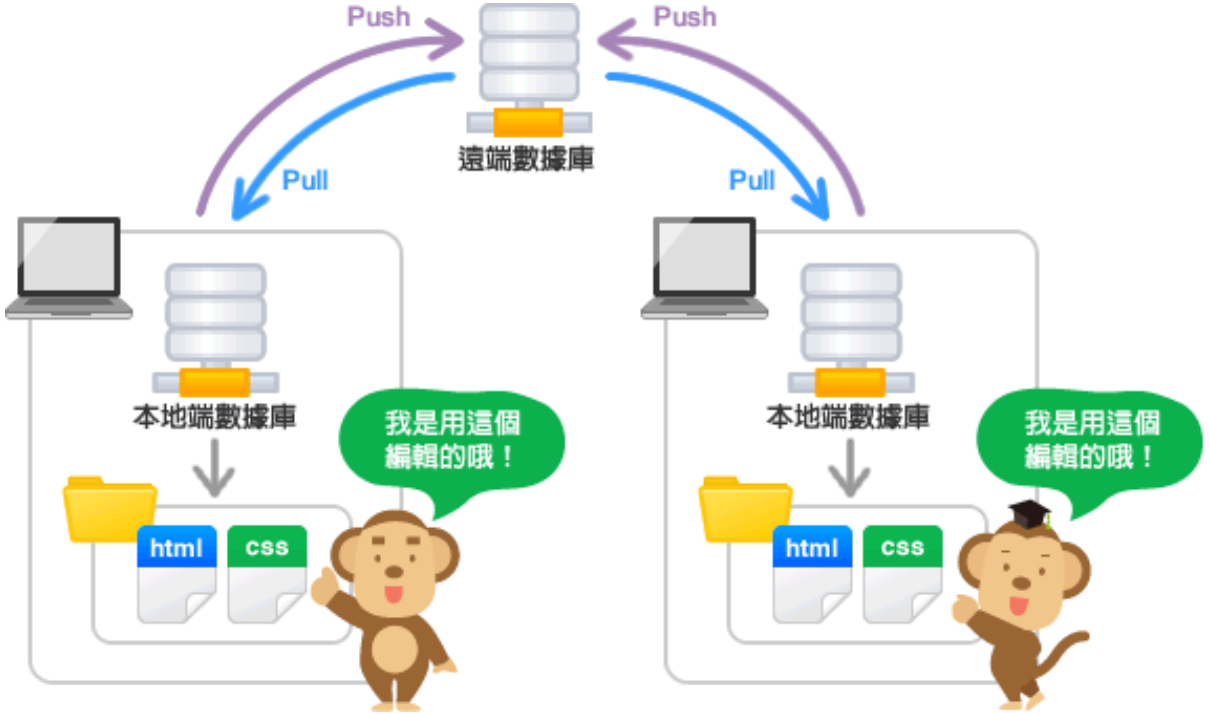
使用Git版本控制系統

- 用於Linux核心開發的版本控制工具。
- 在Git管理檔案的話，更新歷史會保存在Git。所以不需要複製備用的檔案啦。

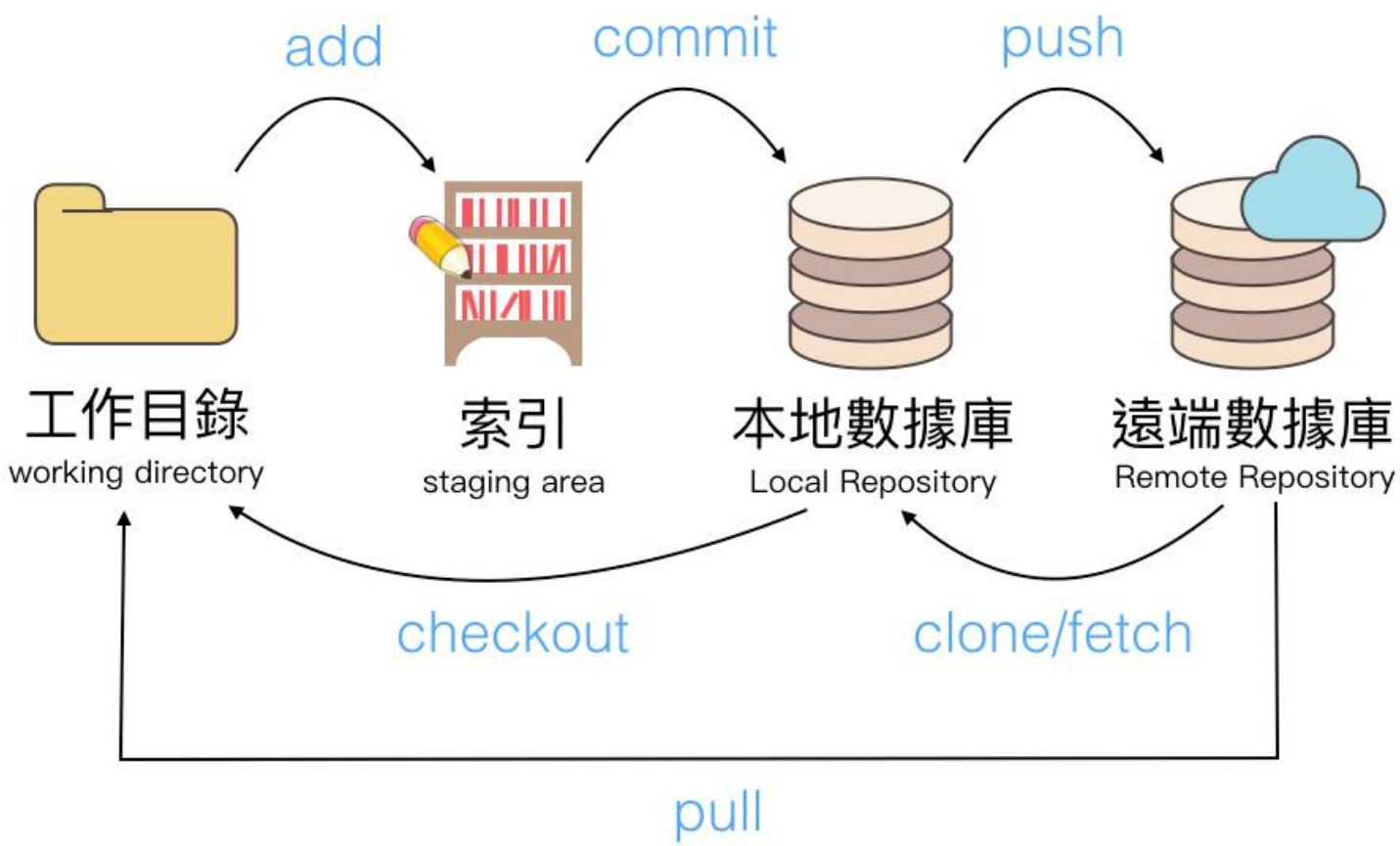
資料來源：連猴子都能懂的Git入門指南

數據庫 (Repository)

- 數據庫 (Repository) 是記錄檔案或目錄狀態的地方
  - 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。



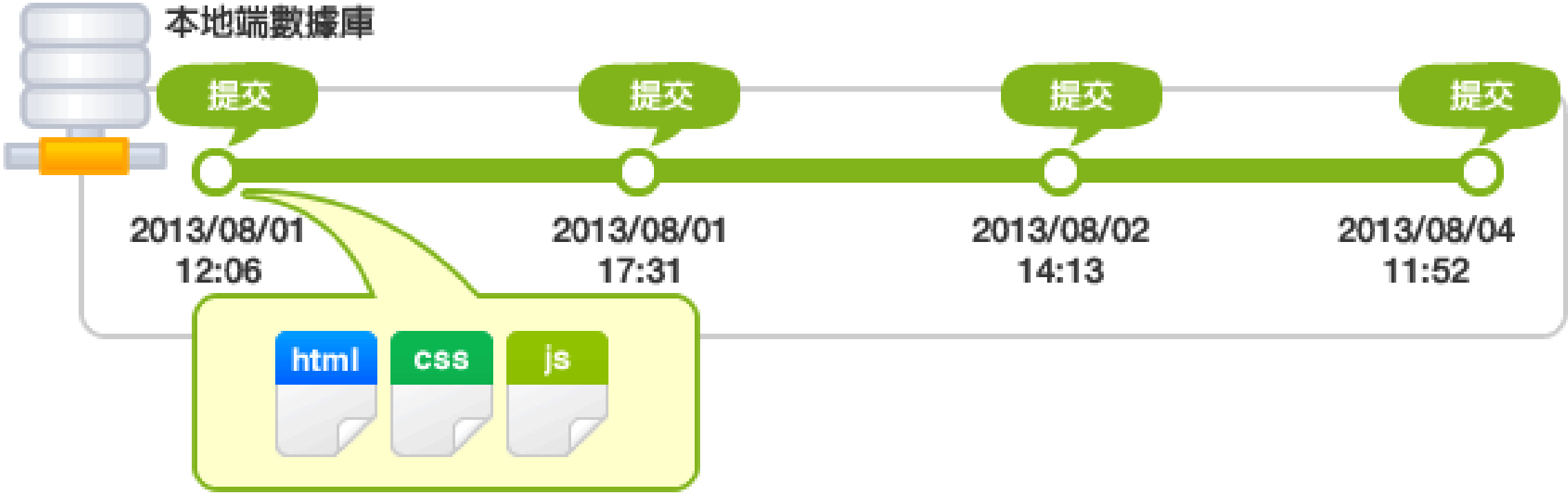
數據庫 (Repository)



https://w3c.hexschool.com/git/7ca21e02

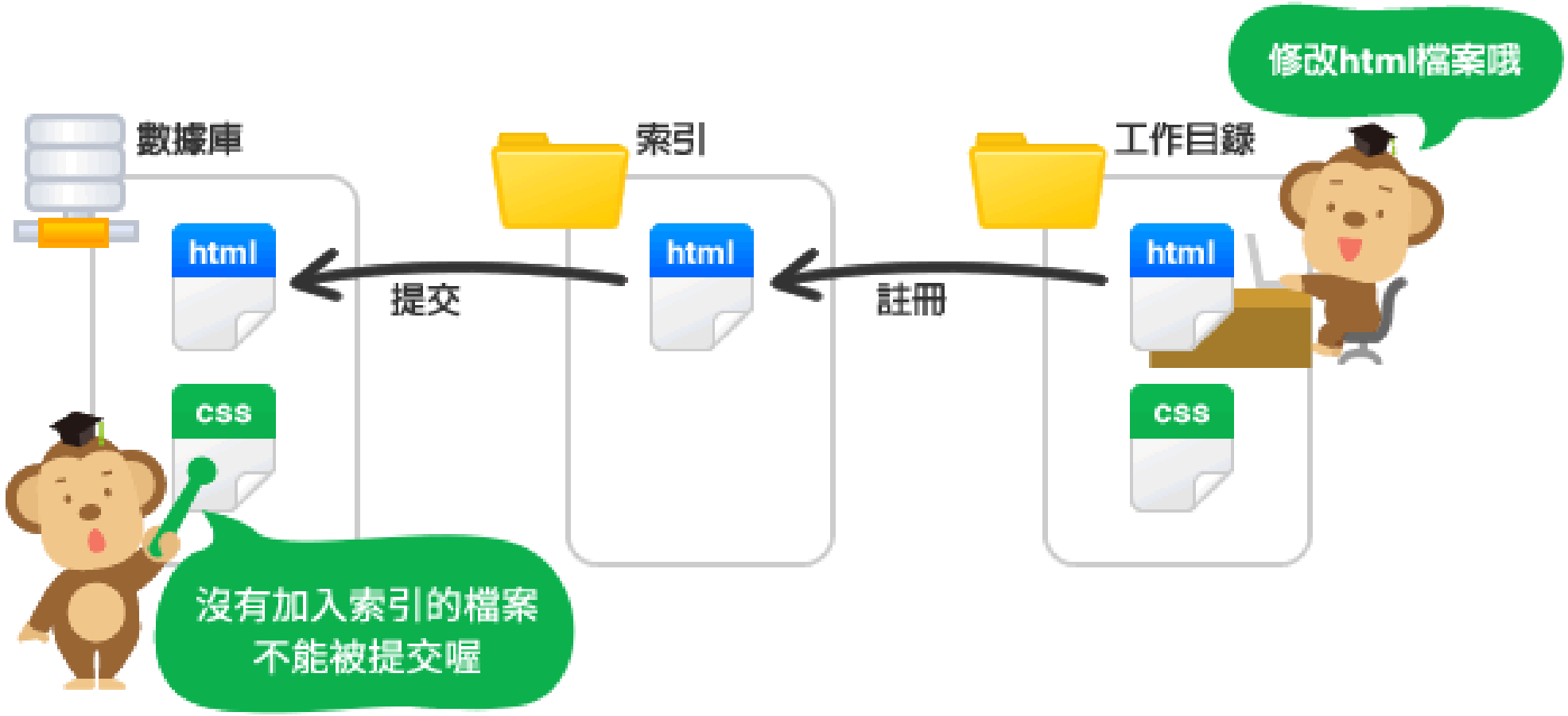
記錄修改的提交

- 執行提交後，數據庫裡會產生上次提交的狀態與現在狀態的差異記錄（也被稱為Revision）。



工作目錄與索引

- 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。



安装 git

- Command Line
  - https://git-scm.com/
  - https://gitforwindows.org/

- Windows(GUI)
  - Sourcetree (https://www.sourcetreeapp.com/)
  - TortoiseGit (https://tortoisegit.org/)

Sourcetree

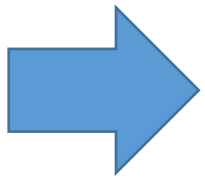
TortoiseGit
Windows Shell Interface to Git

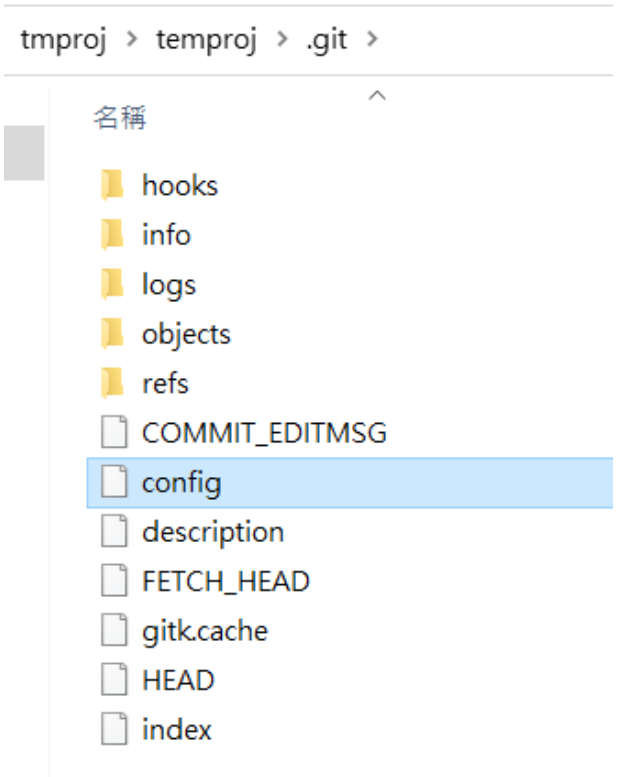
Git 初始設定

- 切換到工作目錄
- 設定姓名、電子郵件

```
git version
git config –global user.name “gbox”
git config –global user.email “lch.gbox@gmail.com”
```

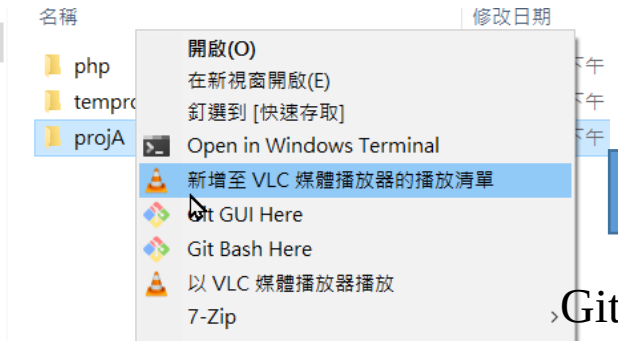
設定檔案
.git/config





- 在該目錄中建立數據庫(Repository)

```
git init
```

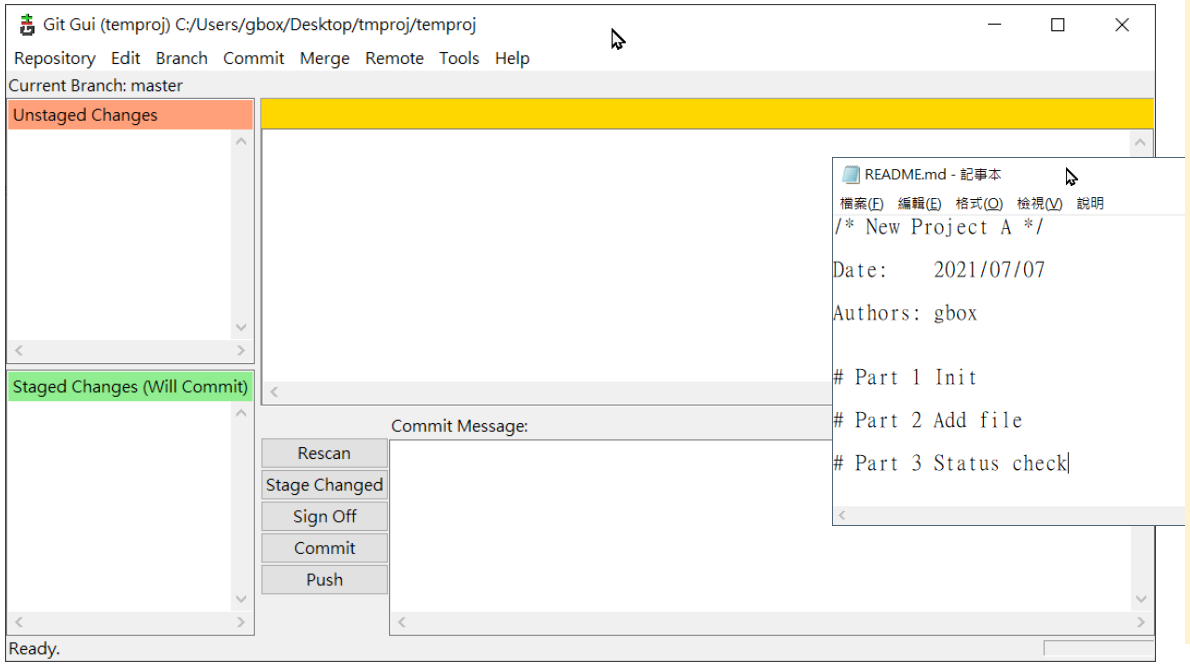


Git GUI Here

git init

開始使用Git

- 查看狀態
  - git status
- 在目錄中建立檔案
  - 如: README.md



D:\work> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

D:\work> git status
On branch master

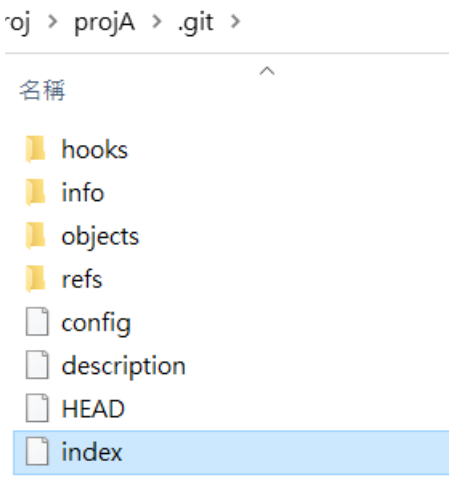
No commits yet

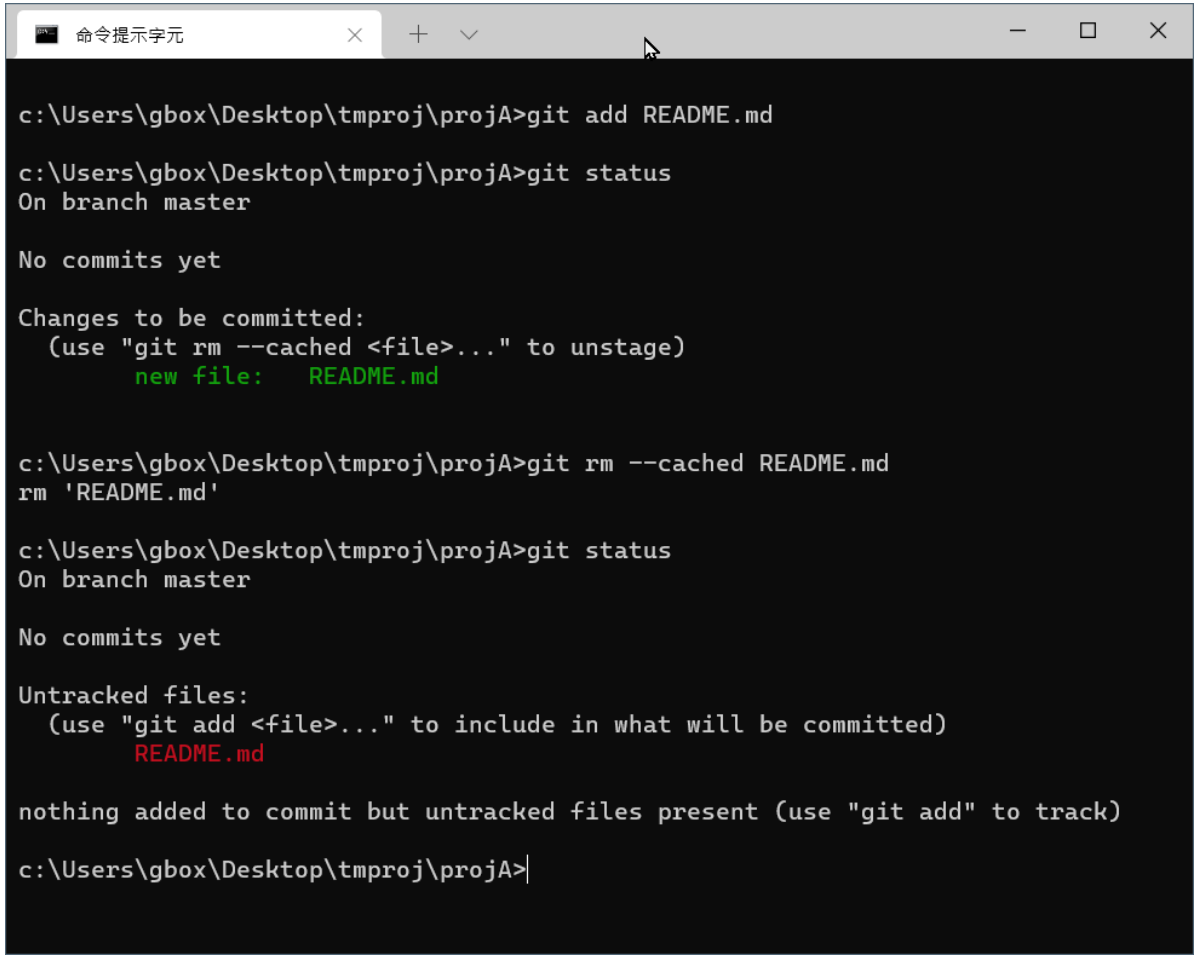
Untracked files:
(use "git add <file>..." to include in what will be committed)
README.md

nothing added to commit but untracked files present (use "git add" to track)

開始使用Git

- 加入新檔案到暫存區(stage changed)
  - git add <filename>
  - git add -A
- 從暫存區刪除檔案
  - git rm –cached <filename>
- 檔案進入index紀錄
  - 尚未commit

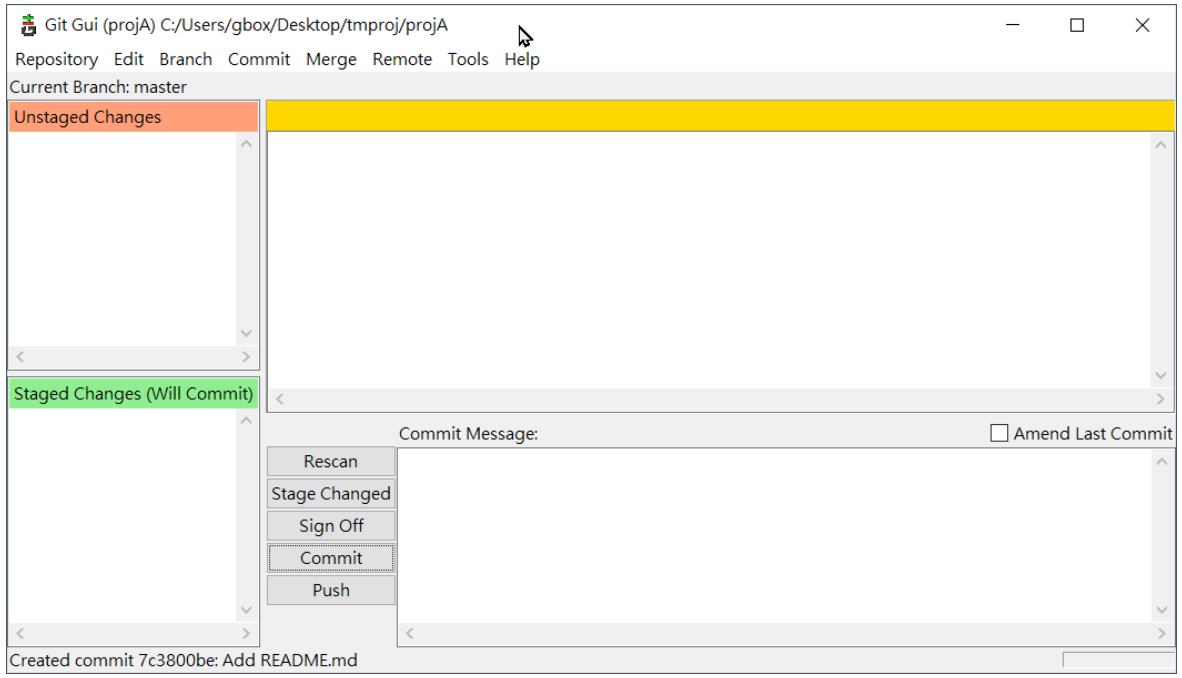


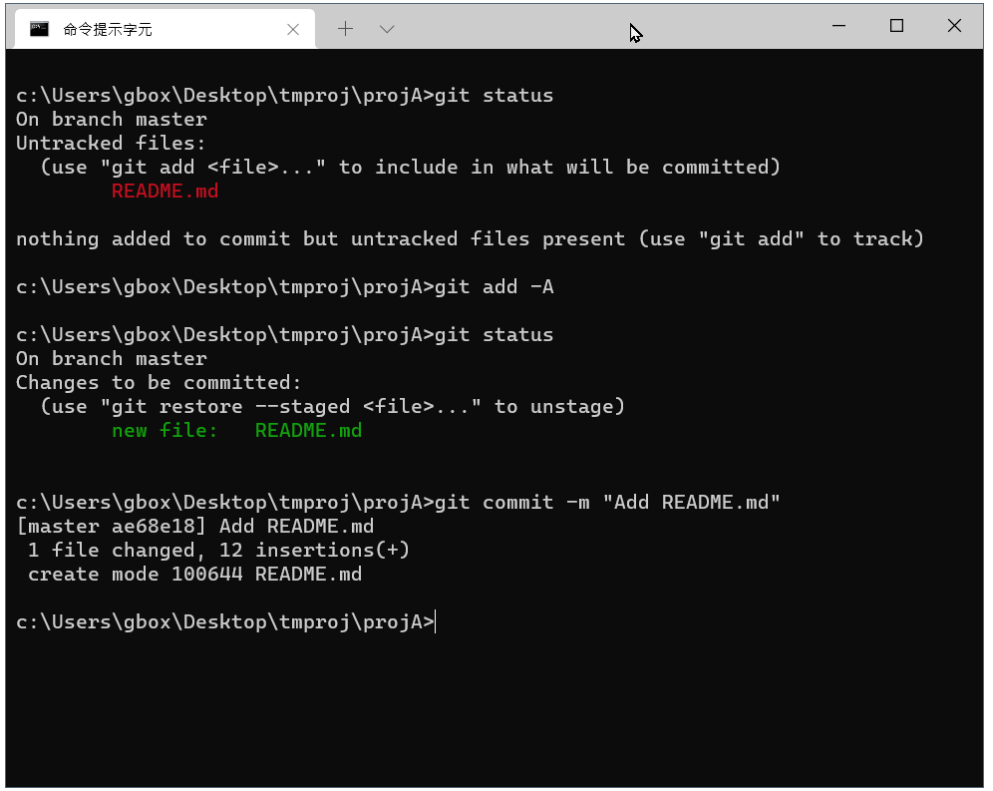


開始使用Git

- 提交到本地數據庫(repository)
  - git commit –m “add README.md”

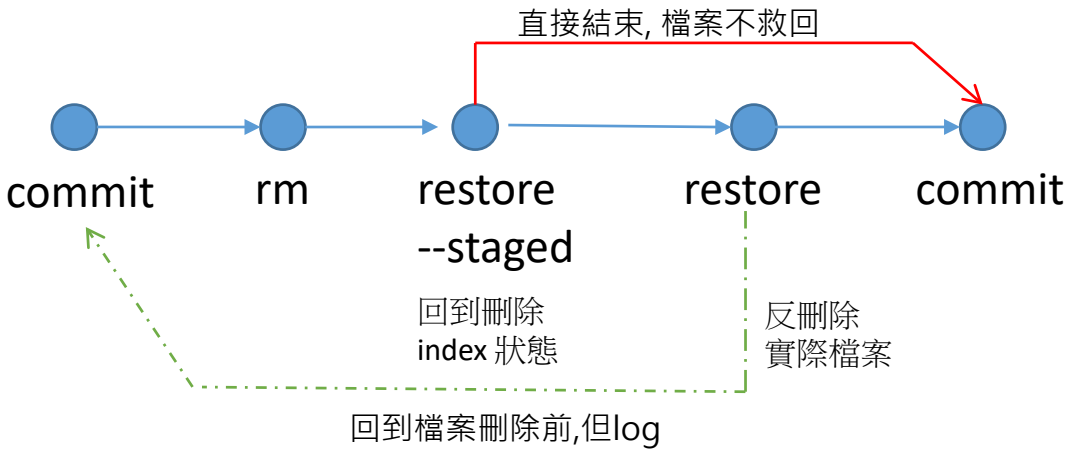
Note:
commit後, staged 所有狀態會寫入數據庫。
視為定稿沒有反悔commit。





開始使用Git

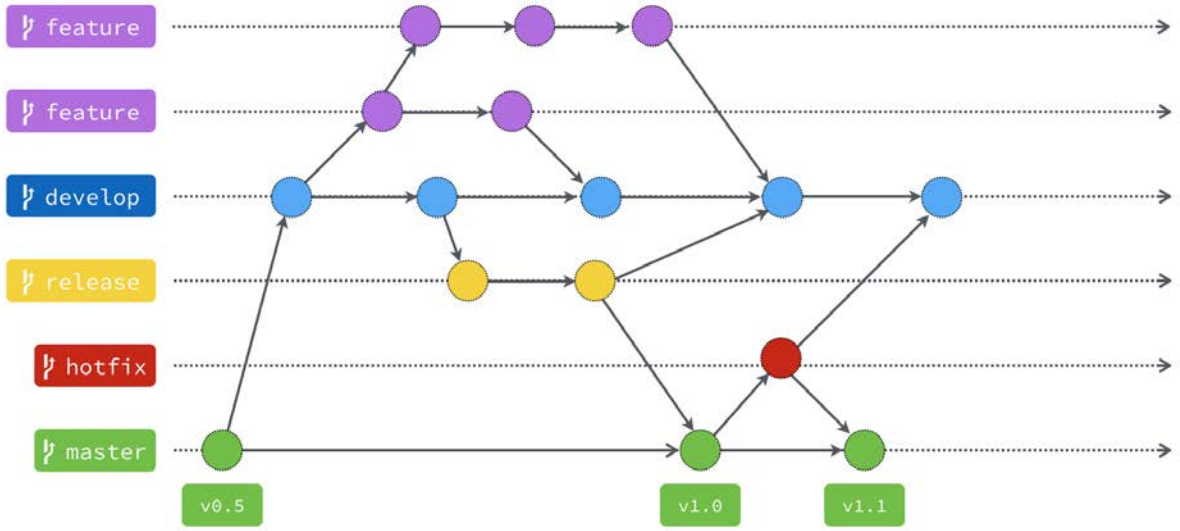
- 刪除檔案
  - git rm <filename>
- 檔案復原
  - git restore –staged <filename>
  - git restore <filename>




```
c:\Users\gbox\Desktop\tmproj\projA>git restore --staged README.md

c:\Users\gbox\Desktop\tmproj\projA>dir
 磁碟區 C 中的磁碟沒有標籤。
 磁碟區序號:  3A73-AC7C

 c:\Users\gbox\Desktop\tmproj\projA 的目錄

2021/07/09  下午 04:42    <DIR>          .
2021/07/09  下午 04:42    <DIR>          ..
               0 個檔案                0 位元組
               2 個目錄  313,520,013,312 位元組可用

c:\Users\gbox\Desktop\tmproj\projA>git restore README.md

c:\Users\gbox\Desktop\tmproj\projA>dir
 磁碟區 C 中的磁碟沒有標籤。
 磁碟區序號:  3A73-AC7C

 c:\Users\gbox\Desktop\tmproj\projA 的目錄

2021/07/09  下午 04:45    <DIR>          .
2021/07/09  下午 04:45    <DIR>          ..
2021/07/09  下午 04:45               124 README.md
               1 個檔案              124 位元組
               2 個目錄  313,519,947,776 位元組可用
```

分支(Branch)

- Branch
  - 分別不同功能與版本
  - 可協作開發
  - 開始會有master branch
  - 遠端數據庫為origin branch
- Head
  - 當前使用branch 最後一次commit



http://nvie.com/posts/a-successful-git-branching-model/

分支(Branch)

- 查看當前branch
  - git branch
- 建立新branch
  - git branch <name>
- 切換到branch
  - git checkout <branch_name>
- 建立並切換到新branch
  - git checkout –b <name>
- 刪除分支
  - git branch -D <name>

D:\>git branch add_module

D:\> git branch
  add_module
* master

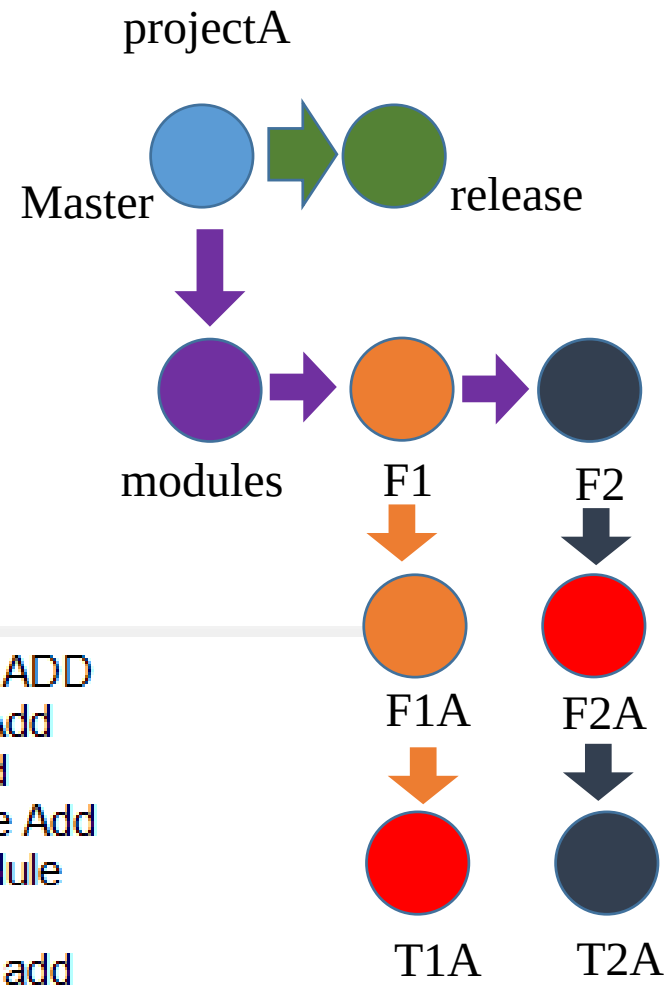
D:\> git checkout add_module
Switched to branch 'add_module'

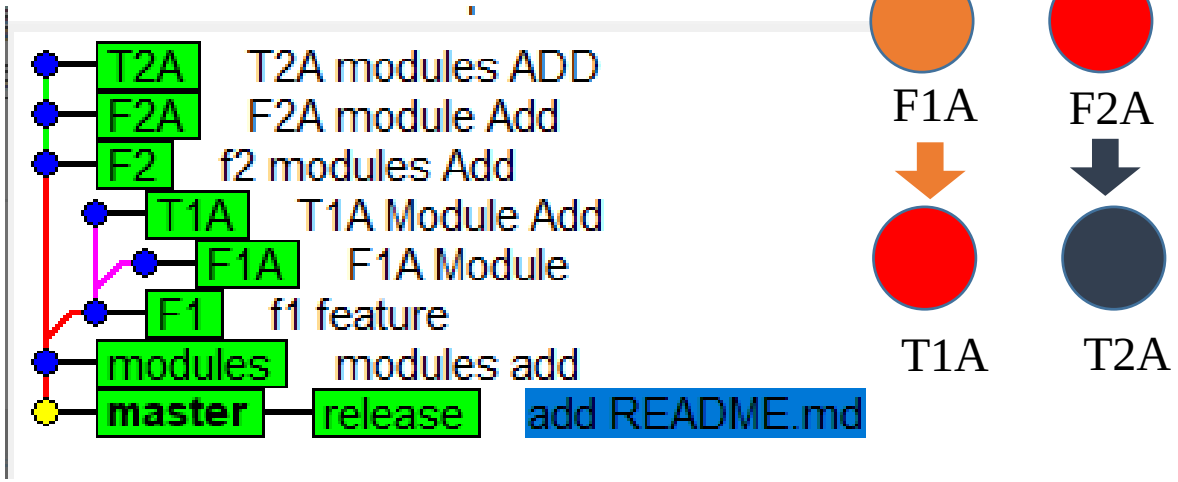
D:\> git branch
* add_module
master

分支(Branch)

分支可以建立多個，互不干擾

- **練習：**
  - 建立一個projectA repository
    - 建立README.md
  - 建立modules & release branch
  - 在modules branch
    - 建立modules 目錄及建立README.md
    - 建立F1 &F2 branch
  - 在F1 branch
    - 建立f1目錄及目錄內建立f1.md
    - 建立F1A &T1A branch
      - 建立F1A&T1A目錄及目錄內建立f1A.md &T1A.md
    - 刪除T1A branch
  - 在F2 branch
    - 建立f2目錄及目錄內建立f2.md
    - 建立F2A &T2A branch
      - 建立F1A&T1A目錄及目錄內建立f1A.md &T1A.md
    - 刪除F2A branch





分支(Branch)

- 刪除分支
  - git branch –d/-D <Branch_name>
- 合併分支
  - git merge <Branch_name>
  - 將指定的<Branch_name>合併至當前分支
  - 衝突解決

- 建立遠端分支
  - git checkout <Branch_name>
  - git push origin <Branch_name>
- 本地<Branch_name> ->遠端改名
  - git push origin <Branch_name> **:** <Remote Branch_name
- 刪除遠端分支
  - git push origin **:**<Branch_name>

分支(Branch)

- 看日誌
  - git log
  - git log --online
  - git log --all --decorate --oneline –graph
- 相關操作紀錄
  - Git reflog

.git/config可以設置別名(alias)

```
[alias]
    co = checkout
    ci  = commit
   dog = --all –decorate –oneline --graph
```

合併分支(merge/rebase)

- 合併基本上是將2分支commit歷史相連接
  - git merge <branch_name>
- 預設模式
  - --ff, Fast-forward
    - 快速進行分之合併動作，以增加行方式合併檔案內容與新增檔案。不進行新的commit
  - --no-ff, recursive
    - 分支變動，個別紀錄，會進行commit
- 將當前branch移動到<branch_name>做為新commit，並且合併兩分支。(合為一支)
  - git rebase <branch_name>
  - 問題：安全、可追蹤
  - 黃金法則：絕對不要再公共分支(master)使用 rebase

合併分支

- 衝突

D:\> git merge F1
CONFLICT (add/add): Merge conflict in a.txt
Auto-merging a.txt
Automatic merge failed; fix conflicts and then commit the result.

D:\> type a.txt
<<<<<<< HEAD
ddd
=======
ccc
>>>>>>> F1

F2

F1

F2 分支中合併F1分支，a.txt發生衝突，git 會將衝突部分合併至檔案中，再有使用者修改決定。

F1

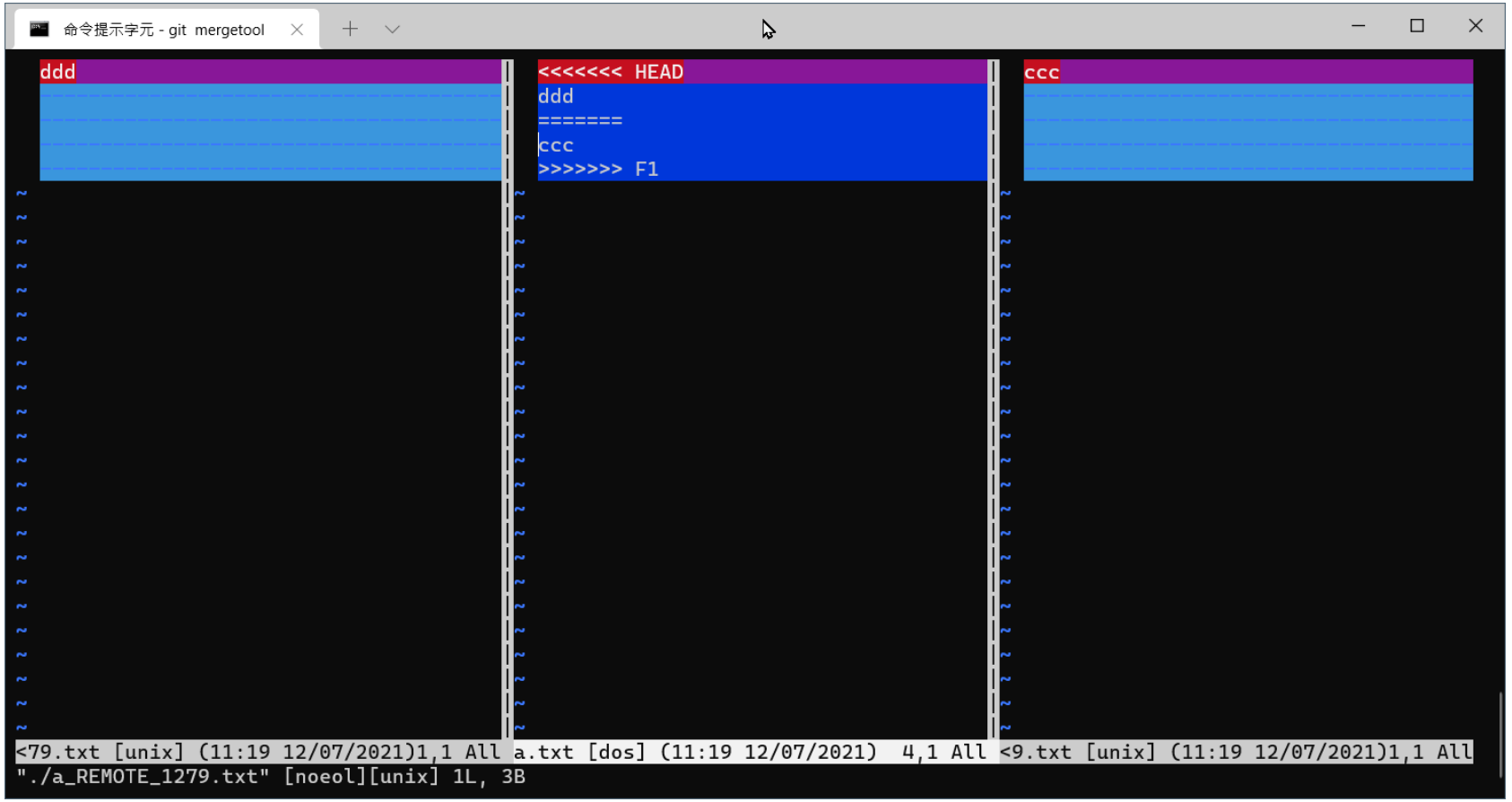
a.txt
ccc

F2

a.txt
ddd

合併分支

- 合併衝突工具
  - git mergetool
  - vimdiff
- 會產出.origin備份檔案**建議刪除**
- **大多會使用工具來進行**



回到過去(設銷變更、回滾撤銷)

- **Reset : 前往或設定HEAD**
- 回到commit/branch
  - 回到上commit狀態：git reset <branch_name>^
  - 回到上2 commit狀態：git reset <branch_name>^^
  - 回到上5次commit狀態：git reset <branch_name>~5
  - 回到特定commit/branch狀態： git reset <branch id>
- Hard/Soft
  - 回到該commit/branch移除不相關檔案：git reset –hard <branch id>
  - 回到該commit/branch保留檔案等待commit： git reset –soft <branch id>

練習：
下列指令意義?
- git reset –HEAD HEAD
- git reset –HEAD HEAD^

回到過去(撤銷變更、回滾撤銷)

- Revert：撤銷某次操作，並包保留此次操作之前和之後的commit, history，並把這次撤銷作為最新的commit。
- 應在公共分支(Master)使用
- 語法與reset用法相同
- 與reset差異在於多commit

gitignore

- 可以建立.gitignore包含不想上傳或需要排除到遠端repository的檔案
- 內容包含副檔名或目錄/.

https://www.toptal.com/developers/gitignore

遠端Repository

- 取得遠端Repository檔案
  - 下載遠端分支：git clone <URI>
  - 下載遠端分支：git pull <URI>
    - Pull = fetch + merge
  - 查看目前遠端分支：git remote -v
- fork
  - 如fork其他人的分支時，要跟自己修改合併
  - 新增遠端上游倉庫：git remote add <new_name> <URI>
  - git fetch <new_name> <*branch_name*>
  - 查看遠程分支 ： git branch –r
  - 合併遠程到本地master： git merge <new_name>/<branch_name>
  - 有貢獻新的用merge,如果只是fork，用rebase

URI：

git://git@url/xx.git

https://url/

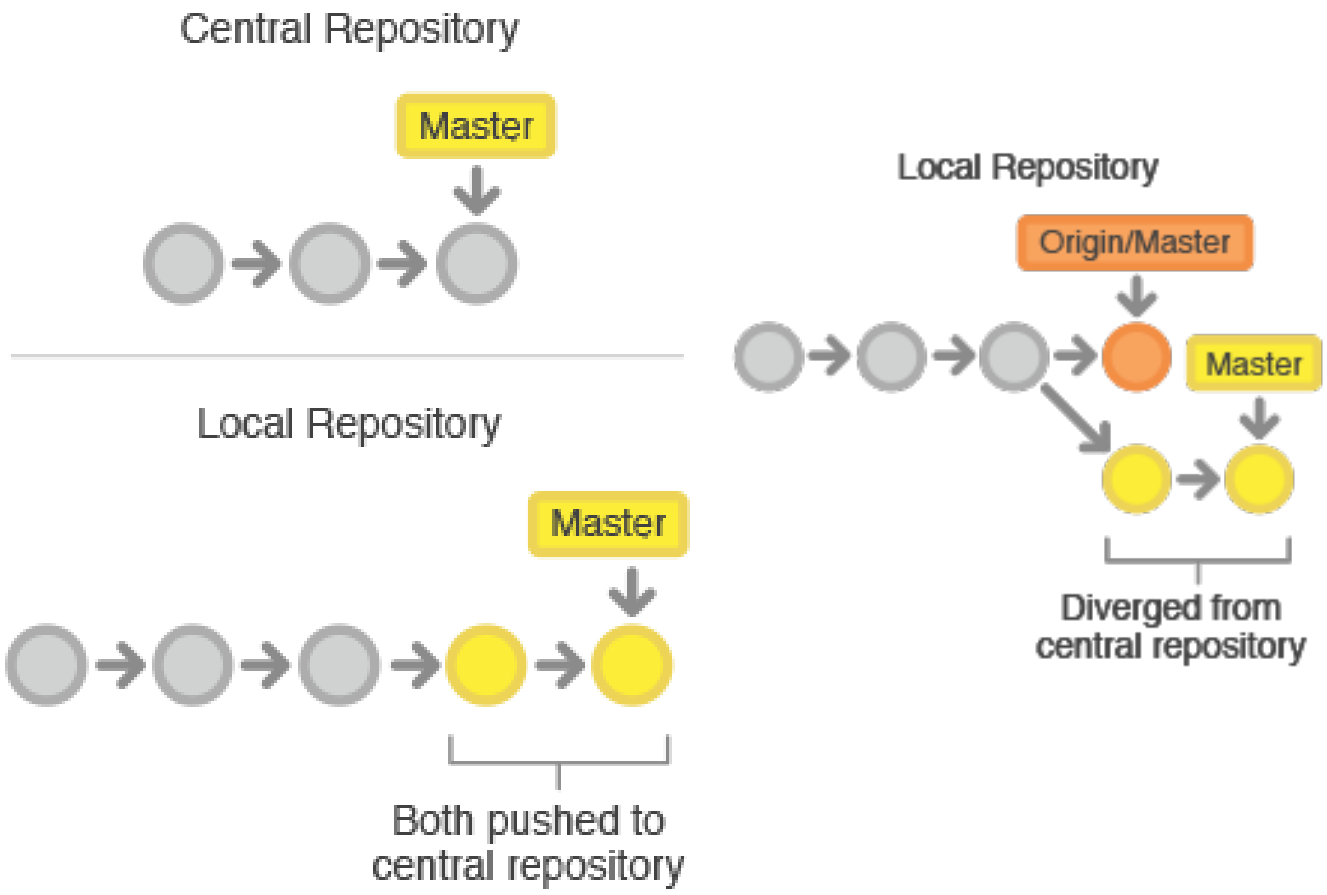
ssh://user@URL

遠端Repository

- 上傳到遠端repository
  - git push
- 權限問題

工作流

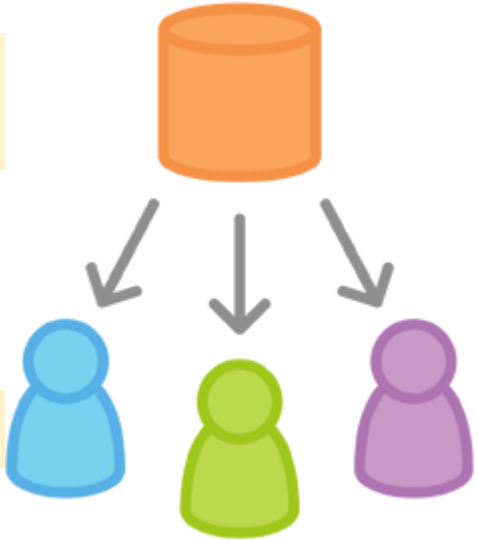
- 中央repository 應該是穩定commit分支
- 在開發者提交自己功能修改到中央前，需要先fetch在中央的新增commit，rebase自己提交到中央提交歷史之上。
- 這樣做的意思是在說，『我要把自己的修改加到別人已經完成的修改上。』最終的結果是一個完美的線性歷史



工作流

- **有人先初始化好中央倉庫**
  - **裸倉庫（bare repository），即沒有工作目錄（working directory）的倉庫**

```
ssh user@host
git init --bare /path/to/repo.git
```



- **所有人克隆中央倉庫**

```
git clone ssh://user@host/path/to/repo.git
```

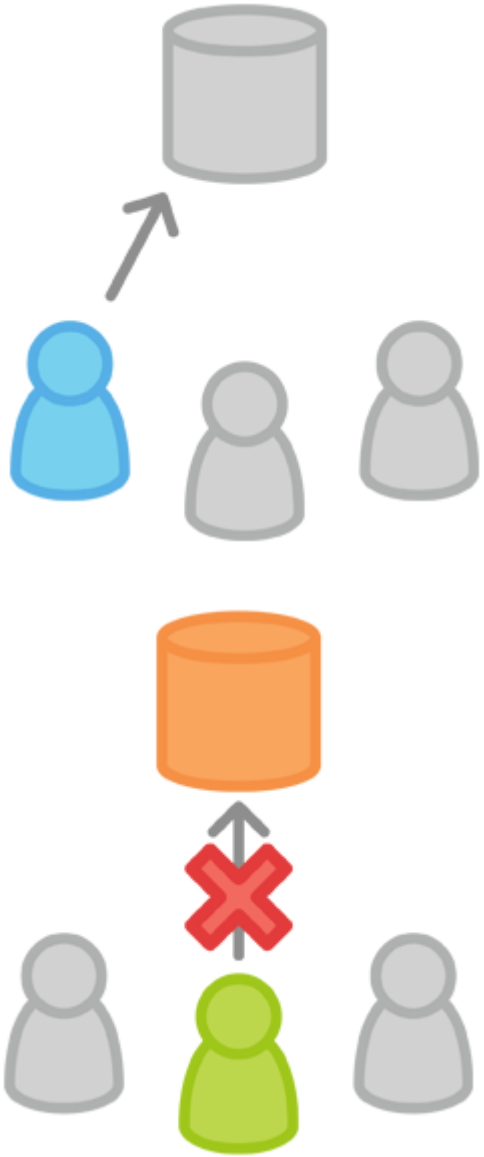
工作流

- **小明發布功能**
  - 小明完成了他的功能開發，會發布他的本地提交到中央倉庫中，這樣其它團隊成員可以看到他的修改。

```
git push origin master
```

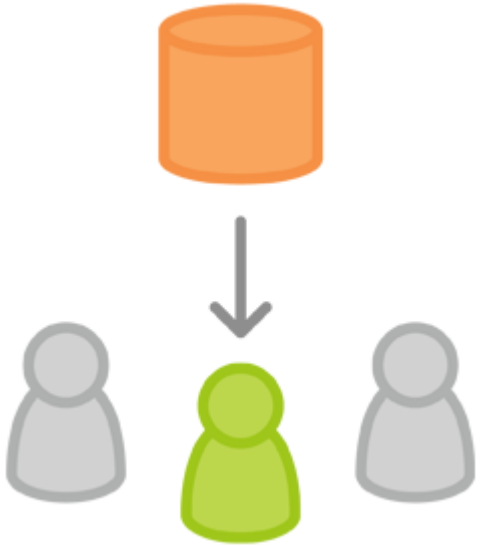
- **小紅試著發布功能**
  - 她的本地歷史已經和中央倉庫有分岐了，Git拒絕操作並給出下面很長的出錯消息她要先pull小明的更新到她的本地倉庫合併上她的本地修改後，再重試。

```
git push origin master

error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')
hint: before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

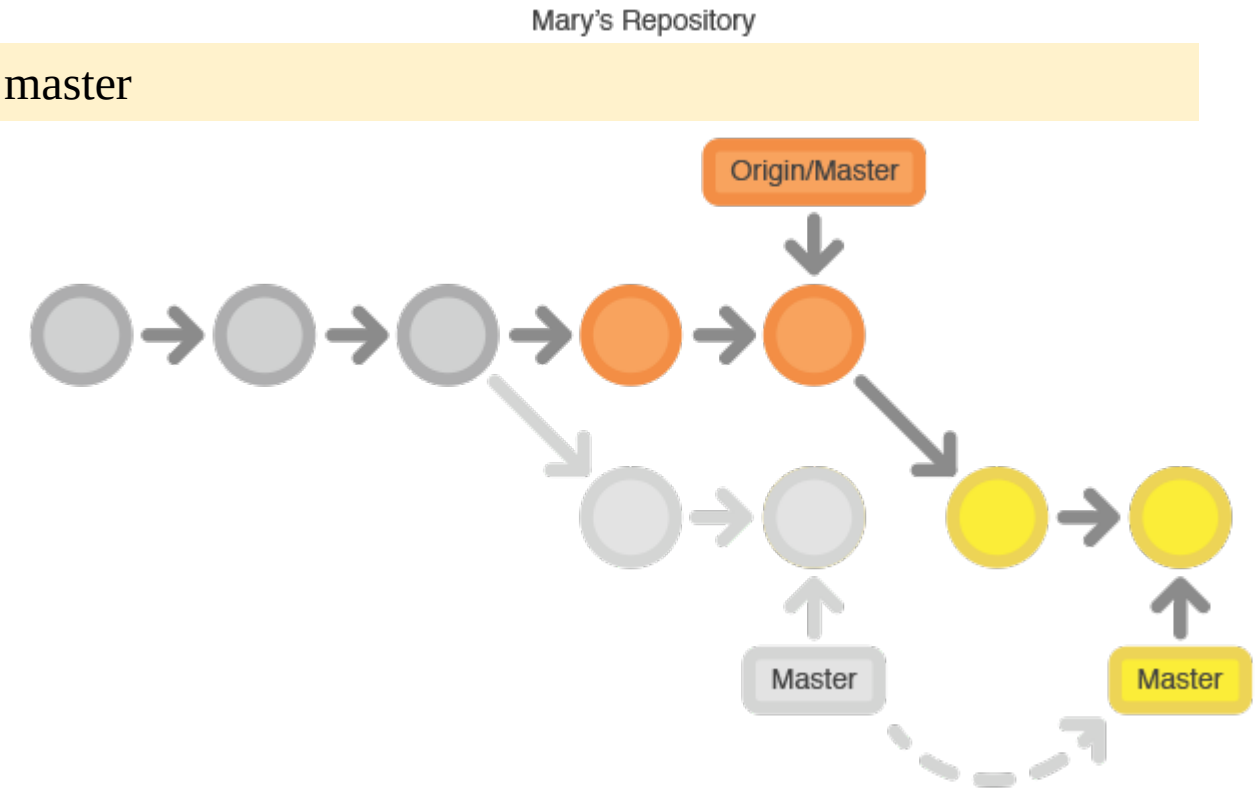


工作流

- 小紅用git pull合併上游的修改到自己的倉庫中。拉取所有上游提交 到小紅的本地倉庫，並嘗試和她的本地修改合併

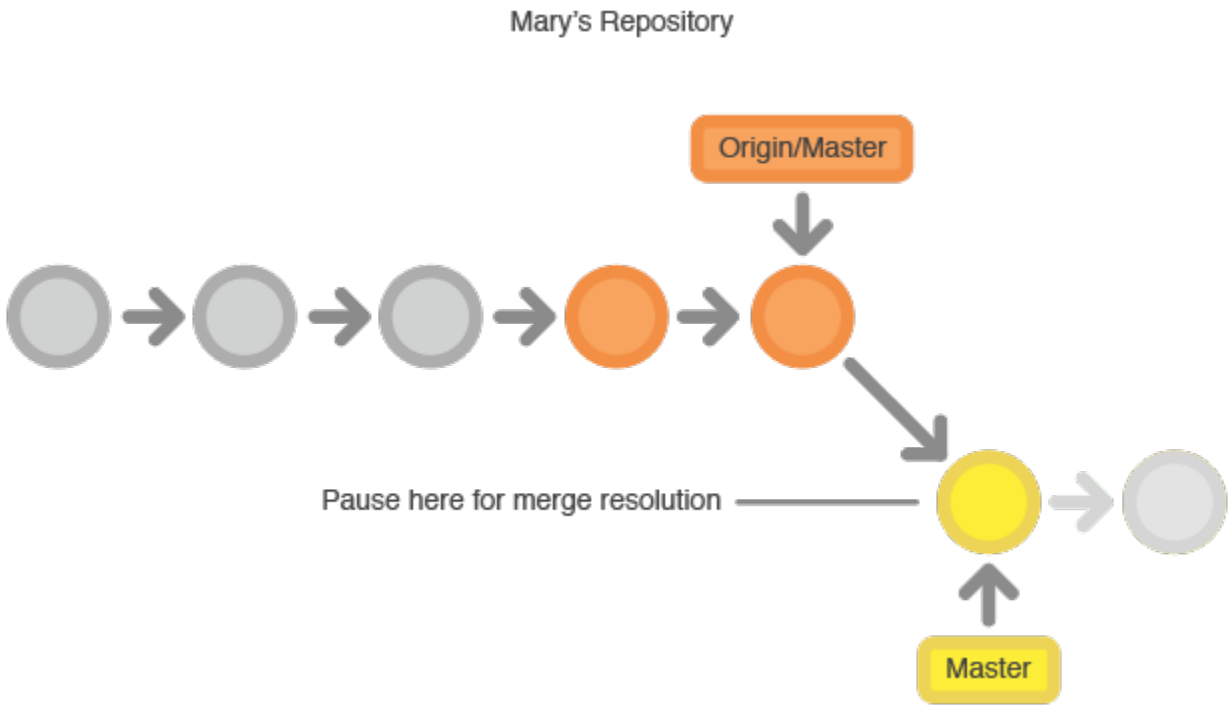


```
git pull --rebase origin master
```



工作流

- rebase操作過程是把本地提交一次一個地遷移到更新了的中央倉庫master分支之上。
  - 這意味著可能要解決在遷移某個提交時出現的合併衝突，而不是解決包含了所有提交的大型合併時所出現的衝突。
  - 這樣的方式讓你盡可能保持每個提交的聚焦和項目歷史的整潔。
  - Git在合併有衝突的提交處暫停rebase過程，輸出下面的信息並帶上相關的指令



CONFLICT (content): Merge conflict in <some-file>

工作流

- 小紅可以簡單的運行git status命令來查看哪裡有問題。衝突文件列在Unmerged paths（未合併路徑）一節中

```
# Unmerged paths:
# (use "git reset HEAD <some-file>..." to unstage)
# (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
# both modified: <some-file>
```

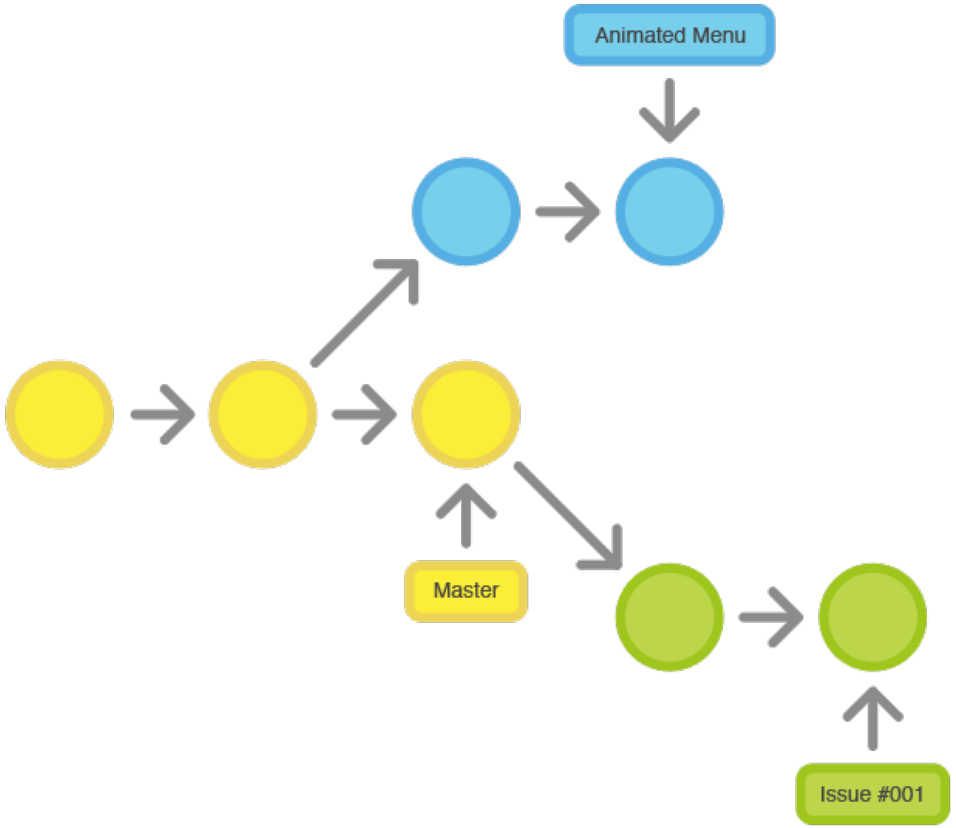
- 小紅編輯這些文件。修改完成後，用老套路暫存這些文件，並讓git rebase完成剩下的事

```
git add < some-file >
git rebase –continue
git push origin master
```

- 如果碰到了衝突，但發現搞不定，不要驚慌。只要執行**git rebase --abort**命令，就可以回到你執行git pull --rebase命令前的樣子

功能分支工作流

- 功能分支工作流背後的核心是所有的功能開發應該在一個專門的分支，而不是在master分支上。

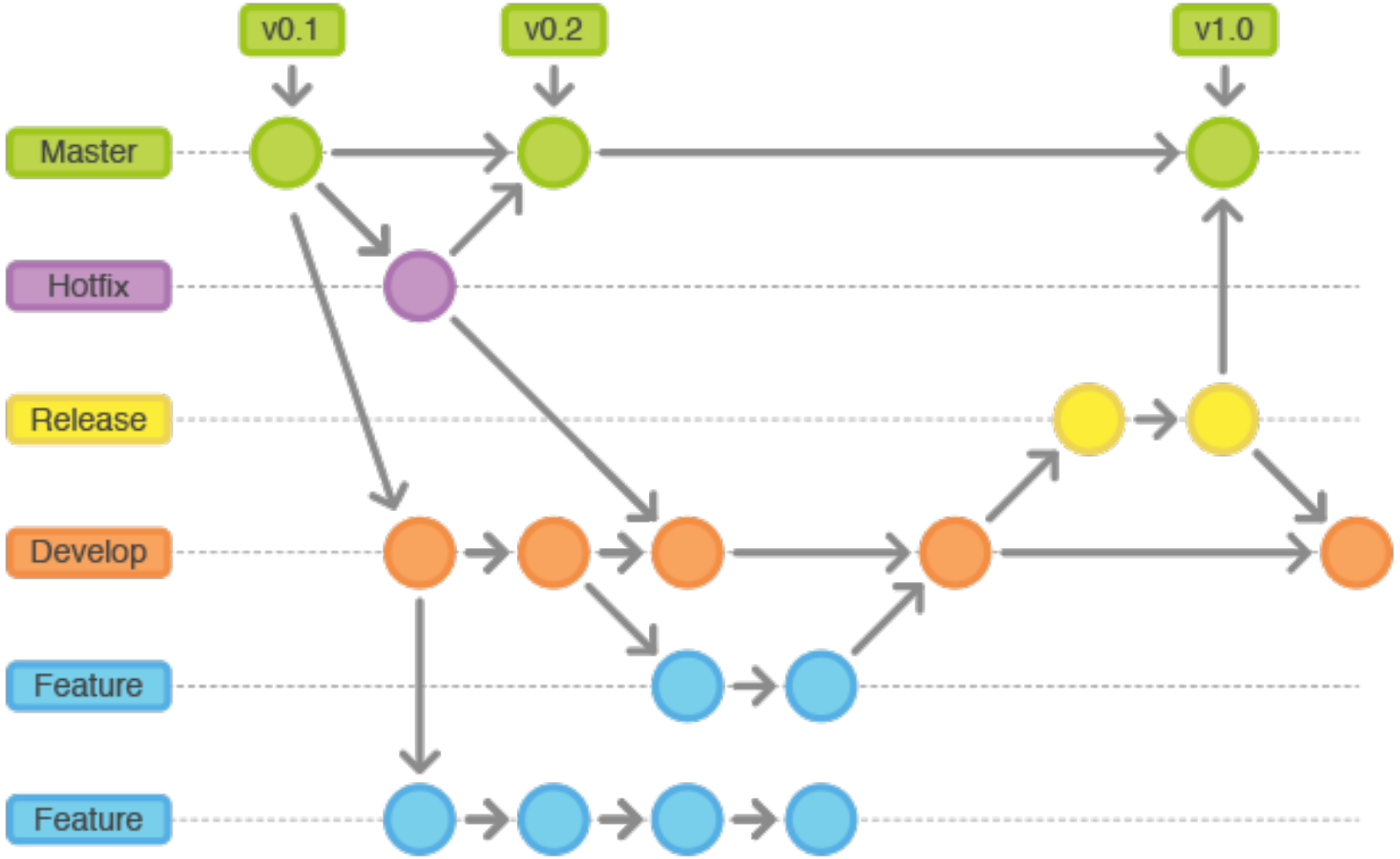


功能分支工作流

- git push -u origin marys-feature
  - 這條命令push marys-feature分支到中央倉庫（origin），-u選項設置本地分支去跟踪遠程對應的分支。設置好跟踪的分支後，小紅就可以使用git push命令省去指定推送分支的參數。

Gitflow工作流

- Gitflow工作流定義了一個圍繞項目發布的嚴格分支模型。



Forking工作流

- Forking工作流是分佈式工作流，充分利用了Git在分支和克隆上的優勢。可以安全可靠地管理大團隊的開發者（developer），並能接受不信任貢獻者（contributor）的提交。
  - Forking工作流要先有一個公開的正式倉庫存儲在服務器上。但一個新的開發者想要在項目上工作時，不是直接從正式倉庫克隆，而是fork正式項目在服務器上創建一個拷貝。