# Git 入門與實踐

蔡宗憲 (Andrew Tsai)

# 為什麼要教 Git

- 版本控制系統是軟體開發最重要的一環
- Git 是目前最流行的版本控制系統之一
- Github 是目前世界上開源軟體 (Open Source Software) 最大的平台
- https://www.udemy.com/git-complete/?siteID=BoHFIyu6APU-_Y89RLcIScBWW3CcLo6_eQ&LSNPUBID=BoHFIyu6APU

# Git 的誕生

- Linux on github, https://github.com/torvalds/linux
- 1991, Linus 創建開源的 Linux
- 2002, BitKeeper
- 2005, Git

# 從情境來看版本控制系統

# 情境一 : 單人修改文件

- 不存檔
- 只存一個檔
  - ~~有些人說子不算很機車~~
- 存很多個檔
  - ~~流水號: 1.txt, 2.txt, 3.txt, 4.txt, …~~
  - ~~隨性編號: aaa.txt, bbb.txt, …~~
  - ~~日期編號: 20171123-1.txt, …~~
  - ~~搞笑編號: 這個只有聰明人才看的到 .txt~~

# 情境二：多人修改文件

- 不確定別人新增的部分的正確性
- 修改非最新文件
- 不小心改到別人新增的部分
- ~~同事離職，文件順便帶走~~
- ~~積怨已深，互相傷害~~

# 情境三:一稿多投 (履歷表)

- 前提
  - 有些部分是給所有公司看的 (姓名, 學歷, ...)
  - 有些部分是針對公司客製化的 (公司所要求的專業技能, ...)
- 狀況
  - 新增學歷
  - 新增公司相關技能 (只有 A & B 公司需要)
- 問題
  - ~~手殘~~:手動複製貼上也貼不完全
  - ~~眼殘~~:複製貼上到錯的位置
  - ~~腦殘~~:忘記自己有沒有做過

# 版本控制系統 (Version Control System, VCS)

- 將檔案集中管理
  - 大家都能隨時取得最新版本
- 每個改變都會被獨立記錄 (原因, 改變時間, 修改人)
  - 可以了解每次改變的原因以及進行了那些改變
  - 可以還原至指定的改變
- 標籤 (Tag)
  - 標記版本號, 用以確定發佈的 內容是否有包含某些改變
- 分支 (Branch)
  - 針對不同目的的發佈

# 版本控制系統在
# 軟體開發流程上的地位

# 代碼審查 (Code Review)

- 利用不同 branch (master branch, feature branch)
- 利用現有工具
  - Gerrit
  - Phabricator

# 需求管理與追蹤

- 前提
  - 每個需求都有相對應的案件編號
  - 每次修改最好只會對應到一個案件編號
  - 在修改提交時，修改記錄要記錄案件編號
- 好處
  - 查看需求時，可以知道相對應的程式碼
  - 查看程式碼時，可以知道相對應的需求

# 持續整合與持續交付

- 持續整合 (Continuous Integration, CI)
- 持續交付 (Continuous Delivery, CD)

# 修改記錄 (change log) 分析與建議

- 找出容易犯錯的工程師~~ (找戰犯)~~
  - 協助熟悉流程
  - 協助熟悉程式碼
- 找出容易出錯的程式碼
  - 程式碼重構 (refactoring)

# 為什麼使用 Git

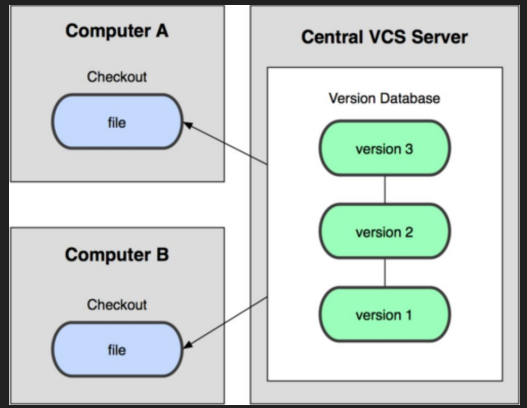
# 分支與合併 (Branching and Merging)

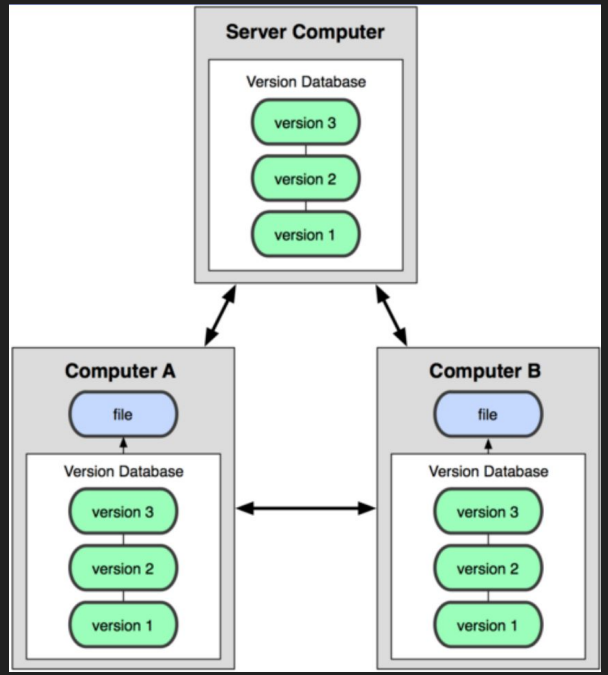
- 可輕易的在分支之間切換
  - Master branch, feature branch
  - 測試
- 當修改在某一個分支後, 可以將修改合併到其他分支
  - Merge
  - Cherry pick
- 潛在問題
  - 亂用分支
  - 分支太多
  - 合併時可能會有問題

# 小且快 (Small and Fast)

# 分散式 (Distributed) (1)

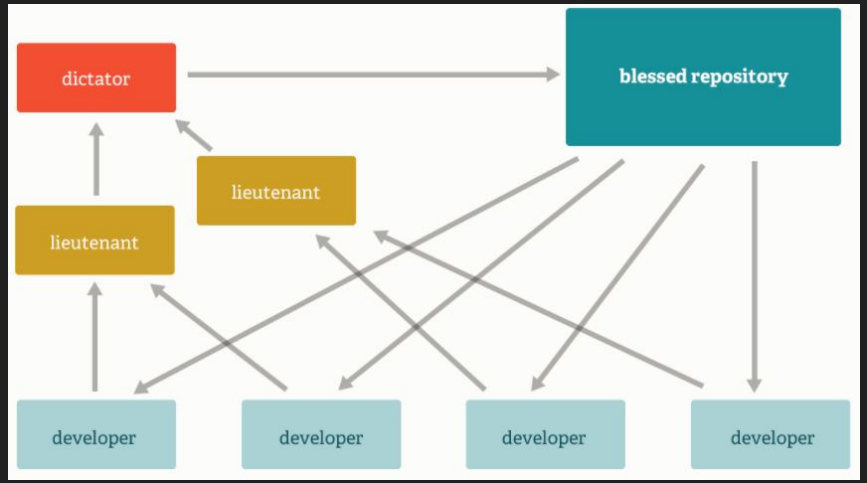
- 本地端也具備版本控制功能
  - 不會因為遠端伺服器連不上而無法進行工作
  - 在提交 (commit) 至遠端司服器前的修改也能進行版本控制
- 多個備份

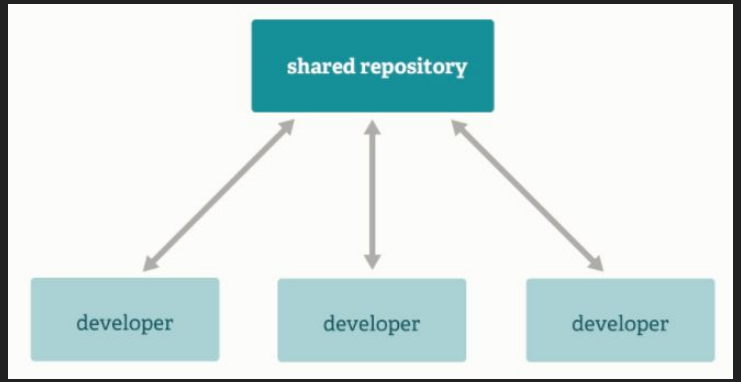


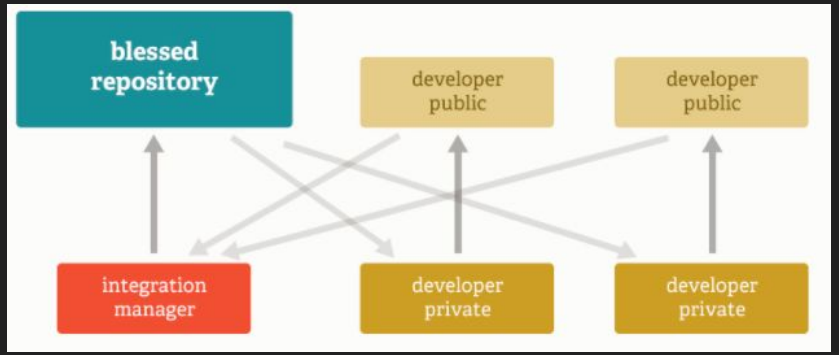



# 分散式 (Distributed) (2)

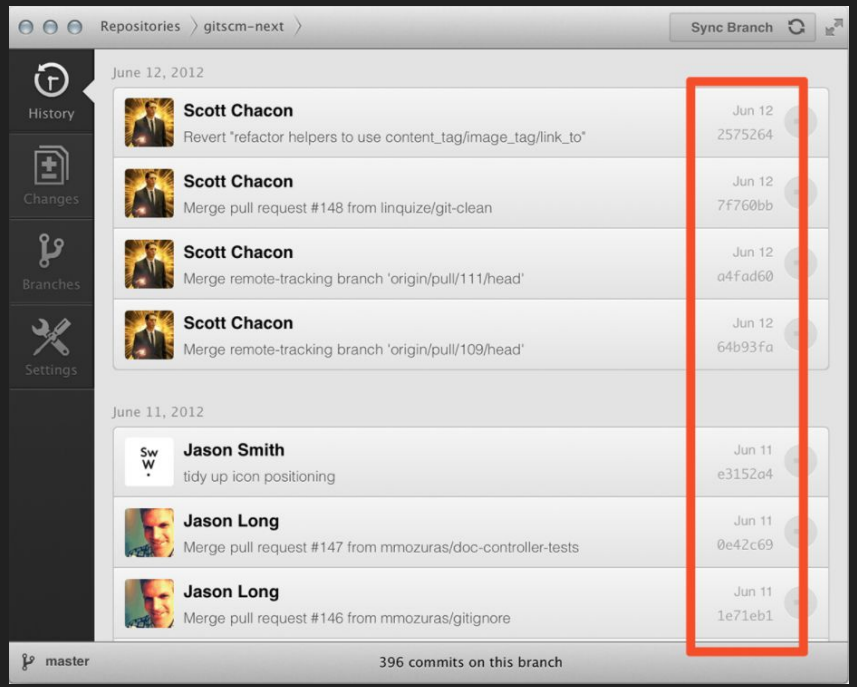
- 多種流程支援



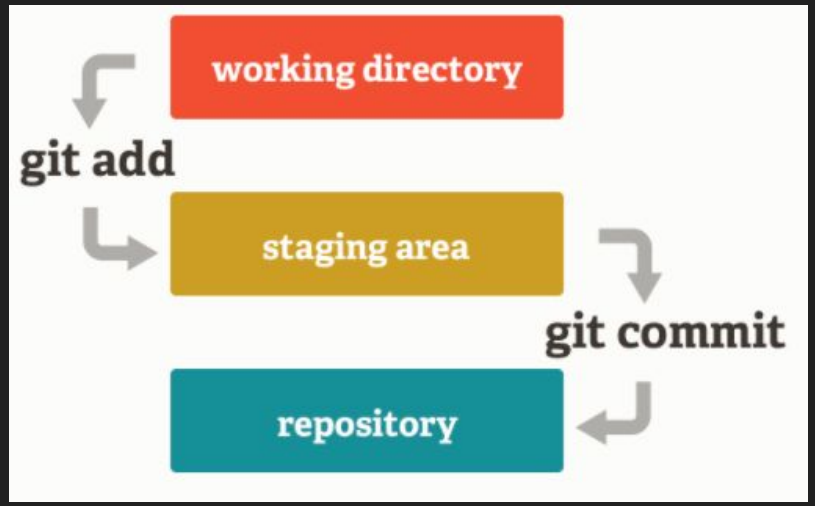



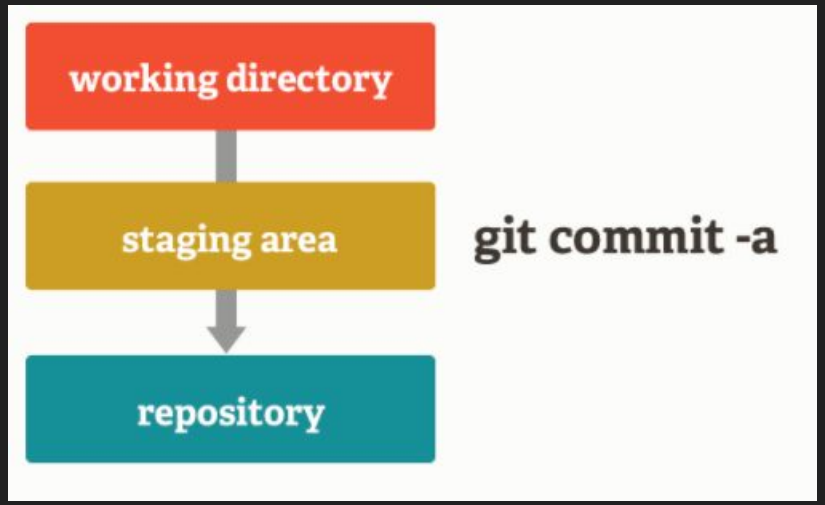



# 資料完整性 (Data Integrity)



# 集結區 (Staging Area)



# 免費且開源 (Free and Open Source)

- https://github.com/git/git

# 他們也都用 Git


Companies & Projects Using Git
Google
facebook
Microsoft
twitter
LinkedIn
NETFLIX
PostgreSQL
android
RAILS
Qt
GNOME
eclipse

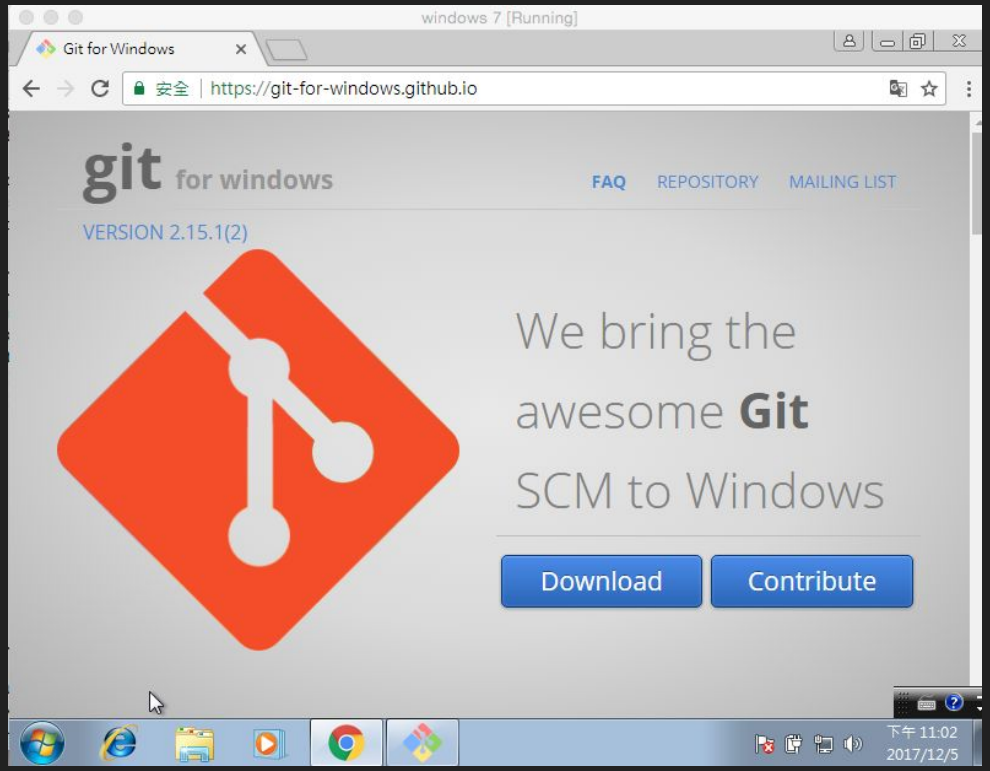
# Git 環境準備

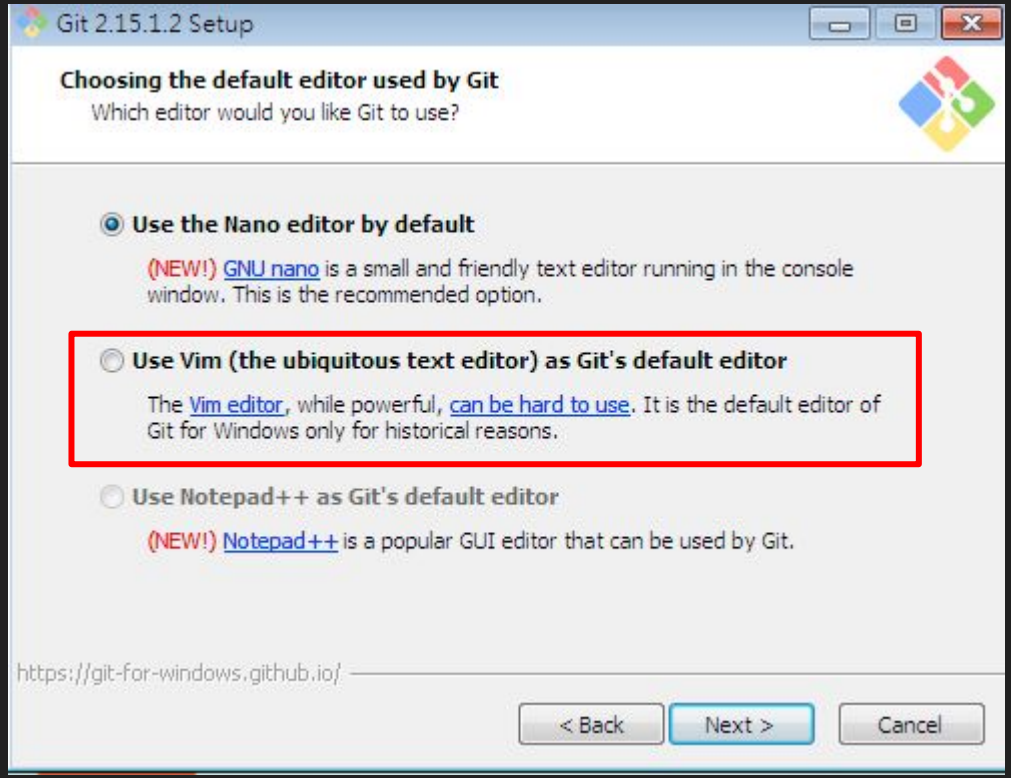
# Git 服務準備

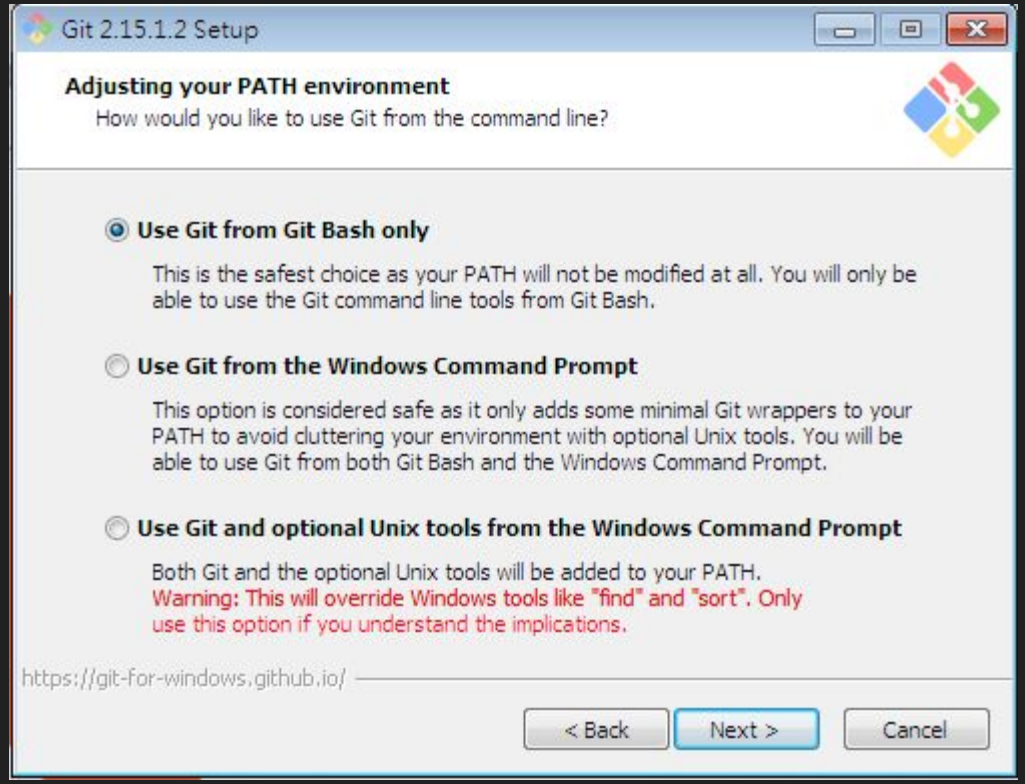
- 自己管理
  - GitLab
  - Phabricator
- 別人管理
  - Github
    - Free for open source (unlimited public repository)
    - https://github.com/
  - Bitbucket
    - Free for small team (unlimited private repository for team less than 5 people)
    - https://bitbucket.org/

# 本地端軟體安裝

- Git for Windows
- GitHub for Windows
- SourceTree
- tortoiseGit

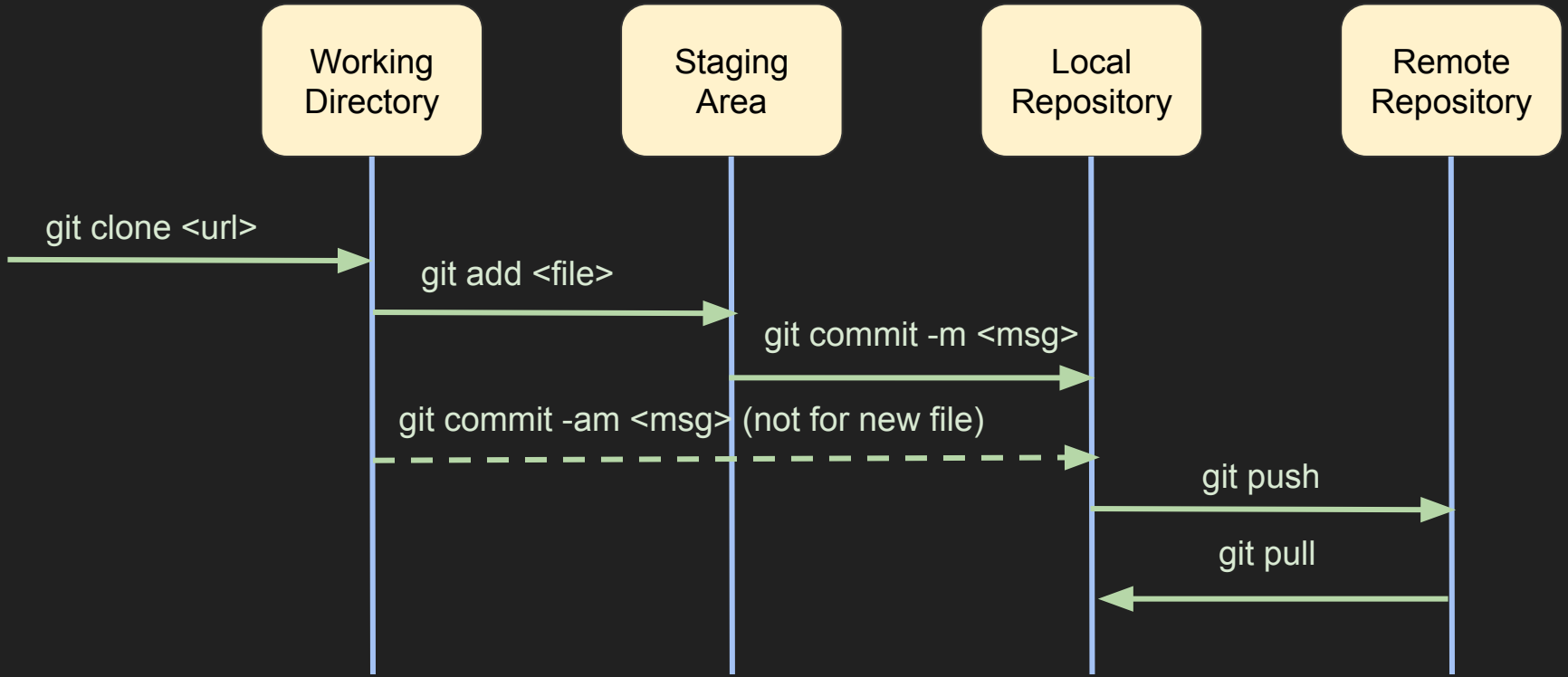

windows 7 [Running]
Git for Windows
安全 | https://git-for-windows.github.io
git for windows
FAQ
REPOSITORY
MAILING LIST
VERSION 2.15.1(2)
We bring the
awesome Git
SCM to Windows
Download
Contribute
下午 11:02
2017/12/5


Git 2.15.1.2 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use the Nano editor by default
(NEW!) GNU nano is a small and friendly text editor running in the console window. This is the recommended option.
Use Vim (the ubiquitous text editor) as Git's default editor
The Vim editor, while powerful, can be hard to use. It is the default editor of Git for Windows only for historical reasons.
Use Notepad++ as Git's default editor
(NEW!) Notepad++ is a popular GUI editor that can be used by Git.
https://git-for-windows.github.io/
< Back
Next >
Cancel

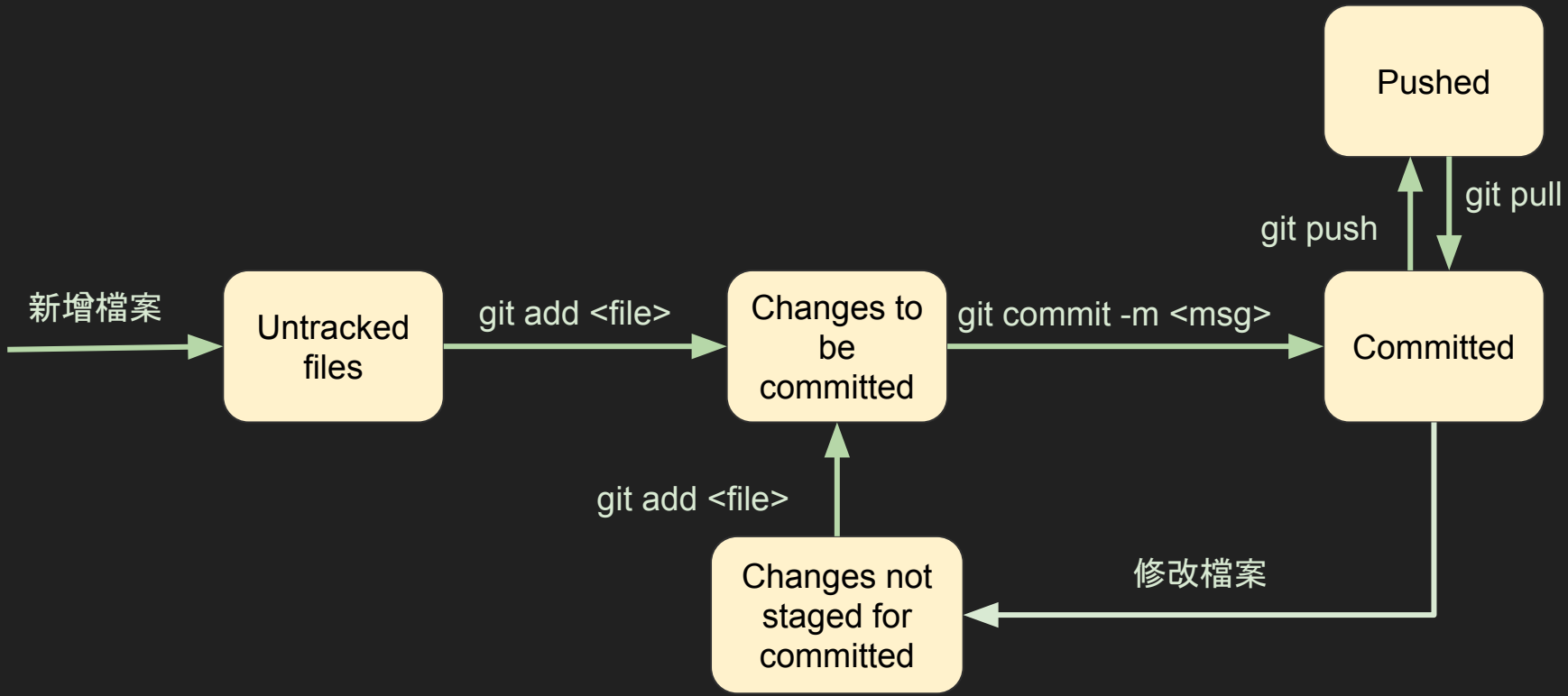

Git 2.15.1.2 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Use Git from the Windows Command Prompt
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.
Use Git and optional Unix tools from the Windows Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://git-for-windows.github.io/
< Back
Next >
Cancel

# 1. Git 正常流程指令

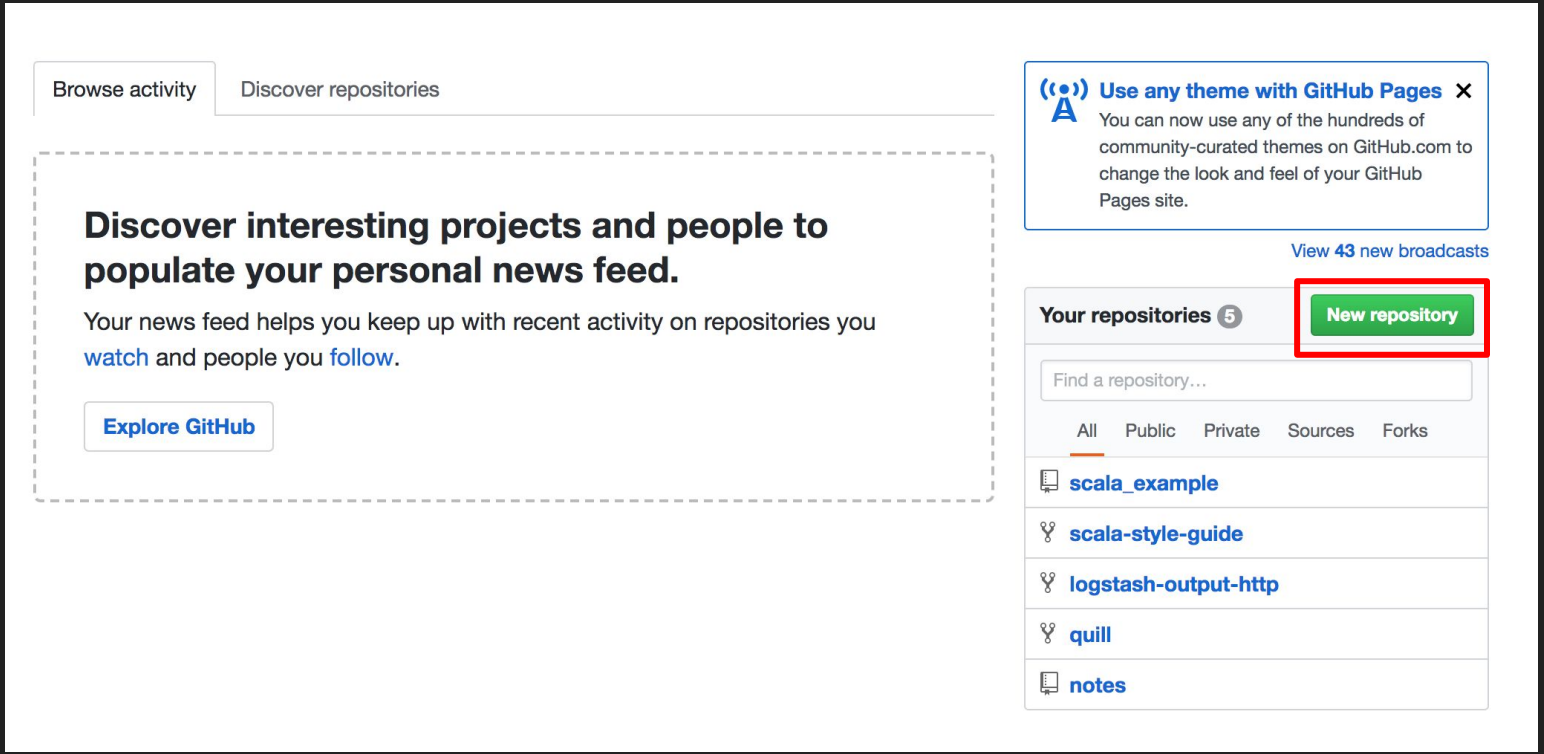
# 空間的轉換


Working Directory
Staging Area
Local Repository
Remote Repository
git clone <url>
git add <file>
git commit -m <msg>
git commit -am <msg> (not for new file)
git push
git pull

# 狀態的轉換


Pushed
git pull
git push
新增檔案
Untracked files
git add <file>
Changes to be committed
git commit -m <msg>
Committed
git add <file>
修改檔案
Changes not staged for committed

# 練習1-1: 建立倉庫 (Repository)



# Create a new repository

A repository contains all the files for your project, including the revision history.

**Owner** apfols / **Repository name** test

Great repository names are short and memorable. Need inspiration? How about **solid-succotash**.

**Description** (optional)

**Public**
Anyone can see this repository. You choose who can commit.

**Private**
You choose who can see and commit to this repository.

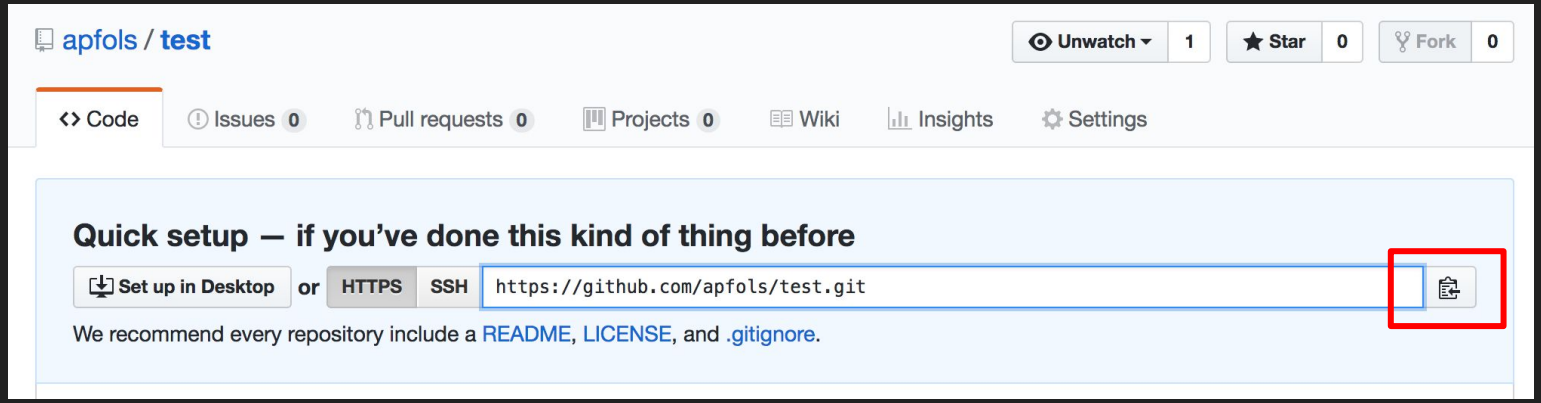
**Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

**Create repository**

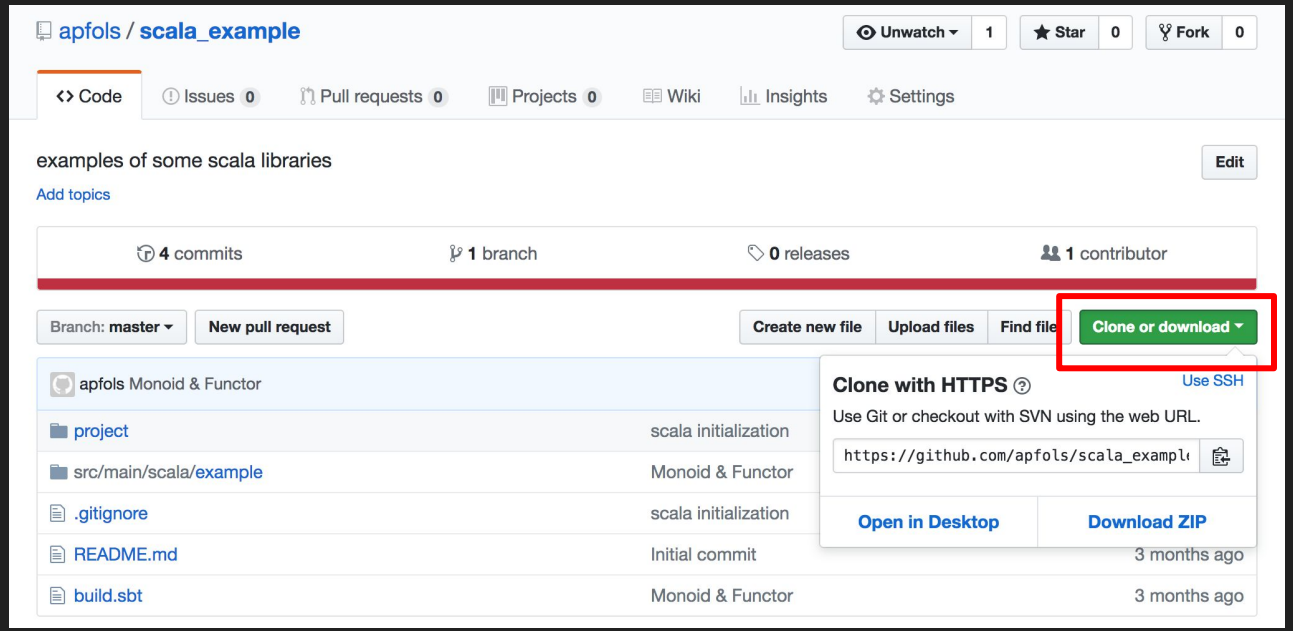
# 練習1-2-1: 建立本地端 (尚未有任何檔案)

- git clone <url>



# 練習1-2-2: 建立本地端 (已有檔案)

- git clone <url>



# 工具1: Git 查詢指令

- 查詢目前檔案狀態
  - git status
  - git status -s
- 查詢目前修改記錄
  - git log
  - git log --oneline
- 練習

# 練習1-3: 新增檔案

- cd <directory>
- echo “hello” > README
- git status

```
apfolsdeMacBook-Pro:course apfols$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README

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

# 練習1-4: 追蹤檔案 (將檔案放入集結區)

- git add README
- git status

```
apfolsdeMacBook-Pro:course apfols$ git add README
apfolsdeMacBook-Pro:course apfols$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README
```

# 練習1-5: 修改在集結區的檔案

- echo “mm” >> README
- git status

```
apfolsdeMacBook-Pro:course apfols$ echo "mm" >> README
apfolsdeMacBook-Pro:course apfols$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README
```

```
apfolsdeMacBook-Pro:course apfols$ git status -s
AM README
```

# 練習1-6: 提交至本地端 (local repository)

- git commit -m “lesson”

```
apfolsdeMacBook-Pro:test apfols$ git commit -m "lesson 1 demo"
[master (root-commit) 2f6eae5] lesson 1 demo
 Committer: apfols <apfols@apfolsdeMacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 2 insertions(+)
 create mode 100644 README
```

- git status

```
apfolsdeMacBook-Pro:test apfols$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)
nothing to commit, working directory clean
```

# 練習1-7: 上傳至遠端 (remote repository)

- git push

```
apfolsdeMacBook-Pro:test apfols$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Username for 'https://github.com': apfols
Password for 'https://apfols@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 222 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/apfols/test.git
 * [new branch]      master -> master
```

- git config --global --edit

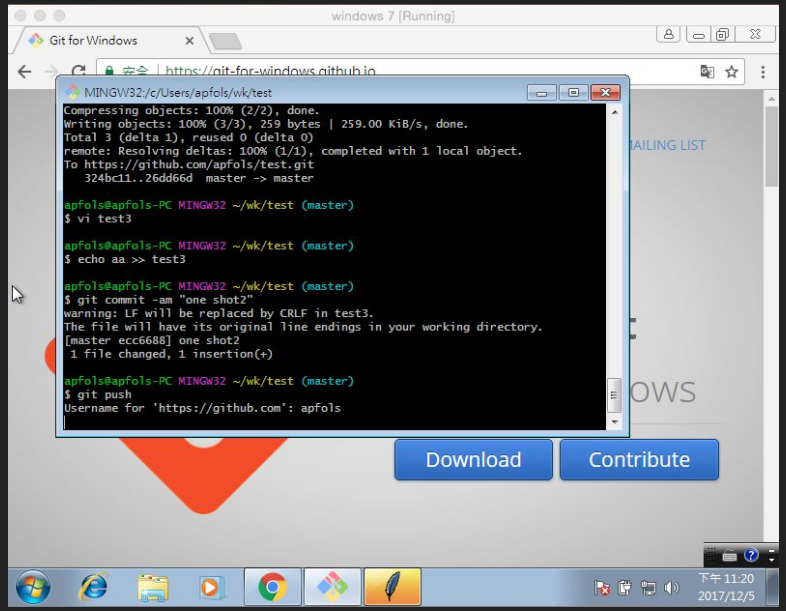
```
# This is Git's per-user configuration file.
[user]
# Please adapt and uncomment the following lines:
        name = apfols
        email = apfols@gmail.com
```

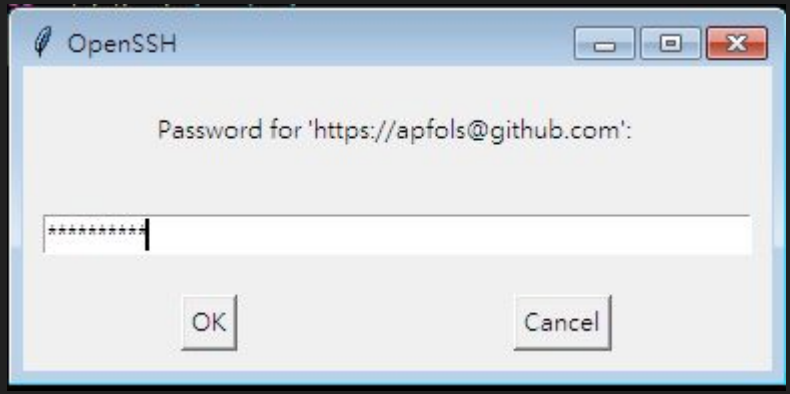
- git config --global push.default matching

```
[master 324bc11] test3
 1 file changed, 1 insertion(+)
 create mode 100644 test3
apfolsdeMacBook-Pro:test apfols$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 288 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/apfols/test.git
   cff3b50..324bc11  master -> master
```

# Windows 環境下的注意要點

```
apfols@apfols-PC MINGW32 ~/wk/test (master)
$ git push
Username for 'https://github.com': |
```



# 練習1-8: 從遠端 (remote repository) 抓取資料

- git pull

```
apfolsdeMacBook-Pro:test apfols$ git pull
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 2), reused 5 (delta 1), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/apfols/test
   324bc11..ecc6688  master     -> origin/master
Updating 324bc11..ecc6688
Fast-forward
 test3 | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

# 為什麼會有這些詭異的訊息

- git push <remote> <branch>
- git remote

```
apfolsdeMacBook-Pro:test apfols$ git remote
origin
```

- git remote -v

```
apfolsdeMacBook-Pro:test apfols$ git remote -v
origin  https://github.com/apfols/test.git (fetch)
origin  https://github.com/apfols/test.git (push)
```

- git push origin master

# 工具2: 標籤

- 列出所有標籤: git tag
- 標籤加說明: git tag -a <標籤名> -m <說明>
- 輕量級標籤: git tag <標籤名>
- 標籤之前的的提交: git tag <標籤名> <提交編號>
- 刪除標籤: git tag -d <標籤名>
- 提交標籤至遠端:
  - 特定標籤: git push origin <標籤名>
  - 全部標籤: git push origin --tags
- 刪除遠端標籤: git push --delete origin <標籤名>

# 練習T2-1: 新增標籤 t2 至遠方

- git tag
- git tag t2 <提交編號>
- git log --tags --pretty="%h %d %s" --decorate=full
- git tag

# 練習T2-2: 刪除遠端的標籤 t2

- git push --delete origin t2
- git log --tags --pretty="%h %d %s" --decorate=full
- git tag

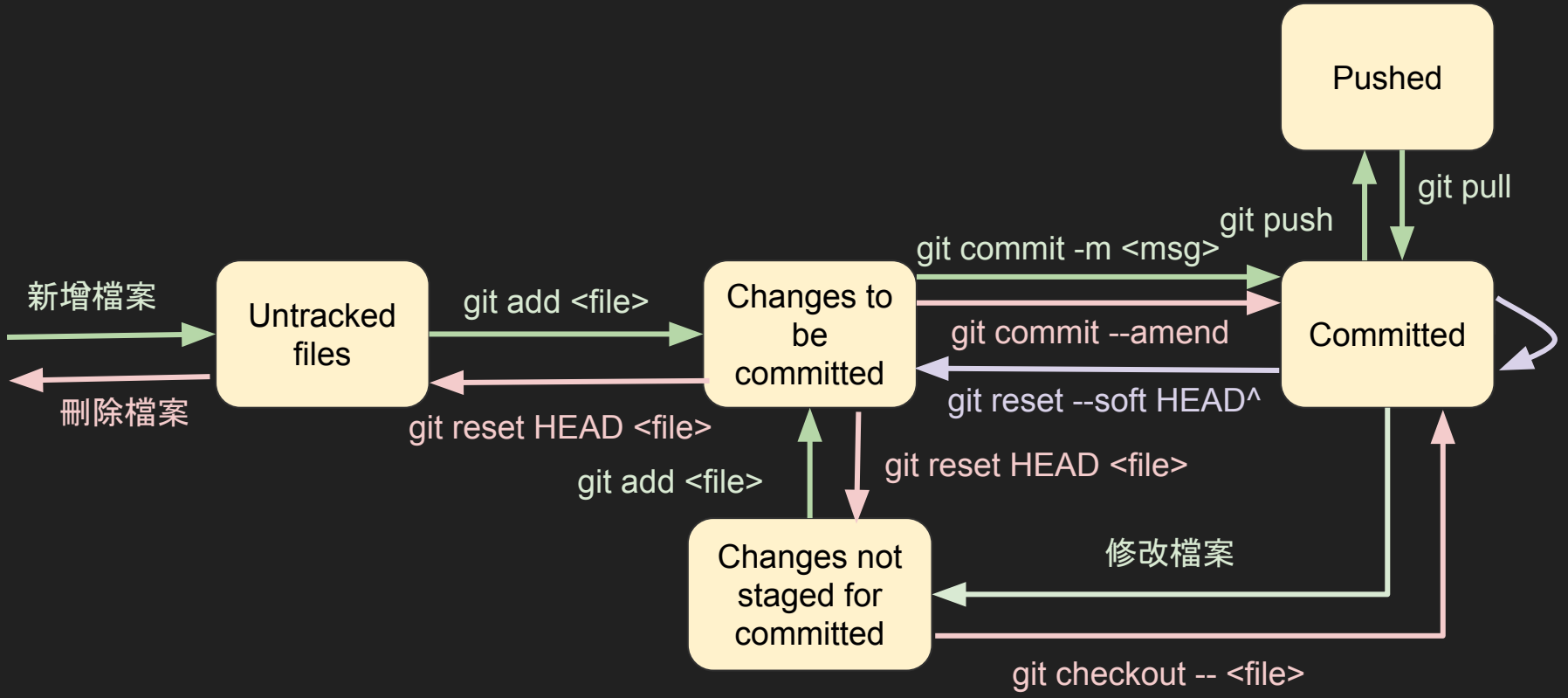
# 工具3: 不同提交間的差異

- git diff <基礎提交編號> <比較提交編號>
- Github compare view
  - https://github.com/blog/612-introducing-github-compare-view
  - https://github.com/apfols/test/compare (在結尾 + “/compare”)

- 練習

# 2. 還原指令

# 狀態轉換圖 again !!


Pushed
git pull
git push
git commit -m <msg>
新增檔案
Untracked files
git add <file>
Changes to be committed
git commit --amend
Committed
刪除檔案
git reset HEAD <file>
git reset --soft HEAD^
git add <file>
git reset HEAD <file>
Changes not staged for committed
修改檔案
git checkout -- <file>

# 工具4: 看不見可是它依舊存在的暫存區

- 情境
  - 永遠處理不完的 “最重要” 問題單
- 理論
  - Queue or stack ?
- 指令
  - 將修改放入暫存區 : git stash
  - 將修改從暫存區中取出 : git stash pop
  - 列出目前暫存區中的修改 : git stash list

# 練習T4-1: 練習使用 stash

- 新增一個檔案 practiceT2.txt, 內容: “stash 練習”
- README 加上一行: “練習T2”
- git status
- git stash
- git status
- git stash list
- README 加上一行 “趕在 T2 之前”
- 提交此次修改
- git stash pop
- git status

```
apfolsdeMacBook-Pro:test apfols$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

        both modified:   README

no changes added to commit (use "git add" and/or "git commit -a")
```

```
<<<<<<< Updated upstream
趕在 T2 之前
=======
練習 T2
>>>>>>> Stashed changes
```

# 練習T4-2: 衝突處理

- 將衝突處理掉
- git add README
- git status

# 練習2-1: 在 push 前發現錯誤 (修改記錄或檔案內容)

- 新增一個檔案 practice2-1.txt 並提交至 local repository, 修改記錄亂打
- git log --oneline
- 在 practice2-1.txt 中加入今天日期
- git add practice2-1.txt
- git commit --amend
- 在編輯畫面中, 把修改記錄改成 “在 push 前發現錯誤” 並儲存
- git log --oneline

# 練習2-2: 將集結區的檔案放回工作區

- 新增檔案 practice2-2.txt, 並提交至 remote repository (log: practice2-2)
- 修改 practice2-2.txt 的內容, 加入今天日期
- git add practice2-2.txt
- git reset HEAD <file> (practice2-2.txt)
- git status

# 練習2-3: 將已修改的檔案變為未修改

- git checkout -- <file> (practice2-2.txt)
- git status

# 其實 “git status” 早就把這些指令告訴你了

```
apfolsdeMacBook-Pro:test apfols$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   test2

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   test3

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

        test4
```

# 回復到不同的版本 (Local Repository)

- 新序號檔案保留：git reset --soft <提交序號>
- 檔案不保留：git reset --hard <提交序號>
- 關鍵字: HEAD, HEAD^

# 練習2-4: 取消已經進行過的修改

- 修改 practice2-1.txt
- git add practice2-1.txt
- 修改 practice2-2.txt
- 新增 practice2-4.txt
- git reset --hard HEAD
- git status
- git log --oneline

# 練習2-5: 復原至上次的修改

- git reset --soft HEAD^
- git status
- git log --oneline

- 試試看
  - 回復到不同的提交編號
  - 之前提交的東西都不見了怎麼辦？

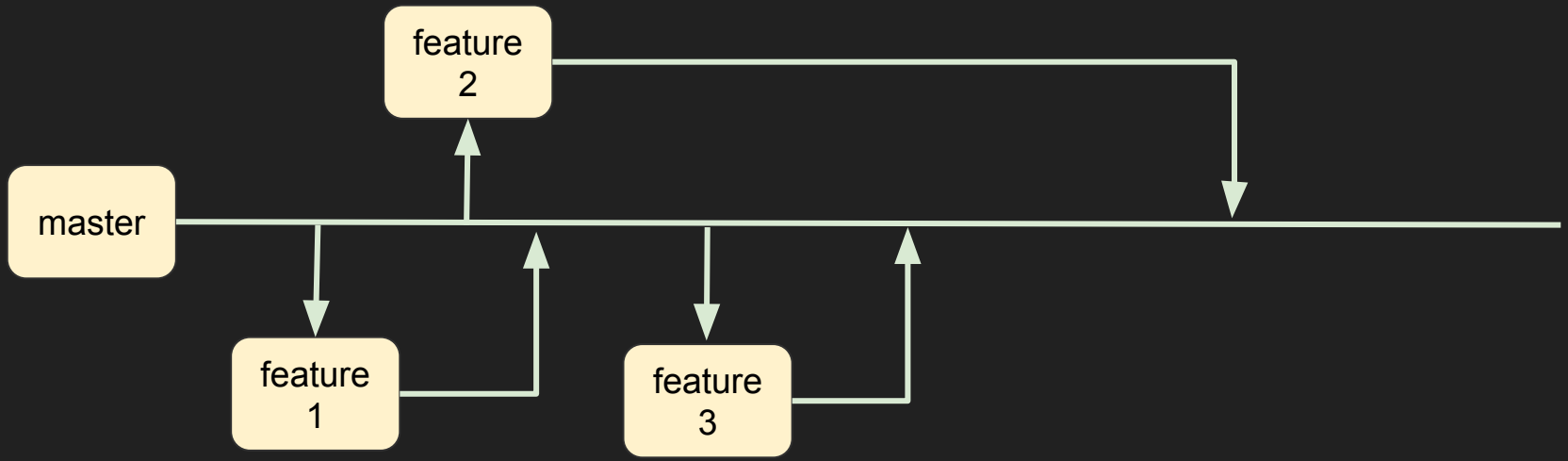
# 回復到不同的版本 (Global Repository)

- 建立回復提交: git revert
- 練習
  - git revert HEAD~2
  - git revert HEAD~3..HEAD~1
- ~~可以搭配 git reset --hard <commit> 使用嗎? (不建議)~~
  - ~~git push -f~~
  - ~~有可能會影響到整個提交的歷史~~

# 3. 分支指令

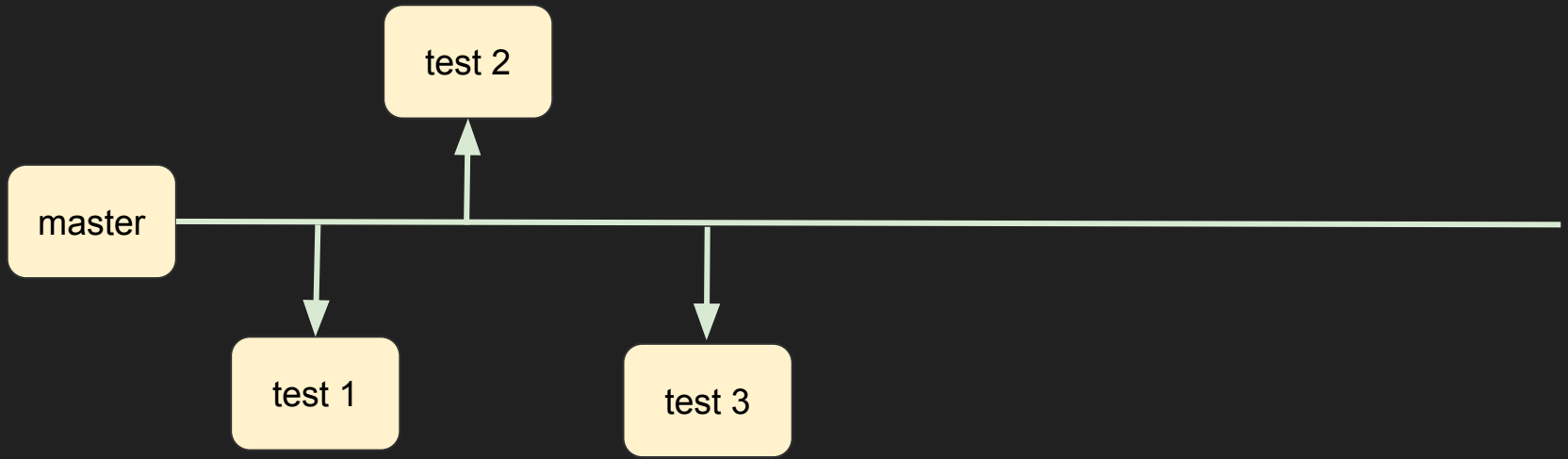
# 使用分支的情境 - master / feature branch

- 通常會使用分支代表該功能開發需要較久的時間, 因此合併時的複雜度度高
- staging branch 變型

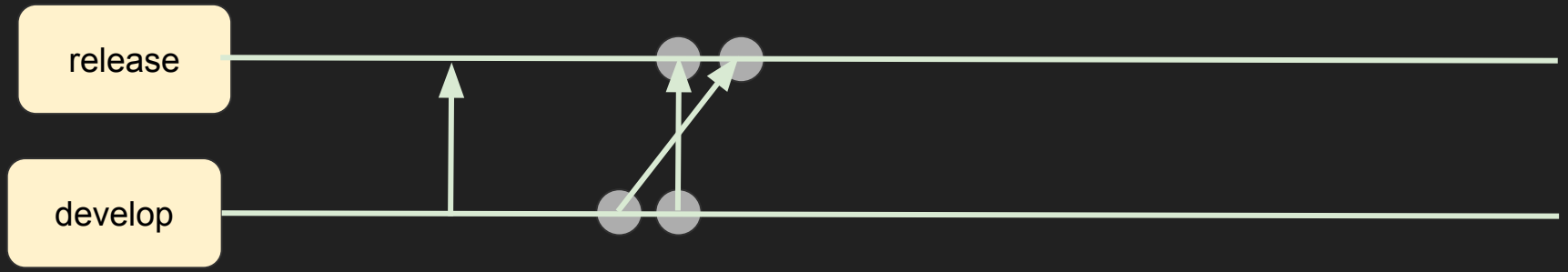


# 使用分支的情境 - 測試

- 避免建立太多不必要的分支 (用完砍掉)
- 命名要有意義

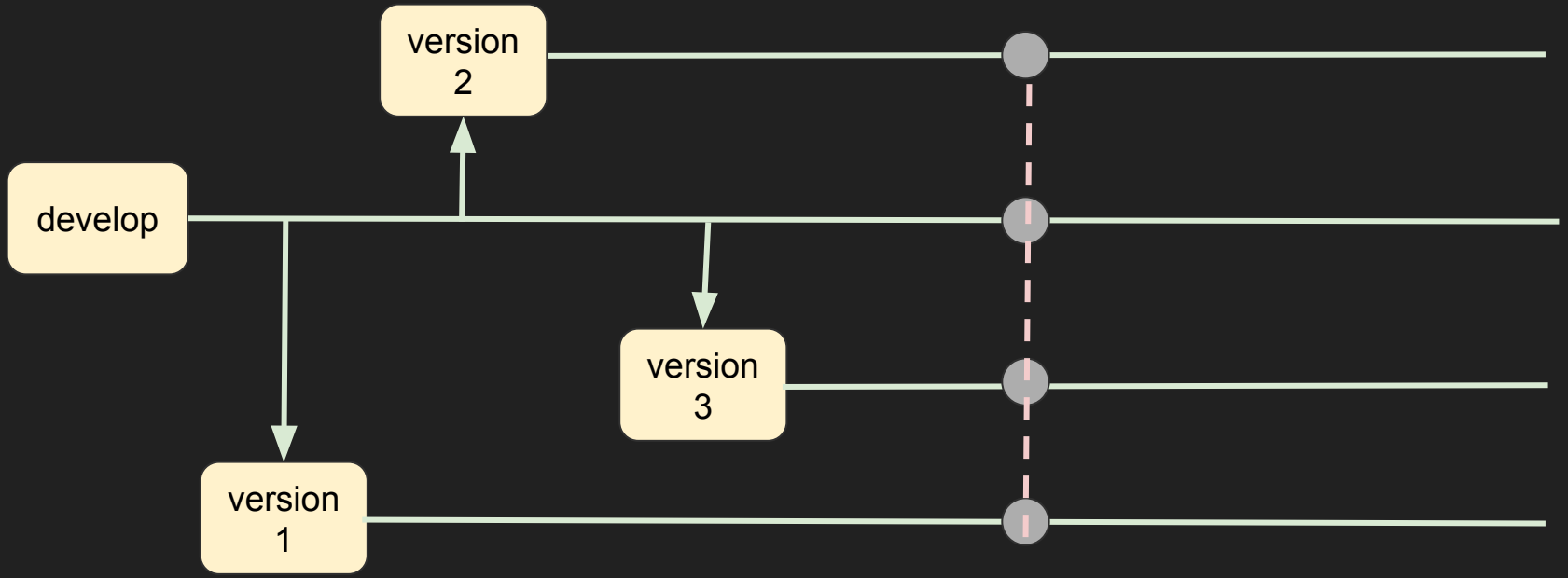


# 使用分支的情境 - 開發與發佈



# 使用分支的情境 - 大版號修改

- 苦命的工程師，只能希望公司及早宣布不維護舊版本



# 分支基本指令

- 查詢本地端分支: git branch
  - 目前分支
  - 所有分支
- 查詢遠端分支: git ls-remote origin
- 建立新分支: git branch <分支名稱>
- 刪除本地端分支: git branch -d <分支名稱>
- 刪除遠端分支: git push origin --delete <分支名稱>
- 切換分支: git checkout <分支名稱>

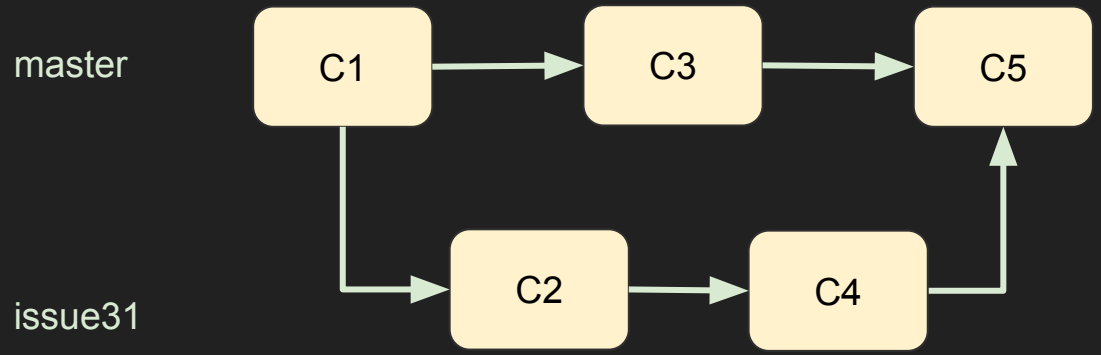
# 練習3-1: 建立分支

- 在 master branch, 建立一個檔案 practice3.txt 相關的提交 “base practice3.txt” 至遠端
- git branch
- git branch issue31
- git checkout issue31
- 建立一個檔案 practice3.txt 相關的提交 “issue31 first commit” 至遠端
- git log --oneline

- 動動腦 : 如果我要建立一個之前在之前提交的分支怎麼辦?

# 分支合併

- 將指定的分支合併至目前分支: git merge <指定的名支名稱>



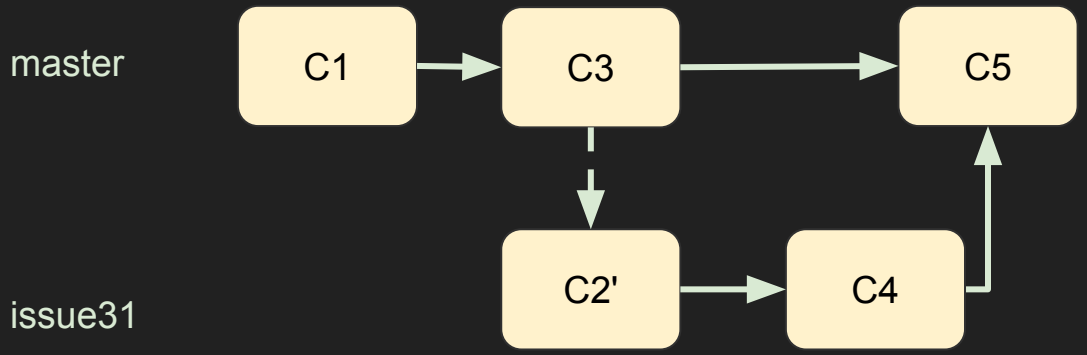
# 練習3-2: 合併分支

- 在新的分支 “hotfix” 中，建立一個新的提交 “fix emergency” 在檔案 practice3.txt 至遠端
- 切換至 master branch
- git merge hotfix
- git push
- git branch -d hotfix
- git push origin --delete hotfix
- git branch

# 練習3-3: 衝突處理

- 在新的分支 “issue31” 中，建立一個新的提交 “finish issue31” 在檔案 practice3.txt 至遠端
- 切換至 master branch
- git merge issue31
- …
- git commit -am “resolve issue31 conflicts”
- git push
- git branch -d issue31
- git push origin --delete issue31
- git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short

# 另一種解決衝突的方法: git rebase <分支名稱>



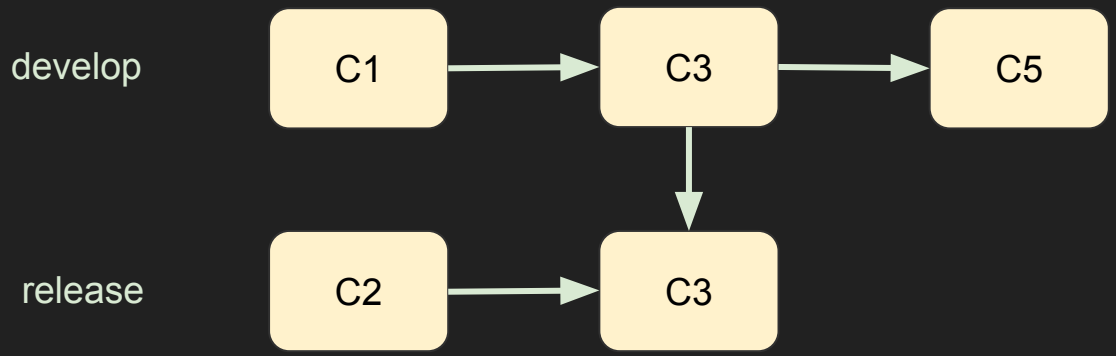
# 練習3-4: 使用 rebase 再解一次衝突

- 重覆練習 3-1, 3-2 的內容, 只是將檔案改成 practice3-4.txt
- 切換至 issue31 branch
- git rebase master
- git add .
- git rebase --continue
- …
- 切換至 master branch
- git merge issue31
- git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short
- 試試看: 在 issue31 提交會發生什麼事?

# Merge 與 Rebase 的比較

- git merge: 反映了實際上提交的順序
- git rebase: 反映了實際上在 git 存在的順序
- 已經提交的改變不應該 rebase

# 挑選需要的提交 - git cherry-pick <提交編號>



# 練習3-5: 挑選需要的功能

- 開新分支 develop
- 轉換至分支 develop
- 建立本地端提交 “feature1”
- 建立本地端提交 “feature2”
- git log --oneline 並複製 “feature2” 的提交編號
- 轉換至分支 master
- git cherry-pick <feature 2 的提交編號>
- git log --oneline

# 指令小撇步

- vim ~/.gitconfig

```
[alias]
    co = checkout
    ci = commit
    st = status
    br = branch
    rs = reset
    lg = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short
```

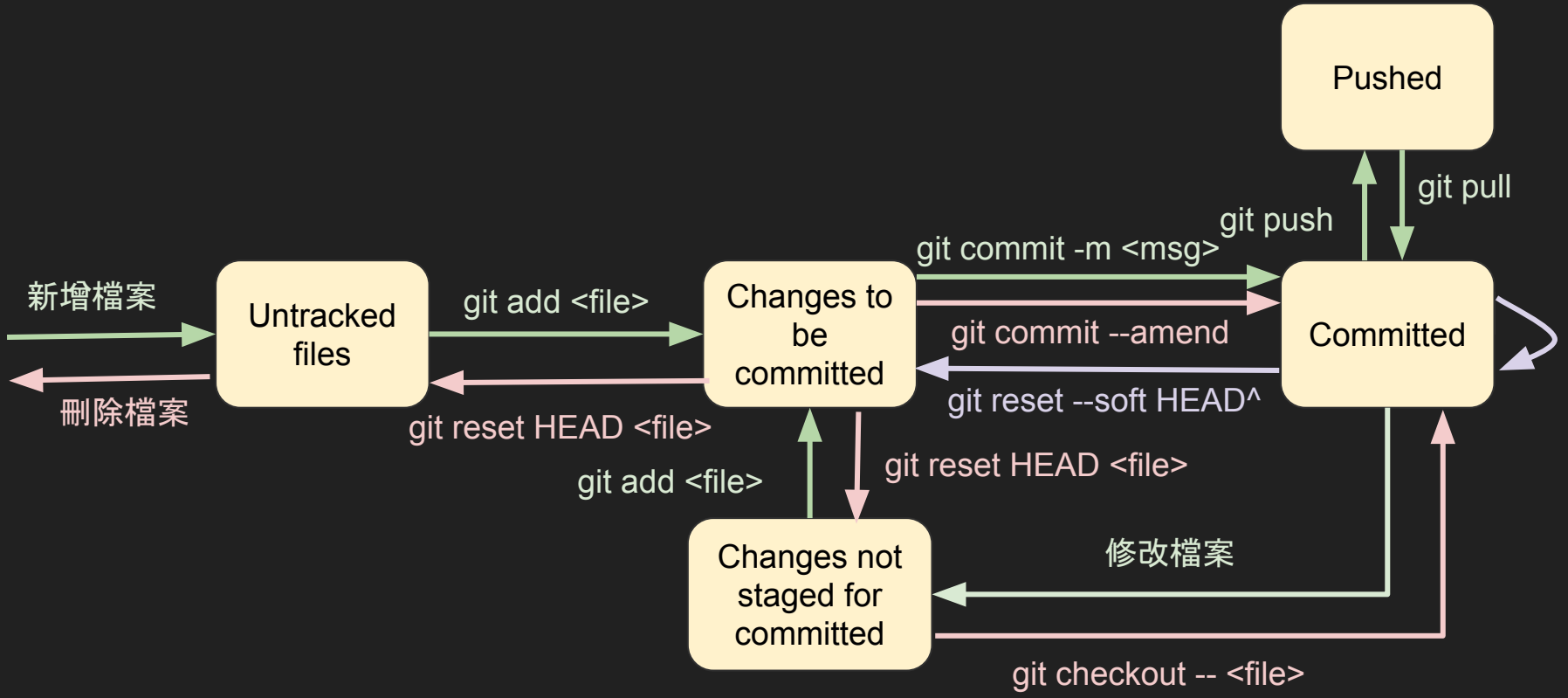
# 再試一試之前的指令

- git br
- git co
- ...

# 總結

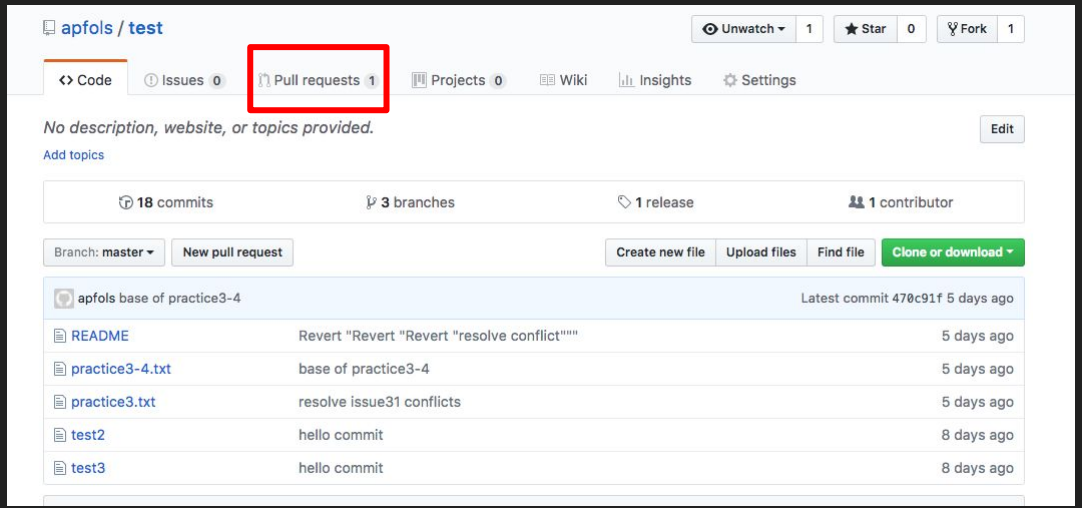
- 版本控制系統的重要性
- Git 的優點
- Git 工具指令 (status, log, tag, diff, stash)
- Git 基本指令 (正常流程)
- Git 還原指令 (修正錯誤)
- Git 分支指令

# Git 流程指令


Pushed
git pull
git push
git commit -m <msg>
新增檔案
Untracked files
git add <file>
Changes to be committed
git commit --amend
Committed
刪除檔案
git reset HEAD <file>
git reset --soft HEAD^
git add <file>
git reset HEAD <file>
Changes not staged for committed
修改檔案
git checkout -- <file>

# One more thing - 在 Github 上的多人合作

- 直接設定共同合作者
  - Settings -> Collaborators
- Pull requests



# Q & A

# 參考資料

- Git 的誕生
- Git 教學
- Git 版本控制系統
- https://git-scm.com/book/en/v2