

# Understand the basic of Git & GitHub



國立台北科技大學電子工程系  
黃士嘉老師

# What is Git

# • What is Git ?



The screenshot shows a Google search interface with the query "git". The search results indicate approximately 215,000,000 results found in 0.38 seconds. The top result is from Wikipedia, titled "git - 維基百科，自由的百科全書 - Wikipedia" with the URL <https://zh.wikipedia.org/zh-tw/Git>. To the right of the text is a small thumbnail image of the Git website's documentation page.

**git** (`/ɡɪt/`，音訊〈說明·資訊〉) 是一個分散式版本控制軟體，最初由林納斯·托瓦茲 (Linus Torvalds) 創作，於2005年以GPL釋出。最初目的是為更好地管理 Linux 內核開發而設計。

[git - 維基百科，自由的百科全書 - Wikipedia](https://zh.wikipedia.org/zh-tw/Git)  
<https://zh.wikipedia.org/zh-tw/Git>

進一步瞭解這項結果    意見回饋

- What is Git ?
  - A version control software



- What is GitHub?

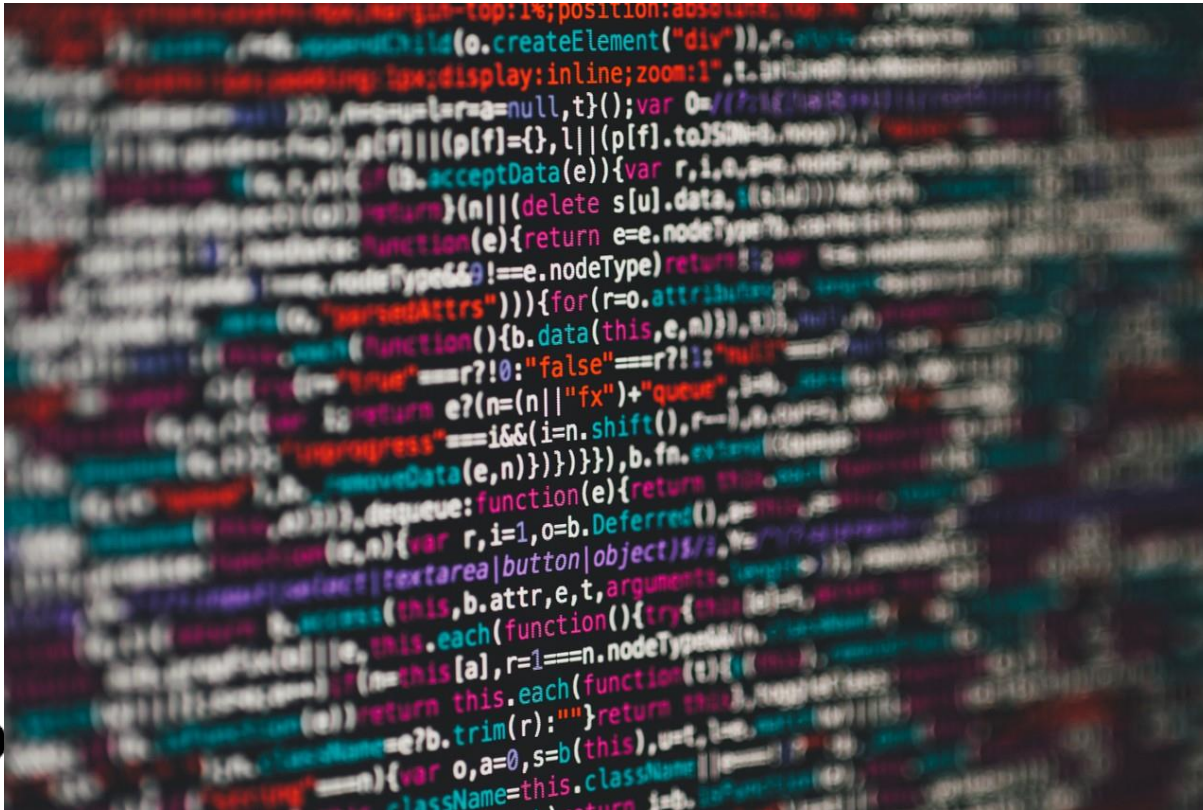
GitHub [\[編輯\]](#)

---

維基百科，自由的百科全書

**GitHub**是一個透過Git進行版本控制的軟體原始碼代管服務，由GitHub公司（曾稱Logical Awesome）的開發者Chris Wanstrath、PJ Hyett和Tom Preston-Werner使用Ruby on Rails編寫而成。



- Version control & Backup
  - 雞蛋不放在同一個籃子裡



Final Project



名稱	大小
期末報告 - 第一版.txt	3 KB
期末報告 - 第二版.txt	6 KB
期末報告 - 第三版.txt	9 KB
期末報告 - 第四版.txt	12 KB

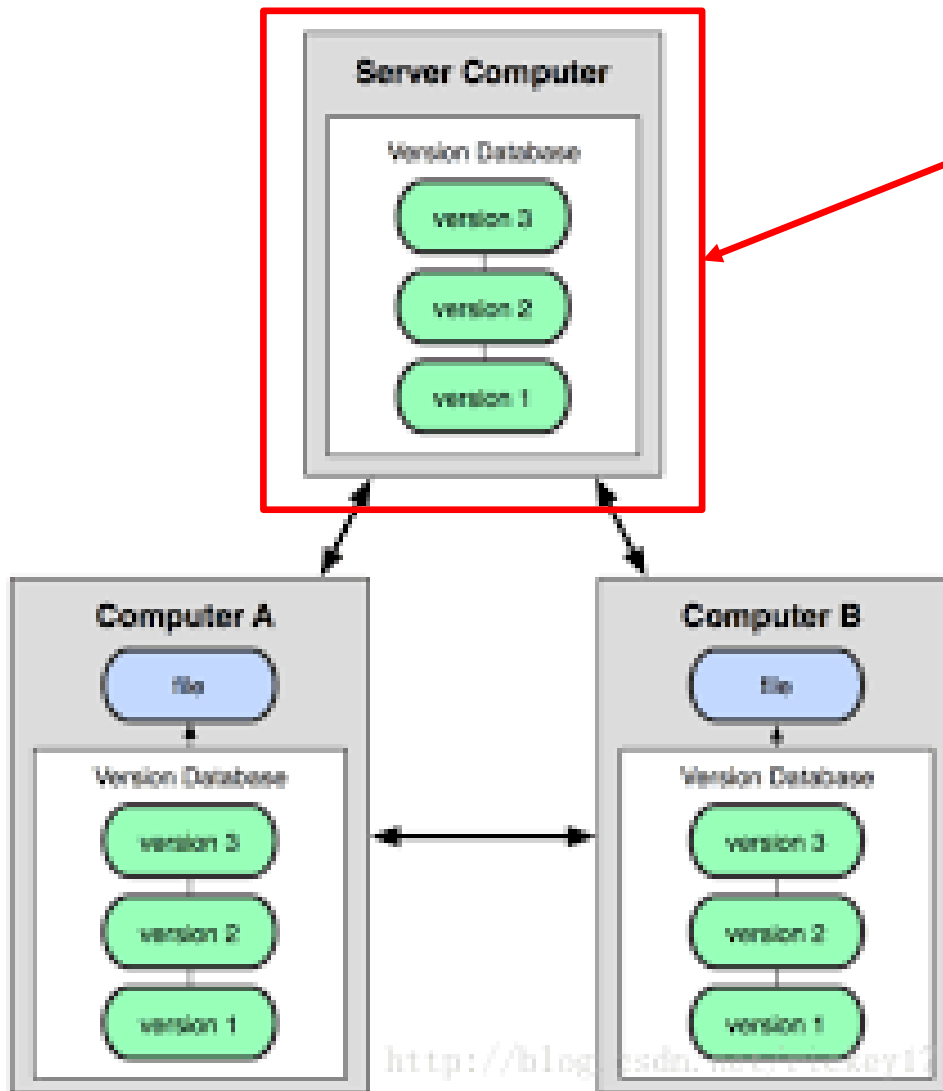
名稱	大小
 .git	
 期末報告.txt	12 KB

[illegible]

- Why version control?



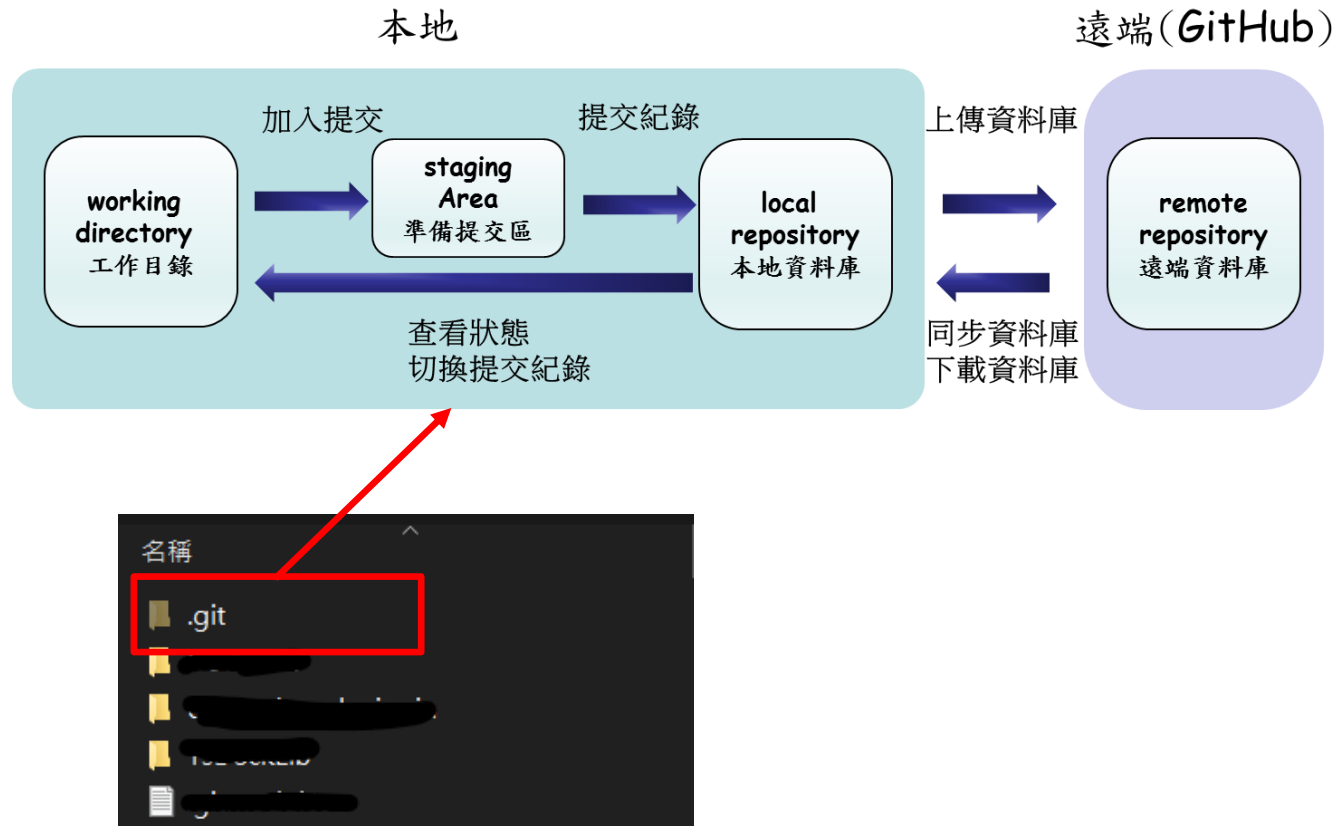
- GitHub(線上軟體原始碼代管服務平台)



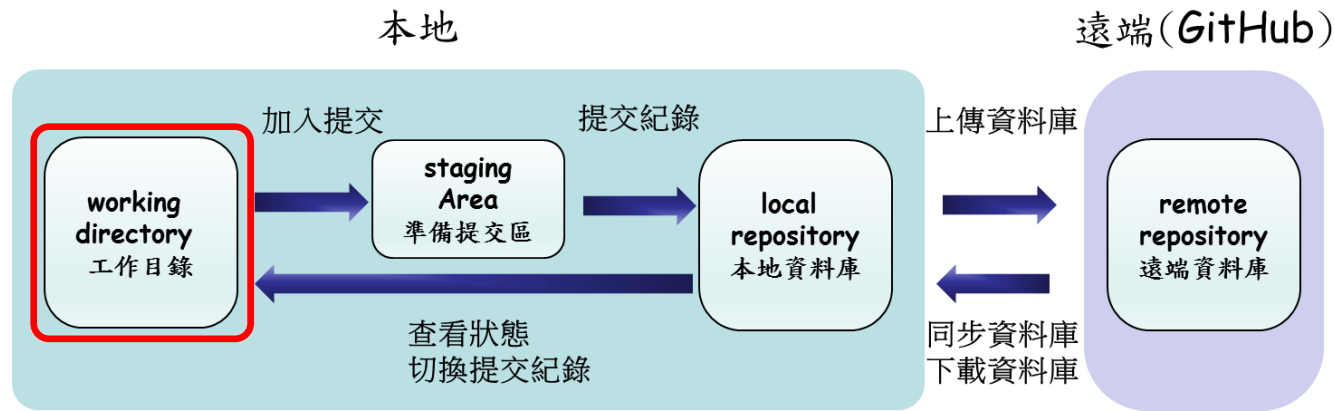


# Workflow of Git

- The work flow of Git



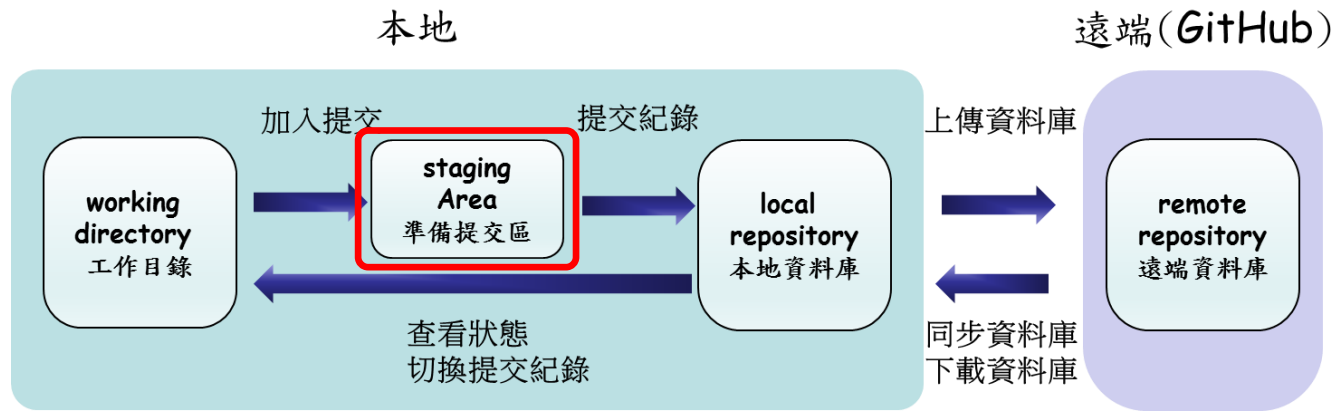
- The work flow of Git



- Working directory

工作目錄主要是存放要被版本控制的檔案資料夾。我們可以選擇一個一般資料夾並在資料夾內建立 **git** 的資料庫(**Repository**)，就能把該資料夾變成 **git** 的工作目錄

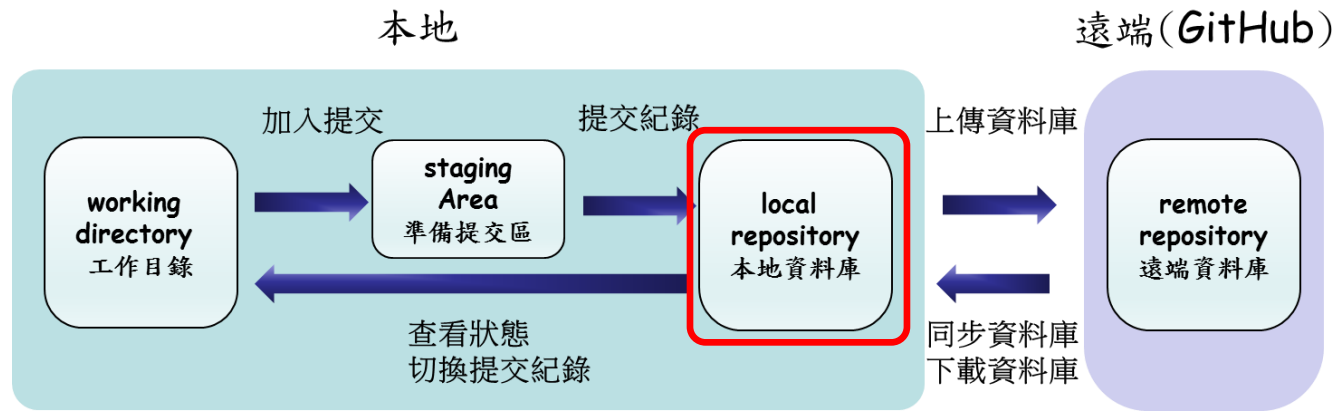
# • The work flow of Git



## - Staging area

準備提交區用於記錄即將要被提交的資料。當在工作目錄下檔案的有進行變更，且我們希望能提交這些變更，我們會將這些資料存入準備提交區之中，這狀態下只有標記那些資料要被提交，但還並未實際的做提交紀錄。

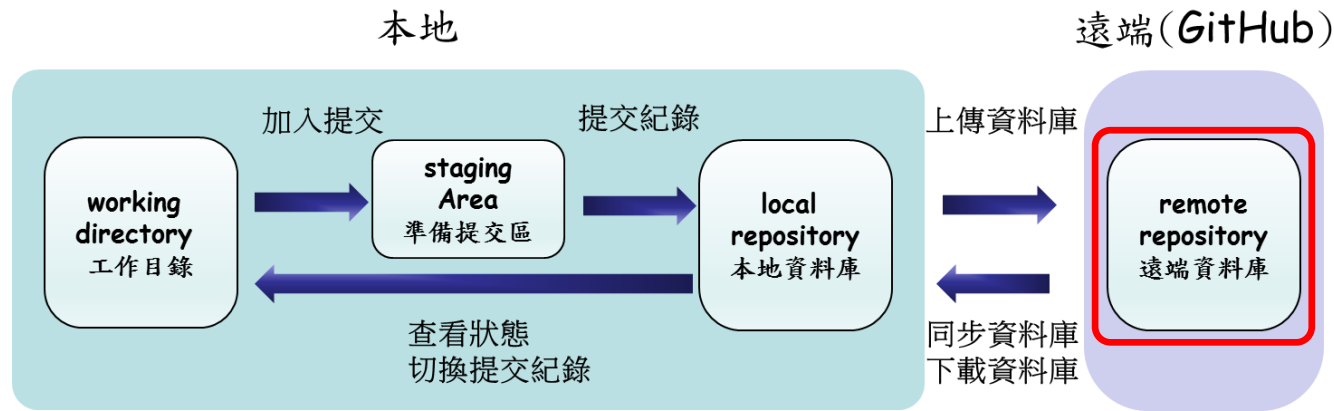
# • The work flow of Git



## - Local repository

即為自己電腦端上的資料庫。當確定好所有要提交的資料都加入到準備提交區之後，可以將準備提交區的資料做提交紀錄，提交後的資料會被記錄成一個提交紀錄保存於資料庫中。

# • The work flow of Git

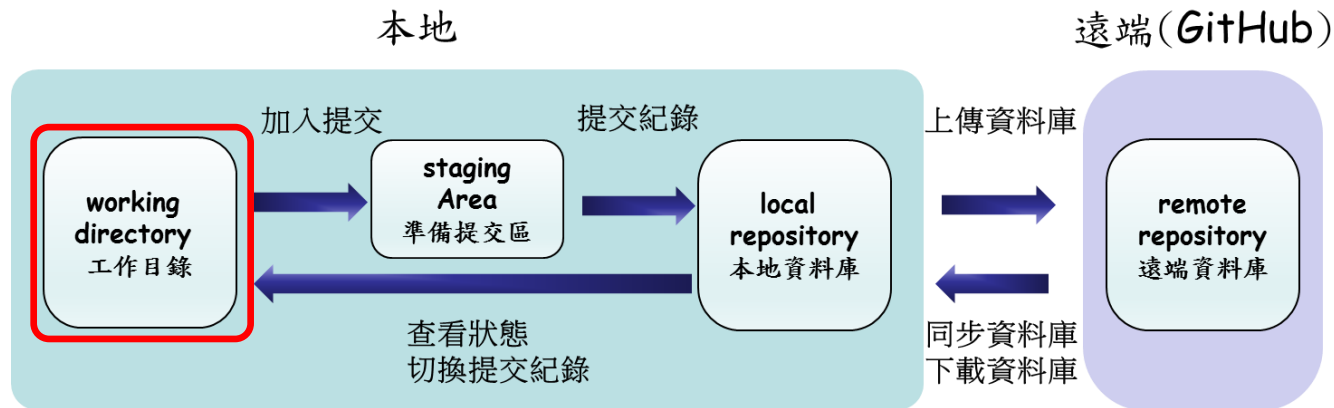


## - Remote repository

即為遠端伺服器上的資料庫。當本地資料備齊後，我們可以透過上傳將資料保存到遠端資料庫，也可以反過來將遠端資料庫的紀錄同步下來。

# Basic command of Git

- Git init
  - 將目前資料夾初始化為working directory



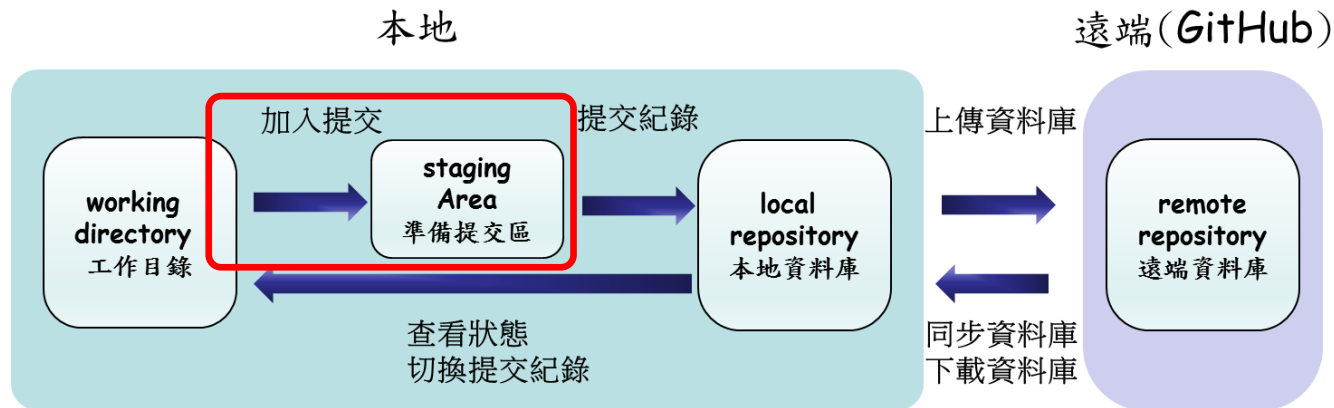
```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0
$ git init
Initialized empty Git repository in C:/Users/Jdway/Desktop/Lab0/.git/

Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0 (master)
$ |
```



## • Git add

- 將資料的變更加入到準備提交區
- 語法：`git add` “添加到準備區的檔案名稱”

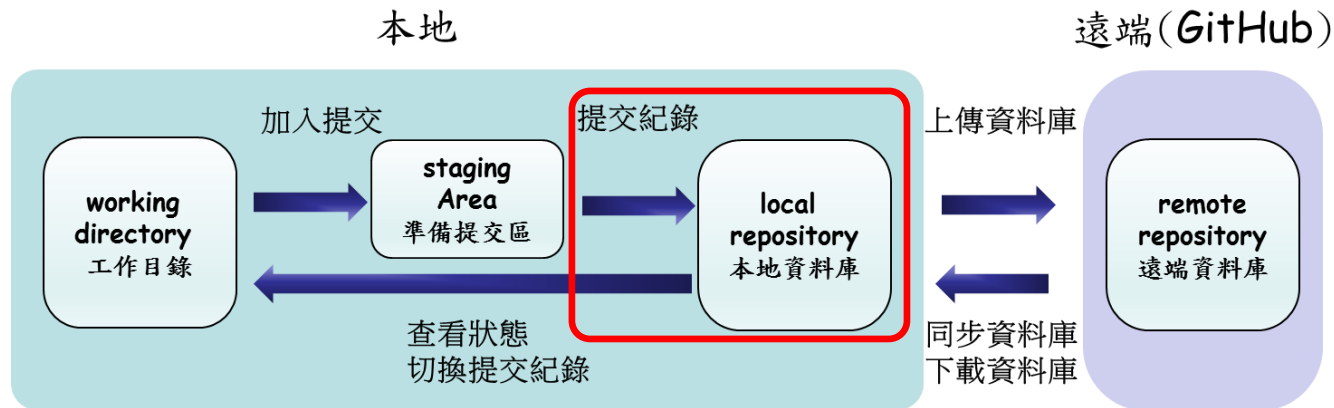


```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0 (master)
$ git add .
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0 (master)
$ |
```

• 代表當前目錄下的所有資料變更

## • Git commit

- 準備提交區的變更寫入到本地資料庫中
- 語法：`git commit -m "本次提交的備註"`

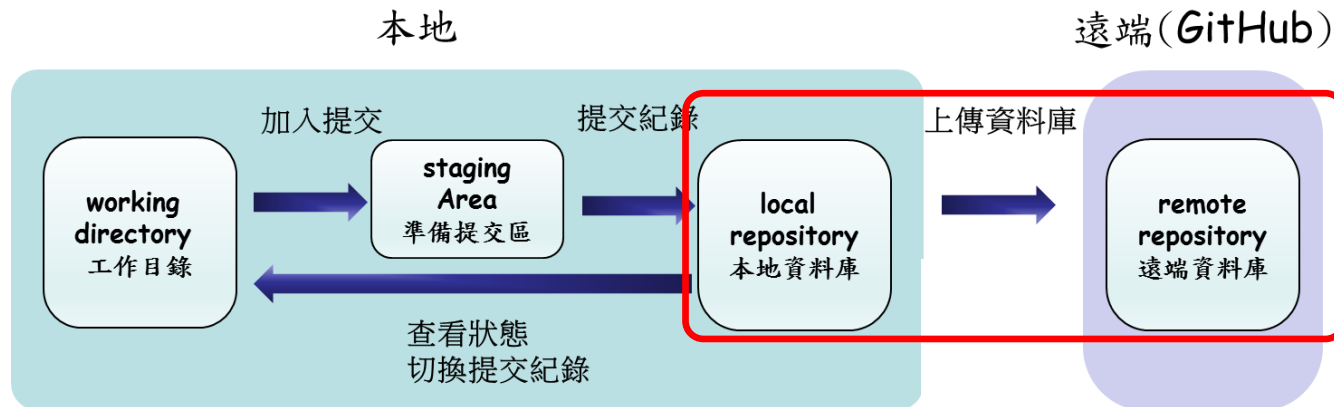


```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0 (master)
$ git commit -m "Commit informations"
[master (root-commit) 4da7ccf] Commit informations
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 asdf

Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab0 (master)
$ |
```

## • Git push

- 將本地資料庫的資料同步到遠端資料庫
- 語法：`git push` 或 `git push origin 「分支名稱」`

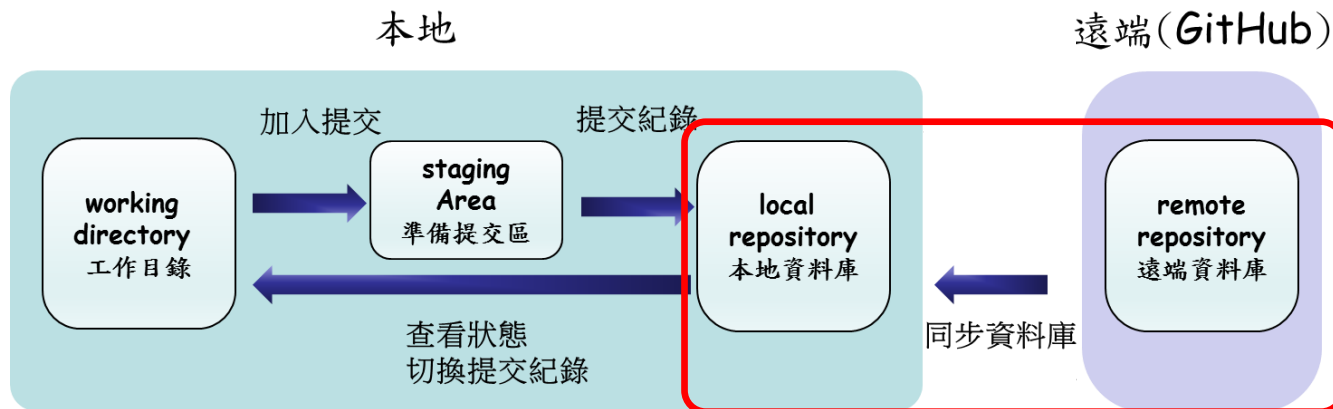


```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab1 (master)
$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 274 bytes | 274.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/davidjaw/Ch0-Lab-1.git
  29c6bed..497653d master -> master

Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab1 (master)
$ |
```

## • Git pull

- 將遠端資料庫的資料同步到本地資料庫
- 語法：`git pull` 或 `git pull origin 「分支名稱」`

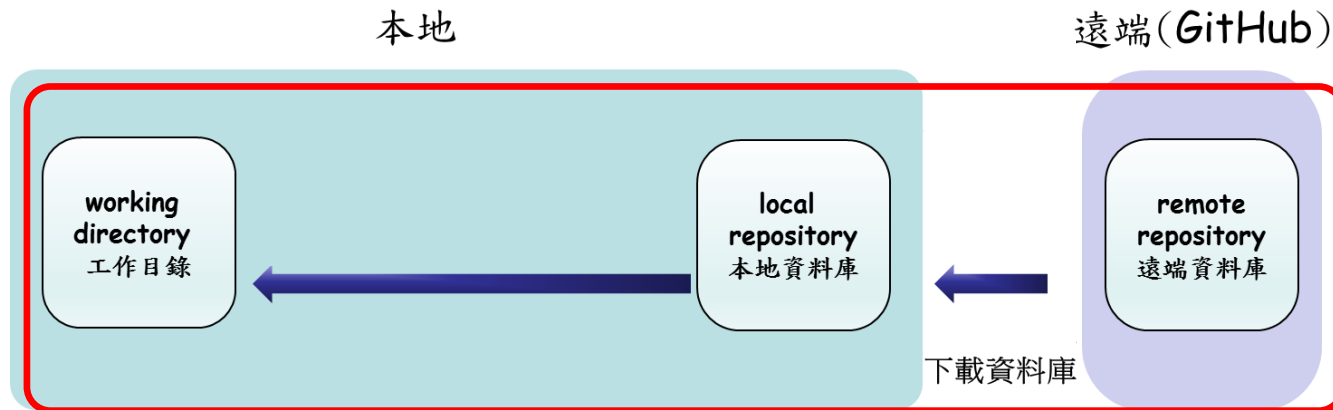


```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab1 (master)
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/davidjaw/Ch0-Lab-1
  497653d..13a4109  master    -> origin/master
Updating 497653d..13a4109
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)

Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop/Lab1 (master)
$ |
```

## • Git clone

- 將遠端資料庫的資料同步到本地資料庫
- 語法：`git clone` 「遠端資料庫網址」



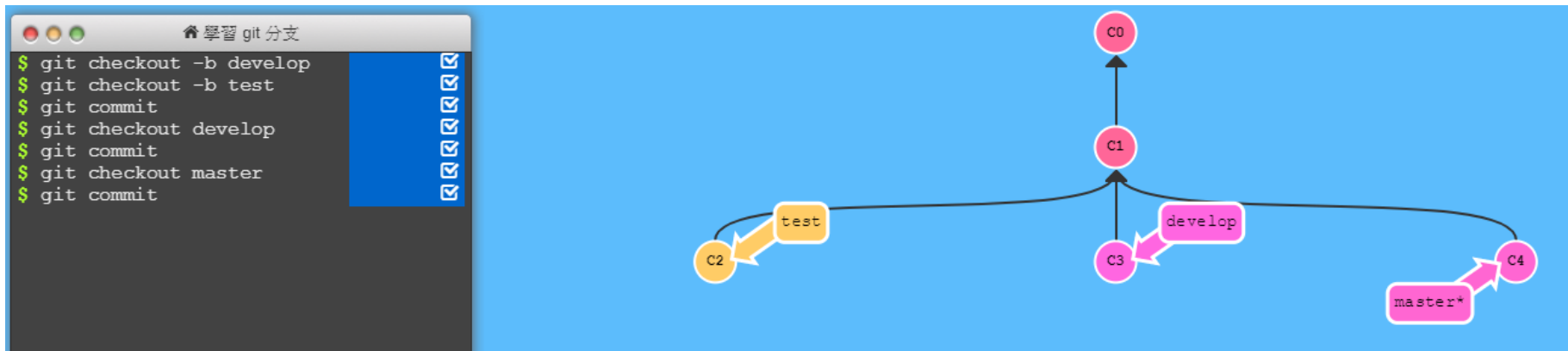
```
Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop
$ git clone https://github.com/davidjaw/Ch0-Lab-1.git
Cloning into 'Ch0-Lab-1'...
remote: Counting objects: 34, done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 34 (delta 7), reused 30 (delta 6), pack-reused 0
Unpacking objects: 100% (34/34), done.

Jdway@DESKTOP-GBM49C1 MINGW64 ~/Desktop
$ |
```

# Advance

- Branch (分支)
  - 在大型專案中，產品會根據不同功能的發展方向做開發，例如：
    - 系統既有的程式加速
    - 系統未有的功能開發
    - 系統當前的程式除錯
  - 因此，多人開發時將會切分出不同的分支以及主分支來進行，如此不會因為其中誰更改了某個功能而導致同步後無法使用
- 視覺化學習網站：<http://learngitbranching.js.org/>
  - 此網站僅提供local指令練習
  - 有許多練習題可以線上做
  - Homework將會在這個網站上完成

- **Git checkout** 「分支名稱」
  - 切換分支：將本地的工作環境切換到該分支上
- **Git checkout -b** 「分支名稱」
  - 創建分支：將本地的工作環境儲存到目前分支上
- **Git branch**
  - 查看本地端分支：將本地的分支顯示於終端機

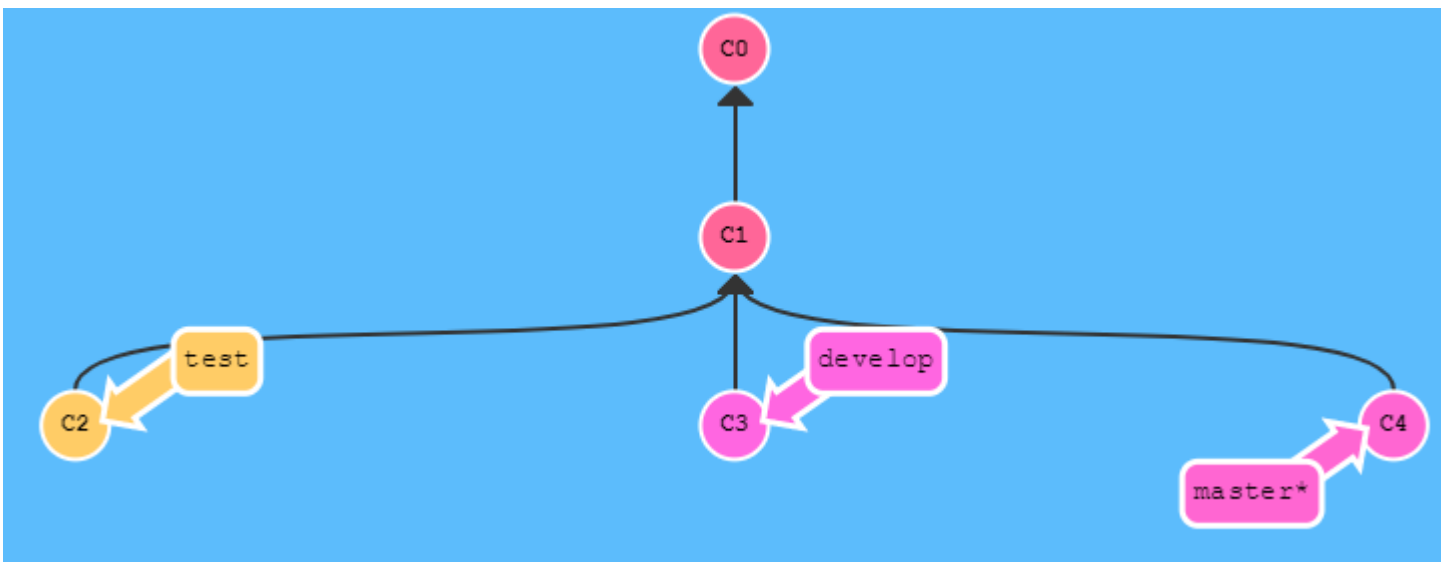




- **Git merge** 「分支名稱」

- 將其他分支所做的改動整合到目前所在的分支上

- 如下圖，目前分支為**master**，我們要將**develop**所做的改動整合到**master**分支上，其指令為「**git merge develop**」



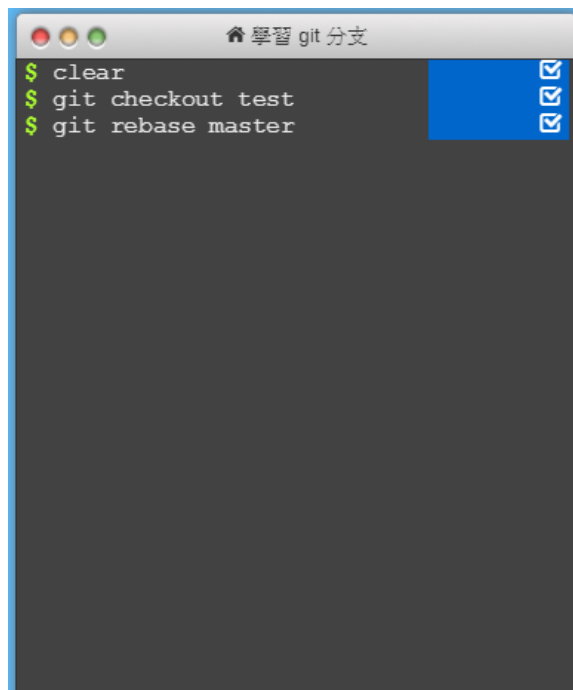




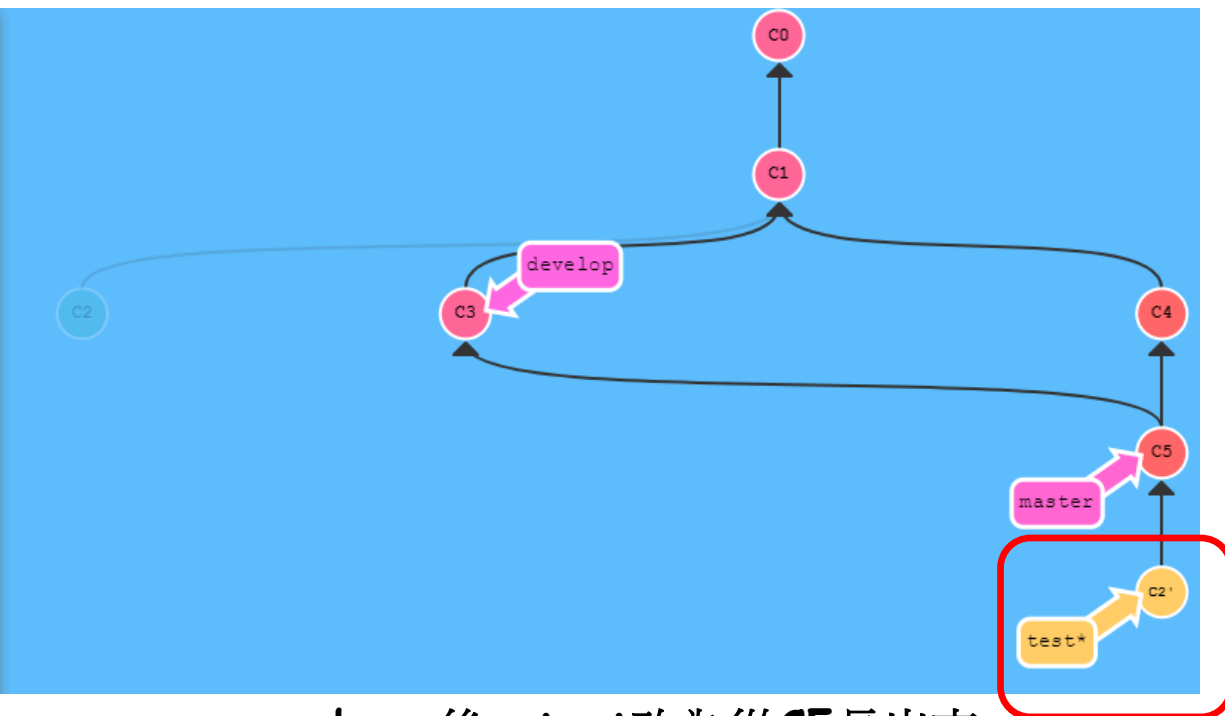
- Git rebase 「分支名稱」

- 將目前基於A點的更動換成基於B點

- 例如：你開發某個功能但尚未完成時，系統進行了改版，你必須要基於改版後的系統繼續進行開發才能確保最後上線時不會有問題



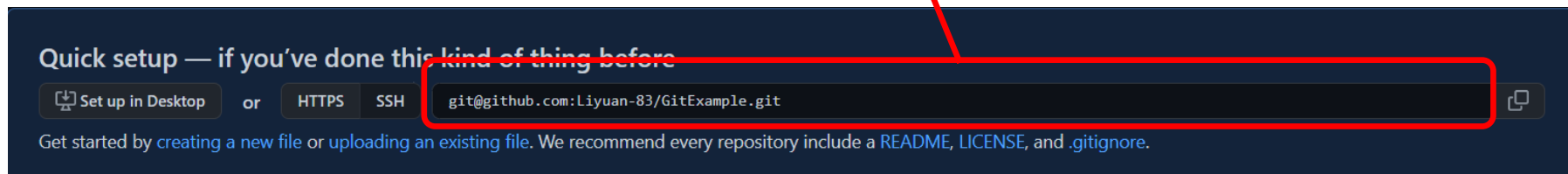
```
$ clear
$ git checkout test
$ git rebase master
```



rebase後，test改為從C5長出來

# Common Git Command Flow

- 建立新專案
  1. `$ git init`
  2. `$ git add .`
  3. `$ git commit -m "First commit(依照修改內容輸入)"`
- 設定並同步至遠端資料庫(Github)
  1. `$ git remote add origin [Github網址]`
  2. `$ git push origin [分支名稱]`



- 日常更新程式
  1. `$ git pull origin [分支名稱]`
  2. `$ git add .`
  3. `$ git commit -m "修正XXXX Bug"`
  4. `$ git push origin [分支名稱]`

# Continuous Integration(CI) /Continuous Delivery(CD)

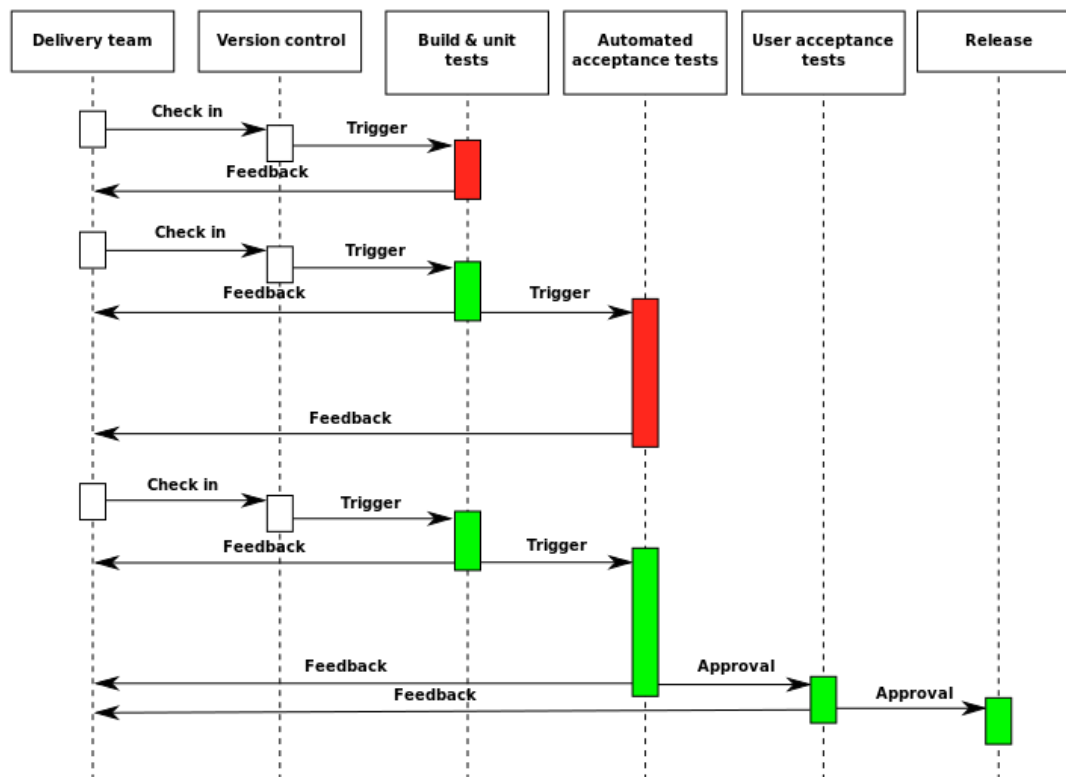


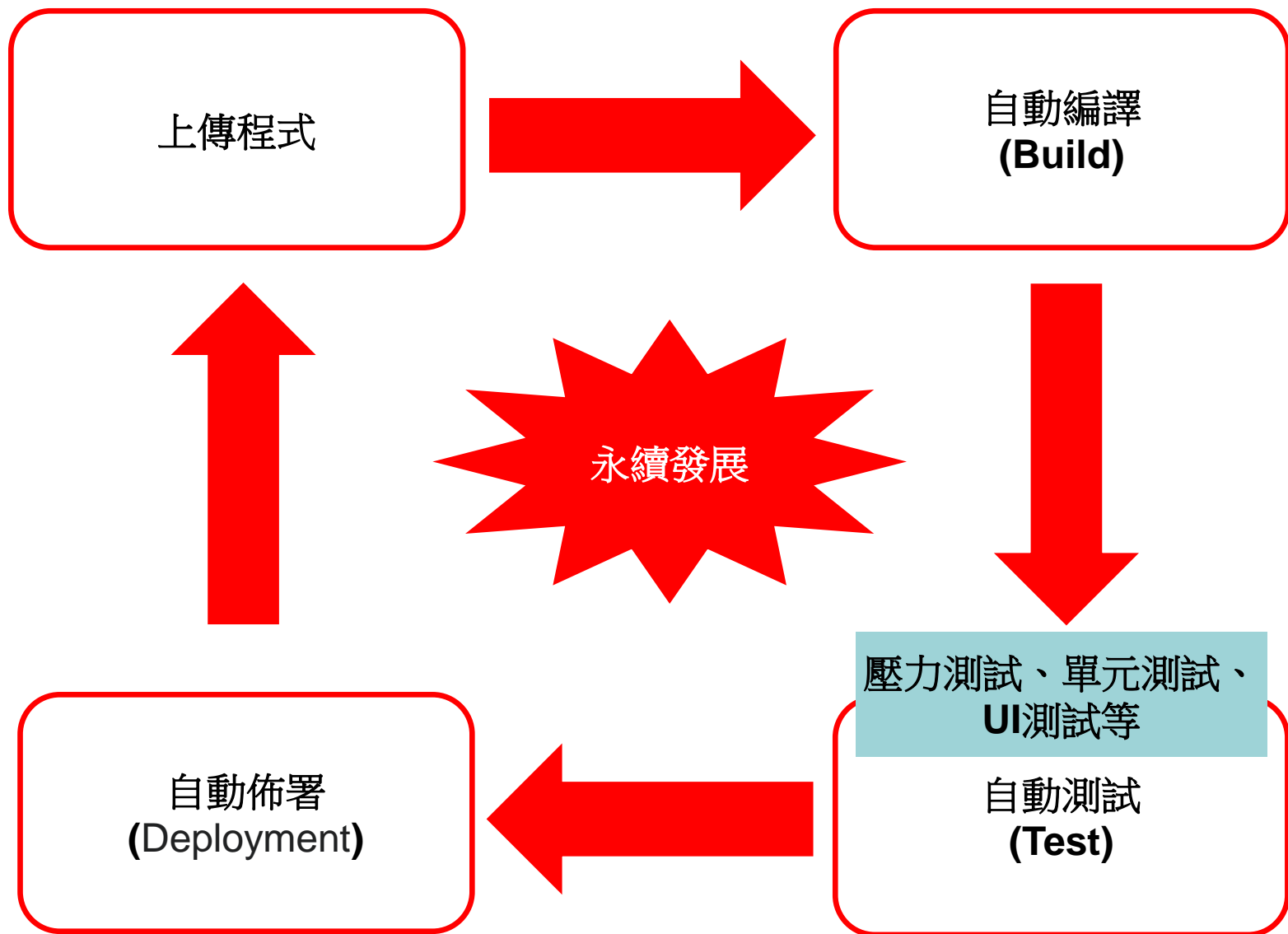
- 工程師的思維
  - 軟體打算用多久?
  - 要給多少人使用?
  - 新功能是否會影響到其他功能?
  - 新加的程式可不可以在其他環境正常編譯?
  - 如何確保每次備份的程式都是有用的?



CI/CD  
(持續整合 / 持續交付)

- 持續整合(Continuous Integration, CI)
  - 就是當開發人員完成一個階段性的程式碼後就經由自動化工具測試、驗證，協助偵測程式碼問題，並建置出即將部署的版本 (Build)。
- 持續交付(Continuous Delivery, CD)





- 常見的**CI/CD**工具

- GitHub

- 服務稱為**GitHub Action**，提供了多項控制**API**，能夠幫助開發者編排、掌握工作流程，在提交程式碼後自動編譯、測試並部署至伺服器，讓每位開發者都能受惠於平台本身自有的**CI/CD**功能。

- GitLab

- **GitLab**主要的服務是提供**git**版本控制系統，其**CI/CD Pipeline**功能簡單又實用，使用者只需要設定於專案根目錄下的「**.gitlab-ci.yml**」檔，便可以開始驅動各種**Pipeline**協助您完成自動化測試及部署。

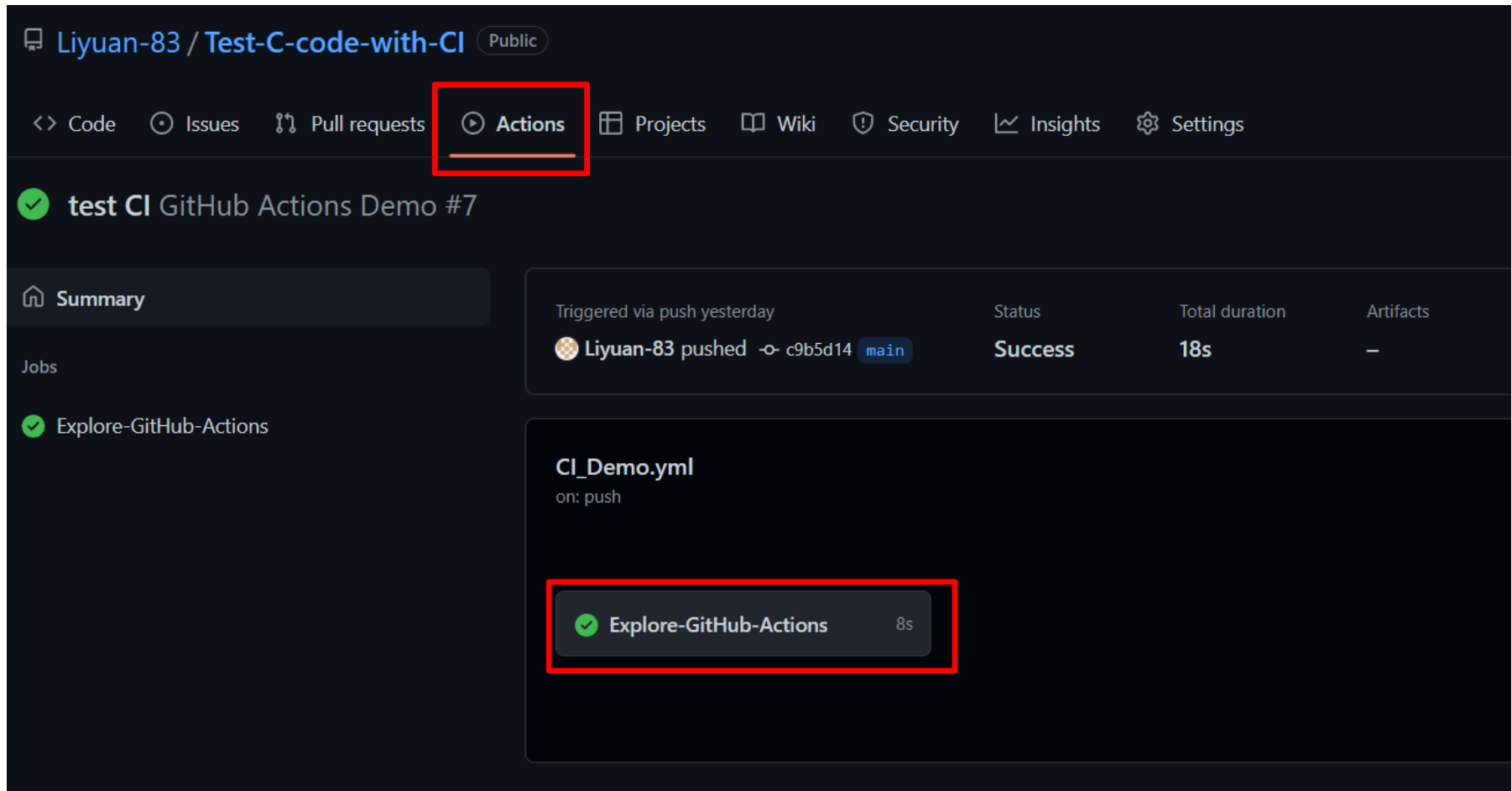
- Anthos

- **Google**推出新的 **Google Cloud** 開放平台，是一個 100% 以軟體為基底的解決方案。您可以在現有的硬體資源上快速啟動並執行，不用特別去翻新技術棧。

- Jenkins

- 2005發布的**CI/CD**工具，是一個開放原始碼(**Open Source**)的專案，有許多第三方外掛工具可使用，也可自行開發功能。

- GitHub Action



The screenshot shows the GitHub interface for a repository named "Liyuan-83 / Test-C-code-with-CI". The "Actions" tab is highlighted with a red box. Below the repository name, the workflow "test CI GitHub Actions Demo #7" is shown with a green checkmark. The "Summary" tab is selected, displaying a table of workflow runs. The first run, triggered by a push from "Liyuan-83" to the "main" branch, is marked as "Success" and took "18s". Below the table, the workflow file "CI\_Demo.yml" is shown, which is triggered on "push". A red box highlights the "Explore-GitHub-Actions" button, which is marked with a green checkmark and took "8s".

Liyuan-83 / Test-C-code-with-CI Public



<> Code Issues Pull requests **Actions** Projects Wiki Security Insights Settings

test CI GitHub Actions Demo #7


Summary

Jobs

Explore-GitHub-Actions

Triggered via push yesterday	Status	Total duration	Artifacts
 Liyuan-83 pushed  c9b5d14 <b>main</b>	<b>Success</b>	<b>18s</b>	—









CI\_Demo.yml  
on: push

 Explore-GitHub-Actions 8s

# ● 執行結果

## Explore-GitHub-Actions




succeeded yesterday in 8s

- >  Set up job
- >  Run echo "🚀 The job was automatically triggered by a push event."
- >  Run echo "💡 This job is now running on a Linux server hosted by GitHub!"
- >  Run echo "🗨️ The name of your branch is refs/heads/main and your repository is Liyuan-83/Test-C-code-with-CI."
- >  Check out repository code
- >  Run echo "💡 The Liyuan-83/Test-C-code-with-CI repository has been cloned to the runner."
- >  Install GCC
- >  Run echo "💻 The workflow is now ready to test your code on the runner."

### ✓ 🚀 Run the resault

```
1 ▶ Run mkdir /home/runner/work/Test-C-code-with-CI/Test-C-code-with-CI/build
6 sh: 1: pause: not found
7 Hello World
8 input is 5
9 input is 6
10 input is 7
11 input is 8
```

Step

- >  Run echo "🍏 This job's status is success."
- >  Post Check out repository code
- >  Complete job

```
name: GitHub Actions Demo
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo "🚀 The job was automatically triggered by a ${ github.event_name } event."
      - run: echo "💡 This job is now running on a ${ runner.os } server hosted by GitHub!"
      - run: echo "🗨️ The name of your branch is ${ github.ref } and your repository is ${ github.repository }."
      - name: Check out repository code
        uses: actions/checkout@v3
      - run: echo "💡 The ${ github.repository } repository has been cloned to the runner."

      - name: Install GCC
        run: sudo apt-get install gcc g++

      - run: echo "💻 The workflow is now ready to test your code on the runner."
      - name: 🚀 Run the resault
        run: |
          mkdir ${ github.workspace }}/build
          g++ ${ github.workspace }}/Lab1-HelloWord/main.cpp -o ${ github.workspace }}/build/test
          echo 5 6 7 8 | ${ github.workspace }}/build/test
      - run: echo "🍏 This job's status is ${ job.status }."
```