Table of Contents

Introduction 1.1

上課須知 1.2

- 相關服務事前註冊 1.2.1
- 事前安裝：自備筆電 1.2.2
- 事前安裝：電腦教室 1.2.3

git 2.1

- Introduction 2.1.1
- 確認環境 2.1.2
- Git 基礎 2.1.3
  - Git 是什麼？ 2.1.3.1
  - 為什麼我們需要 Git 2.1.3.2
  - Git vs SVN 2.1.3.3
  - 初始化專案 init 2.1.3.4
  - 設定檔 2.1.3.5
  - 練習題：建立一個裸容器 2.1.3.6
  - 觀念講解：索引 2.1.3.7
  - 觀念講解：認識 Git 物件 2.1.3.8
- Git 之 CRUD 2.1.4
  - 工作區、暫存區、儲存庫 2.1.4.1
  - 狀態 status 2.1.4.2
  - 新增 add 2.1.4.3
  - 送交 commit 2.1.4.4
  - 練習題：新增一個送交紀錄 2.1.4.5
  - 檢視 log / show 2.1.4.6

比對差異 diff 2.1.4.7
刪除 rm 2.1.4.8
練習題：刪除檔案 2.1.4.9
重新命名 mv 2.1.4.10
觀念講解： Git Flow 2.1.5
整體概念 2.1.5.1
master 分支 2.1.5.2
feature 分支 2.1.5.3
hotfix 分支 2.1.5.4
分支 branch 2.1.6
分支 branch CRUD 2.1.6.1
切換 checkout 2.1.6.2
練習題：新增一個分支 2.1.6.3
合併 merge 2.1.6.4
觀念解說：解決衝突 2.1.6.5
練習題：解決衝突 2.1.6.6
暫存 stash 2.1.6.7
練習題：將檔案放入暫存區 2.1.6.8
修改送交 2.1.7
還原 reset 2.1.7.1
資料還原 revert 2.1.7.2
(進階指令) rebase 2.1.7.3
觀念講解：reset vs revert 2.1.7.4
練習題：還原狀態 2.1.7.5
遠端協作 2.1.8
容器 2.1.8.1
remote 2.1.8.2

複製 clone 2.1.8.3
拉 pull 2.1.8.4
更新 fetch 2.1.8.5
練習題：透過 fetch 指令取得最新送交紀錄 2.1.8.6
部署 push 2.1.8.7
練習題：發佈送交 2.1.8.8
標籤 tag 2.1.8.9
練習題：發佈 tag 2.1.8.10
Github 2.1.9
issue 2.1.9.1
clone 2.1.9.2
fork 2.1.9.3
pull request 2.1.9.4
練習題：操作 Github 2.1.9.5
補充 2.1.10
blame 2.1.10.1
grep 2.1.10.2
tig 2.1.10.3
實用小技巧 2.1.10.4
更多資源 2.1.10.5
Jenkins 3.1
基本概念介紹 3.1.1
豐田式生產 3.1.1.1
敏捷式開發 3.1.1.2
持續整合 3.1.1.3
安裝 Jenkins 3.1.1.4
Quick Start 3.1.2

Lab 101 3.1.2.1
Lab 102 3.1.2.2
Lab 103 3.1.2.3
Lab 104 3.1.2.4
CI flow 簡介 3.1.3
持續整合實作以 Java + git 為例 3.1.4
Lab 進行方式說明 3.1.4.1
Lab 201 建置 3.1.4.2
build-package 3.1.4.2.1
build-archive 3.1.4.2.2
Lab 202 測試 3.1.4.3
test 3.1.4.3.1
plugin-JUnit 3.1.4.3.2
plugin-Cobertura 3.1.4.3.3
Lab 203 部署 3.1.4.4
ssh-遠端連線設置 3.1.4.4.1
使用 ssh 進行 deploy 3.1.4.4.2
Lab 204 Task 連結 與 Build Tigger 3.1.4.5
定期建置 Tigger 3.1.4.5.1
Pull SCM Tigger 3.1.4.5.2
Task 連結-測試成功則進行部署 3.1.4.5.3
搭配 Jenkins 2.0 之 Pipeline 進行建置 3.1.5
基礎練習 3.1.5.1
使用 Pipeline 進行建置 3.1.5.2
使用 GitHub Organization 進行建置 3.1.5.3
使用 Multibranch Pipeline 進行建置 3.1.5.4
設定 3.1.6

環境變數 3.1.6.1
ssh 與 scp 3.1.6.2
進階設定 3.1.7
security 3.1.7.1
timezone 3.1.7.2
master/slave 3.1.7.3
Docker 4.1
Meet Docker 4.1.1
What is Docker 4.1.1.1
Docker vs VMs 4.1.1.2
Why Docker 4.1.1.3
Image 4.1.1.4
Container 4.1.1.5
Repository 4.1.1.6
Registry 4.1.1.7
command reference 4.1.2
Lab 101: Docker Image 4.1.3
help 4.1.3.1
search 4.1.3.2
pull 4.1.3.3
images 4.1.3.4
inspect 4.1.3.5
save, load 4.1.3.6
rmi 4.1.3.7
Lab 102: Docker Container 4.1.4
create, start, stop, ps 4.1.4.1
run 4.1.4.2

run -d 4.1.4.3
exec, attach 4.1.4.4
rm 4.1.4.5
Lab 103: Registry 4.1.5
distribution private 4.1.5.1
distribution public 4.1.5.2
Lab 104: dockerfile 4.1.6
CMD vs entrypoint 4.1.6.1
Lab 201: Docker volume 4.1.7
create, inspect 4.1.7.1
direct volume host filesystem 4.1.7.2
volume form container 4.1.7.3
Lab 202: Docker network 4.1.8
link 4.1.8.1
expose 4.1.8.2
create 4.1.8.3
port 4.1.8.4
透過 docker-compose 使用 network 4.1.8.5
Lab 203: Docker Machine 的使用 (1 hour) 4.1.9
install 4.1.9.1
drive 4.1.9.2
create 4.1.9.3
env 4.1.9.4
ls 4.1.9.5
使用 docker-machine 連結已存在之遠端 docker engine 4.1.9.6
upgrade 4.1.9.7
rm 4.1.9.8

Lab 301: 使用 docker 建置專案運行環境 (1 hour) 4.1.10
運行環境安裝 4.1.10.1
進入 Container 進行建置 4.1.10.2
透過 Dockerfile 進行建置 4.1.10.3
Lab 401: java 範例專案使用 docker 進行建置 (1 hour) 4.1.11
範例專案 4.1.11.1
安裝 4.1.11.2
測試 4.1.11.3
運行 4.1.11.4
打包 4.1.11.5
Lab 402: docker-compose 4.1.12
run and up 4.1.12.1
其他指令 4.1.12.2
Lab 403: java 範例專案使用 docker-compose 進行建置 (1 hour) 4.1.13
練習：製作 dev mode service 4.1.13.1
範例專案 4.1.13.2
安裝 4.1.13.3
測試 4.1.13.4
運行 4.1.13.5
打包 4.1.13.6
Lab 404: java 範例專案搭配 jenkins docker 進行部署 (1 hour) 4.1.14
production image 製作 4.1.14.1
練習：docker production 發布流程 4.1.14.2
練習：使用 production image 運行專案 4.1.14.3
練習：搭配 jenkins 建立 production deploy task 4.1.14.4
Lab 501: java 範例專案搭配 jenkins Pipeline 使用 Docker 進行專案建置 (1 hour) 4.1.15

Appendix 4.2
vt-x amd-v 異常造成 VM 無法開啟 4.2.1
Ubuntu install docker 4.2.2
Mac OS X install docker 4.2.3
docker 參考資料 4.2.4
git 參考資料 4.2.5

AgileWorks

本教材由創科資訊研發團隊謝宗穎、林彥宏等共同編撰。

敏捷軟體開發培訓課程，預約請洽創科資訊。

2016 年企業培訓課程包含：

- Docker 從基礎到進階實戰班
- Jenkins 從基礎到進階實戰班
- 持續整合與自動化測試實作（含 Jenkins 與 Docker）

持續整合與自動化測試實作班

適合對象

企業內訓 / 從基礎到進階

學習目的

透過 Learning by Doing 從做中學的方式，瞭解持續整合與自動化測試的方法論與專案實作方式，包括 Jenkins 與 Docker 的工具實務應用，幫助學員能在日常工作中實際應用。

- 理解敏捷開發中持續整合的價值與目的
- 學習如何搭配 Jenkins 完成專案自動化建置，包含單元測試及整合測試
- 學習如何透過 Docker 配置開發與部署的軟體環境，並應用在持續整合的流程中
- 以常用的開發工具（Java / Node.js 等）提供自動化專案建置與測試的範例

課程大綱

面對快速變動的系統開發需求，想要維持軟體的高品質要求，持續整合是重要的關鍵。Jenkins 是實現 CI 的工具首選，而 Docker 則是目前最受歡迎的輕量級虛擬化技術，透過這兩種工具與現有各種原始碼版本管理方案的整合，讓企業更容易實現持續整合流程中，關於自動化軟體建置及測試，以及快速驗證軟體版本演進的需求。

鋼鐵人創造了 Jarvis 為他打理大小事，讓他的主人可以專心在創意的發想以及對付強大的敵人。在軟體開發同樣的我們也可以透過機器人管家 Jenkins，來為我們打理開發流程中的大小事，讓我們專注在最有價值的工作上。本課程將從豐田式生產（Lean Production）切入，引導學習持續整合的目的，並介紹軟體開發流程的各個階段如何搭配 CI 運作。另外也將一步步帶領學員，使用 Jenkins 進行自動化建置、測試與部署的實作，以及搭配 Docker 容器進行測試與部署，透過此課程將領引大家打造一個軟體專案的管家，關注開發流程的每個一環節。

課程特色

採用 Workshop 形式授課，每個單元提供實作練習。 強調每位學員皆能動手完成實作練習，搭配 AgileWorks 教學平台提供線上教材與 VM Lab 操作環境。 基礎課程包含 TDD 觀念、專案自動化建置實務、Jenkins 基本功能講解及常用 Jenkins Plugins 功能說明等。 進階課程包含 Jenkins Cluster 建置、專案自動化測試實務及搭配 Docker 容器的測試部署整合等。 可依照開發團隊需求調整搭配的軟體工具（例如 Git、SVN；Java、PHP、Node.js 等）

備註

為確保課程品質及學員滿意度，Workshop 總人數建議以 30 人為限。

時數

建議安排 12 小時（分成 2 次上課，每次 6 小時，例如：上午 2 小時、下午 4 小時）

課程實績

iThome、資策會、關貿網路、台灣大哥大、中華電信學院、…等。

授課講師

謝宗穎 / 創科資訊共同創辦人

> 實務上的 Web Full Stack 開發經驗，熟悉 Java 以及 JavaScript 開發技術，對開發流程優化有高度興趣，提倡 TDD 開發方式，以及透過 CI Jenkins 進行自動化，利用 Docker 輔助開發或測試環境建置，技術團隊管理者。
>
> 培訓經驗：曾於 JSDC 2015；JCConf Taiwan 2015；Container Summit 2014 / 2015 等技術大會擔任講者，並在 QNAP、關貿網路、資策會與 iThome 等機構擔任技術相關演講及工作坊講師。專題演講經驗有台灣大學、台中科大、正修科大與義守大學等校。

劉艾霖 / 創科資訊軟體顧問

> 實務上的 Web Full Stack 開發經驗，熟悉 Java 以及 JavaScript 開發技術，專精於網站開發、架構設計與撰寫前後端自動化測試。是位全端工程師、後端工程師、測試開發工程師、企業內訓講師，具有 8 年軟體開發相關資歷，目前從事企業技術教學、技術顧問、軟體開發。
>
> 培訓經驗：曾於 JSDC 2016、Modern Web 2016、Testing Day 2017 等技術大會擔任講者，並於各大院校、資策會、IThome、企業內訓等機構擔任技術研獎及 Workshop 講者。

持續整合與自動化測試：相關服務事前註冊

註冊 GitHub 帳號

本課程會帶你實機操作，課程中將會需要下載範例專案，以及進行 GitHub 與 Docker Hub 串接運用。

如果你還沒有 GitHub 帳號，請在 GitHub 官方網站進行註冊。

註冊 Docker Hub 帳號

如果你還沒有 Docker Hub 帳號，請到 Docker Hub 網站 進行註冊。

持續整合與自動化測試：自備筆電課前準備

本課程以實機操作實作練習為主，為維持課程品質及確保學習進度，我們假設您已經熟悉：

1. Linux 基本操作指令
2. Git 版本控制工具基本操作指令

請先完成以下軟體安裝：

1. Google Chrome 瀏覽器
2. VirtualBox 5.x 以上版本虛擬機器軟體及 Extension Pack

並確保電腦規格符合以下建議條件：

1. 建議使用 64 位元作業系統
2. 作業系統 Windows、Mac 或 Linux 皆可
3. 建議 8GB 以上實體記憶體（最低需求為 4GB）
4. 至少 40GB 磁碟可用空間

我們在課程中會提供實作練習專用的 VM Image，以方便搭配教材內容。

安裝 VirtualBox

依照您的作業系統類型，安裝 5.x 以上最新版本的 VirtualBox 軟體，建議同時安裝「VirtualBox Extension Pack」以支援虛擬機器的進階功能。

VirtualBox 下載頁面

https://www.virtualbox.org/wiki/Downloads

安裝 VirtualBox Extension Pack 之步驟可以參考:

http://www.arthurtoday.com/2011/01/oracle-vm-virtualbox-40-extension-pack_14.html

常見的 VirtualBox 安裝問題：

- vt-x amd-v 異常造成 VM 無法開啟，參考下面文章設置:

https://www.qnap.com/zh-hk/how-to/faq/article/%E5%A6%82%E4%BD%95%E5%95%9F%E7%94%A8-intel-vt-x-%E5%92%8C-amd-svm

註冊 GitHub 帳號

本課程會帶你實機操作，課程中將會需要下載範例專案，以及進行 GitHub 與 Docker Hub 串接運用。

如果你還沒有 GitHub 帳號，請在 GitHub 官方網站:

https://github.com/

進行註冊。

註冊 Docker Hub 帳號

如果你還沒有 Docker Hub 帳號，請到 Docker Hub 網站:

https://hub.docker.com/account/signup/

進行註冊。

持續整合與自動化測試：電腦教室課前準備

請先完成以下軟體安裝：

1. Google Chrome 瀏覽器
2. VirtualBox 5.x 以上版本虛擬機器軟體及 Extension Pack

並確保電腦規格符合以下建議條件：

1. 建議使用 64 位元作業系統
2. 作業系統 Windows、Mac 或 Linux 皆可
3. 建議 8GB 以上實體記憶體（最低需求為 4GB）
4. 至少 40GB 磁碟可用空間

我們在課程中會提供實作練習專用的 VM Image，以方便搭配教材內容。

安裝 VirtualBox

依照您的作業系統類型，安裝 5.x 以上最新版本的 VirtualBox 軟體，建議同時安裝「VirtualBox Extension Pack」以支援虛擬機器的進階功能。

VirtualBox 下載頁面 https://www.virtualbox.org/wiki/Downloads

安裝 VirtualBox Extension Pack 之步驟可以參考這份文件。

常見的 VirtualBox 安裝問題：

- vt-x amd-v 異常造成 VM 無法開啟

Introduction

講師

劉艾霖

> 實務上的 Web Full Stack 開發經驗，熟悉 Java 以及 JavaScript 開發技術，專精於網站開發、架構設計與撰寫前後端自動化測試。是位全端工程師、後端工程師、測試開發工程師、企業內訓講師，具有 8 年軟體開發相關資歷，目前從事企業技術教學、技術顧問、軟體開發。
>
> 培訓經驗：曾於 JSDC 2016、Modern Web 2016、Testing Day 2017 等技術大會擔任講者，並於各大院校、資策會、IThome、企業內訓等機構擔任技術研獎及 Workshop 講者。

上課注意事項

需事前安裝以下軟體

- 如果你還沒有 GitHub 帳號，請在 GitHub 官方網站進行註冊。
- 建議使用 Google Chrome 瀏覽器
- VirtualBox 5.x 以上版本虛擬機器軟體及 Extension Pack

並確保電腦規格符合以下建議條件：

1. 建議使用 64 位元作業系統
2. 作業系統 Windows、Mac 或 Linux 皆可
3. 建議 8GB 以上實體記憶體（最低需求為 4GB）
4. 至少 40GB 磁碟可用空間

我們在課程中會提供實作練習專用的 VM Image，以方便搭配教材內容。

License

MIT © alincode

VM

- 安裝 VirtualBox

Cloud 9

```
帳號：user
密碼：password
```

打開 IDE

http://localhost:9083

如果字太小

Go to Preferences > User Settings > Editor > Terminal. There you will find the Font Size option.

Git 是什麼

- 它是一個版本控制系統 (VCS)
- 利用分散式的方式開發
- 快速且有效率
- 支援並且鼓勵分支的開發環境

語法結構

```
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path]
[--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objec
ts] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespa
ce=<name>]
           <command> [<args>]
user@ubuntu-xenial:~/workspace/git-sample/alincode-hello$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path]
[--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objec
ts] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespa
ce=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize
an existing one
```

```
work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlin
k
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the
 index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that intro
duced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and workin
g tree, etc
   merge      Join two or more development histories together
   rebase     Forward-port local commits to the updated upstr
eam head
   tag        Create, list, delete or verify a tag object sig
ned with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another reposito
ry
   pull       Fetch from and integrate with another repositor
y or a local branch
   push       Update remote refs along with associated object
s

'git help -a' and 'git help -g' list available subcommands an
```

```
d some
concept guides. See 'git help <command>' or 'git help <concep
t>'
to read about a specific subcommand or concept.
```

為什麼我們需要 Git

- 離線
- 快速
- 更好的 merge 機制

Git VS SVN

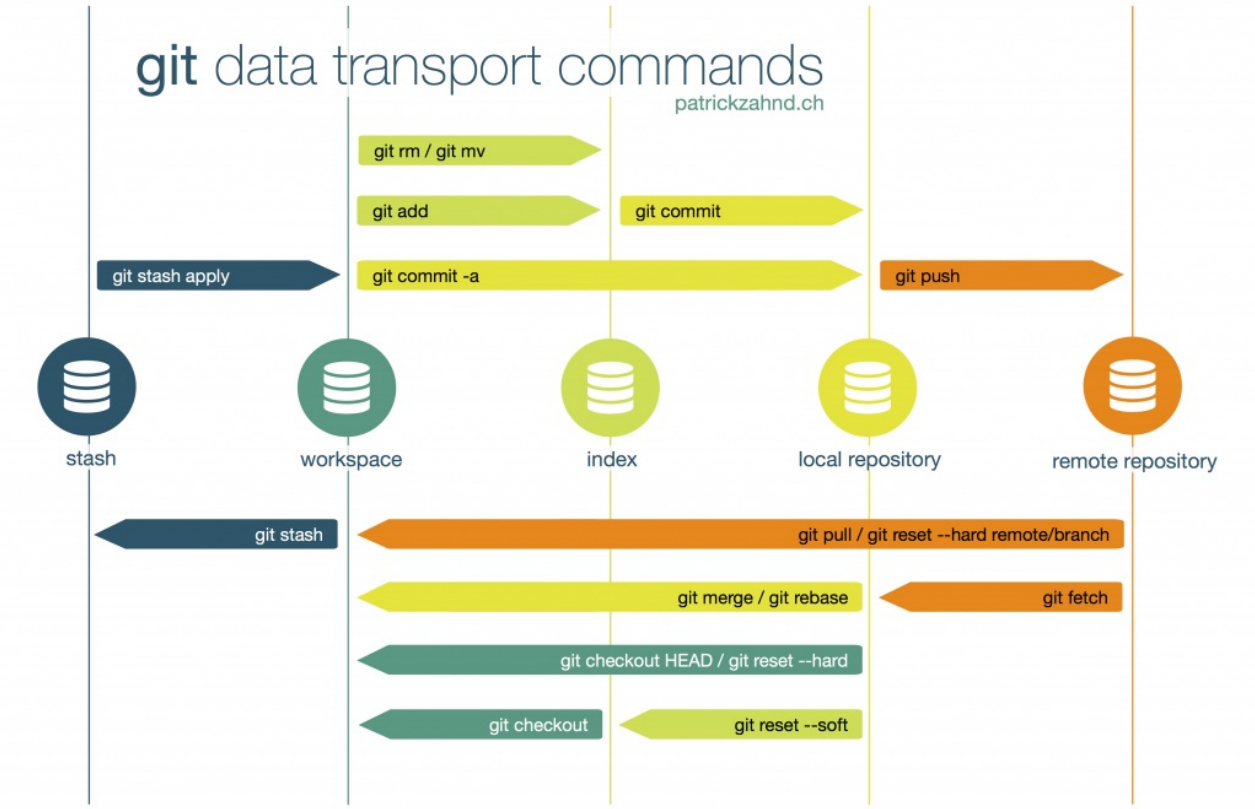
Git

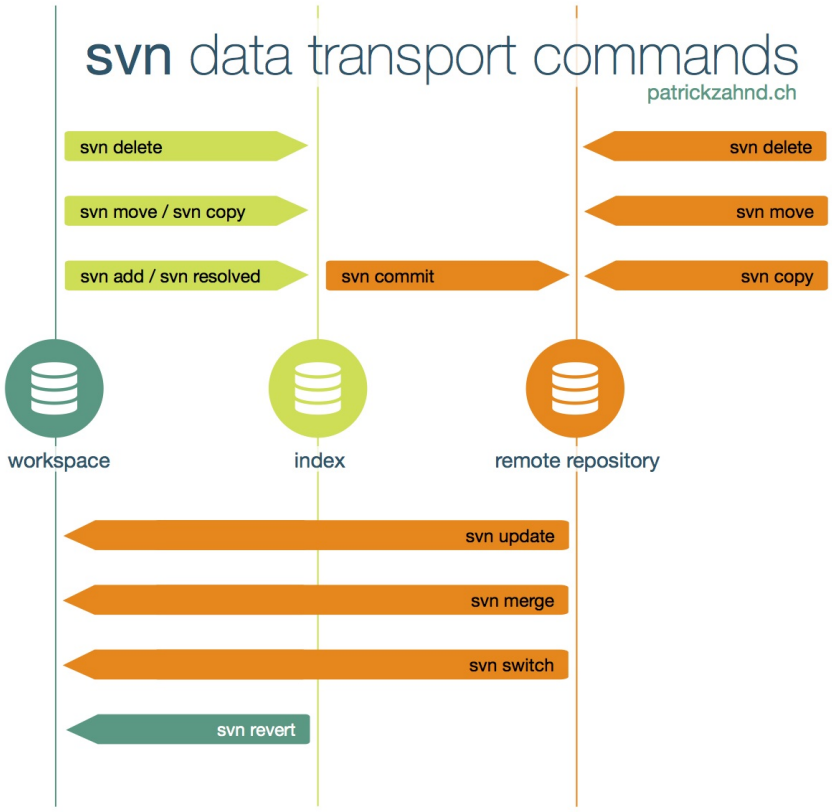
- Straight merge 預設的合併模式，會有全部的被合併的 branch commits 記錄加上一個 merge-commit，看線圖會有兩條 Parents 線，並保留所有 commit log。

SVN

- Squashed commit 壓縮成只有一個 merge-commit，不會有被合併的 log。

transport





比較

- Git 的「送交」與「發布」步驟是分開的
- Git 沒有唯一的真實歷史紀錄，在分散式開發環境中，大家都是平等的。
- 時間戳不重要，重點是「父子關係」。

git init

裸容器 (bare)：一個沒有工作目錄的容器，目錄通常會取為 xxx.git

使用情境

- 初始化專案

常用範例

範例	說明
git init	
git init --bare	初始化裸容器

語法結構

```
usage: git init [-q | --quiet] [--bare] [--template=<template
-directory>] [--shared[=<permissions>]] [<directory>]

    --template <template-directory>
                          directory from which templates will
 be used
    --bare                create a bare repository
    --shared[=<permissions>]
                          specify that the git repository is
to be shared amongst several users
    -q, --quiet           be quiet
    --separate-git-dir <gitdir>
                          separate git dir from working tree
```

設定檔

設定檔的位置

```
Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
```

- git config --global 的設定內容會被寫入 ~/.gitconfig
- git config 的設定會被寫入 .git/config

編輯設定檔的方式

- `vi .git/config`
- 使用 `git config` 指令

常用範例

範例	說明
git config l	列出所有設定值
git config push.default matching	
git config --global user.name "demo_user"	
git config --global user.email "demo_user@demo.com"	
git config --local user.name "demo_user"	設定名稱
git config --local user.email "demo_user@demo.com"	設定信箱
git config alias.tree "log --oneline --decorate --graph"	設定 tree 暱稱
git config alias.l "log --all --decorate --graph --oneline"	設定 l 暱稱
git config core.editor "vim"	修改預設編輯器

推薦設定的 alias

```
alias.co=checkout
alias.ci=commit -m
alias.st=status
alias.pl=pull
alias.ps=push
alias.df=diff
alias.br=branch
alias.ap=add -p
alias.aa=add .
alias.cp=checkout -p
alias.cc=checkout .
alias.ss=stash
alias.spp=stash pop
alias.sdd=stash drop
alias.rb=rebase
alias.type=cat-file -t
alias.dump=cat-file -p
alias.clear=remote prune origin
alias.tree=log --oneline --decorate --graph
alias.cm=commit
alias.lg=log --color --graph --pretty=format:'%Cred%h%Creset
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Crese
t' --abbrev-commit --
alias.logg=log --all --graph --pretty=format:'%Cred%h%Creset
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Crese
t' --abbrev-commit --date=relative
```

語法結構

```
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    -f, --file <file>     use given config file
```

```
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-regex]
    --get-all             get all values: key [value-regex]
    --get-regexp          get values for regexp: name-regex [
value-regex]
    --get-urlmatch        get value specific for the URL: sec
tion[.var] URL
    --replace-all         replace all matching variables: nam
e value [value_regex]
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-rege
x]
    --unset-all           remove all matches: name [value-reg
ex]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    -e, --edit            open an editor
    --get-color           find the color configured: slot [de
fault]
    --get-colorbool       find the color setting: slot [stdou
t-is-tty]

Type
    --bool                value is "true" or "false"
    --int                 value is decimal number
    --bool-or-int         value is --bool or --int
    --path                value is a path (file or directory
name)

Other
    -z, --null            terminate values with NUL byte
    --name-only           show variable names only
    --includes            respect include directives on looku
p
```

練習題：建立一個裸容器

1. 透過 `mkdir ex01` 指令，新增一個資料夾叫 `ex01`
2. 透過 `git init --bare` 指令，進行專案初始化。
3. 完成裸容器的建置

索引 (Index)

- 索引是一個動態且暫時的二進位檔案，此檔案描述整個容器的目錄結構。
- Git 的一個關鍵特色在於，它可以讓你使用有條理，且完整的步驟來修改索引的內容。
- 索引允許將持續增加的開發步驟及送交的修改之間區隔開來。

索引的名稱由來

- 依照物件內容，透過 SHA1 演算法運算，產出 160 個位元得值，使用 40 位 16 進位的數字呈現。範例：c21132707b7c5a945dc6187655981cb5d2b67887

認識 Git 物件

物件型態

blob 物件

實際檔案內容

tree 物件

`git write-tree`

```
100644 blob f3b9a76bc4e9ccad8e613a8f00d279ae023e81d9    .giti
gnore
040000 tree 2ce507750fe9d51b9a587934cadb8dcb4d08e358    .vsco
de
100644 blob 95e9b0b25b5295ea09df687ce87921dc359fbcb0    LICEN
SE
100644 blob 5302934326ee2082005ebf0a7cd3a932a75353a0    READM
E.md
100644 blob ee0e01483aefc3326749380bd9f47f13181cb14c    SUMMA
RY.md
040000 tree e3a43292c1c3b9d75731ffa28e2f8b287654efc3    comma
nd
040000 tree 4af7150c15ed99abc02e935619af322da8620903    found
ation
040000 tree 671d79384113b2b73961f254d25013bfe89d25e3    git-f
low
100644 blob 9464f843f5dc01ede201ee872218db64eb905dc1    githu
b.md
040000 tree ca88d0944c8ac48c7750eed3ab534f6f1911969f    mise
040000 tree 05f4dc2fd37ead780727bd55d122c23b204f3cab    pract
ice
040000 tree a4006ed3288ce93305ebd8bb79be648c2955d101    prepa
re
100644 blob cf61a2ddec9e7b2d7e73e1400b1ff0c2ff24d31a    resou
rce.md
040000 tree a54bcc2e0a588a95dac720af08e400245120f9ee    setup
100644 blob c9ceb128cc0a241fd3d088fa9f14972916502f40    tips.
md
```

commit 物件 (送交物件)

```
tree 81745419c7ba83f077eaaf86bd2fafd3fb9b8b64
parent f512170b13fa45e065c219fad3a89843b0002b4e
author alincode <alincode@gmail.com> 1526312241 +0800
committer alincode <alincode@gmail.com> 1526312241 +0800

update
```

- 樹狀物件的名稱，該物件顯示檔案的關聯
- 新版本作者的姓名以及新版本修改的時間
- 送交者的姓名及送交時間
- 此修訂版本 commit 時，留下的描述

tag 物件

```
git rev-parse v0.0.1
git cat-file -p d8044b536a827ea34e638a3da76dae9740922896
```

```
tree 16ae2442a6869cb7be3aeec47129721218d8074d
parent fb7f269f31cd9fe29712ed2a4b7f4cde208ed6e3
parent 8ccde2dc957c45a2575ec1b385cfda93c9ed1b60
author Your Name <you@example.com> 1526307625 +0000
committer Your Name <you@example.com> 1526307625 +0000

Merge branch 'dev'
```

物件儲存

- 當 Git 要儲存一個物件時，它使用內容雜湊演算值，而不是檔案名稱。若有兩個存在不同目錄，但內容相同的檔案，兩個檔案的 SHA1 會相同。
- SHA1 的前兩位數，會作為 `git/object/前兩位數` 的放置位置。

git ls-files -s

```
100644 aff6388cffe192b412dfca52ddeee7a7ebd8961e 0       hello
.txt
```

find .git

```
.git/objects/af
.git/objects/af/f6388cffe192b412dfca52ddeee7a7ebd8961e
```

git write-tree

```
bb9e55860a20ae3a1c441d5e2d8a5e0261a3f2ff
```

git cat-file -p cbfd7a8e6c7bd0eec9f524b42bdce9ff31bac2e3

將內容反組譯

```
100644 blob aff6388cffe192b412dfca52ddeee7a7ebd8961e    hello
.txt
100644 blob 14be0d41c639d701e0fe23e835b5fe9524b4459d    hello
2.txt
```

Git 內部檔案

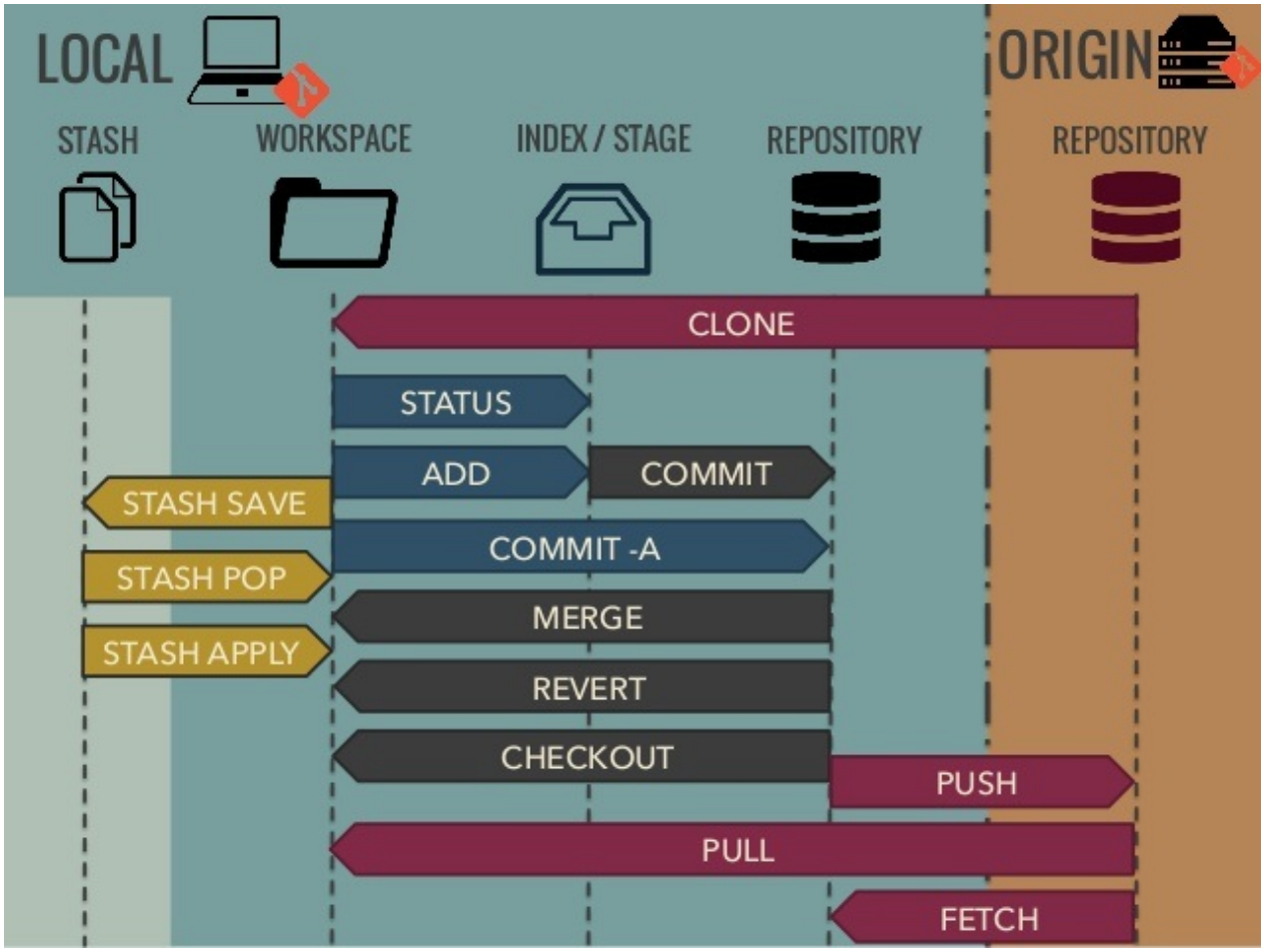
find .git

```
.git
.git/branches
.git/COMMIT_EDITMSG
.git/config
.git/description
.git/HEAD
.git/hooks
```

```
.git/hooks/applypatch-msg.sample
.git/hooks/commit-msg.sample
.git/hooks/post-update.sample
.git/hooks/pre-applypatch.sample
.git/hooks/pre-commit.sample
.git/hooks/pre-push.sample
.git/hooks/pre-rebase.sample
.git/hooks/pre-receive.sample
.git/hooks/prepare-commit-msg.sample
.git/hooks/update.sample
.git/index
.git/info
.git/info/exclude
.git/logs
.git/logs/HEAD
.git/logs/refs
.git/logs/refs/heads
.git/logs/refs/heads/master
.git/logs/refs/remotes
.git/logs/refs/remotes/origin
.git/logs/refs/remotes/origin/HEAD
.git/logs/refs/remotes/origin/master
.git/objects
.git/objects/04
.git/objects/04/1f29da84578ee14f4f5e5147a576d0580cea9d
.git/objects/info
.git/objects/pack
.git/objects/pack/pack-28fa62ca95b17883a7d6a1b6d9b2013a7d9ab7
51.idx
.git/objects/pack/pack-28fa62ca95b17883a7d6a1b6d9b2013a7d9ab7
51.pack
.git/packed-refs
.git/refs
.git/refs/heads
.git/refs/heads/master
.git/refs/remotes
.git/refs/remotes/origin
.git/refs/remotes/origin/HEAD
```

```
.git/refs/remotes/origin/master
.git/refs/tags
```

工作區、暫存區、儲存庫



檔案階段	檔案所在位置	說明
已修改	Working Directory(目前的目錄)	經過修改的檔案
已暫存	Staging area(暫存區)	要提交的變動清單
已提交	Repository(容器)	已提交的檔案及變動記錄

git status 指令

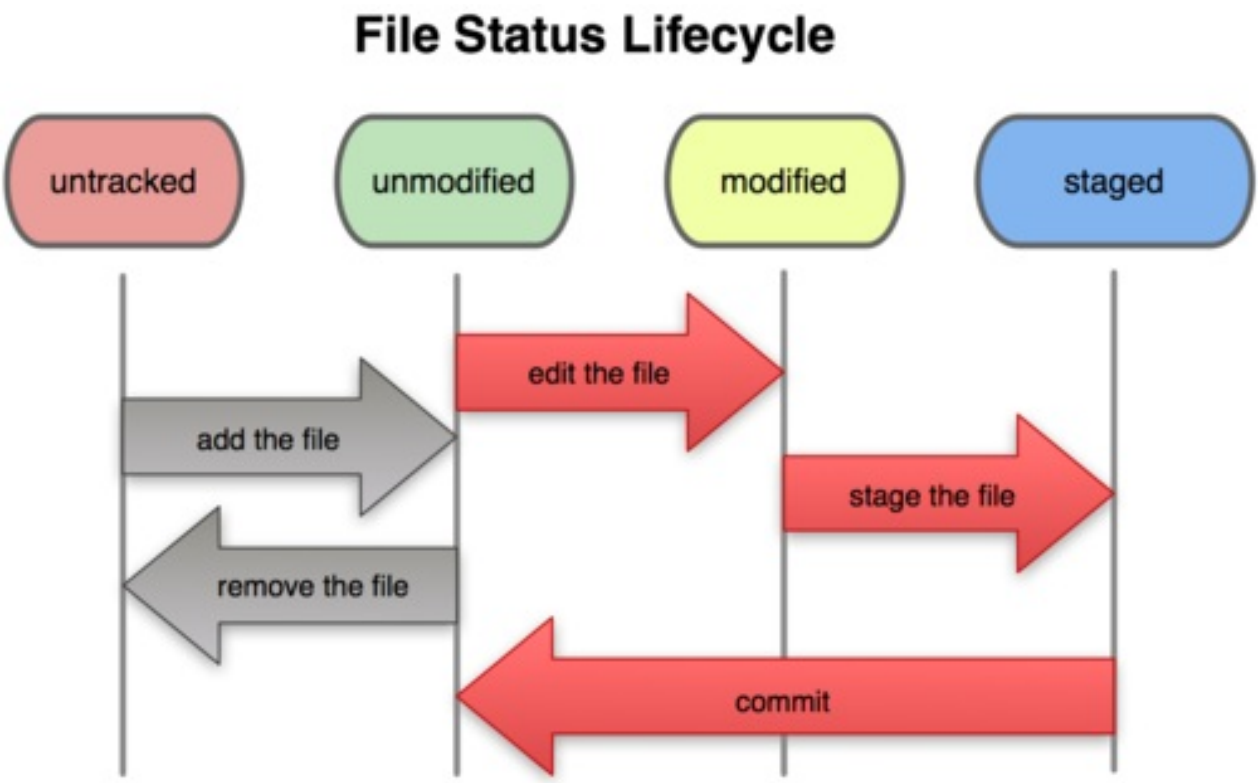
使用情境

- 觀看檔案的狀態

狀態

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

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:   command/config.md
        modified:   command/status.md
        modified:   command/tag.md
        modified:   foundation/git-vs-svn.md
        modified:   foundation/index.md
        modified:   foundation/object.md
        modified:   foundation/space.md
        modified:   foundation/what.md
        modified:   foundation/why.md
        modified:   tips.md

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

        command/assets/status.png
        command/assets/status2.png
```

- untracked：Untracked files（未被追蹤的檔案）
- unmodified：Changes not staged for commit（被更動但尚未要提交的檔案）
- modified: Changes to be committed（將要提交的檔案）
- staged



檔案分類

- 被追蹤
- 不被追蹤
- 被忽略

.gitignore

想忽略的檔案，如暫存檔、log、編譯後的檔案

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Runtime data
pids
*.pid
*.seed
```

```
*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSC
over
lib-cov

# Coverage directory used by tools like istanbul
coverage

# nyc test coverage
.nyc_output

# Grunt intermediate storage (http://gruntjs.com/creating-plu
gins#storing-task-files)
.grunt

# Bower dependency directory (https://bower.io/)
bower_components

# node-waf configuration
.lock-wscript

# Compiled binary addons (http://nodejs.org/api/addons.html)
build/Release

# Dependency directories
node_modules/
jspm_packages/

# Typescript v1 declaration files
typings/

# Optional npm cache directory
.npm

# Optional eslint cache
.eslintcache
```

```
# Optional REPL history
.node_repl_history

# Output of 'npm pack'
*.tgz

# Yarn Integrity file
.yarn-integrity

# dotenv environment variables file
.env
```

常用範例

範例	說明
git status	看目前檔案的狀態

語法結構

```
usage: git status [<options>] [--] <pathspec>...

    -v, --verbose         be verbose
    -s, --short           show status concisely
    -b, --branch          show branch information
    --porcelain           machine-readable output
    --long                show status in long format (default
)
    -z, --null            terminate entries with NUL
    -u, --untracked-files[=<mode>]
                          show untracked files, optional mode
s: all, normal, no. (Default: all)
    --ignored             show ignored files
    --ignore-submodules[=<when>]
                          ignore changes to submodules, optio
nal when: all, dirty, untracked. (Default: all)
    --column[=<style>]    list untracked files in columns
```

git add 指令

將檔案放入索引。將檔案備份到「物件儲存」中，使用 SHA1 算出名稱並且建立索引，將檔案放入準備狀態。

常用範例

範例	說明
git add README.txt	
git add .	將資料先暫存到 staging area

語法結構

```
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored file
s
    -u, --update          update tracked files
    -N, --intent-to-add   record only the fact that the path
will be added later
    -A, --all             add changes from all tracked and un
tracked files
    --ignore-removal      ignore paths removed in the working
 tree (same as --no-all)
    --refresh             don't add, only refresh the index
    --ignore-errors       just skip files which cannot be add
ed because of errors
    --ignore-missing      check if - even missing - files are
 ignored in dry run
```

git commit 指令

- 送交
- 版本和容器變動的最小單位

使用情境

- 新增送交
- 更改送交

常用範例

範例	說明
git commit -m "commit message"	新增一筆送交紀錄
git commit -a -m "commit message"	git add . + git commit -m
git commit –amend -m "修改成新的 message"	改變最後一次提交紀錄

語法結構

```
usage: git commit [<options>] [--] <pathspec>...

    -q, --quiet           suppress summary after successful c
ommit
    -v, --verbose         show diff in commit message templat
e

Commit message options
    -F, --file <file>     read message from file
    --author <author>     override author for commit
    --date <date>         override date for commit
```

```
    -m, --message <message>
                          commit message
    -c, --reedit-message <commit>
                          reuse and edit message from specifi
ed commit
    -C, --reuse-message <commit>
                          reuse message from specified commit
    --fixup <commit>      use autosquash formatted message to
 fixup specified commit
    --squash <commit>     use autosquash formatted message to
 squash specified commit
    --reset-author        the commit is authored by me now (u
sed with -C/-c/--amend)
    -s, --signoff         add Signed-off-by:
    -t, --template <file>
                          use specified template file
    -e, --edit            force edit of commit
    --cleanup <default>   how to strip spaces and #comments f
rom message
    --status              include status in commit message te
mplate
    -S, --gpg-sign[=<key-id>]
                          GPG sign commit

Commit contents options
    -a, --all             commit all changed files
    -i, --include         add specified files to index for co
mmit
    --interactive         interactively add files
    -p, --patch           interactively add changes
    -o, --only            commit only specified files
    -n, --no-verify       bypass pre-commit hook
    --dry-run             show what would be committed
    --short               show status concisely
    --branch              show branch information
    --porcelain           machine-readable output
    --long                show status in long format (default
)
```

```
     -z, --null            terminate entries with NUL
     --amend               amend previous commit
     --no-post-rewrite     bypass post-rewrite hook
     -u, --untracked-files[=<mode>]
                           show untracked files, optional mode
s: all, normal, no. (Default: all)
```

練習題：新增一個送交紀錄

1. 透過 `mkdir ex02` 指令，新增一個資料夾叫 `ex02`
2. 透過 `git init` 指令，進行專案初始化。
3. 透過 `echo 123 >> README.md` 指令，在資料夾中，新增一個檔案叫 `README.md`
4. 透過 `git add .` 指令，將所有更變加入 index 中
5. 透過 `git commit -m 'init'` 指令，將更變送交給 git
6. 透過 `echo 123 >> README2.md` 指令，在資料夾中，新增一個檔案叫 `README2.md`
7. 透過 `git add .` 指令，將所有更變加入 index 中
8. 透過 `git commit -m 'add README2 file'` 指令，將更變送交給 git
9. 完成第一個版本控制範例

log / show 指令

使用情境

- 查看送交內容

常用指令範例

範例	說明
git show README.md	
git log [HEAD]	查看commit歷史
git log master	
git log --oneline	
git log --author="alincode"	只查看特定人的送交紀錄
git log -p hello.js	
git log --follow README.md	
git log --oneline --abbrev-commit --all --graph --decorate --color	
git log --graph --oneline --all --decorate	顯示所有分之並圖形化

- –graph：選項會用文本模式排列出commit 節點的演進圖
- –oneline：會用最精簡的方式顯示．
- –all：會顯示所有分支的commit紀錄。
- –decorate：表示要標示分支的名稱。

語法結構

```
usage: git log [<options>] [<revision-range>] [[--] <path>...
]
   or: git show [<options>] <object>...

    -q, --quiet           suppress diff output
    --source              show source
    --use-mailmap         Use mail map file
    --decorate[=...]      decorate options
    -L <n,m:file>         Process line range n,m in file, cou
nting from 1
```

git diff 指令

差異比較

使用情境

- 檢查更動
- 判斷該文件是否發成衝突或者解決衝突是否成功
- git diff 用在衝突檔案時只會顯示衝突的部分，而不會顯示只有一邊修改的部分

常用範例

範例	說明
git diff --ours/--theirs	
git diff	比對差異

語法結構

```
usage: git diff [<options>] [<commit> [<commit>]] [--] [<path
>...]
```

git rm 指令

刪除檔案

常用範例

範例	說明
git rm README.txt	將檔案從 repo 中刪除

語法結構

```
usage: git rm [<options>] [--] <file>...

    -n, --dry-run         dry run
    -q, --quiet           do not list removed files
    --cached              only remove from the index
    -f, --force           override the up-to-date check
    -r                    allow recursive removal
    --ignore-unmatch      exit with a zero status even if not
hing matched
```

練習題：刪除檔案

1. 透過 `git rm README2.md` 指令，刪除 README2.md 檔案
2. 透過 `git commit -m 'remove readme2'` 指令，將刪除檔案的紀錄送交給 repo。

git mv 指令

常用範例

範例	說明
git mv README.txt README2.txt	將檔案重新命名

語法結構

```
usage: git mv [<options>] <source>... <destination>

    -v, --verbose         be verbose
    -n, --dry-run         dry run
    -f, --force           force move/rename even if target ex
ists
    -k                    skip move/rename errors
```

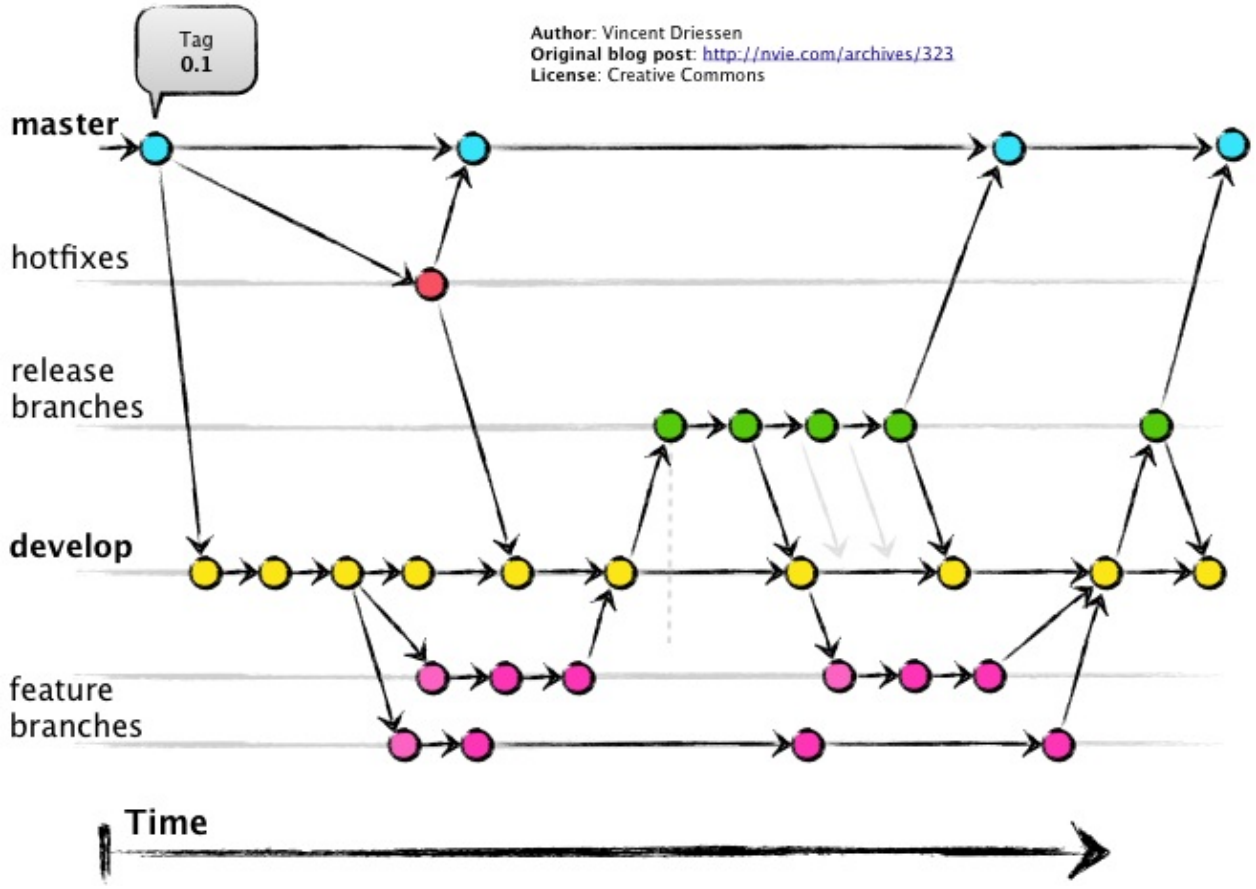
Git Flow 整體概念

分支 branch

在開發軟體時，可能同時會有多人在開發同一功能或修復錯誤，也可能會有多個發佈版本的存在，並且需要針對每個版本進行維護。為了能支援同時進行數個功能的增加或版本控制，Git具備了分支的功能。分支是為了將修改記錄的整體流程分開儲存，讓分開的分支不受其他分支的影響，所以在同一個數據庫裡可以同時進行多個不同的修改。

使用情境

- 可以讓你的系統依據不同的需求分別進行開發，又不互相影響。



建議分之命名

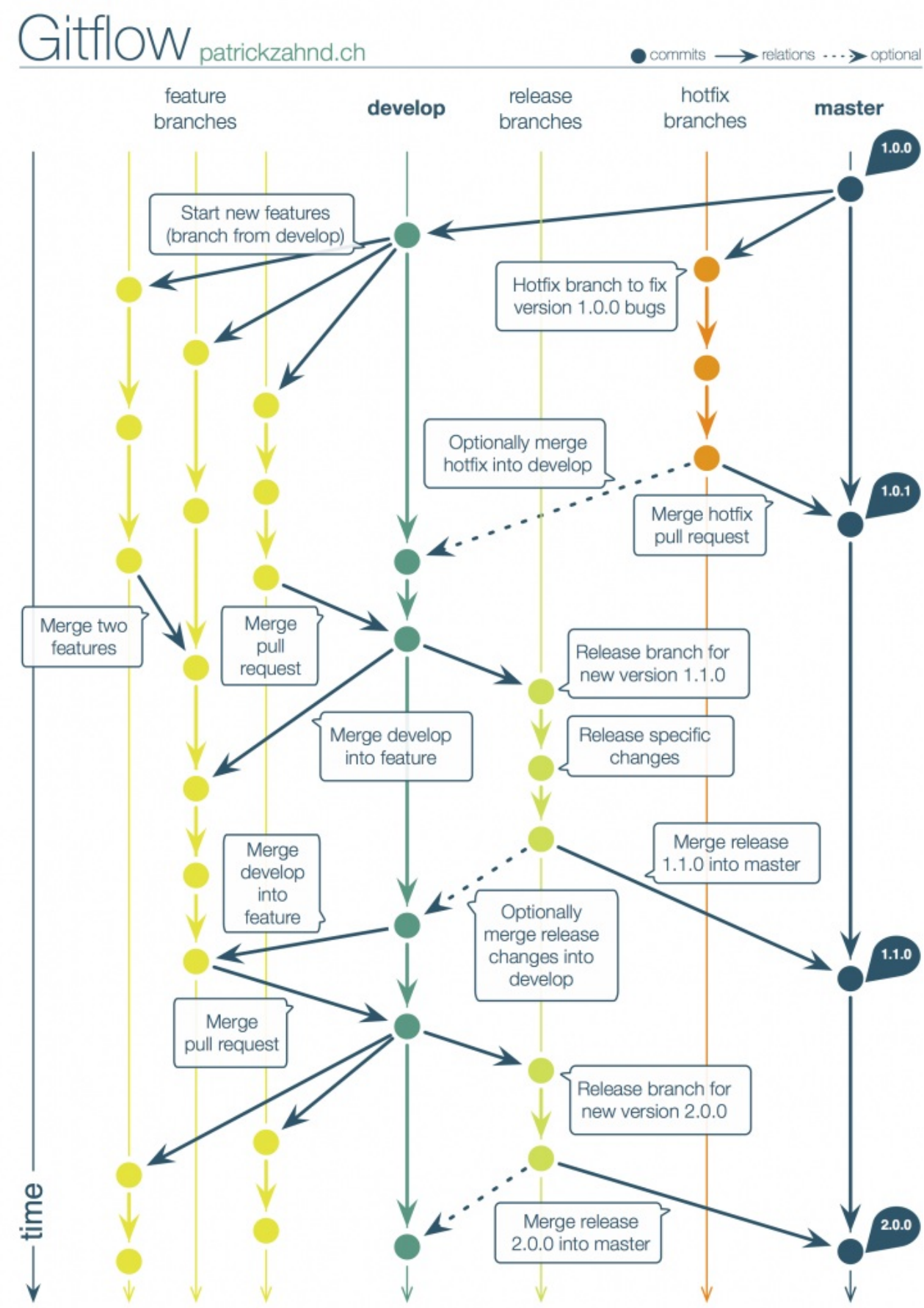
```
git checkout -b hotfix/issue-1
git checkout -b feature/issue-1
```

主要 branch

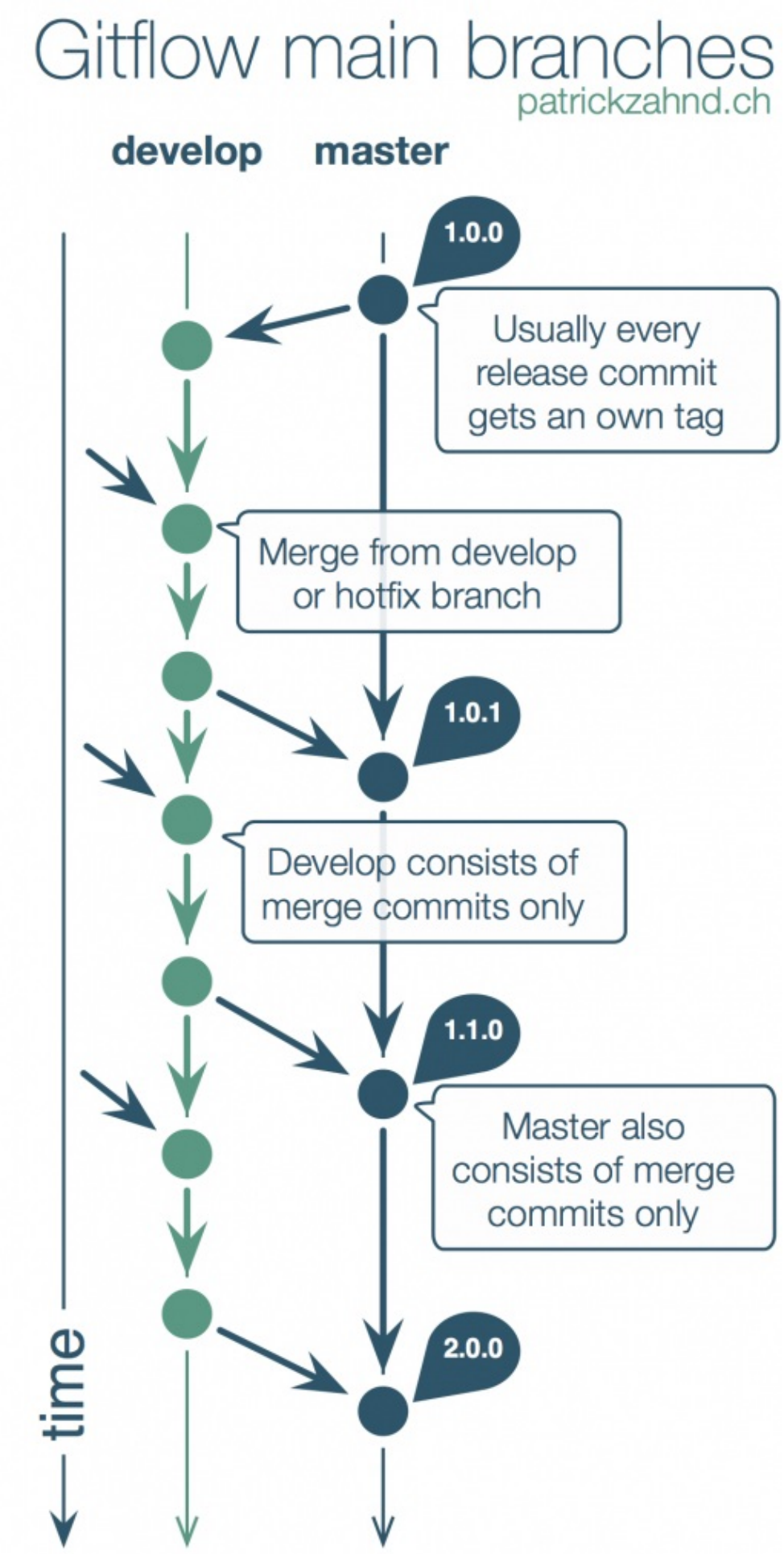
- master: 主要版本，只接受 develop 和 Release 的 merge
- develop: 所有 Feature 開發都從這分支出去，完成後 merge 回來

支援 branch

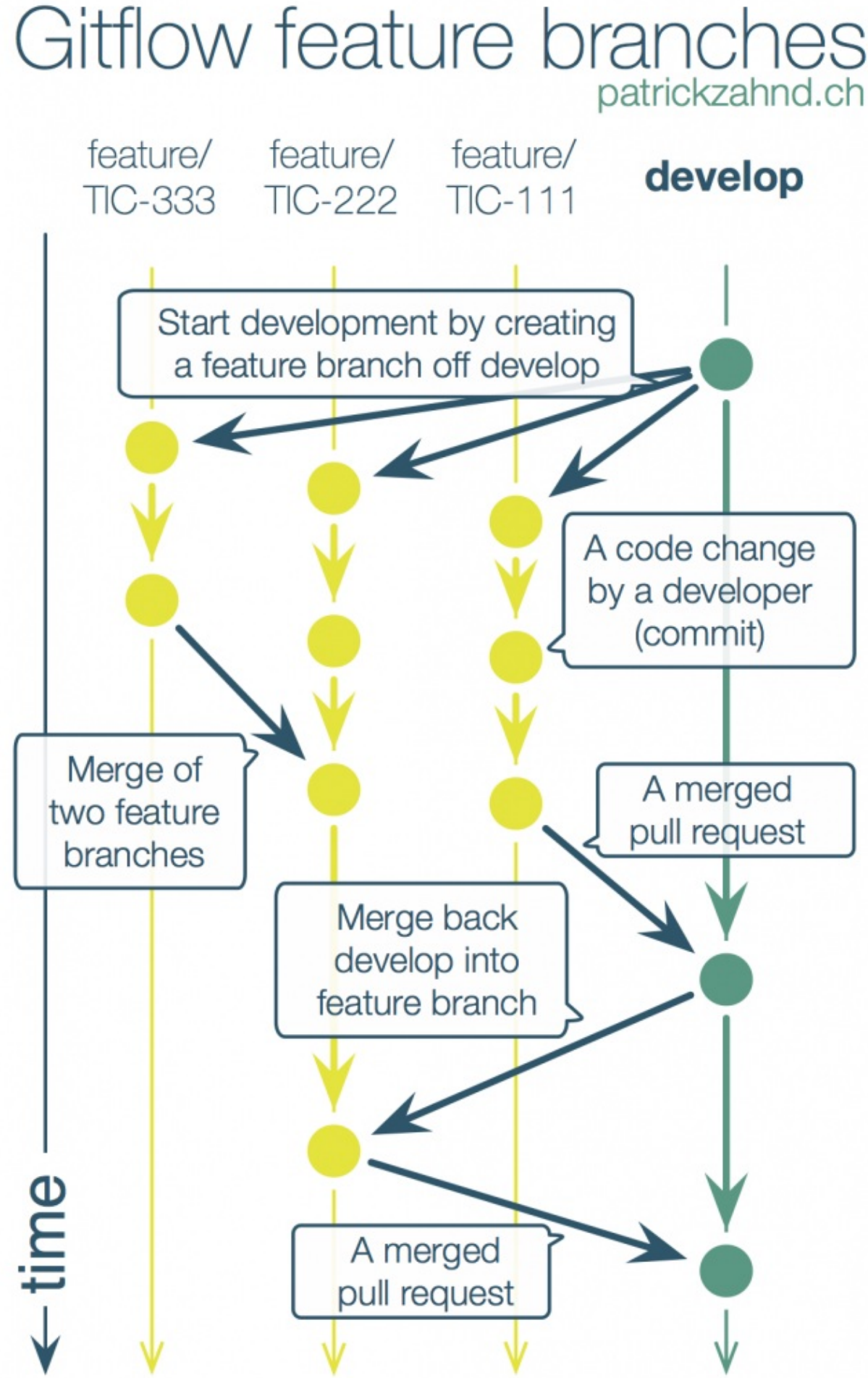
- feature branches：從 develop 分支出來，當功能開發修改完成後 merge 回 develop
- release branches：從 develop 分支出來，是準備釋出的版本，只修改版本號與 bug，完成後 merge 回 develop 與 master，並在 master 標上版本號的 tag
- hotfix branches：從 master 分支出來，主要是處理已釋出版本需要立即修改的錯誤，完成後 merge 回 develop 與 master，並在 master 標上版本號的 tag


Gitflow
patrickzahnd.ch
commits
relations
optional
feature branches
develop
release branches
hotfix branches
master
1.0.0
Start new features (branch from develop)
Hotfix branch to fix version 1.0.0 bugs
Optionally merge hotfix into develop
Merge hotfix pull request
1.0.1
Merge two features
Merge pull request
Release branch for new version 1.1.0
Merge develop into feature
Release specific changes
Merge release 1.1.0 into master
Merge develop into feature
Optionally merge release changes into develop
1.1.0
Merge pull request
Release branch for new version 2.0.0
Merge release 2.0.0 into master
2.0.0
time

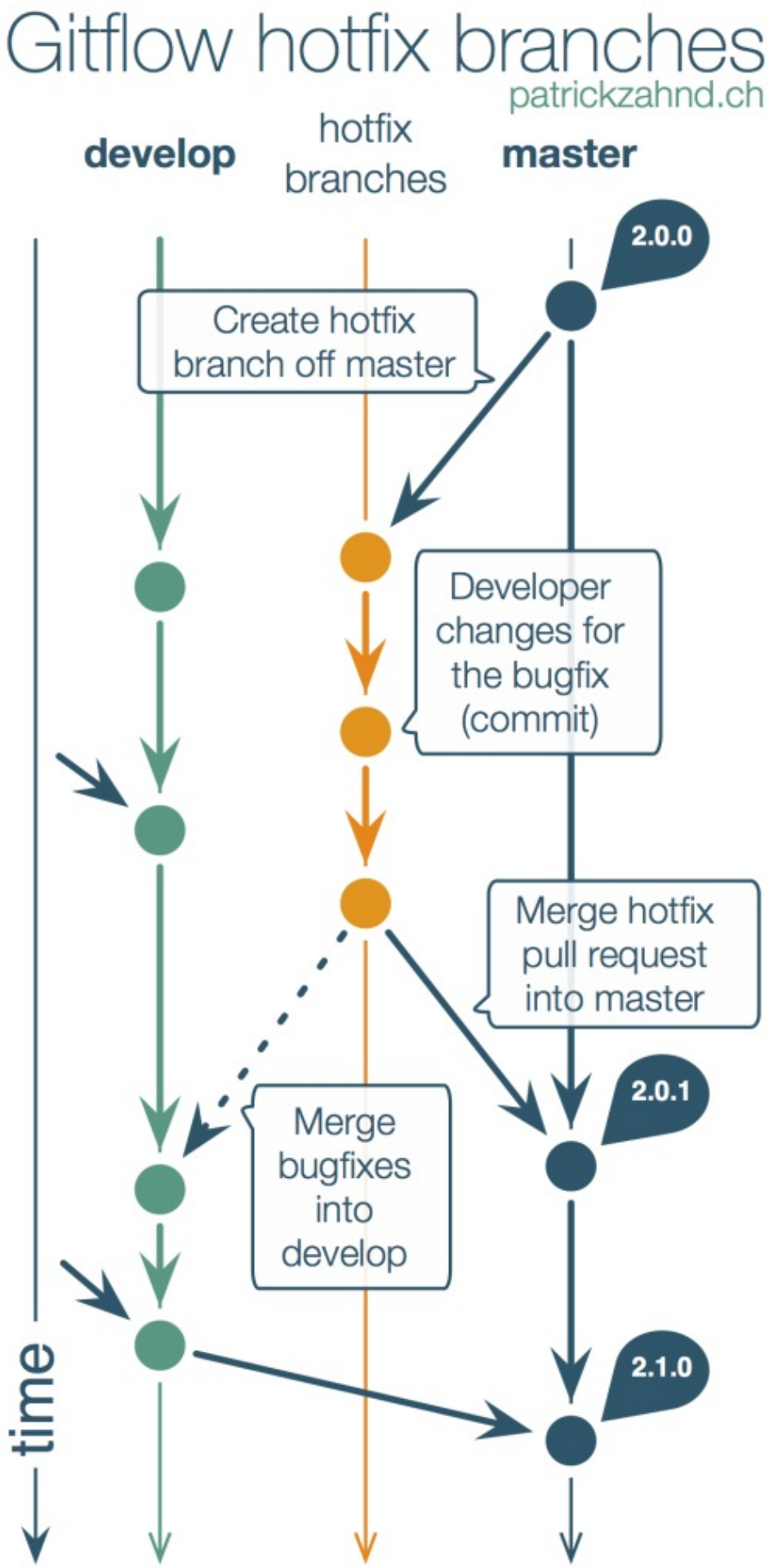
Master 分支


Gitflow main branches
patrickzahnd.ch
develop
master
1.0.0
Usually every release commit gets an own tag
Merge from develop or hotfix branch
1.0.1
Develop consists of merge commits only
1.1.0
Master also consists of merge commits only
2.0.0
time

Feature 分支


Gitflow feature branches
patrickzahnd.ch
feature/ TIC-333
feature/ TIC-222
feature/ TIC-111
develop
Start development by creating a feature branch off develop
A code change by a developer (commit)
Merge of two feature branches
A merged pull request
Merge back develop into feature branch
A merged pull request
time

hotfix 分支


Gitflow hotfix branches
patrickzahnd.ch
develop
hotfix branches
master
2.0.0
Create hotfix branch off master
Developer changes for the bugfix (commit)
Merge hotfix pull request into master
2.0.1
Merge bugfixes into develop
2.1.0
time

git branch CRUD

git branch --all ```

- master remotes/origin/HEAD -> origin/master remotes/origin/master ```

名稱

- 絕對名稱
- 參照名稱

```
refs/head
refs/remotes
refs/tags
```

常用範例

範例	說明
git branch dev	新增 dev 分支
git branch	列出 local 分支
git branch -r	列出遠端分支
git branch -a	列出所有分支
git branch -m dev dev2	修改分支名稱
git branch -d dev2	刪除分支
git branch -D dev2	強迫刪除分支 (即使分支，還沒被 merge 過)

相關聯指令

範例	說明
git push origin dev	發佈 dev 分支

語法結構

```
usage: git branch [<options>] [-r | -a] [--merged | --no-merg
ed]
   or: git branch [<options>] [-l] [-f] <branch-name> [<start
-point>]
   or: git branch [<options>] [-r] (-d | -D) <branch-name>...
   or: git branch [<options>] (-m | -M) [<old-branch>] <new-b
ranch>
   or: git branch [<options>] [-r | -a] [--points-at]
```

git checkout 指令

checkout 可用於將特定版本檔案取出，無論是資料夾或檔案皆可。

使用情境

- 切換分支
- 發現檔案改錯，用git checkout filename把檔案還原到最新的版本

常用指令範例

範例	說明
git checkout README.md	取出 README.md 檔案
git checkout docs/	取出特 docs 資料夾
git checkout dev	取出 dev 分支
git checkout -b dev	新增 dev 分支，並同時切換到 dev 分之上
git checkout — README.md	將 README.md 恢復到上一次 Commit 的狀態

語法結構

```
usage: git checkout [<options>] <branch>
   or: git checkout [<options>] [<branch>] -- <file>...

    -q, --quiet           suppress progress reporting
    -b <branch>           create and checkout a new branch
    -B <branch>           create/reset and checkout a branch
    -l                    create reflog for new branch
    --detach              detach the HEAD at named commit
    -t, --track           set upstream info for new branch
    --orphan <new-branch>
                          new unparented branch
    -2, --ours            checkout our version for unmerged f
iles
    -3, --theirs          checkout their version for unmerged
 files
    -f, --force           force checkout (throw away local mo
difications)
    -m, --merge           perform a 3-way merge with the new
branch
    --overwrite-ignore    update ignored files (default)
    --conflict <style>    conflict style (merge or diff3)
    -p, --patch           select hunks interactively
    --ignore-skip-worktree-bits
                          do not limit pathspecs to sparse en
tries only
    --ignore-other-worktrees
                          do not check if another worktree is
 holding the given ref
    --progress            force progress reporting
```

練習題：新增一個分支

1. 使用 `git branch develop` 指令來建立 `develop` 分支
2. 使用 `git checkout develop` 指令來切換至 `develop` 分支
3. 新增一個叫 `develop.txt` 檔案 `git add develop.txt`
4. 透過 `git commit -m 'add develp.txt file'` 送交 commit
5. 確認已有新的分支產生 `git log --graph --oneline --decorate --all`

git merge 指令

- 合併
- 修改內容的歷史記錄會維持原狀，但是合併後的歷史紀錄會變得更複雜。
- 預設會以 fast-forward 的模式進行。

Fast-Forward Merge

當你目前的位置 (HEAD) 是某個要被 merge branch 上的 commit 的 root commit ，在沒有任何新的 commit 的情形下，要 merge 回來的時候，就會觸發 fast-forward 。

由於在原本的分支上沒有新的變更紀錄需要被 merged ，預設就會觸發 fast-forward merge ，就會直接把這個 commit HEAD 移動到要被 merge 的 commit 的位置，也不會新增一個 merge commit 。

merge 之前

```
* 1ea846c (HEAD -> dev) 4
* 096a417 3
* 3c20c25 (master) 2
* 72d8959 1
* 337f2b6 Initial Commit
```

merge 之後

```
user@ubuntu-xenial:~/workspace/git-sample/merge-sandbox$ git merge dev
Updating 3c20c25..1ea846c
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
user@ubuntu-xenial:~/workspace/git-sample/merge-sandbox$ git log --graph --oneline --decorate --all
* 1ea846c (HEAD -> master, dev) 4
* 096a417 3
* 3c20c25 2
* 72d8959 1
* 337f2b6 Initial Commit
```

No Fast-Forward Merge

```
git merge dev —-no-ff
```

```
user@ubuntu-xenial:~/workspace/git-sample/merge-sandbox$ git merge dev --no-ff
Merge made by the 'recursive' strategy.
 README.md | 2 ++
 1 file changed, 2 insertions(+)
user@ubuntu-xenial:~/workspace/git-sample/merge-sandbox$ git log --graph --oneline --decorate --all
*   d8044b5 (HEAD -> master) Merge branch 'dev'
|\
| * 8ccde2d (dev) 4
| * 60d5637 3
|/
* fb7f269 2
* aef5b93 1
* e7de681 Initial Commit
```

使用情境

- 在開發一個功能時，通常都會開一支新的分支，使用 --no-ff 可以讓成員在日後可以很清楚辨識不同的功能，所包含的送交歷史紀錄有哪些。
- GitHub merge pull request 的策略，也是使用 --no-ff ，也是讓開發者可以方便辨別。

缺點

- 即使一個很小的修改，也會拉出一條支線圖。

常用範例

範例	說明
git merge master	合併 master 分支的資料，到目前分支
git merge –abort	放棄合併

實際操作流程

```
git pull
git log --graph --oneline --decorate --all
git diff master
git merge master
```

語法結構

```
usage: git merge [<options>] [<commit>...]
   or: git merge [<options>] <msg> HEAD <commit>
   or: git merge --abort

    -n                    do not show a diffstat at the end o
f the merge
    --stat                show a diffstat at the end of the m
erge
    --summary             (synonym to --stat)
    --log[=<n>]           add (at most <n>) entries from shor
tlog to merge commit message
    --squash              create a single commit instead of d
oing a merge
    --commit              perform a commit if the merge succe
eds (default)
    -e, --edit            edit message before committing
    --ff                  allow fast-forward (default)
    --ff-only             abort if fast-forward is not possib
le
    --rerere-autoupdate   update the index with reused confli
ct resolution if possible
    --verify-signatures   Verify that the named commit has a
valid GPG signature
    -s, --strategy <strategy>
                          merge strategy to use
    -X, --strategy-option <option=value>
                          option for selected merge strategy
    -m, --message <message>
                          merge commit message (for a non-fas
t-forward merge)
    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --abort               abort the current in-progress merge
    --progress            force progress reporting
    -S, --gpg-sign[=<key-id>]
```

```
                              GPG sign commit
      --overwrite-ignore      update ignored files (default)
```

觀念解說：解決衝突 (conflict)

如果遠端 repo 跟 本地端 repo 在同個地方(同行)都發生修改的情況下，這時會導致 Git 無法自動判斷，那一份才是正確的，於是發出衝突緊訊告知使用者需要手動排解衝突。

發生衝突時，執行 `git diff` 可以看到類似。

```
<<<<<<< HEAD
123 456

hello world
=======
123 789

hello world
>>>>>>> dev
```

練習題：解決衝突

情境設定

建立一個裸容器

```
cd ~/workspace/git-sample
mkdir hello.git
cd hello.git
git init --bare
```

模擬 john 的行為

```
cd ..
git clone ~/workspace/git-sample/hello.git john-hello
cd ~/workspace/git-sample/john-hello
git config user.email "john@demo.com"
git config user.name "john"
echo "123" << hello.txt
git add .
git commit -m 'create hello file'
```

模擬 lily 的行為

```
cd ~/workspace/git-sample/
git clone ~/workspace/git-sample/hello.git lily-hello
cd ~/workspace/git-sample/lily-hello
git config user.email "lily@demo.com"
git config user.name "lily"
```

john 編輯檔案

vi hello.txt

```
123 456
```

```
git add .
git commit -m 'add 456'
git push -u origin master
```

lily 編輯檔案

vi hello.txt

```
123 789
```

```
git add .
git commit -m 'add 789'
git push -u origin master
```

解決衝突

```
git pull
git diff
# vi 編輯衝突
git add .
git commit
git push -u origin master
```

git stash 指令

暫存

使用情境

- 通常用於臨時必須要切換到其他分支，但現在工作做到一半還不能 commit 時，就先把修改存起來，切到其他分支事情做完之後，再回來把修改取出來，繼續剛剛未完成的工作。

常用範例

範例	說明
git stash	暫時儲存當前目錄
git stash list	列出所有暫存區的資料
git stash pop	取出最新的一筆, 並移除
git stash clear	把 stash 都清掉
git stash drop	清掉單一筆暫存

語法結構

```
usage: git stash list [<options>]
   or: git stash show [<stash>]
   or: git stash drop [-q|--quiet] [<stash>]
   or: git stash ( pop | apply ) [--index] [-q|--quiet] [<sta
sh>]
   or: git stash branch <branchname> [<stash>]
   or: git stash [save [--patch] [-k|--[no-]keep-index] [-q|-
-quiet]
                         [-u|--include-untracked] [-a|--all] [<
message>]]
   or: git stash clear
```

練習題：將檔案放入暫存區，並模擬 git flow 流程。

1. 透過 `git checkout develop`
2. 透過 `git checkout -b feature/issue-1`
3. 透過 `echo "123" >> feature.md` 指令，新增一個檔案
4. 透過 `git add .` 指令，將檔案建立好索引
5. 透過 `git stash` 指令，將檔案暫時放置在暫存區
6. 透過 `git checkout master`
7. 透過 `git checkout -b hotfix/issue-2`
8. 透過 `echo "123" >> hotfix.md` 指令，新增一個檔案
9. 透過 `git commit -m 'I am hotfix'` 指令，新增一個送交紀錄
10. 透過 `git checkout feature/issue-1` 指令，回到 `feature/issue-1` 分支。
11. 透過 `git stash pop` 指令，將存擋取出。
12. 透過 `git add .` 指令，將檔案建立好索引
13. 透過 `git commit -m 'I am feature'` 指令，新增一個送交紀錄

git reset 指令

- --sort: 轉變為 uncommit 狀態
- --hard: 檔案會消失

使用情境

- 修正還未發佈的送交紀錄
- 將新增並已加入索引的檔案，還原到沒加入索引之前。

常用範例

範例	說明
git reset --sort HEAD	
git reset —-hard	回覆到沒修改的狀態

語法結構

```
usage: git reset [--mixed | --soft | --hard | --merge | --kee
p] [-q] [<commit>]
   or: git reset [-q] <tree-ish> [--] <paths>...
   or: git reset --patch [<tree-ish>] [--] [<paths>...]

    -q, --quiet           be quiet, only report errors
    --mixed               reset HEAD and index
    --soft                reset only HEAD
    --hard                reset HEAD, index and working tree
    --merge               reset HEAD, index and working tree
    --keep                reset HEAD but keep local changes
    -p, --patch           select hunks interactively
    -N, --intent-to-add   record only the fact that removed p
aths will be added later
```

git revert 指令

使用情境

- 修正已經發佈的送交紀錄，新增一個反向的 rollback 送交紀錄。

常用範例

範例	說明
git revert HEAD	回到前一次 commit 的狀態

語法結構

```
usage: git revert [<options>] <commit-ish>...
   or: git revert <subcommand>

    --quit                end revert or cherry-pick sequence
    --continue            resume revert or cherry-pick sequen
ce
    --abort               cancel revert or cherry-pick sequen
ce
    -n, --no-commit       don't automatically commit
    -e, --edit            edit the commit message
    -s, --signoff         add Signed-off-by:
    -m, --mainline <n>    parent number
    --rerere-autoupdate   update the index with reused confli
ct resolution if possible
    --strategy <strategy>
                          merge strategy
    -X, --strategy-option <option>
                          option for merge strategy
    -S, --gpg-sign[=<key-id>]
                          GPG sign commit
```

git rebase 指令 (須謹慎使用)

- 重新指定位置
- 修改內容的歷史記錄會接在要合併的分支後面，合併後的歷史記錄會比較清楚簡單，但是會比使用 merge 更容易發生衝突。

使用情境

- 可進行重新排序、編輯、移除、擠壓、拆散提交
- 僅適用於還沒發佈的送交紀錄

優點

可以不產生分支線和額外的 merge commit

缺點

- 等於改變提交記錄，僅適合修改還沒發佈的本機端送交紀錄
- 移動的commit若屬於多個分支，則每個分支都要重新指定位置。
- 經常無法自動合併成功的情況，一直得手動排除衝突，過度追求線圖的清晰，反而失去 Git 的部分優點。

常用範例

範例	說明
git rebase master	
git rebase –abort	如果執行 rebase 指令後出現衝突的情況，可以使用這個指令取消 rebase 的操作。 Git repo 會恢復到還沒有執行 rebase 之前的狀態。
git rebase –continue	執行rebase 指令後出現衝突的情況，而且我們已經編輯好發生衝突的文件，接著就可以執行 `git add` 指令， 把新的修改內容加入 Git 索引中，最後再執行這個指令，完成rebase的操作。

語法結構

```
usage: git rebase [-i] [options] [--exec <cmd>] [--onto <newb
ase>] [<upstream>] [<branch>]
   or: git rebase [-i] [options] [--exec <cmd>] [--onto <newb
ase>] --root [<branch>]
   or: git-rebase --continue | --abort | --skip | --edit-todo

Available options are
    -v, --verbose         display a diffstat of what changed
upstream
    -q, --quiet           be quiet. implies --no-stat
    --autostash           automatically stash/stash pop befor
e and after
    --fork-point          use 'merge-base --fork-point' to re
fine upstream
    --onto ...            rebase onto given branch instead of
 upstream
    -p, --preserve-merges
                          try to recreate merges instead of i
gnoring them
    -s, --strategy ...    use the given merge strategy
    --no-ff               cherry-pick all commits, even if un
changed
```

```
    -m, --merge           use merging strategies to rebase
    -i, --interactive     let the user edit the list of commi
ts to rebase
    -x, --exec ...        add exec lines after each commit of
 the editable list
    -k, --keep-empty         preserve empty commits during re
base
    -f, --force-rebase    force rebase even if branch is up t
o date
    -X, --strategy-option ...
                          pass the argument through to the me
rge strategy
    --stat                display a diffstat of what changed
upstream
    -n, --no-stat         do not show diffstat of what change
d upstream
    --verify              allow pre-rebase hook to run
    --rerere-autoupdate   allow rerere to update index with r
esolved conflicts
    --root                rebase all reachable commits up to
the root(s)
    --autosquash         move commits that begin with squash
                          move commits that begin with squash
!/fixup! under -i
    --committer-date-is-author-date
                          passed to 'git am'
    --ignore-date         passed to 'git am'
    --whitespace ...      passed to 'git apply'
    --ignore-whitespace   passed to 'git apply'
    -C ...                passed to 'git apply'
    -S, --gpg-sign[=...]  GPG-sign commits

Actions:
    --continue            continue
    --abort               abort and check out the original br
anch
    --skip                skip current patch and continue
    --edit-todo           edit the todo list during an intera
```

```
ctive rebase
```

Reset vs Revert

- Reset：直接修改舊的送交歷史紀錄，不會多增加一個節點。
- Revert：回到指定的 commit 節點的前一個節點的狀態，執行完畢後會新增一個 commit 節點。

練習題：還原狀態

1. 編輯 README.md
2. 透過 `git checkout -- README.md` 指令，還原修改的內容
3. 編輯 README.md
4. 透過 `git add .` 指令，將修改的內容加入索引。
5. 透過 `git status` 指令，查看目前檔案狀態。
6. 透過 `git reset HEAD README.md` 指令，將檔案變成 Unstaged (移除索引)。

容器

容器的概念

遠端容器

- 分享程式碼的第一步，你必須要將自己的容器與其他的容器，建立關聯性才可以交換資料。
- 遠端容器是一個參照到另一個容器的分式。
- 一個容器中可以定義任何數量的遠端容器。

管理容器

當你發布後，任何一個使用者都可能會複製容器，你應該將它視為一個靜態的容器，並且避免「複寫」任何分支的歷史紀錄。

git remote 指令

常用範例

範例	說明
git remote	顯示遠端來源名稱
git remote -v	顯示遠端來源詳細資訊
git remote add origin http://git.xx.com.tw/project.git	加入遠端來源
git remote rename origin origin2	修改遠端來源的名字
git remote rm origin2	刪除叫 origin2 的遠端連接

語法結構

```
usage: git remote [-v | --verbose]
   or: git remote add [-t <branch>] [-m <master>] [-f] [--tag
s | --no-tags] [--mirror=<fetch|push>] <name> <url>
   or: git remote rename <old> <new>
   or: git remote remove <name>
   or: git remote set-head <name> (-a | --auto | -d | --delet
e | <branch>)
   or: git remote [-v | --verbose] show [-n] <name>
   or: git remote prune [-n | --dry-run] <name>
   or: git remote [-v | --verbose] update [-p | --prune] [(<g
roup> | <remote>)...]
   or: git remote set-branches [--add] <name> <branch>...
   or: git remote get-url [--push] [--all] <name>
   or: git remote set-url [--push] <name> <newurl> [<oldurl>]
   or: git remote set-url --add <name> <newurl>
   or: git remote set-url --delete <name> <url>

    -v, --verbose         be verbose; must be placed before a
 subcommand
```

git clone 指令

複製一份遠端專案至本地端

常用範例

範例	說明
git clone https://github.com/alincode/git-workshop	複製遠端容器
git clone ~/remote-repo alincode-repo	複製遠端容器，並指定資料夾名稱
git clone ~/remote-repo	複製遠端容器

語法結構

```
usage: git clone [<options>] [--] <repo> [<dir>]

    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --progress            force progress reporting
    -n, --no-checkout     don't create a checkout
    --bare                create a bare repository
    --mirror              create a mirror repository (implies
 bare)
    -l, --local           to clone from a local repository
    --no-hardlinks        don't use local hardlinks, always c
opy
    -s, --shared          setup as shared repository
    --recursive           initialize submodules in the clone
    --recurse-submodules  initialize submodules in the clone
    --template <template-directory>
                          directory from which templates will
 be used
    --reference <repo>    reference repository
    --dissociate          use --reference only while cloning
    -o, --origin <name>   use <name> instead of 'origin' to t
rack upstream
    -b, --branch <branch>
                          checkout <branch> instead of the re
mote's HEAD
    -u, --upload-pack <path>
                          path to git-upload-pack on the remo
te
    --depth <depth>       create a shallow clone of that dept
h
    --single-branch       clone only one branch, HEAD or --br
anch
    --separate-git-dir <gitdir>
                          separate git dir from working tree
    -c, --config <key=value>
                          set config inside the new repositor
y
```

git pull 指令

拉回遠端 repo 最新版送交歷史紀錄到本地端 repo

- clone vs pull
- git pull = git fetch + git merge

常用範例

範例	說明
git pull	將遠端的資料更新到本地端，並合併

語法結構

```
usage: git pull [<options>] [<repository> [<refspec>...]]

    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --progress            force progress reporting

Options related to merging
    -r, --rebase[=<false|true|preserve>]
                          incorporate changes by rebasing rat
her than merging
    -n                    do not show a diffstat at the end o
f the merge
    --stat                show a diffstat at the end of the m
erge
    --log[=<n>]           add (at most <n>) entries from shor
tlog to merge commit message
    --squash              create a single commit instead of d
oing a merge
    --commit              perform a commit if the merge succe
```

```
eds (default)
    --edit                edit message before committing
    --ff                  allow fast-forward
    --ff-only             abort if fast-forward is not possib
le
    --verify-signatures   verify that the named commit has a
valid GPG signature
    -s, --strategy <strategy>
                          merge strategy to use
    -X, --strategy-option <option=value>
                          option for selected merge strategy
    -S, --gpg-sign[=<key-id>]
                          GPG sign commit

Options related to fetching
    --all                 fetch from all remotes
    -a, --append          append to .git/FETCH_HEAD instead o
f overwriting
    --upload-pack <path>  path to upload pack on remote end
    -f, --force           force overwrite of local branch
    -t, --tags            fetch all tags and associated objec
ts
    -p, --prune           prune remote-tracking branches no l
onger on remote
    --recurse-submodules[=<on-demand>]
                          control recursive fetching of submo
dules
    --dry-run             dry run
    -k, --keep            keep downloaded pack
    --depth <depth>       deepen history of shallow clone
    --unshallow           convert to a complete repository
    --update-shallow      accept refs that update .git/shallo
w
    --refmap <refmap>     specify fetch refmap
```

git fetch 指令

執行 fetch，可以取得遠端數據庫的最新歷史記錄。取得的提交會導入在自動建立的分支中，並可以切換這個名為 FETCH_HEAD 的分支。

使用情境

執行 pull，遠端數據庫的內容會自動合併。但是，有時候只是想確認遠端數據庫的內容卻不是真的想合併，在這種情況下，請使用 fetch。

常用指令範例

範例	說明
git fetch	更新所有遠端容器
git fetch origin	更新指定遠端容器

語法結構

```
usage: git fetch [<options>] [<repository> [<refspec>...]]
   or: git fetch [<options>] <group>
   or: git fetch --multiple [<options>] [(<repository> | <gro
up>)...]
   or: git fetch --all [<options>]

    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --all                 fetch from all remotes
    -a, --append          append to .git/FETCH_HEAD instead o
f overwriting
    --upload-pack <path>  path to upload pack on remote end
    -f, --force           force overwrite of local branch
    -m, --multiple        fetch from multiple remotes
    -t, --tags            fetch all tags and associated objec
ts
    -n                    do not fetch all tags (--no-tags)
    -p, --prune           prune remote-tracking branches no l
onger on remote
    --recurse-submodules[=<on-demand>]
                          control recursive fetching of submo
dules
    --dry-run             dry run
    -k, --keep            keep downloaded pack
    -u, --update-head-ok  allow updating of HEAD ref
    --progress            force progress reporting
    --depth <depth>       deepen history of shallow clone
    --unshallow           convert to a complete repository
    --update-shallow      accept refs that update .git/shallo
w
    --refmap <refmap>     specify fetch refmap
```

練習題：透過 fetch 指令取得最新送交紀錄

前置環境

```
git clone https://github.com/agileworks-tw/git-hello-world
```

講師操作

```
vi README.md
git add .
git commit -m 'add my name'
git push
```

學員操作

```
git fetch origin
git log --graph --oneline --decorate --all
git merge origin/master
git log --graph --oneline --decorate --all
```

git push 指令

使用情境

- 將本地端新增的送交紀錄，發佈至遠端

常用範例

範例	說明
git push origin master -u	從本地端的 master branch 上傳到 origin 的 master branch
git push origin master	發佈 master
git push origin v0.0.1	發佈標籤

語法結構

```
usage: git push [<options>] [<repository> [<refspec>...]]

    -v, --verbose         be more verbose
    -q, --quiet           be more quiet
    --repo <repository>   repository
    --all                 push all refs
    --mirror              mirror all refs
    --delete              delete refs
    --tags                push tags (can't be used with --all
 or --mirror)
    -n, --dry-run         dry run
    --porcelain           machine-readable output
    -f, --force           force updates
    --force-with-lease[=<refname>:<expect>]
                          require old value of ref to be at t
his value
    --recurse-submodules[=<check|on-demand|no>]
                          control recursive pushing of submod
ules
    --thin                use thin pack
    --receive-pack <receive-pack>
                          receive pack program
    --exec <receive-pack>
                          receive pack program
    -u, --set-upstream    set upstream for git pull/status
    --progress            force progress reporting
    --prune               prune locally removed refs
    --no-verify           bypass pre-push hook
    --follow-tags         push missing but relevant tags
    --signed[=<yes|no|if-asked>]
                          GPG sign the push
    --atomic              request atomic transaction on remot
e side
```

練習題：發佈送交

1. `git init`
2. `git add .`
3. `git commit -m 'update'`
4. `git remote add origin https://github.com/username/git-sandbox.git`
5. `git push -u origin https://github.com/username/git-sandbox.git`
6. 確認結果

git tag 指令

標籤

使用情境

- 里程碑管理
- 準備發佈一個穩定版本，給予容易識別的版本命名

常用範例

範例	說明
git tag	
git rev-parse v0.0.1	
git tag v0.0.1 sha1234	新增 tag
git tag -d v0.0.1	刪除 tag

```
$ git rev-parse v0.0.1
d93a61550216134ea18ff5db907f066fd8beb866
```

```
$ git cat-file -p d93a61550216134ea18ff5db907f066fd8beb866

tree cbfd7a8e6c7bd0eec9f524b42bdce9ff31bac2e3
parent 59da3738f9ccf00e1246098a65259b713e848926
author alincode <alincode@gmail.com> 1526267514 +0000
committer alincode <alincode@gmail.com> 1526267514 +0000

add hello2 file
```

語法結構

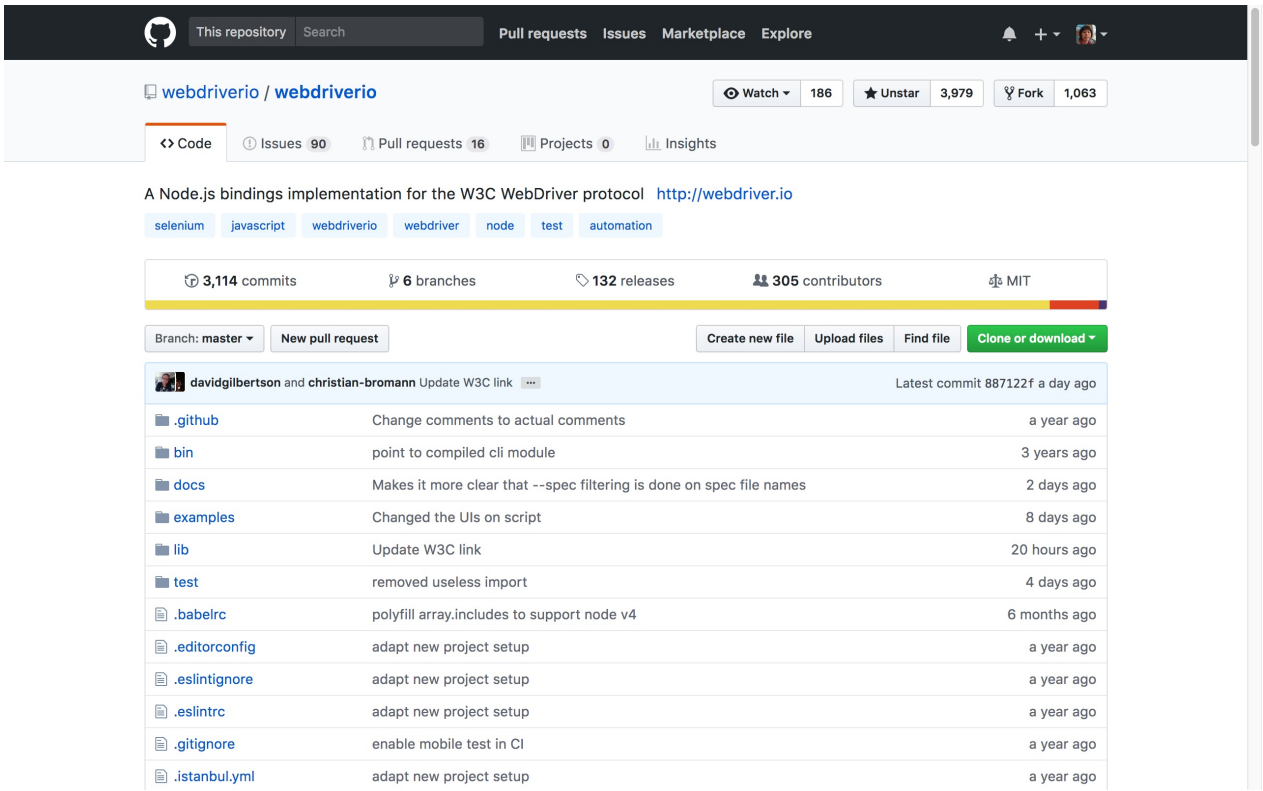
```
usage: git tag [-a | -s | -u <key-id>] [-f] [-m <msg> | -F <f
ile>] <tagname> [<head>]
   or: git tag -d <tagname>...
   or: git tag -l [-n[<num>]] [--contains <commit>] [--points
-at <object>]
                [--format=<format>] [--[no-]merged [<commit>]
] [<pattern>...]
   or: git tag -v <tagname>...

    -l, --list            list tag names
    -n[<n>]               print <n> lines of each tag message
    -d, --delete          delete tags
    -v, --verify          verify tags

Tag creation options
    -a, --annotate        annotated tag, needs a message
    -m, --message <message>
                          tag message
    -F, --file <file>     read message from file
    -s, --sign            annotated and GPG-signed tag
    --cleanup <mode>      how to strip spaces and #comments f
rom message
    -u, --local-user <key-id>
                          use another key to sign the tag
    -f, --force           replace the tag if exists
    --create-reflog       create a reflog

Tag listing options
    --column[=<style>]    show tag list in columns
    --contains <commit>   print only tags that contain the co
mmit
    --merged <commit>     print only tags that are merged
    --no-merged <commit>  print only tags that are not merged
    --sort <key>          field name to sort on
    --points-at <object>  print only tags of the object
    --format <format>     format to use for the output
```

練習題：發佈 tag

1. 透過 `git tag v0.01` 指令，建立標籤
2. 透過 `git tag` 指令，列出所有標籤
3. 透過 `git tag -d v0.01` 指令，刪除標籤
4. 透過 `git tag v0.0.1 -m '新增了一個偉大的功能'` 指令，建立標籤
5. 透過 `git show v0.0.1` 指令，查看標籤詳細
6. 透過 `git tag v0.0.1` 指令，建立標籤
7. 透過 `git push origin v0.0.1` 指令，發佈標籤

Github

https://github.com/webdriverio/webdriverio



watch 追蹤

- 會收到更變通知
  - 郵件
  - 快訊

star 給一個星星

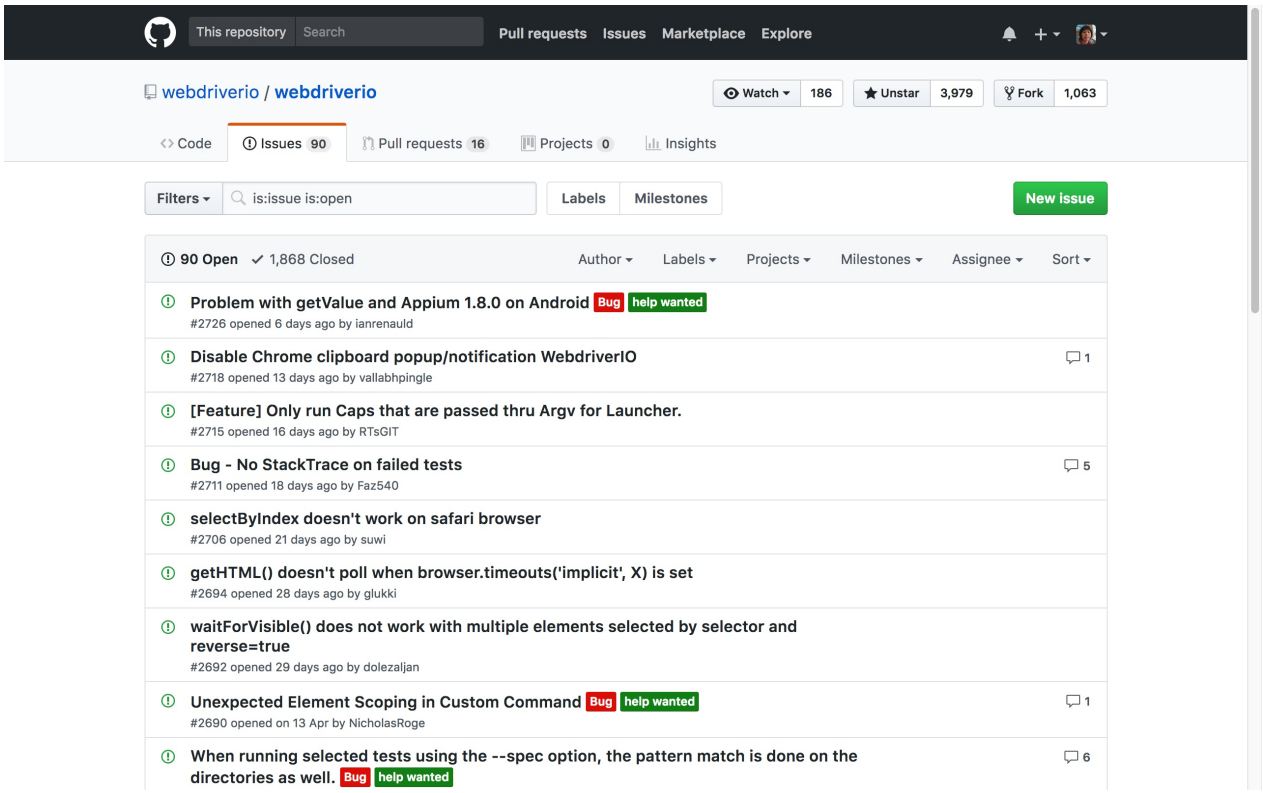
例如像 facebook 按「讚」

Fork

fork 是發生在「整個容器」的階層上，branch 是發生在「單一容器」中。

Github issue

查看 issue 清單



新增一個 issue

This repository Search Pull requests Issues Marketplace Explore

webdriverio / webdriverio

Watch 186 Unstar 3,979 Fork 1,063

Code Issues 90 Pull requests 16 Projects 0 Insights

Title

Write Preview

The problem

Briefly describe the issue you are experiencing (or the feature you want to see added to WebdriverIO). Tell us what you were trying to do and what happened instead. Remember, this is _not_ a place to ask questions. For that, join the Gitter chat room ([![Gitter](https://badges.gitter.im/Join Chat.svg)](https://gitter.im/webdriverio/webdriverio?utm_source=badge&utm_medium=badge&utm_campaign=pr-badge&utm_content=badge))!

Environment

* WebdriverIO version:
* Node.js version:
* [Standalone mode or wdio testrunner](http://webdriver.io/guide/getstarted/modes.html):
* if wdio testrunner, running synchronous or asynchronous tests:
* Additional wdio packages used (if applicable):

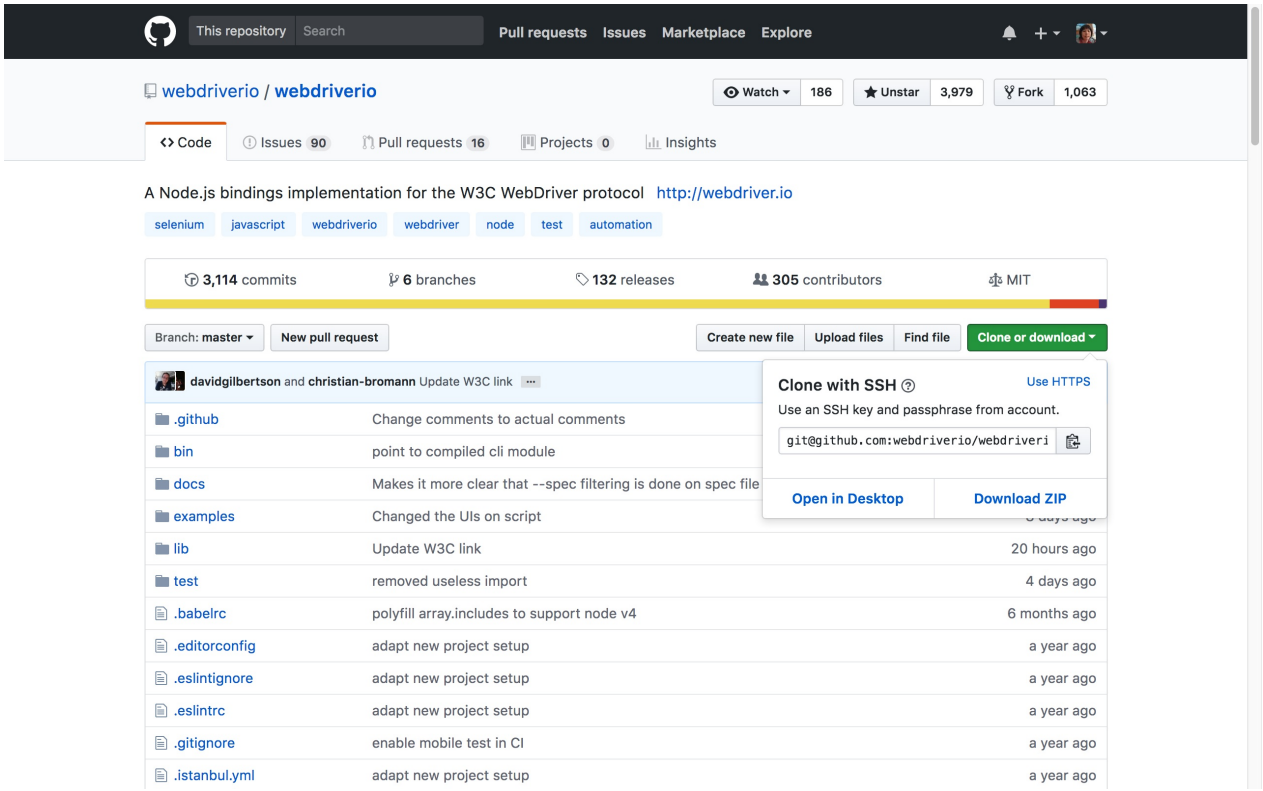
Details

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Submit new issue

Github clone

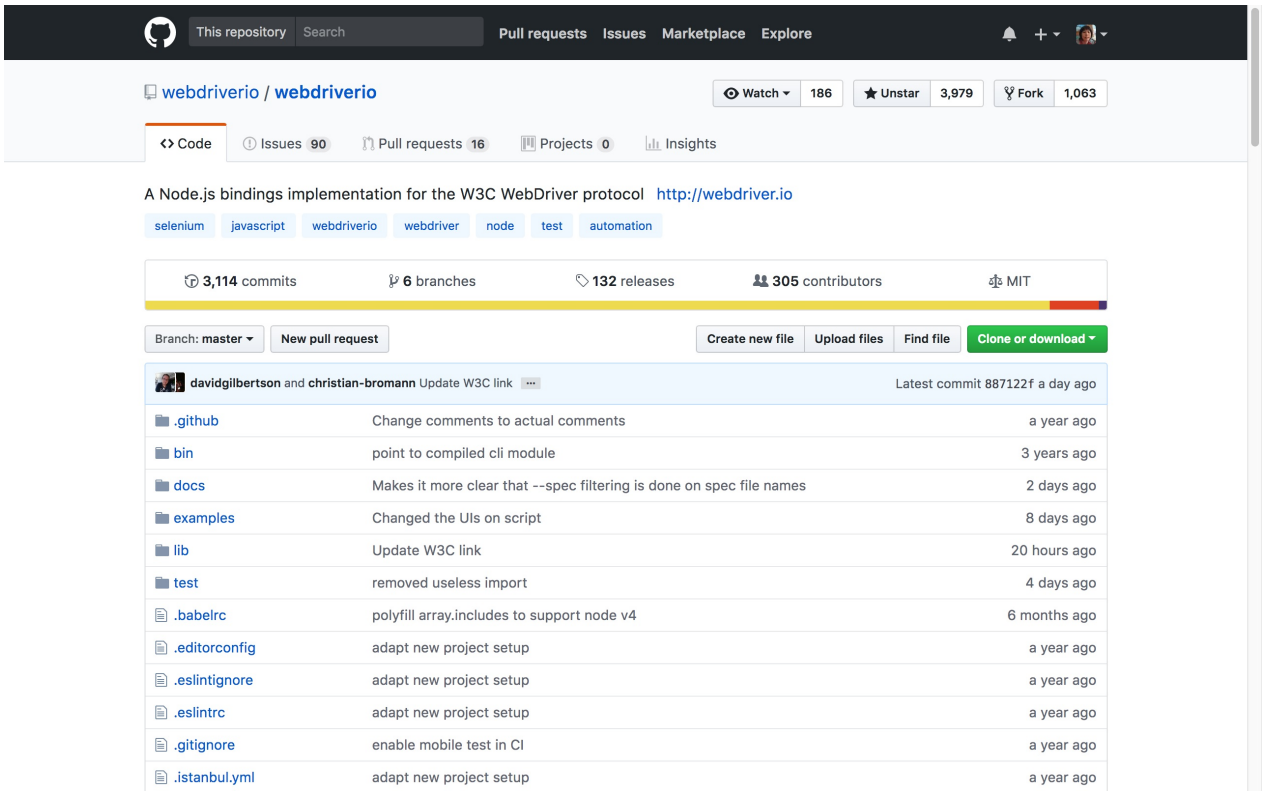



```
git clone git@github.com:webdriverio/webdriverio.git
// or
git clone https://github.com/webdriverio/webdriverio.git
```

Github Fork

使用情境

- 常發生在長期無人維護的專案，但你需要使用並修改。
- 你需要用某個專案當基底，開發出一條完全新的路線。



This repository Search Pull requests Issues Marketplace Explore

alincode / webdriverio
forked from webdriverio/webdriverio

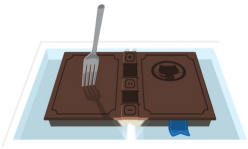
Unwatch 1 Star 0 Fork 1,064

Code Pull requests 0 Projects 0 Insights Settings

Forking webdriverio/webdriverio

It should only take a few seconds.

Refresh



© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

This repository Search Pull requests Issues Marketplace Explore

alincode / webdriverio
forked from webdriverio/webdriverio

Unwatch 1 Star 0 Fork 1,064

Code Pull requests 0 Projects 0 Insights Settings

A Node.js bindings implementation for the W3C WebDriver protocol http://webdriver.io

Edit

Add topics

3,114 commits 6 branches 132 releases 305 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

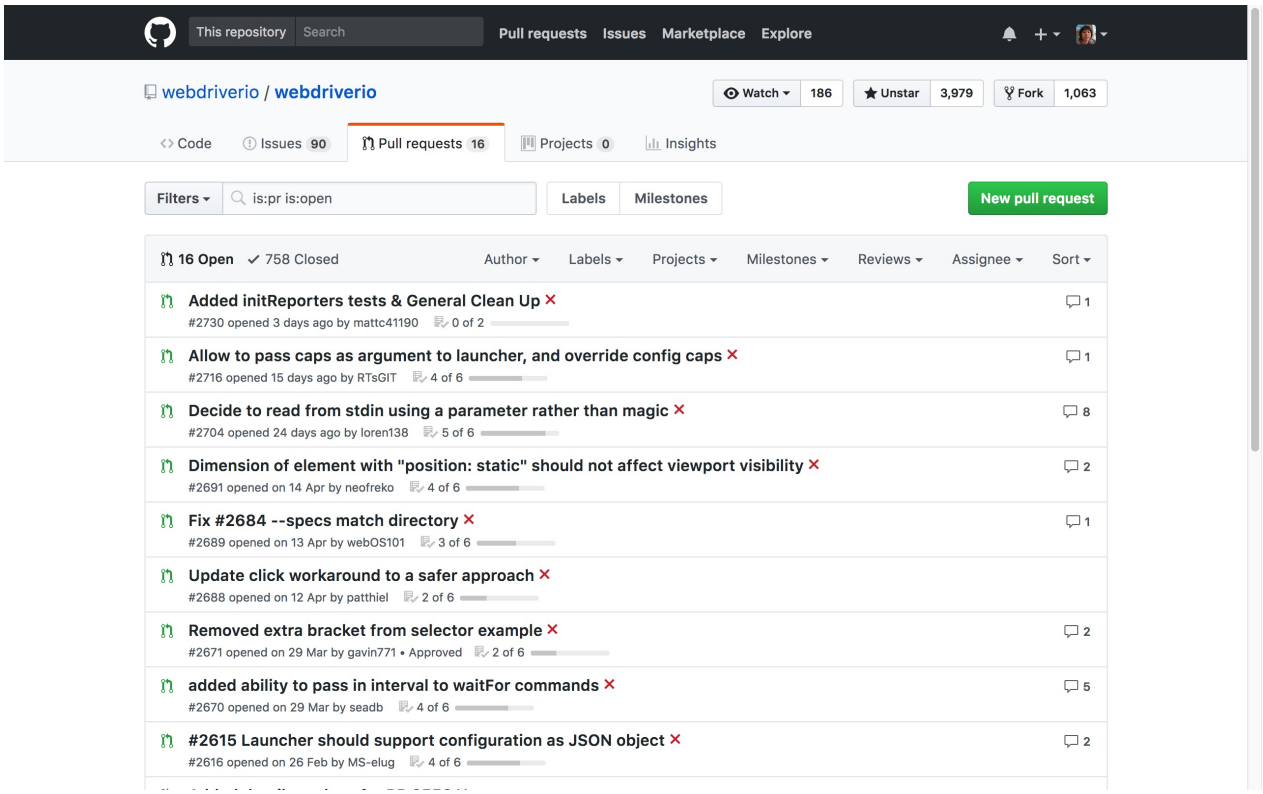
This branch is even with webdriverio:master. Pull request Compare

davidgilbertson and christian-bromann Update W3C link Latest commit 887122f a day ago

.github	Change comments to actual comments	a year ago
bin	point to compiled cli module	3 years ago
docs	Makes it more clear that --spec filtering is done on spec file names	2 days ago
examples	Changed the UIs on script	8 days ago
lib	Update W3C link	20 hours ago
test	removed useless import	4 days ago
.babelrc	polyfill array.includes to support node v4	6 months ago
.editorconfig	adapt new project setup	a year ago
.eslintignore	adapt new project setup	a year ago
.eslintrc	adapt new project setup	a year ago
.gitignore	enable mobile test in CI	a year ago

Github Pull Request

查看 PR



新增一個 PR

This repository Search Pull requests Issues Marketplace Explore

webdriverio / webdriverio

Watch 186 Unstar 3,979 Fork 1,063

Code Issues 90 Pull requests 16 Projects 0 Insights

Compare changes

Compare changes across branches, commits, tags, and more below. If you need to, you can also compare across forks.

base fork: webdriverio/webdriverio base: master head fork: webdriverio/webdriverio compare: master

Create pull request Choose different branches or forks above to discuss and review changes.

Compare and review just about anything

Branches, tags, commit ranges, and time ranges. In the same repository and across forks.

EXAMPLE COMPARISONS	
end-session	on 12 Apr
asyncAwaitElements	on 7 Nov 2017
v5	on 4 Aug 2017
strategy	on 12 Feb 2016
webdriverjs	on 18 Aug 2014

mattc41190 added some commits 3 days ago

`Refactor launcher test (#1)` ··· Verified `af03712`

`eslint fixes` ✕ `385603c`

練習題：操作 Github

- 練習題：在 Github 上，新增一個叫 profile 的 repository
- 練習題：直接在 Github 網站上編輯檔案
- 練習題：新增一個 ISSUE
- 練習題：Fork 一個專案
- 練習題：發一個 Pull Rquest (PR)
- 練習題：進行代碼審查 Code Review

git blame 指令

使用情境

- 追蹤 bug，查看這行程式誰的。

常用範例

範例	說明
git blame -L 2,5 hello.txt	
git blame -L 2 hello.txt	
git blame -L 2, hello.txt	
git blame -L ,5 hello.txt	追蹤修改紀錄

語法結構

```
usage: git blame [<options>] [<rev-opts>] [<rev>] [--] <file>

    <rev-opts> are documented in git-rev-list(1)

    --incremental         Show blame entries as we find them,
 incrementally
    -b                    Show blank SHA-1 for boundary commi
ts (Default: off)
    --root                Do not treat root commits as bounda
ries (Default: off)
    --show-stats          Show work cost statistics
    --score-debug         Show output score for blame entries
    -f, --show-name       Show original filename (Default: au
to)
    -n, --show-number     Show original linenumber (Default:
```

```
off)
    -p, --porcelain       Show in a format designed for machi
ne consumption
    --line-porcelain      Show porcelain format with per-line
 commit information
    -c                    Use the same output mode as git-ann
otate (Default: off)
    -t                    Show raw timestamp (Default: off)
    -l                    Show long commit SHA1 (Default: off
)
    -s                    Suppress author name and timestamp
(Default: off)
    -e, --show-email      Show author email instead of name (
Default: off)
    -w                    Ignore whitespace differences
    --minimal             Spend extra cycles to find better m
atch
    -S <file>             Use revisions from <file> instead o
f calling git-rev-list
    --contents <file>     Use <file>'s contents as the final
image
    -C[<score>]           Find line copies within and across
files
    -M[<score>]           Find line movements within and acro
ss files
    -L <n,m>              Process only line range n,m, counti
ng from 1
    --abbrev[=<n>]        use <n> digits to display SHA-1s
```

git grep 指令

使用情境

- 快速查看送交紀錄

常用範例

範例	說明
git grep "Hello"	查現在版本是否有 "Hello" 的字串

語法結構

```
usage: git grep [<options>] [-e] <pattern> [<rev>...] [[--] <
path>...]

    --cached              search in index instead of in the w
ork tree
    --no-index            find in contents not managed by git
    --untracked           search in both tracked and untracke
d files
    --exclude-standard    ignore files specified via '.gitign
ore'

    -v, --invert-match    show non-matching lines
    -i, --ignore-case     case insensitive matching
    -w, --word-regexp     match patterns only at word boundar
ies
    -a, --text            process binary files as text
    -I                    don't match patterns in binary file
s
    --textconv            process binary files with textconv
filters
```

```
    --max-depth <depth>   descend at most <depth> levels

    -E, --extended-regexp
                          use extended POSIX regular expressi
ons
    -G, --basic-regexp    use basic POSIX regular expressions
 (default)
    -F, --fixed-strings   interpret patterns as fixed strings
    -P, --perl-regexp     use Perl-compatible regular express
ions

    -n, --line-number     show line numbers
    -h                    don't show filenames
    -H                    show filenames
    --full-name           show filenames relative to top dire
ctory
    -l, --files-with-matches
                          show only filenames instead of matc
hing lines
    --name-only           synonym for --files-with-matches
    -L, --files-without-match
                          show only the names of files withou
t match
    -z, --null            print NUL after filenames
    -c, --count           show the number of matches instead
of matching lines
    --color[=<when>]      highlight matches
    --break               print empty line between matches fr
om different files
    --heading             show filename only once above match
es from same file

    -C, --context <n>     show <n> context lines before and a
fter matches
    -B, --before-context <n>
                          show <n> context lines before match
es
    -A, --after-context <n>
```

```
                            show <n> context lines after matche
s
    -NUM                    shortcut for -C NUM
    -p, --show-function     show a line with the function name
before matches
    -W, --function-context
                            show the surrounding function

    -f <file>               read patterns from file
    -e <pattern>            match <pattern>
    --and                   combine patterns specified with -e
    --or
    --not
    (
    )
    -q, --quiet             indicate hit with exit status witho
ut output
    --all-match             show only matches from files that m
atch all patterns

    -O, --open-files-in-pager[=<pager>]
                            show matching files in the pager
    --ext-grep              allow calling of grep(1) (ignored b
y this build)
```

git tig

```
2018-03-19 04:48 Pavel Roskin          o Don't allow pos->lineno to underflow
2018-03-19 07:47 Jonas Fonseca         M─┐ Merge pull request #799 from proski
2018-03-18 22:47 Pavel Roskin          │ o Annotate fall-through in case state
2018-03-18 22:33 Pavel Roskin          │ o Let strncpy() terminate the destina
2018-03-18 22:34 Pavel Roskin          │ o Drop "const" from return values, it
2018-03-16 21:24 Jonas Fonseca         o─┘ Print the ncurses and readline vers
2018-03-13 02:32 rofl0r                o display.h: fix build against netbsd-c
2018-03-13 02:31 rofl0r                o display.c: fix build with netbsd-curs
2018-03-13 03:26 Thomas Koutcher       o Fix file argument (#788)
2018-03-13 03:23 Björn Andersson       o New boolean option "send-child-enter"
2018-03-13 03:09 Björn Andersson       o Update make config for cygwin/ncurses
2018-02-07 03:23 Jonas Fonseca         M─┐ Merge pull request #780 from hwangc
2018-02-06 23:05 hwangcc23             │ o Fix usage of sizeof(finder->line)
2018-02-06 22:22 hwangcc23             │ o Fix dead code
2018-01-31 23:52 hwangcc23             │ o Fix the resource leak
2018-01-30 20:12 Jonas Fonseca         o─┘ <tig-2.3.3> tig-2.3.3
2018-01-27 21:01 harshavardhan         o updated https to https (#777)
2018-01-09 09:25 Jonas Fonseca         o Change refs_tags type to size_t
2017-12-21 19:46 Alexander Droste      o Revert "Handle \n like \r (#758)" (#7
2017-12-19 11:22 Jonas Fonseca         o Fix #164 by catching SIGHUP
2017-12-18 16:58 Jonas Fonseca         o <tig-2.3.2> tig-2.3.2
2017-12-18 12:07 Jonas Fonseca         o Only check for busy loop when no view
2017-12-14 21:14 Jonas Fonseca         o <tig-2.3.1> tig-2.3.1
2017-12-14 20:30 Jonas Fonseca         o Update NEWS
2017-12-14 11:00 Stephen               o Parallelize the Travis CI build and m
2017-12-14 13:45 David O'Trakoun       o tig(1): Fix env var checked (#752)
2017-12-14 19:43 Christian Brabandt    o Handle \n like \r (#758)
2017-12-14 12:46 Jonas Fonseca         o Use initscr to ensure proper TTY setu
2017-12-12 22:01 Jonas Fonseca         o Fix #164: Add workaround to detect bu
2017-12-12 21:35 Jonas Fonseca         o Move loop updating views to separate
2017-12-11 14:38 Jonas Fonseca         o Fix formatting of the Windows install
2017-10-16 13:03 Matt                  o Added another installation method (#7
2017-09-30 18:30 Jonas Fonseca         M─┐ Merge pull request #725 from roland
2017-08-14 13:10 Roland Walker         │ o restore tty attrs+flush before/afte
2017-08-14 13:05 Roland Walker         │ o save and restore TTY attributes
2017-09-30 13:09 Roland Walker         o │ preserve/restore TTY attributes in
2017-09-29 15:38 Roland Walker         o │ raise timeout in test which sleeps
[main] bc722c9612638d757ca911b31d7b11564af5c3de - commit 17 of 2441          1%
```

```
commit bc722c9612638d757ca911b31d7b11564af5c3de
Refs: tig-2.3.3-1-gbc722c9
Author:     hwangcc23 <hwangcc@csie.nctu.edu.tw>
AuthorDate: Wed Jan 31 23:52:13 2018 +0800
Commit:     hwangcc23 <hwangcc@csie.nctu.edu.tw>
CommitDate: Wed Jan 31 23:52:13 2018 +0800

    Fix the resource leak

    Fix the file descriptor leak.
---
 src/display.c | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

diff --git a/src/display.c b/src/display.c
index 974f601..6f04255 100644
--- a/src/display.c
+++ b/src/display.c
@@ -408,8 +408,10 @@ save_view(struct view *view, const char *path)
                        enum_name(get_line_type_name(line->type)),
                        line->selected);

-               if (!view->ops->get_column_data(view, line, &column_data))
+               if (!view->ops->get_column_data(view, line, &column_data)) {
+                       fclose(file);
                        return true;
+               }

                if (column_data.box) {
                        const struct box *box = column_data.box;
[diff] bc722c9612638d757ca911b31d7b11564af5c3de - line 1 of 30          100%
```

實用小技巧

新增空的資料夾

```
.gitkeeper
```

救回誤刪的檔案

```
# 查看已刪除的檔案
git ls-files -d

# 將已刪除的檔案還原
git ls-files -d | xargs git checkout —
```

救回誤刪的分支

若你不小心移除了分支或是其他參照，你可以使用 `git reflog` 指令來還原。

快速尋找臭蟲

- 善用 grep
- 善用 blame 文件逐行追溯
- 善用 bisect 二分查找

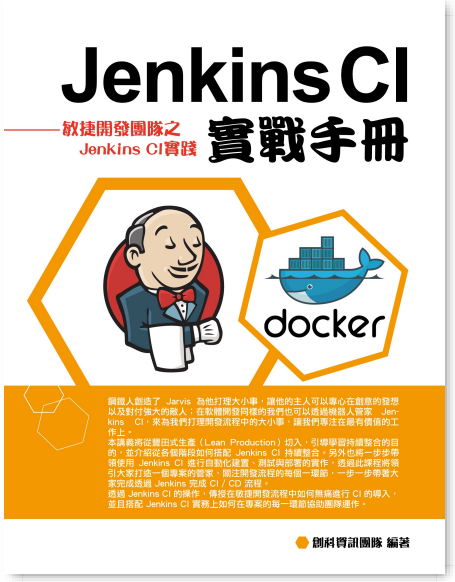
更多資源

- Git Magic - 前言

參考來源

- gitster/git-htmldocs

Jenkins CI 實戰手冊



鋼鐵人創造了 Jarvis 為他打理大小事，讓他的主人可以專心在創意的發想以及對付強大的敵人；在軟體開發同樣的我們也可以透過機器人管家 Jenkins CI，來為我們打理開發流程中的大小事，讓我們專注在最有價值的工作上。

本講義將從豐田式生產（又稱為精實生產；Lean production ）切入，引導學習持續整合的目的，並介紹從各個階段如何搭配 Jenkins CI。另外也將一步步帶領使用 Jenkins CI 進行自動化建置、測試與部署的實作，透過此課程將領引大家打造一個專案的管家，關注開發流程的每個一環節，一步一步帶著大家完成透過 Jenkins 完成 Continuous Integration（持續整合）及 Continuous Delivery（持續交付；CD）流程。

透過 Jenkins CI 的操作，傳授在敏捷開發流程中如何無痛進行 CI 的導入，並且搭配 Jenkins CI 實務上如何在專案的每一環節協助團隊運作

閱讀須知

本講義之範例實作皆搭配 Ubuntu Linux 作業系統，請於課前先向活動主辦單位取得 VirtualBox 虛擬機器映像檔。

適合對象

適合對開發流程自動化感興趣的 IT 從業人員，或是導入敏捷開發卻無法敏捷的團隊學習，適合不排斥在 Linux 命令列輸入指令的人，因為大部份自動化程序都需要透過命令列輸入指令執行。

本講義將針對 Web 相關應用如 Web Application、API Server 相關開發應用進行演示（因 App 開發流程與 Web 有所差異，特別提醒 App 開發者留意）講師本身熟悉 Java 以及 JavaScript 開發，因此課程將以 Node.js 範例專案進行演練，但實際 CI 的導入應用觀念與流程，可適用於大多數程式語言及開發工具。

講座與課程

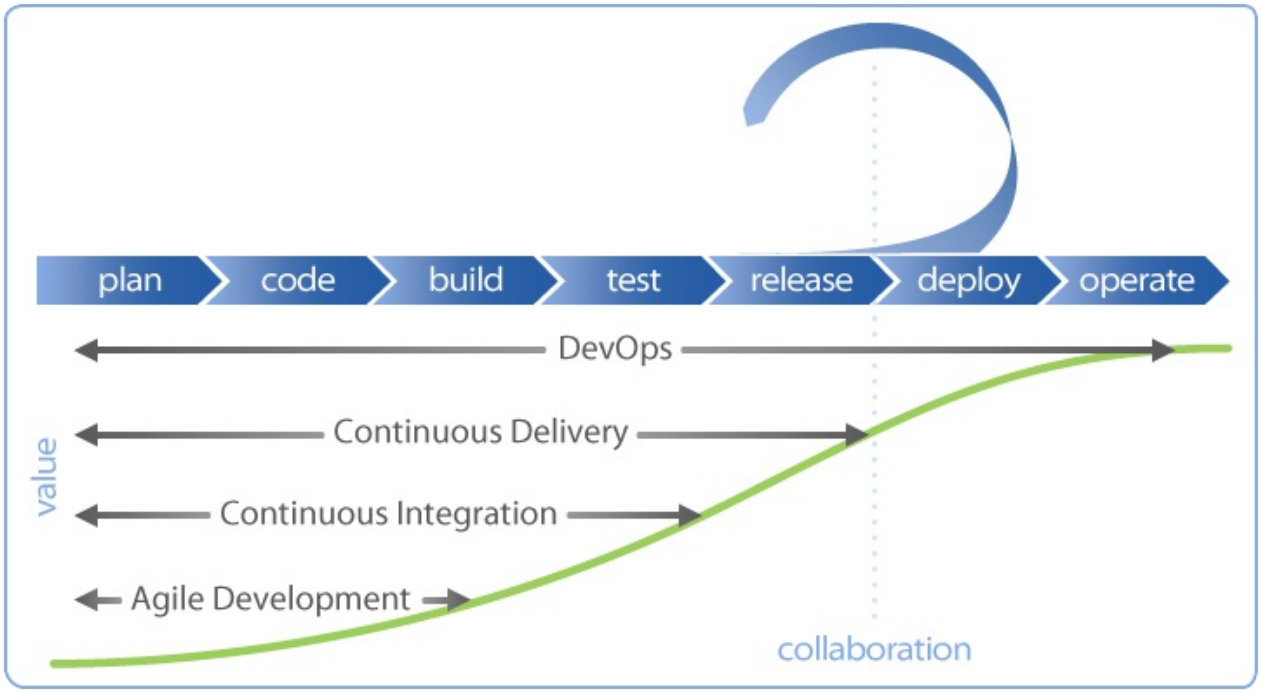
本講義提供給參加 Jenkins CI 課程活動之學員。

- 資策會 舉辦之 DevOps：Jenkins + Docker 持續整合實務班
- iThome 舉辦之 Jenkins 持續整合實戰講堂
- 企業或學校辦理相關軟體開發培訓課程請聯繫 workshop@trunksys.com

講義作者

本講義由創科資訊研發團隊的 Spooky 及 Kyle 編著，如需要分享請先獲得授權許可。

基本概念



上圖中可以看到每個階段都有其對應要處理的事項，而關於這些不同階段的事項所要解決的事情如下：

1. Agile Development：有效率的溝通，做最有價值的事。
2. Continuous Integration：自動化測試，自動化建置，讓重覆的事情可以自動執行。
3. Continuous Delivery：讓專案可以自動發佈，隨時有穩定版本可以運作，加速 test 驗證。
4. DevOps：確保運行環境正常，負責將上述各個機制建置及維護，還有一旦 production deployment 之後的維運。

當然，實際狀況絕對不是 DevOps 這個職位的人單一可以完成上述所有的整合，必須要專案開發人員與及規劃時皆已考慮才有可能順暢的進行。

上述每一個環節都是要避免所有不必要的浪費

1. Agile Development：避免用昂貴的人力（你）無效的溝通。
2. Continuous Integration：避免用昂貴的人力（你）進行重覆測試，避免用昂貴的人力（你）進行重覆專案的建置及測試。

3. Continuous Delivery：避免用昂貴的人力（你）處理 produciton deployment 的瑣碎細節。

除了解決不必要的浪費，還有避免人為錯誤，一旦所有程序都自動化後，錯誤將容易被排除驗證。

就像豐田式生產（TPS, Lean Production）所提倡的理念：杜絕浪費；每一次的浪費一旦被避免，也就表示有更多時間資源可以進行有價值的事！

上述內容節錄自：企業端如何推動持續整合開發流程

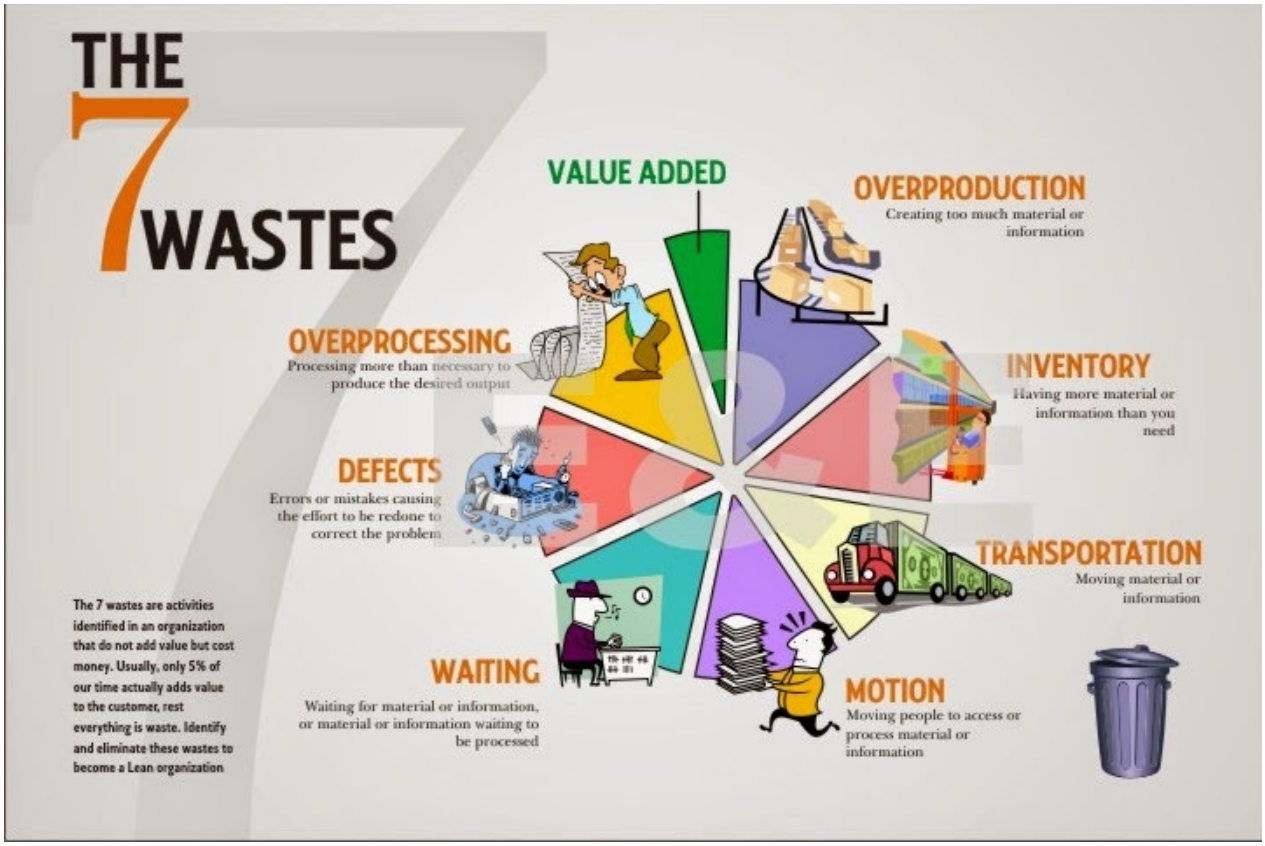
刊登於 ithome：【JSDC客座文章】企業端如何推動持續整合開發流程

豐田式生產

如果想瞭解持續整合的目的，就需要先知道豐田式生產管理（Toyota Management）的概念，豐田式生產又稱為精實生產（Lean Producton）。

根據 wiki：豐田生產方式

> 豐田式生產主要目標，是杜絕負荷過重（muri）、不平準（mura）和一切的浪費（muda）。TPS 把浪費歸納成七種：
>
> 1. 等待的浪費；
> 2. 搬運的浪費；
> 3. 不良品的浪費；
> 4. 動作的浪費；
> 5. 加工的浪費；
> 6. 庫存的浪費；
> 7. 製造過多（早）的浪費。



TPS 背後的理念是杜絕浪費，因而不再需要庫存。基本原則是「豐田模式」（The Toyota Way），概述如下：

1. 正確的流程方能產生正確結果
2. 發展員工與事業夥伴，以為組織創造價值
3. 持續解決根本問題是組織型學習的驅動力

從製造業的角度來看，能夠避免不必要的浪費跟重工，相對的效率就會變快，我們可以把 TPS 歸納的七種浪費就軟體開發的角度來看：

1. 等待的浪費，搬運的浪費：程式開發週期除了寫程式之外 build, test, deploy 都需要處理，但可以自動化
2. 不良品的浪費：程式錯誤一再發生，沒辦法杜絕，若能在每個環節測試確認，則可以確定每個動作沒有問題
3. 動作的浪費，加工的浪費：可以自動化，就不需要人的接入
4. 庫存的浪費：軟體沒有庫存壓力
5. 製造過多（早）的浪費：重覆的動作，就是一種浪費

所有在製造業所會發生的問題，大部份軟體開發也會有同樣的問題，我們可以借鏡其精神，與豐田式生產相互輝映，發展出敏捷式開發（agile），下一章節將近行介紹。

敏捷式開發

說明持續整合，就不得不說明敏捷軟體開發（Agile），跟豐田式生產一樣，敏捷式開發也致力於杜絕所有可能的浪費。

參考 Wikipedia 中的說明：敏捷軟體開發

> 敏捷軟體開發（英語：Agile software development），又稱敏捷開發，是一種從 1990 年代開始逐漸引起廣泛關注的一些新型軟體開發方法，是一種應對快速變化的需求的一種軟體開發能力。它們的具體名稱、理念、過程、術語都不盡相同，相對於「非敏捷」，更強調程式設計師團隊與業務專家之間的緊密協作、面對面的溝通（認為比書面的文檔更有效）、頻繁交付新的軟體版本、緊湊而自我組織型的團隊、能夠很好地適應需求變化的代碼編寫和團隊組織方法，也更注重軟體開發中人的作用。

敏捷開發宣言

- 個人與互動 **重於** 流程與工具：避免資訊的錯誤，浪費溝通的時間。
- 可用的軟體 **重於** 詳盡的文件：若詳盡的文件就是可用的軟體，可避免維護兩個地方，且文件就是運作結果本身。
- 與客戶合作 **重於** 合約協商：客戶才是需求的本身，與客戶合作將更接近目標，不會花錯力氣。
- 回應變化 **重於** 遵循計劃：唯一不變的就是變，計畫趕不上變化，如果只是不知變通的依計劃進行，往往知道方向錯了已來不及。

持續整合改善

有了可運行的規格以及每次異常產生的規格，接著就要讓日常累積的開發能量，能每日運行規格成為品質的後盾。

搭配 CI 定時產出報表，隨時注意專案的健康度，即早發現問題即早治療，在這要說明如何搭配 CI（Continous Integration），讓這一連串的活動能夠持續運行。

Daliy Build（or Nightly Build）

1. 每天早上以及中午進行一次建置
2. 建置前需要通過所有的規格運行
3. 一旦發生錯誤，一律需要當天確認，並且當天修正完畢
4. 一旦建置完成 deploy 到 develop 機器進行測試

Weekly Build

1. 每個週末選定一天進行建置
2. 建置前需要通過所有的規格運行
3. week build 可選擇是否 deploy 到 production 機器

每天每週的建置，正如豐田式生產一樣，發生問題即早面對並解決，避免產出不良品，修正完成再繼續往前，將異常或是浪費降到最低。

結論

程式開發是一連串的協同合作，人為疏失是沒有辦法根絕的，但卻可以有機制讓問題及早發現，TDD 正是一個良好的開發流程，讓開發過程包含了驗證機制，確保開發能量能夠被累積並且重覆使用，隨著專案的成長，偵錯的能樣也會同步成長。

日常有 TDD 的習慣，導入 CI 水到渠成，利用 CI 讓專案定時被健康檢查，專案上線或是更新不再提心吊膽，在令人安心的環境下開發，更能專注於系統功能的完善，我想有遵循 TDD 開發流程的團隊是幸福的，就像有爸媽呵護一樣不用擔心受怕 :)

持續整合流程從豐田式生產而來，其所提倡的核心目標是剛剛好，不浪費，以及透明，並且透過持續整合 Jenkins 的幫忙，讓專案有“心跳”，若沒有正常跳動盡早急救！

Jenkins 可以在 Windows、Linux 或 Mac OS X 環境執行，一般來說我們會盡可能選擇跟 Production / Testing 伺服器接近的環境來運行。

本書範例皆以 Ubuntu Linux 演示

因為 Ubuntu Linux 相容多數 Open Source 軟體，也被用於許多 Production Server 環境，因此使用 Ubuntu Linux 作為 Jenkins 入門的系統，可以避免很多干擾學習的問題發生。

建議搭配 VirtualBox 虛擬機器使用 Ubuntu Linux + Jenkins，我們也為學習者提供已安裝設定完成的 VirtualBox Image，請向課程主辦單位取得檔案下載位址。

安裝 Java 軟體

建議使用 JDK 1.8 以上版本。

安裝 java 8 & maven

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

選擇 java 8 作為預設 java version

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

安裝 Jenkins 軟體

參考 Jenkins 官方的 Ubuntu Linux 安裝說明

```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key
 | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/
 > /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install jenkins
```

管理 Jenkins 伺服器

重新啟動 Jenkins 伺服器。

```
sudo service jenkins restart
```

Jenkins Quick Start

安裝提示

1. 安裝 Ubuntu Linux 16.04 LTS
2. 安裝 Java

```
sudo apt-get install openjdk-8-jdk
```

3. 安裝 Jenkins

```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.
key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stab
le binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install jenkins
```

4. 設定 Jenkins

```
# /etc/default/jenkins
HTTP_PORT=8080
JAVA_ARGS="-Djava.awt.headless=true -Xmx1024m"
```

5. 設定 sudo 權限與免密碼

```
# /etc/passwd
jenkins:x:111:117:Jenkins,,,:/var/lib/jenkins:/bin/bash

# /etc/sudoers
%sudo   ALL=(ALL:ALL) NOPASSWD: ALL

# /etc/group
sudo:x:27:user,jenkins
```

6. 管理 Jenkins

```
# Upgrade
sudo apt-get update && sudo apt-get upgrade jenkins

# Start, Restart, Stop
sudo service jenkins start
sudo service jenkins restart
sudo service jenkins stop
```

Hello! Jenkins!

難度： 簡易

目標： 認識 Jenkins Job 基礎操作。

專案： https://github.com/agileworks-tw/java-hello-world

練習：

1. 新增作業（Jenkins Job）名稱： `HelloWorld`
2. 使用 Shell 指令檢查 JDK 版本

   ```
   java -version
   javac -version
   ```

3. 使用 Git SCM，設定 Git Repository URL

   ```
   https://github.com/agileworks-tw/java-hello-world
   ```

4. 使用 `javac` 指令進行編譯、 `java` 指令執行
5. 使用 curl 指令操作 Remote API： `curl http://localhost:8080/job/HelloWorld/build`
6. 自訂字串參數： `WHO` ，並帶入 Shell 指令
7. 設定 Git 分支（branch）

   ```
   with-name-argument
   ```

8. 使用 curl 指令操作參數化建置 Remote API： `http://localhost:8080/job/HelloWorld/buildWithParameters?WHO=Mary`
9. 使用 Git Parameter 參數化建置切換分支： `BRANCH_NAME` ，並帶入 Git SCM 設定

10. 利用 Remote API 回傳 XML 或 JSON 格式的 Jenkins Job 定義資料
11. 使用 Jenkins CLI 刪除 Job

參考資料

下面範例需要用到使用者的 token 可在下面網址取得

http://{jenkins_url}/user/{username}/configure

Remote API 範例

```
curl http://<user>:<token>@localhost:8080/job/HelloWorld/api/
json | python -m json.tool
curl http://<user>:<token>@localhost:8080/job/HelloWorld/api/
json | python -m json.tool
```

```
curl http://<user>:<token>@@localhost:8080/job/HelloWorld/api
/json | xmllint --format -
```

Jenkins CLI 範例

```
java -jar jenkins-cli.jar -auth <user>:<token> -s http://loca
lhost:8080/ delete-job HelloWorld
```

關於 cli 部分，可以參考 http://{jenkins_url}/cli

若要透過 command line 安裝 plugin 也可以透過 cli 完成，如：

```
java -jar jenkins-cli.jar -s http://{jenkins_url}:9088/ install-
plugin SOURCE ... [-deploy] [-name VAL] [-restart]
```

Test Report: JUnit v.s. TAP

難度： 簡易

目標： 瞭解 Jenkins 的測試報表（Reporting）功能。

使用 YSlow for PhantomJS。

PhantomJS is a headless WebKit with JavaScript API. YSlow for PhantomJS is a command line script that allows page performance analysis from live URLs, unlike YSlow for Command Line (HAR) where a pre-generated HAR file is needed in order to analyze page performance.

http://yslow.org/phantomjs/

練習：

1. 新建 Free-Style Jenkins Job： `YSlow`
2. 執行 Shell 指令檢查 `phantomjs` 版本
3. 下載並解壓縮： `http://yslow.org/yslow-phantomjs-3.1.8.zip`
4. 執行 `phantomjs` 產生 JUnit 格式的測試報告 yslow.xml
5. 使用 Post-build actions 發佈 JUnit 測試結果報告
6. 改用 TAP Test Results 輸出測試結果

PhantomJS + YSlow 指令範例：

```
# junit
phantomjs yslow.js -i grade -threshold "B" -f junit http://lo
calhost:8080/ > yslow.xml

# tap
phantomjs yslow.js -i grade -threshold "B" -f tap http://loca
lhost:8080/ > yslow.tap
```

Markdown Converter

難度： 簡易

目標： 使用 Jenkins 處理轉檔任務並保存結果。

專案： https://github.com/agileworks-tw/markdown-example

練習：

1. 建立 Free-Style Jenkins Job： `Markdown`
2. 執行 Shell 指令檢查 `pandoc` 版本
3. 設定 Git Repository URL
4. 將 `README.md` 轉成 html 文件格式
5. 保存 `README.html` 文件

```
pandoc -s README.md -o README.html
```

Using Gradle

難度： 普通

目標： 使用 Gradle 自動化建置與測試 Java 專案。

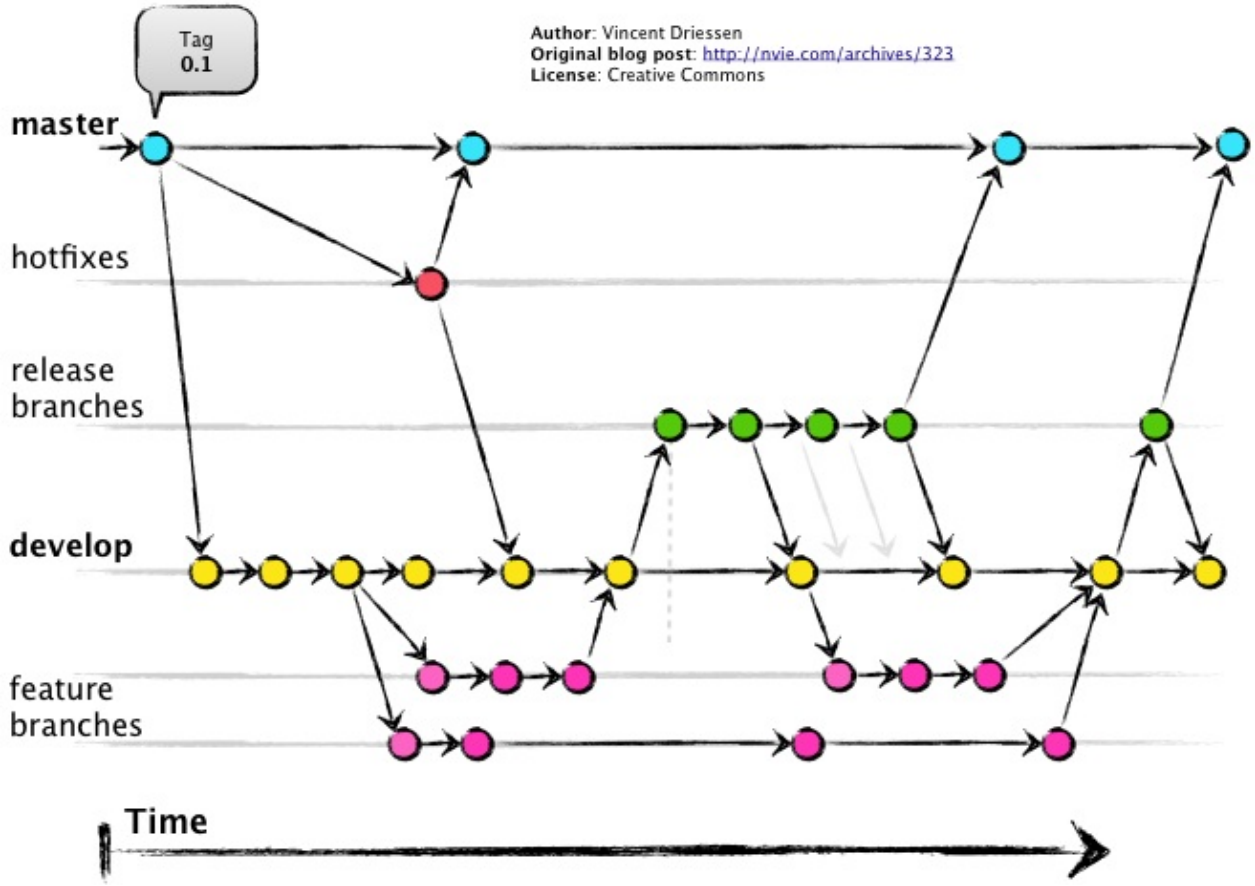
專案： https://github.com/agileworks-tw/gradle-example

練習：

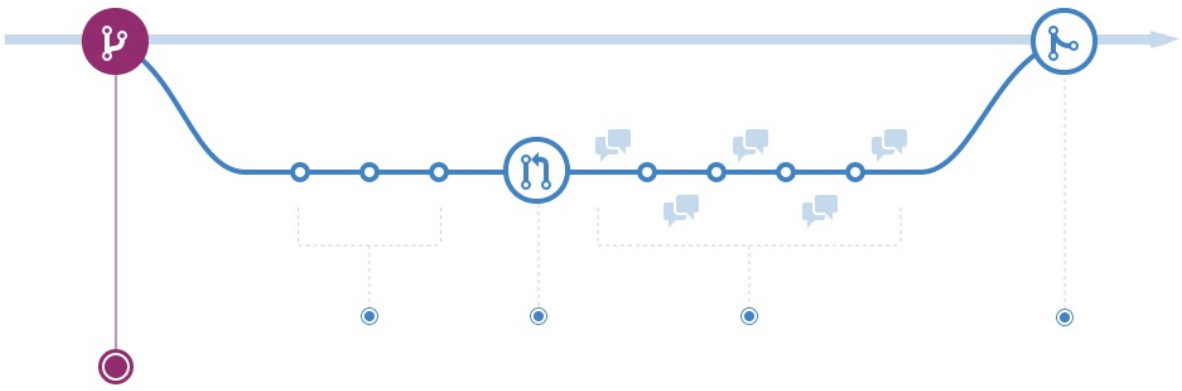
- 建立 Free-Style Jenkins Job： `gradle-example`
- 執行 Shell 指令檢查 `gradle` 版本
- 設定 Git Repository URL
- 執行 `gradle build` 指令
- 封存產生的 .jar 檔案
- 建立並封存 JUnit Report
- 建立並封存 JavaDoc Report

專案開發各個階段如何搭配 CI

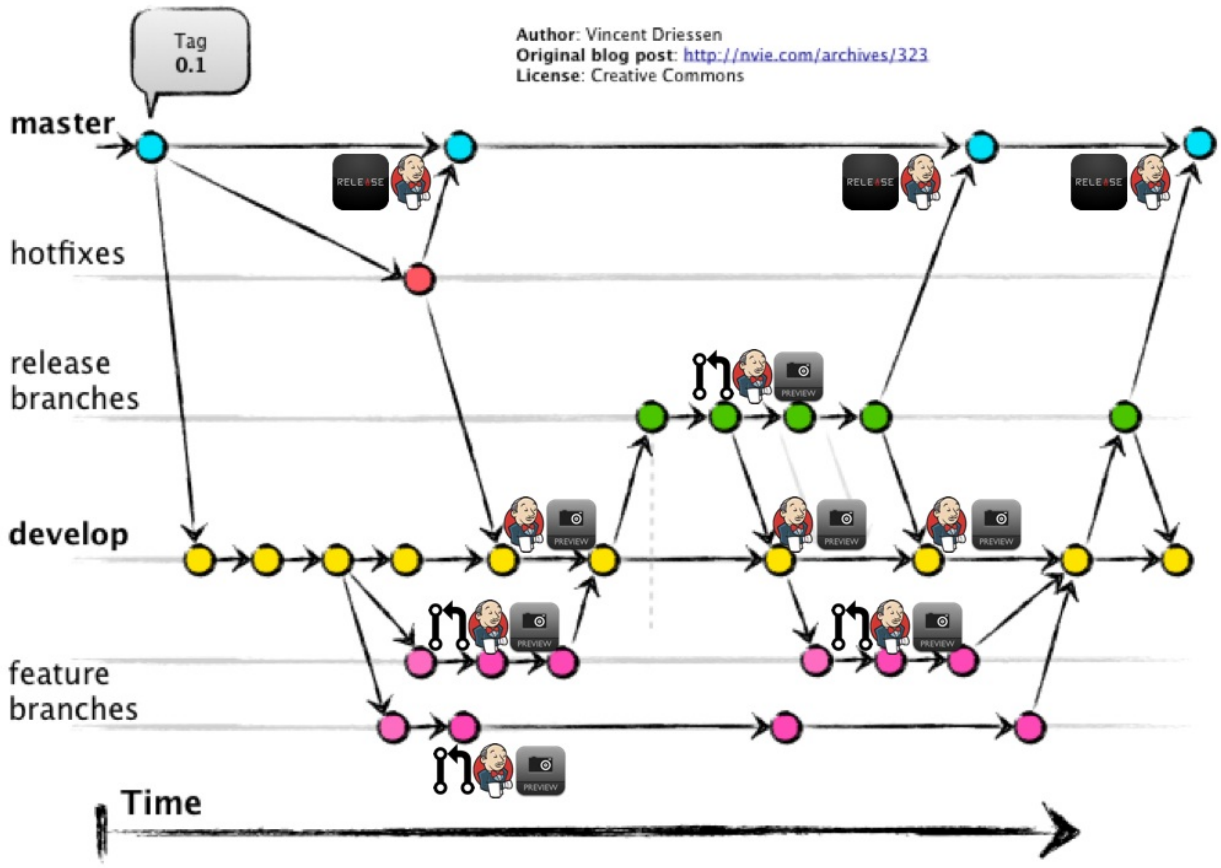
Git-Flow



GitHub Flow



Git-Flow with GitHub Flow and Jenkins



Lab 進行方式說明

相關 Lab 將會透過下面步驟進行

1. 使用 command line 完成並驗證
2. 使用 Jenkins Task 進行建置設置
3. 確認 Jenkins 執行結果

透過 command line 的操作能夠了解實際建置的運作原理，更近一步的可以了解整合過程中如何進行除錯與驗證

Build 自動建置

使用時機

1. Daily Build
2. 週期性建置
3. 進行正式機部署時事先建置

Task 實作

先設置 git Repository 使用：

```
https://github.com/agileworks-tw/spring-boot-sample
```

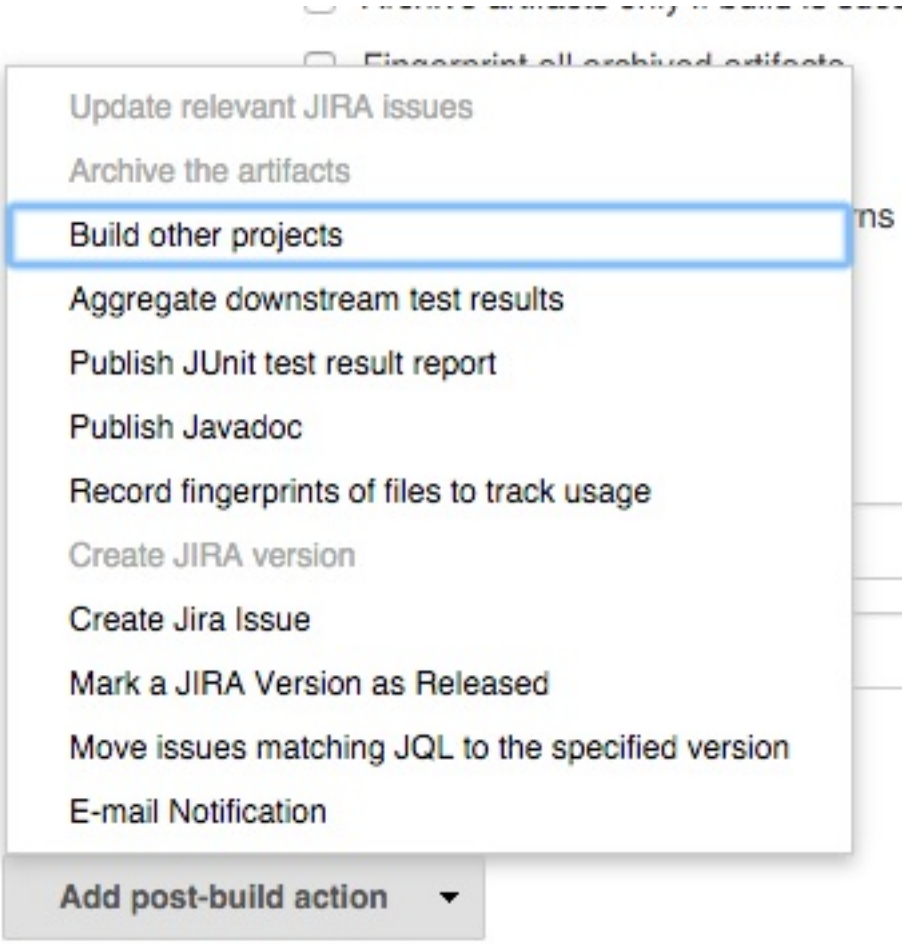
Shell 指令

```
#!/bin/bash
mvn package
```

build archive

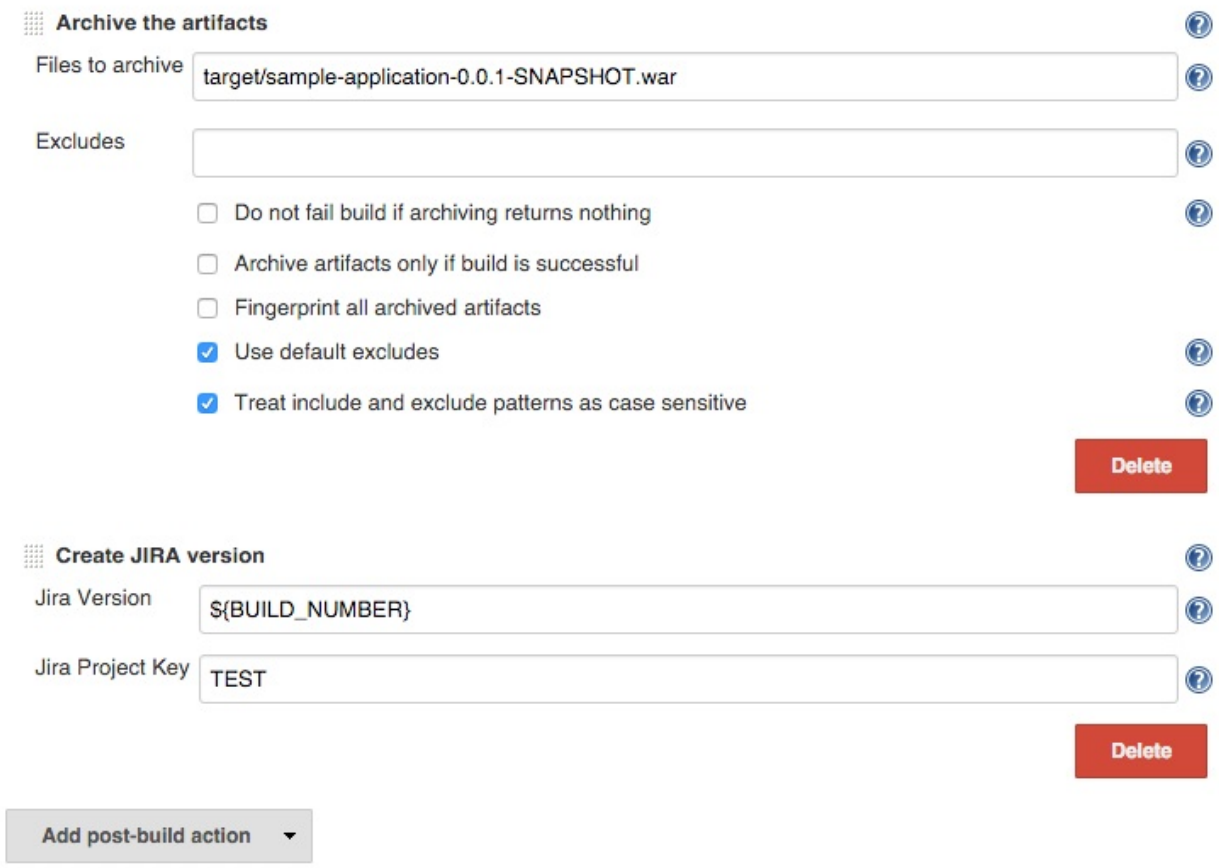
Archive the artifacts

參考下圖進行選曲



選擇 `Archive the artifacts` 此選項

設置要封存的檔案

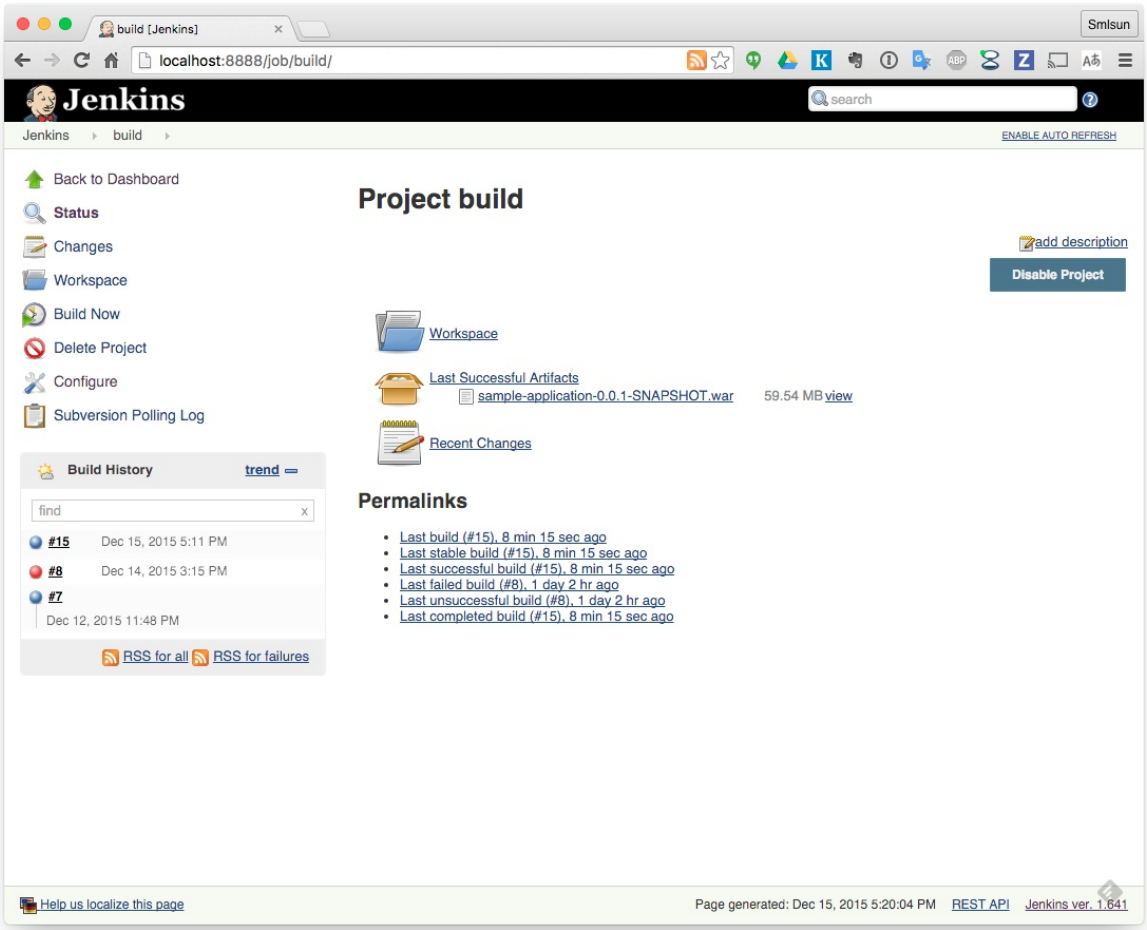


此例而言，我們可以填入 `target/sample-application-0.0.1-SNAPSHOT.war`

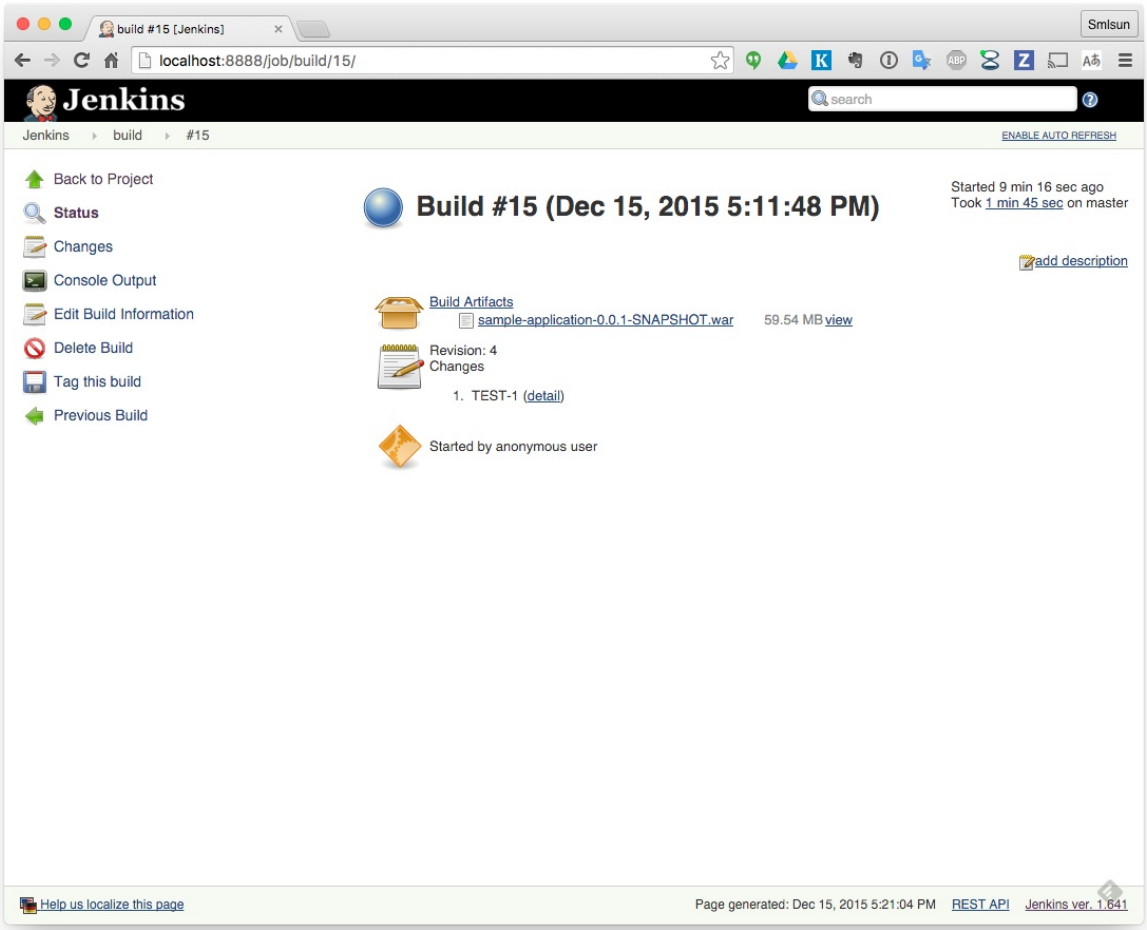
透過 `mvn -Pprod package` 此指令將會產生上述檔案

檢視封存結果

overview


build [Jenkins]
localhost:8888/job/build/
Jenkins
search
Jenkins
build
ENABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
Subversion Polling Log
Build History
trend
find
#15 Dec 15, 2015 5:11 PM
#8 Dec 14, 2015 3:15 PM
#7 Dec 12, 2015 11:48 PM
RSS for all
RSS for failures
Project build
add description
Disable Project
Workspace
Last Successful Artifacts
sample-application-0.0.1-SNAPSHOT.war 59.54 MB view
Recent Changes
Permalinks
Last build (#15), 8 min 15 sec ago
Last stable build (#15), 8 min 15 sec ago
Last successful build (#15), 8 min 15 sec ago
Last failed build (#8), 1 day 2 hr ago
Last unsuccessful build (#8), 1 day 2 hr ago
Last completed build (#15), 8 min 15 sec ago
Help us localize this page
Page generated: Dec 15, 2015 5:20:04 PM
REST API
Jenkins ver. 1.641

task

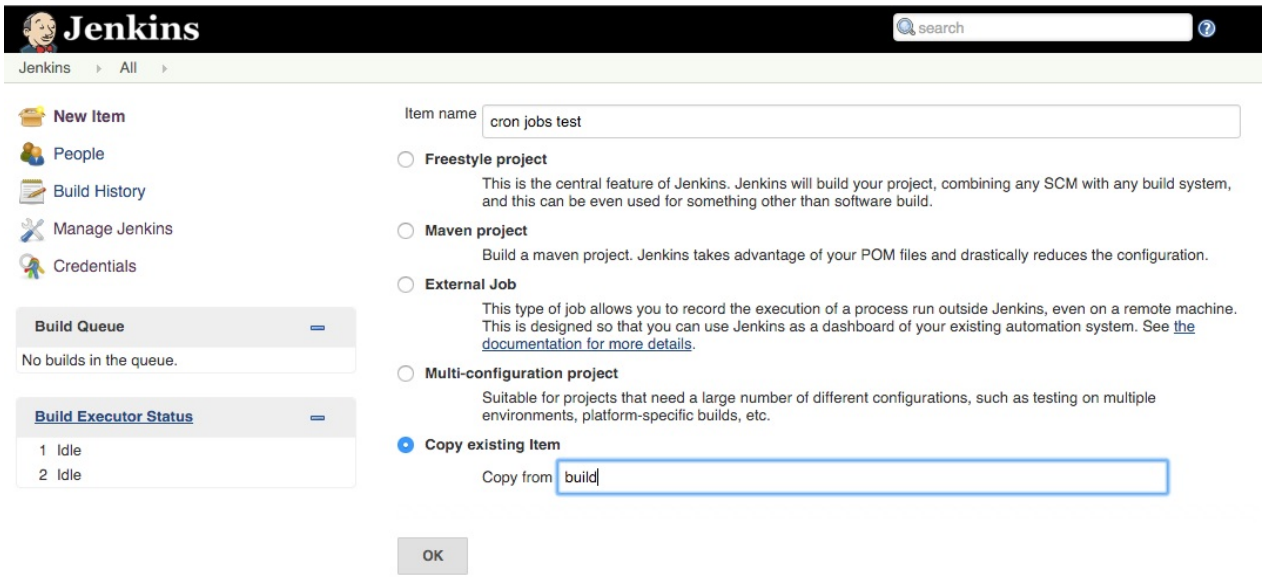


可以看到 jenkins 將會提供連結讓使用者可以進行下載

test

延續 task/build

我們可以建置透過 build task 直接複製如下圖：



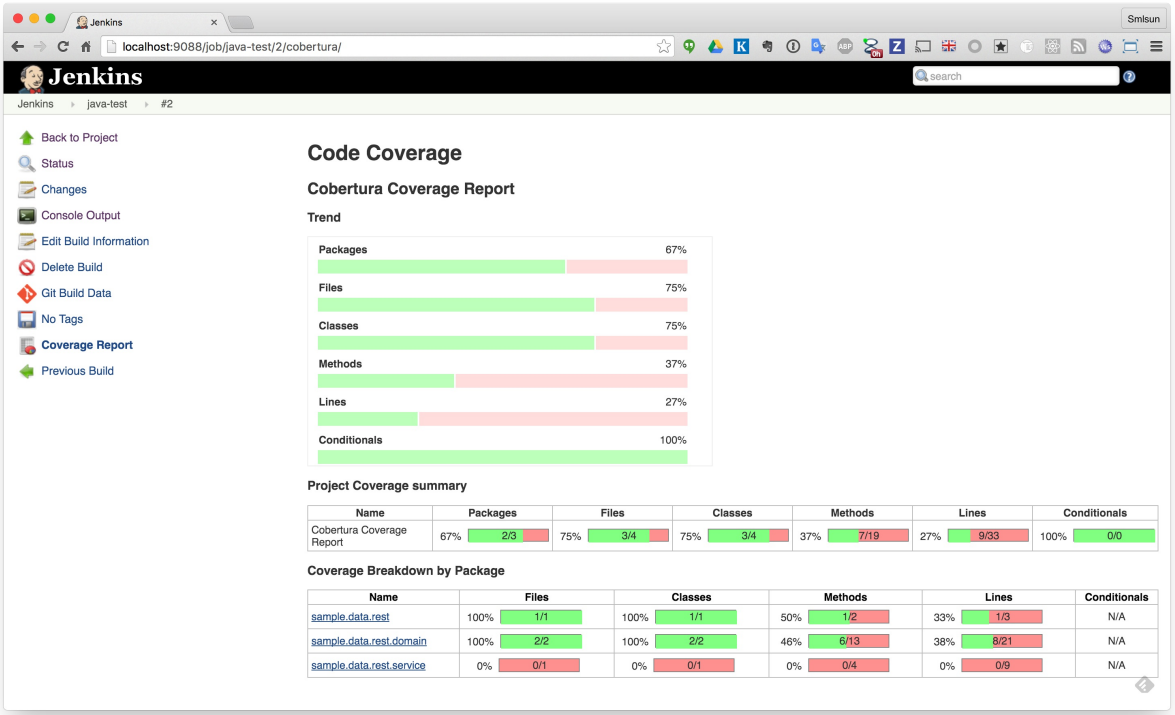
建置指令

```
#!/bin/bash
mvn clean cobertura:cobertura test
```

設置 Cobertura 呈現測試覆蓋率

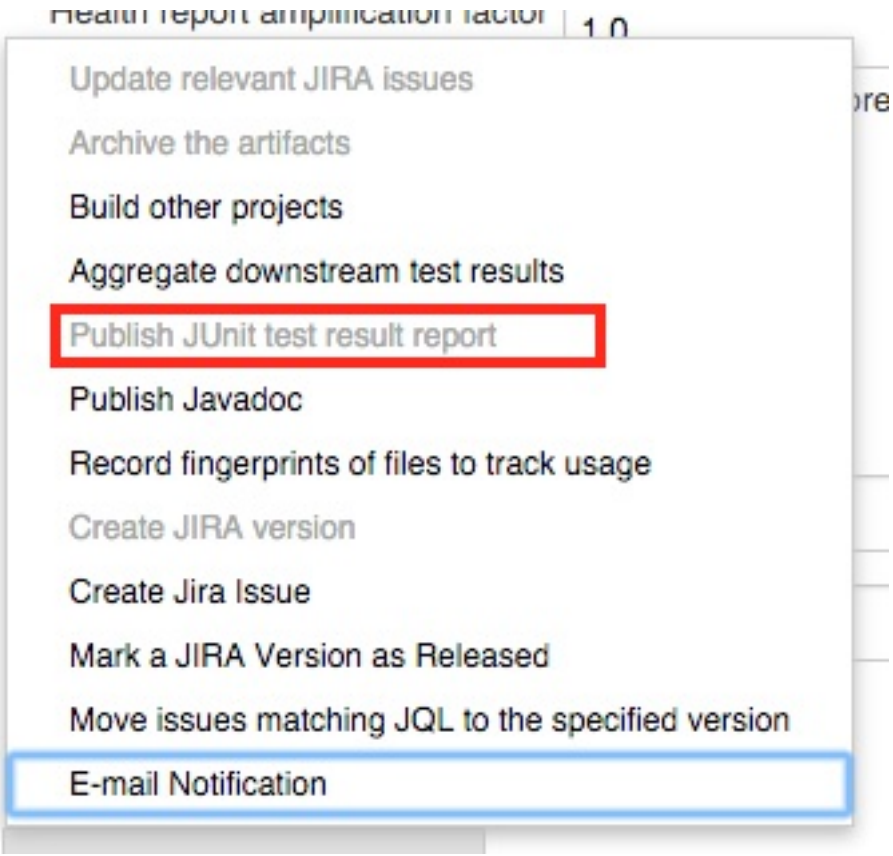
report 位置： `target/site/cobertura/coverage.xml`

執行結果

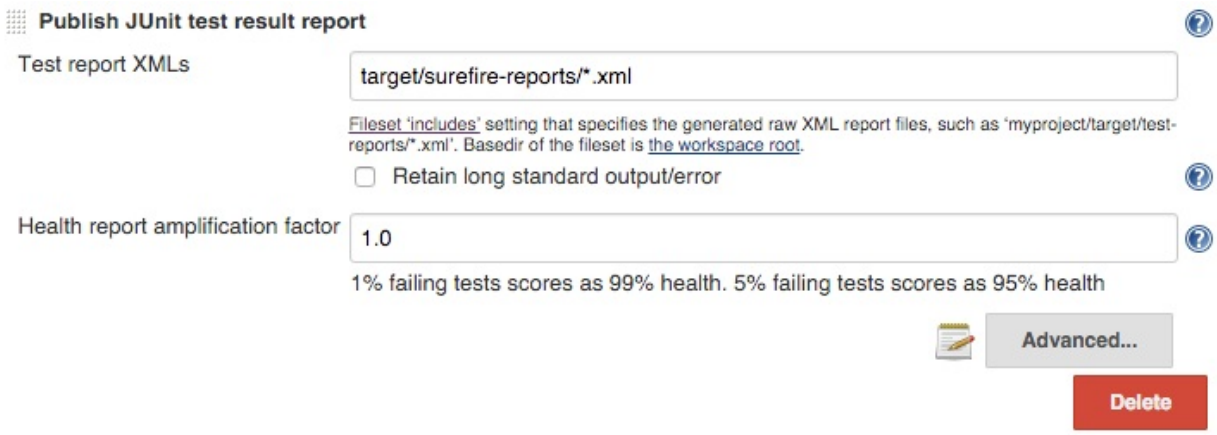

Jenkins
localhost:9088/job/java-test/2/cobertura/
Jenkins
Jenkins › java-test › #2
Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
Git Build Data
No Tags
Coverage Report
Previous Build
Code Coverage
Cobertura Coverage Report
Trend
Packages 67%
Files 75%
Classes 75%
Methods 37%
Lines 27%
Conditionals 100%
Project Coverage summary
Name | Packages | Files | Classes | Methods | Lines | Conditionals
Cobertura Coverage Report | 67% 2/3 | 75% 3/4 | 75% 3/4 | 37% 7/19 | 27% 9/33 | 100% 0/0
Coverage Breakdown by Package
Name | Files | Classes | Methods | Lines | Conditionals
sample.data.rest | 100% 1/1 | 100% 1/1 | 50% 1/2 | 33% 1/3 | N/A
sample.data.rest.domain | 100% 2/2 | 100% 2/2 | 46% 6/13 | 38% 8/21 | N/A
sample.data.rest.service | 0% 0/1 | 0% 0/1 | 0% 0/4 | 0% 0/9 | N/A

JUnit report

設置 task output report

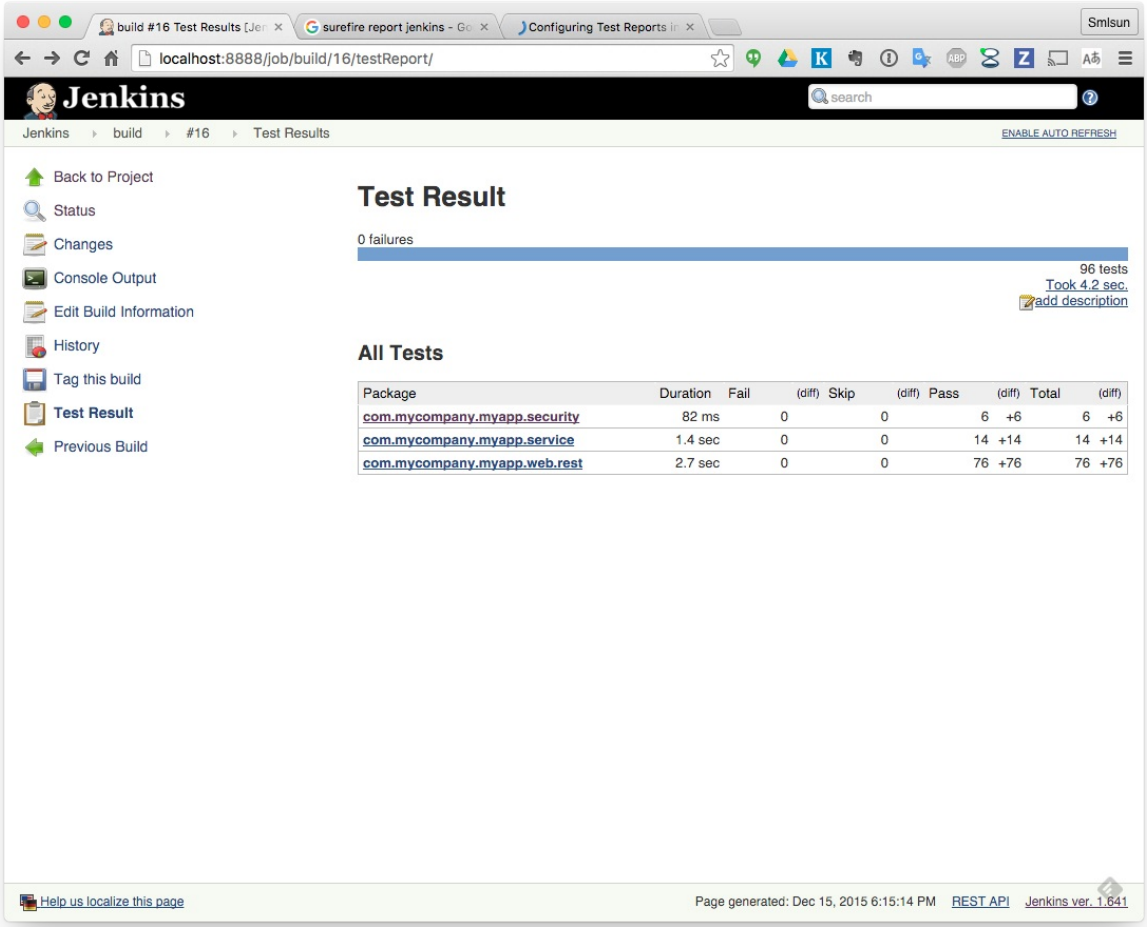


設定讀取 report 位置



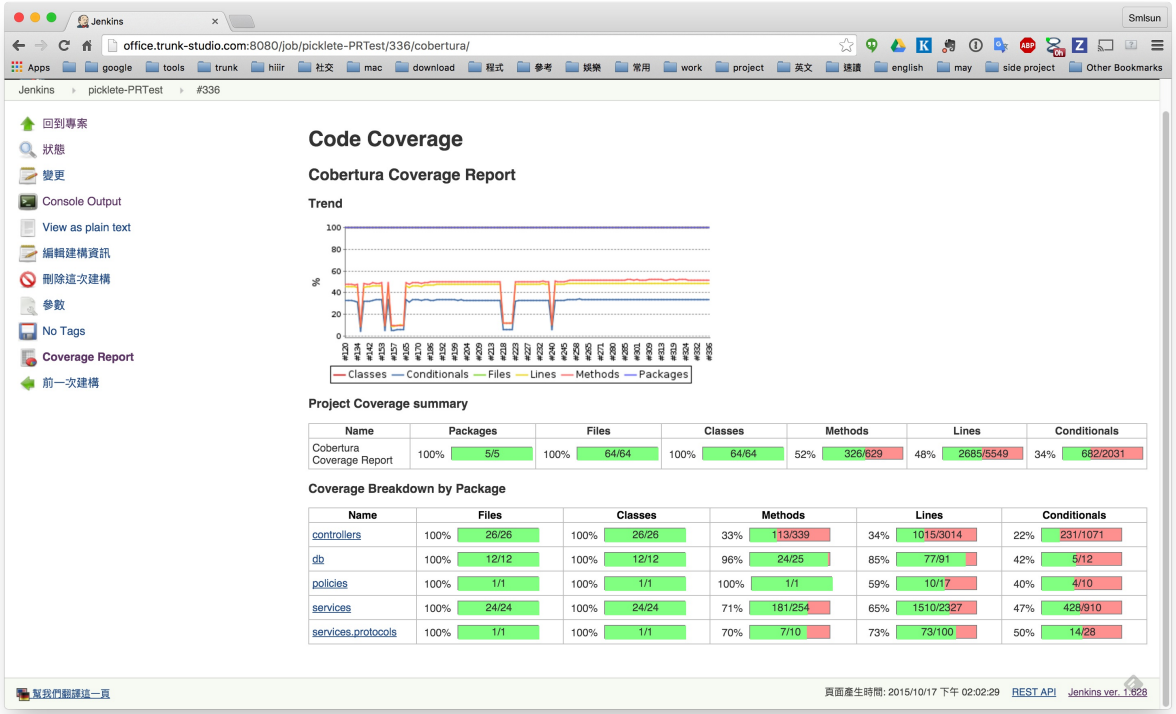
如上圖所示，將 Test report XMLs 設置為 `target/surefire-reports/*.xml`

檢視 test report



Cobertura

程式覆蓋率



task 設置畫面

picklete-PRTest Config [Je

office.trunk-studio.com:8080/job/picklete-PRTest/configure

Jenkins › picklete-PRTest › 組態設定

建置後動作

Publish Cobertura Coverage Report

Cobertura xml report pattern: **/coverage/cobertura-coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use **/target/site/cobertura/coverage.xml). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root.
Cobertura must be configured to generate XML reports for this plugin to function.

Consider only stable builds

Include only stable builds, i.e. exclude unstable and failed ones.

Fail builds if no reports

fail builds if No coverage reports are found.

Fail unhealthy builds

Unhealthy projects will be failed.

Fail unstable builds

Unstable projects will be failed.

Health auto update

Auto update threshold for health on successful build.

Stability auto update

Auto update threshold for stability on successful build.

Zoom coverage chart

Zoom the coverage chart and crop area below the minimum and above the maximum coverage of the past reports.

Maximum number of builds: 0

Only graph the most recent N builds in the coverage chart, 0 disables the limit.

Source Encoding: ASCII

Encoding when showing files.

Coverage Metric Targets

Methods	刪除	40.0	20.0	30.0
Lines	刪除	40.0	20.0	30.0
Conditionals	刪除	40.0	20.0	30.0

新增

Configure health reporting thresholds.
For the row, leave blank to use the default value (i.e. 80).
For the and rows, leave blank to use the default values (i.e. 0).

刪除

新增建置後動作

儲存 套用

SSH

為 Jenkins User 建立 SSH Keys

SSH 在自動化的過程扮演很重要的角色，因此我們需要先將 jenkins 的 ssh key 產生出來

透過 jenkins user 建立 ssh key

首先透過下面指令切換到 jenkins 這個 user

```
sudo su - jenkins
```

接著透過下面指令產生 ssh key

```
ssh-keygen -t rsa
```

default 會產生在 `~/.ssh` 將會有 `id_rsa` , `id_rsa.pub` 這兩個檔案

將建立好的 public key 加入到要傳輸的目標機器

```
cat ~/.ssh/id_rsa.pub >>~/.ssh/authorized_keys
```

有了 ssh key 以便後續自動化進行處理，在 jenkins user 底下執行上述指令，也就表示：

> 把本機當作 stage machine

這樣做的好處是，一旦 stage machine 在其他位置

只要改變 ssh server ip 並且再把 id_rsa.pub 加入對象機器即可，其他程序皆相通

如此，可以透過下面指令進行登入

```
ssh jenkins@localhost
```

若設置正確，將不需要 password 即可成功登入。

除了可以登入遠端機器之外，我們可以透過 ssh 對遠端進行指令的執行，如：

```
ssh jenkins@localhost id
```

會看到類似下面的結果

```
uid=112(jenkins) gid=117(jenkins)
groups=117(jenkins),27(sudo),999(docker)
```

使用 scp 傳輸檔案

scp 為用於檔案傳輸的指令，其所使用的驗證方式跟 ssh 一樣，因此只要 ssh key 設置好之後，使用 scp 也將不需要密碼

指令範例：

```
scp filename jenkins@localhost:~/path/to/upload
```

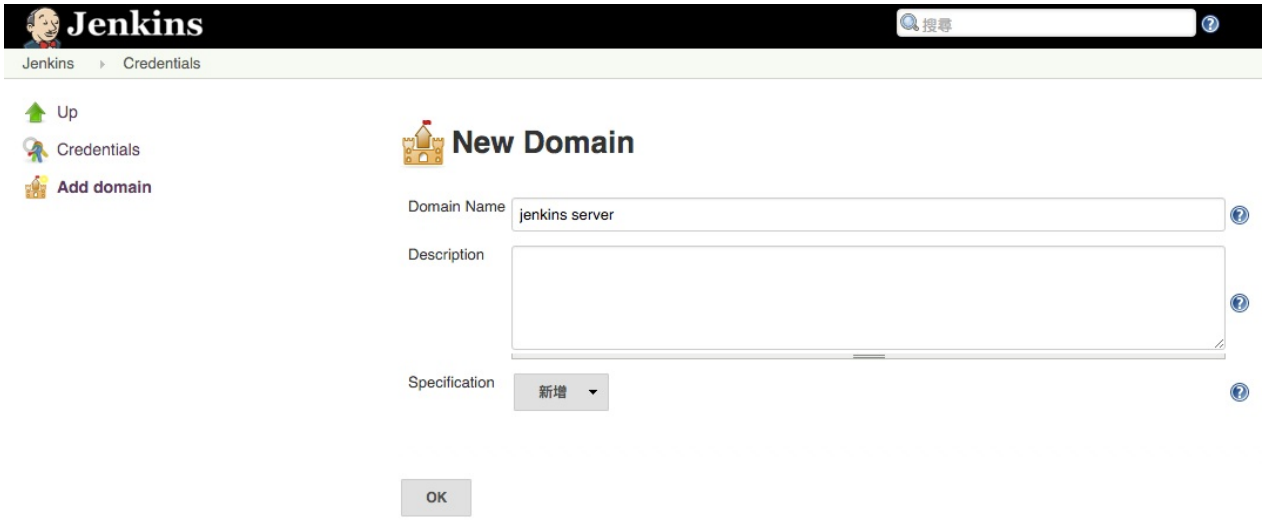
ssh 與 scp 組合應用範例

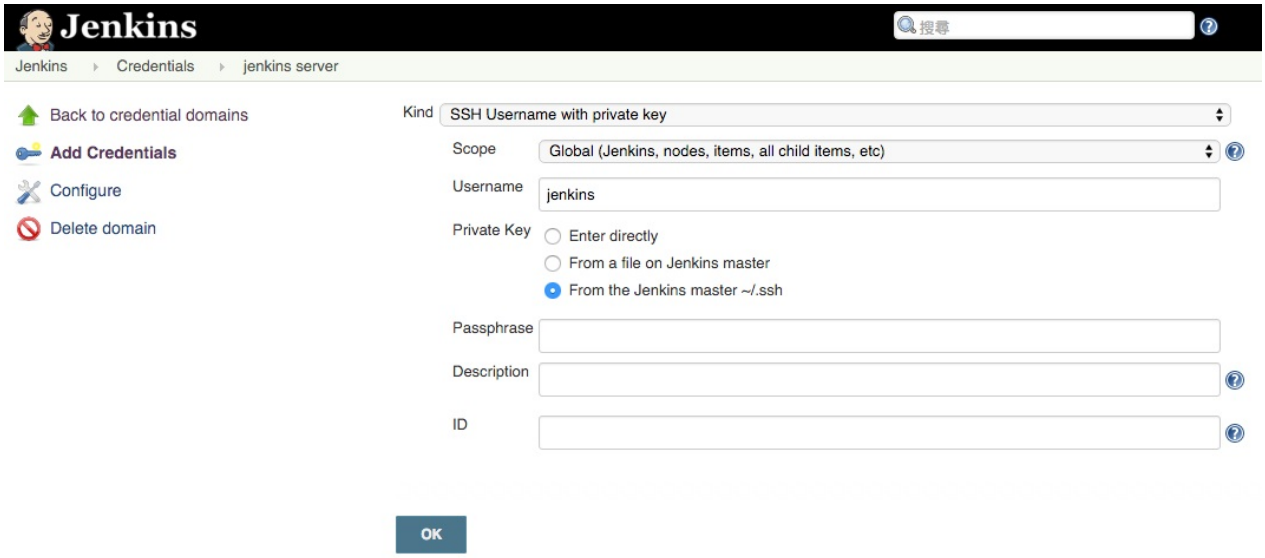
此部分將展示如何使用 ssh 建立資料夾，再將檔案透過 scp 傳輸到對應的資料夾

```
touch test-upload-file.md
ssh jenkins@localhost mkdir ~/test-upload-folder
scp test-upload-file.md jenkins@localhost:~/test-upload-folde
r
ssh jenkins@localhost ls ~/test-upload-folder
```

Jenkins 設定 ssh credentials

若要在 Jenkins 串接其他服務需要用到 ssh 認證的狀況，可參考下面設置





新增 ssh server

我們可以先把 jenkins CI server 視為 development 測試環境，因此我們需要進行 ssh server 相關設置

相關設置參考：plugin/publish over ssh 透過該章節設置，我們可以確保 ssh 連結正常

部署

release 指令使用 Makefile

```
deploy-default:
    ssh jenkins@localhost mkdir -p deploy/release
    scp target/spring-boot-sample-data-rest-0.1.0.jar jenkins
@localhost:deploy/release
    - ssh jenkins@localhost 'kill `cat deploy/release/run.pid
`'
    ssh jenkins@localhost 'java -jar deploy/release/spring-bo
ot-sample-data-rest-0.1.0.jar > /dev/null 2>&1 & echo $$! > "
deploy/release/run.pid"'
```

幾個重點如下：

```
- ssh jenkins@localhost 'kill `cat deploy/release/run.pid`'
```

上述指令開頭的 `-` 表示該指令執行失敗不影響最後結果。

`/dev/null 2>&1 & echo $$! >` 其中 `$$` 若在 terminal 執行只需要一個 `$` 因為 `$` 在 makefile 中為特殊字元，透過 `$$` 來進行跳脫，若在 terminal 執行，完整指令如下

`ssh jenkins@localhost 'java -jar deploy/release/spring-boot-sample-data-rest-0.1.0.jar > /dev/null 2>&1 & echo $! > "deploy/release/run.pid"'`

jenkins release bash shell

實際在 jenkins 中的 bash shell 只需要執行下面指令

```
make deploy-default
```

即可完成上述多行指令的執行

定時執行測試

使用時機

較常使用於測試任務，針對第三方 API 定時進行測試，確保相關應用運作正常

Ex: Facebook, Google 廣告 API

設置方式

設置每 5 分鐘執行一次，指令如下：

```
H/5 * * * *
```

建置觸發程序

- [ ] Build after other projects are built
- [x] 定期建置

排程

No schedules so will never run

這個欄位大致跟 cron 的語法一樣 (再加上一點點調整)。 明確定義如下，每一行都該是由 TAB 或空白隔開的五個欄位:

分 時 日 月 星期

分 一小時內的第幾分鐘 (0-59)

時 一天內的哪一小時 (0-23)

日 一個月內的第幾天 (1-31)

月 月份 (1-12)

星期 星期幾 (0-7)，0 跟 7 都代表星期日。

要在欄位裡指定多個值時，可以使用下列運算符號 (按照優先權排序):

- '*' 代表所有可以用的數值。
- 'M-N' 代表範圍區間，例如 "1-5"
- 'M-N/X' 或 '*/X' 代表區間內每隔 X 的數值，例如在「分」欄位裡的 "*/15" 表示 "0,15,30,45"，而 "1-6/2" 則是 "1,3,5"
- 'A,B,...,Z' 可以指定多個值，例如 "0,30" 或 "1,3,5"

為了在定期排程的時的系統負擔平均一點，可以使用 'H' 符號。 例如，大家常會把每天要執行的作業設成 "0 0 * * *"，可是這樣子會讓事情都擠到每天午夜時分去。 相較之下，設定成 "H H * * *" 也一樣每天都會跑一次，但是 Jenkins 會分散到不同時段執行。

'H' 符號也能指定範圍。就像 "H H(0-7) * * *" 代表子夜開始到早上 7:59 前的某個時間。

您可以把 'H' 符號想像成是在某個區間裡隨機挑出來的數值。 但事實上它只是作業名稱的 Hash 值，而不是隨機函式，所以對固定一個專案而言，這個數值是固定的。

空行，或是以 '#' 開頭的那幾行都會被當作註解忽略掉。

另外也支援 "@yearly" (每年), "@annually" (每季), "@monthly" (每月), "@weekly" (每週), "@daily" (每天), "@midnight" (半夜) 及 "@hourly" (每小時)。

```
範例 # 每 1 分鐘
     * * * * *
     # 整點後第 5 分鐘
     5 * * * *
```

設置 Poll SCM

使用時機

用於偵測是否有 commit 被 push 的事件，若有的話則進行建置

設置方式

test 成功自動進行 preview

test 成功自動進行 release 也是非常簡單，請自由發揮，不過一般來說 production release 會是比較嚴謹控管，建議由人員控制時機點

jenkins 2.0

課程難度：進階

Jenkins 2.0 版現在已經發佈。該版本增加了一種新的定義構建流程的方式，改善了新用戶的體驗。

Jenkins workflow plugin 並不是什麼新東西，但是2.0版本將提供一種新的核心特性（名字就叫 Pipeline），該特性允許用戶在 DSL（Domain Specific Language）的幫助下定義他們的構建流程。DSL 是 Groovy 其中之一的特性，允許把「建構設定(build definitioin)」當作一般程式碼撰寫。因此，建構設定就不只是定義在 Jenkins 的 task，而是可以被版本控制。

其實在此之前，有一些其他 Plugin 用來解決同樣的問題，但是 Jenkins 的創建者 Kohsuke Kawaguchi 對 InfoQ 說 Pipeline 不只是一款 Plugin。

實際上 Pipeline 是由一系列 Plugin 組成的一種意義重大的子系統。關於 2.0 版，我們其中一部分想法是想讓用戶拋棄「從內核開始然後安裝插件」的心態。取而代之的是，當你拿到 Jenkins 2.0 版的時候，你就會得到我們認為能覆蓋 80% 使用場景的功能，其中有些來自內核，有些來自 Plugin。 把管道功能作為核心特性也是業界中其他同行的做法。

紐約市的一名軟體工程師 Jacques Chester 認為，在關注 Jenkins Pipeline 功能的開發者也應該瞭解一下 Concourse。在一次 InfoQ 的採訪中，Chester 說Jenkins 的 Pipeline Plugin 系統其實是為其固有缺陷上的 Patch。

該系統將「通過 Plugin 而實現 Concourse 從設計之初就具有的部分功能從而重塑 Jenkins。Concourse 的基因是可以對整合流程進行版本控制，用完即可丟棄的構建環境以及可以將任務計劃委派給工具而不是手動組裝」。

另外，2.0 版的「Getting Started」有了一種新的體驗，在安裝插件方面為用戶提供一些建議。

課程內容

此章節將介紹 Jenkins 2.0 的新特性，以及在前面章節關於 Docker 的應用進行整合，透過 Pipeline 進行 CI 建置。

資料來源

http://www.infoq.com/cn/news/2016/04/jenkins-2-beta-pipeline-system

Pipeline 使用

Pipeline 是以 Groovy Script 作為撰寫，為 Java 的動態語言。

第一個 Pipeline: Hello World

```
echo 'hello from Pipeline'
```

執行結果

```
Started by user anonymous
[Pipeline] echo
hello from Pipeline
[Pipeline] End of Pipeline
Finished: SUCCESS
```

以上面的例子來說，完整的語法為：

```
echo("hello from Pipeline");
```

我們可以省略 `;`，`()` 並且 `''`，`""` 皆可以使用。

建立簡單的 Pipeline

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'
  def mvnHome = tool 'M3'
  sh "${mvnHome}/bin/mvn test"
}
```

其中

```
git url: 'https://github.com/agileworks-tw/spring-boot-sample
.git'
```

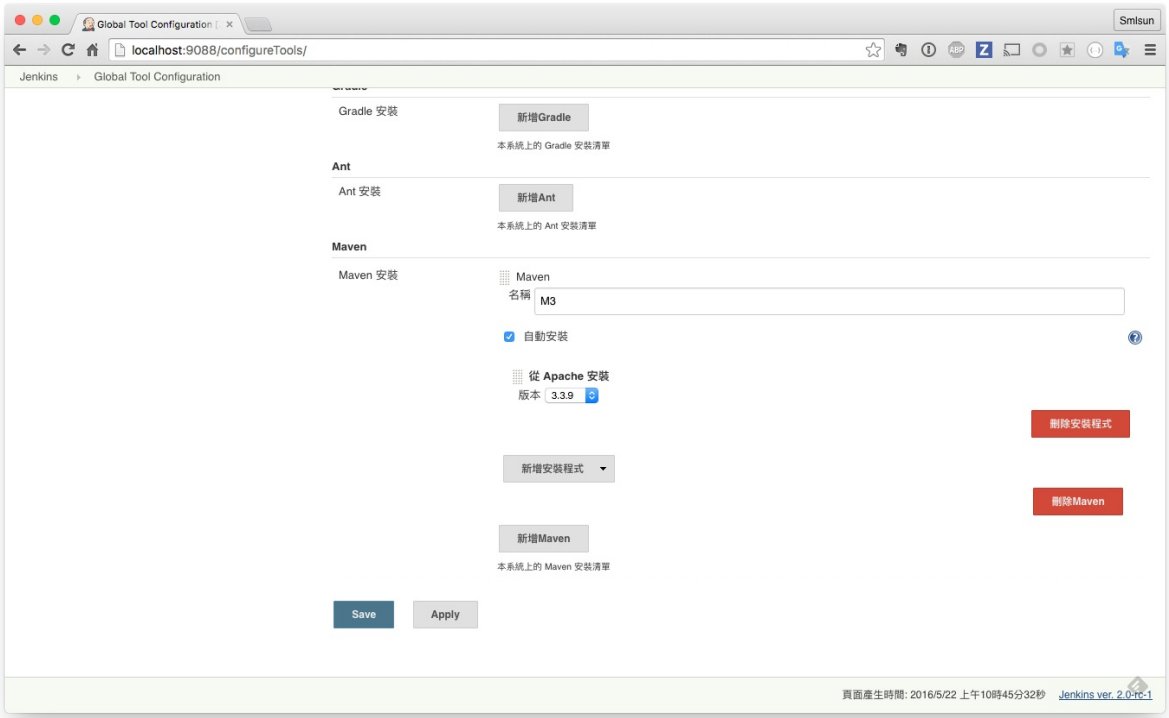
也可以指定 branch，指令如下

```
git url: 'https://github.com/agileworks-tw/spring-boot-sample
.git', branch: 'master'
```

若沒有指定的話，預設為 Master，完整程式碼如下

```
git([url: 'https://github.com/agileworks-tw/spring-boot-sampl
e.git', branch: 'master'])
```

tool 實際上是讀取 jenkins 中的 `Global Tool Configuration` 所設定的建置工具，如下圖：




```
def mvnHome = tool 'M3'
```

就是取得該 tool 定義的結果來進行專案建置。

而下面指令

```
sh "${mvnHome}/bin/mvn test"
```

則為使用該 tool 利用 mvn 進行專案 test，而實際指令為

```
sh([script: mvnHome + '/bin/mvn test'])
```

可以看到簡化不少

for windows 可以改為

```
bat "${mvnHome}\\bin\\mvn test"
```

管理環境變數

在 Pipeline 我們可以使用 env 來存取 jenkins 所設置的環境變數，我們也可以使用 env 來設置新的環境變數。

延續上一個範例：

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'
  def mvnHome = tool 'M3'
  env.PATH = "${mvnHome}/bin:${env.PATH}"
  sh 'mvn test'
}
```

透過 env 的使用，我們就不需要使用 `${mvnHome}/bin/mvn` 這麼長的程式路徑。

上述的處理方式，只有當你使用但一 slave 時比較沒有問題，有時候我們需要指定某項建置需要特定的環境變數時，我們可以改為

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'
  withEnv(["PATH+MAVEN=${tool 'M3'}/bin"]) {
    sh 'mvn test'
  }
}
```

`withEnv(["PATH+MAVEN=${tool 'M3'}/bin"]) {` 得意思就是原有的 `PATH` 變數，加上新的 `Maven` 環境變數

如此就可以限制該環境變數限制在特定的 scope 一但執行完成，環境變數就會還原成原本的內容。

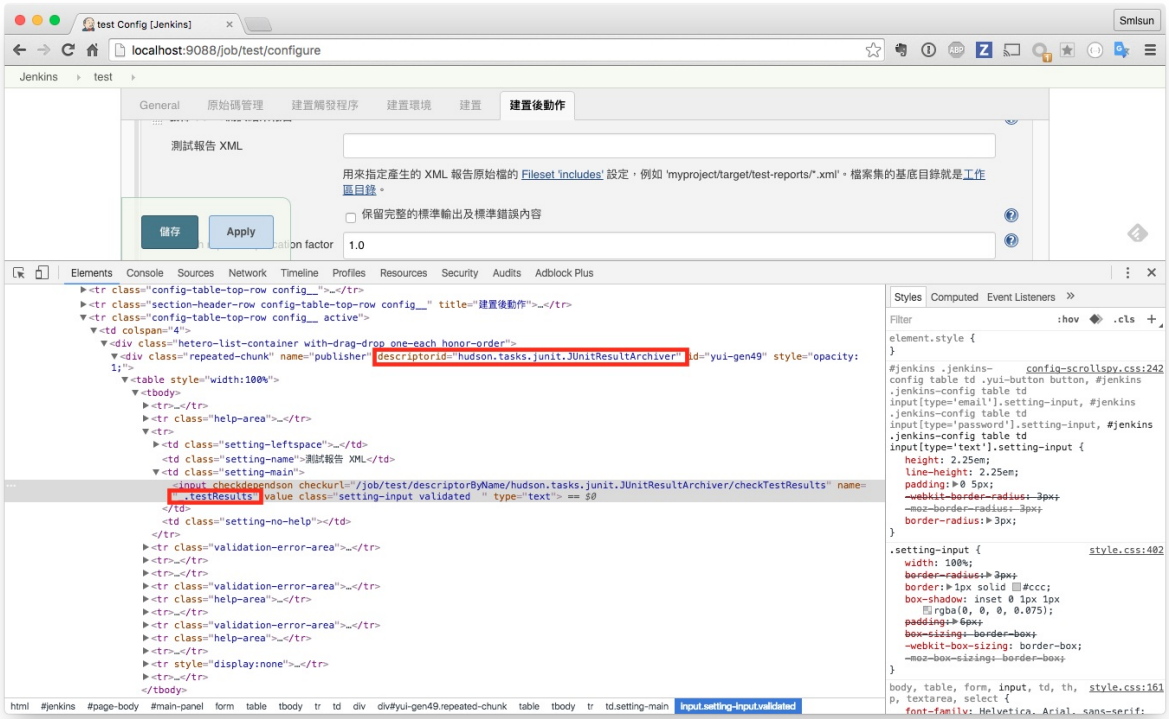
除此之外，一些 Jnekins 本來在建置過程中就會可用的變數也可以透過 env 直接存取，比如 `env.BUILD_TAG` 可在 groovy script 中使用，或是在 `sh` 時使用 `$BUILD_TAG` 進行該變數存取。

紀錄測試結果，以及建置封存

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'
  def mvnHome = tool 'M3'
  sh "${mvnHome}/bin/mvn test package"
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.j
ar', fingerprint: true])
  step([$class: 'JUnitResultArchiver', testResults: '**/targe
t/surefire-reports/TEST-*.xml'])
}
```

其中 `$class` 為縮寫，以 JUnitResultArchiver 為例，完整路徑為 `hudson.tasks.junit.JUnitResultArchiver`

可以從 jenkins UI 找到一些線索，如下圖：



指定特定 Slave 進行建置

假設已經有設置 jenkins slave 節點

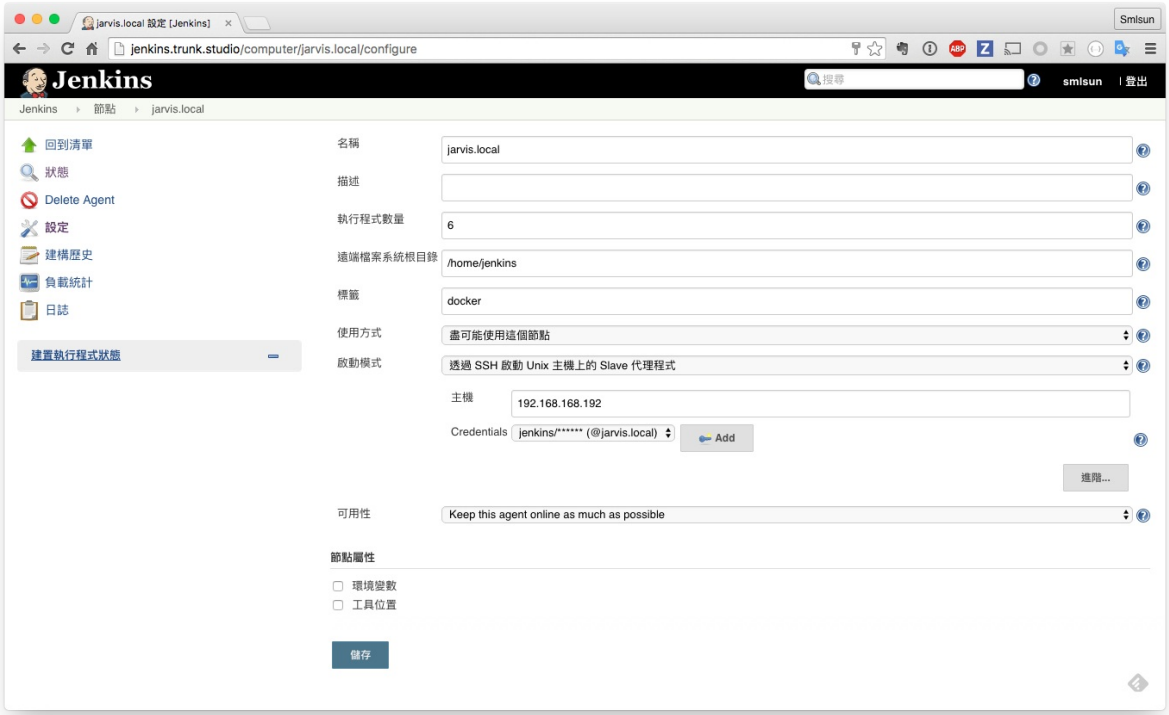
```
node('master') {
    // as before
}
```

我們可以透過上述方式指定特定的 node 進行建置，若沒有指定將由 jenkins 自行判斷

除了節點的 name，也可以指定特定的 label，如

```
node('unix && 64bit') {
    // as before
}
```

定義方式可參考下圖



若點選建置，你將可以看到下列訊息輸出

```
Running on <yourslavename> in /<slaveroot>/workspace/<jobname>
```

中途暫停

```
node('remote') {
  input 'Ready to go?'
  // rest as before
}
```

使用情境如：用於確認 QA 是否正確，若確認完成進行 production deploy，又或者輸入特定參數。

使用 try... catch... final

我們在進行建置時，就像在寫程式一樣，一定會有另外產生，同樣我們可以透過 `try... catch... final` 來定義，如：

```
try{
  stage 'Approve, go production'
  def url = 'http://localhost:8000/'
  input message: "Does staging at $url look good? ", ok: "Dep
loy to production"
}finally{
  sh "docker-compose stop server"
}
```

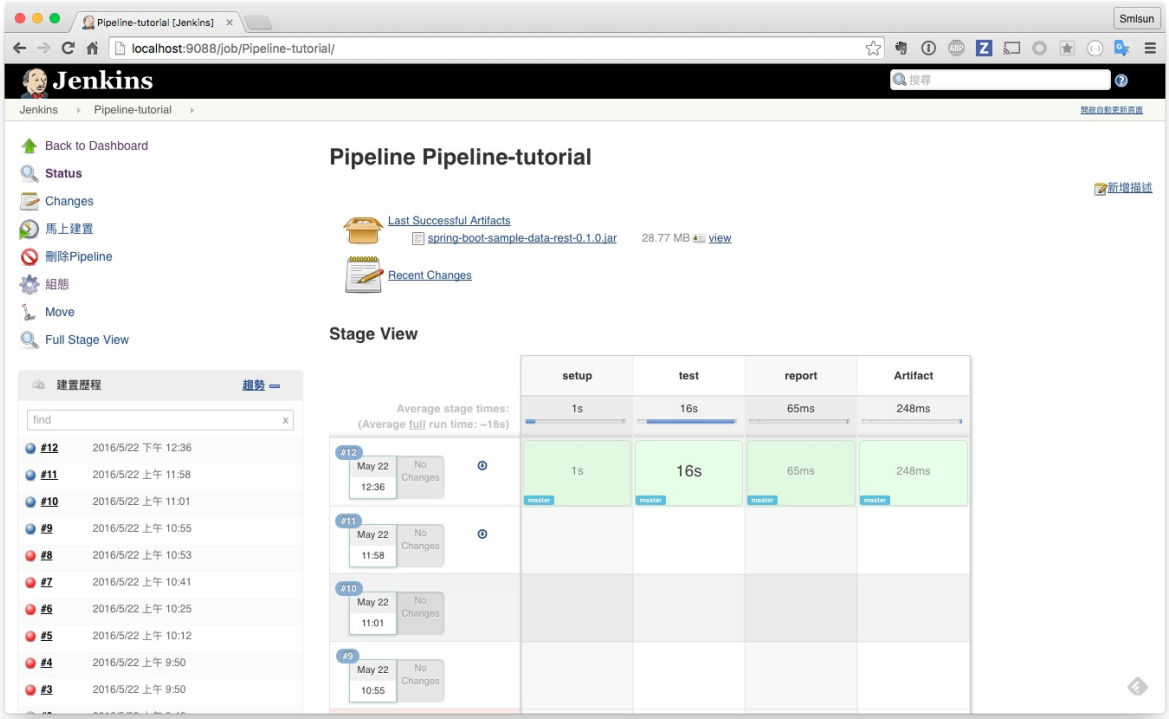
使用 docker 進行 QA 驗證，無論最後是否要要進行 production deploy 皆需要將 docker 停止。

stage 定義

stage 的定義，可以讓你的建置步驟更清楚，可讀性更好

```
node {
  stage 'setup'
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'
  def mvnHome = tool 'M3'

  stage 'test'
  sh "${mvnHome}/bin/mvn test package"


  stage 'report'
  step([$class: 'JUnitResultArchiver', testResults: '**/targe
t/surefire-reports/TEST-*.xml'])

  stage 'Artifact'
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.j
ar', fingerprint: true])

}
```

且 jenkins 在 task 中 stage view 進度上可以看到每個 stage 的執行過程與結果，如下圖：


Pipeline-tutorial [Jenkins]
localhost:9088/job/Pipeline-tutorial/
Jenkins
Jenkins › Pipeline-tutorial
Back to Dashboard
Status
Changes
馬上建置
刪除Pipeline
組態
Move
Full Stage View
建置歷程
趨勢
#12 2016/5/22 下午 12:36
#11 2016/5/22 上午 11:58
#10 2016/5/22 上午 11:01
#9 2016/5/22 上午 10:55
#8 2016/5/22 上午 10:53
#7 2016/5/22 上午 10:41
#6 2016/5/22 上午 10:25
#5 2016/5/22 上午 10:12
#4 2016/5/22 上午 9:50
#3 2016/5/22 上午 9:50
Pipeline Pipeline-tutorial
新增描述
Last Successful Artifacts
spring-boot-sample-data-rest-0.1.0.jar 28.77 MB view
Recent Changes
Stage View
setup
test
report
Artifact
Average stage times:
(Average full run time: ~18s)
1s
16s
65ms
248ms
#12 May 22 12:36 No Changes
#11 May 22 11:58 No Changes
#10 May 22 11:01 No Changes
#9 May 22 10:55 No Changes

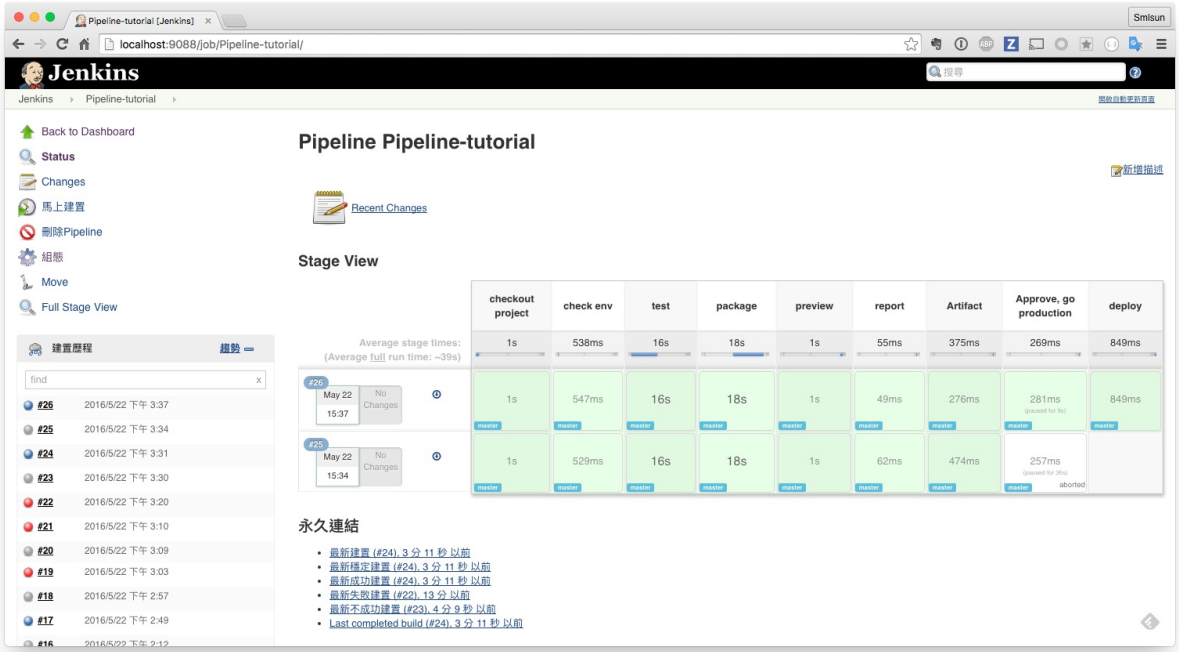
使用 Pipeline 進行建置

我們可以把前面章節所完成的相關 task 濃縮為單一個 Jenkinsfile

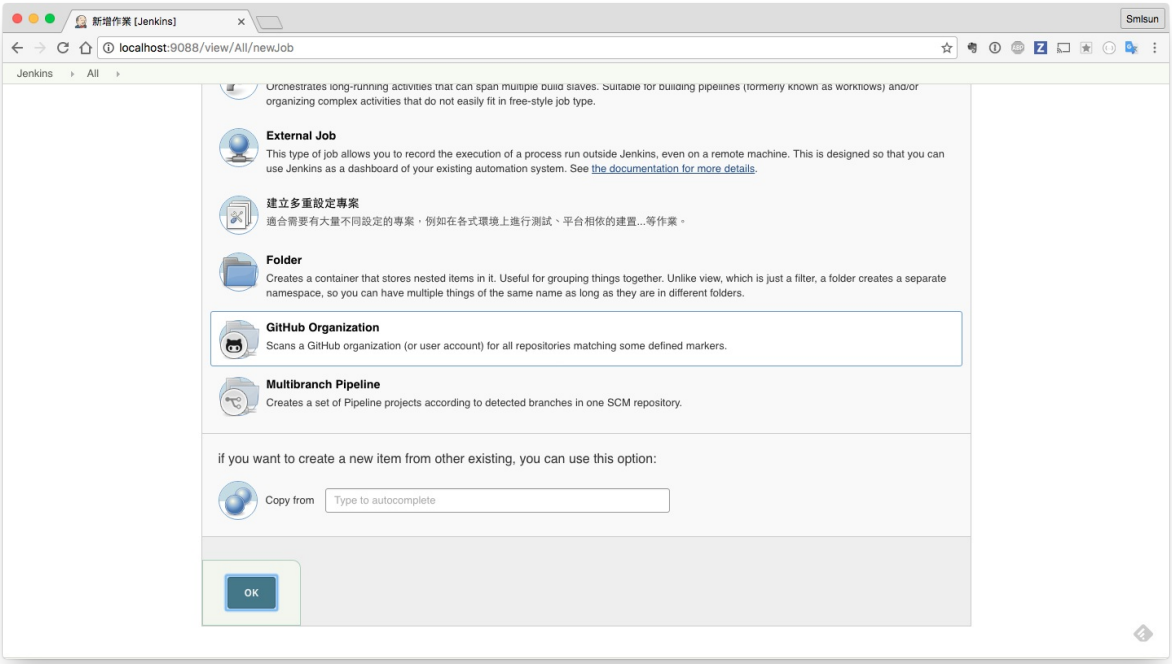
```
node {

  stage 'checkout project'
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'

  stage 'check env'

  sh "mvn -v"
  sh "java -version"

  stage 'test'
  sh "mvn test"

  stage 'package'
  sh "mvn package"

  stage 'preview'
  sh 'make deploy-default'

  stage 'report'
  step([$class: 'JUnitResultArchiver', testResults: '**/targe
t/surefire-reports/TEST-*.xml'])

  stage 'Artifact'
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.j
ar', fingerprint: true])

  try{
    stage 'Approve, go production'
    def url = 'http://localhost:8000/'
    input message: "Does staging at $url look good? ", ok: "D
```

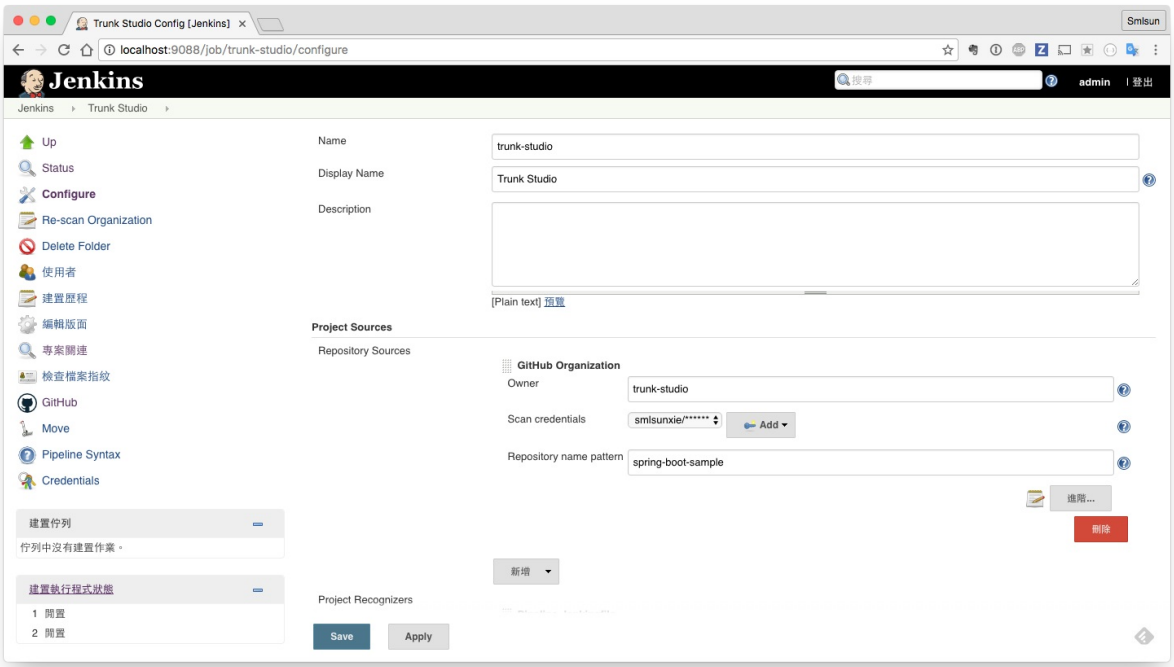
```
eploy to production"
  }finally{
    sh "ssh jenkins@localhost 'kill `cat deploy/release/run.p
id`'"
  }

  stage 'deploy'
  sh 'make deploy-default'
}
```

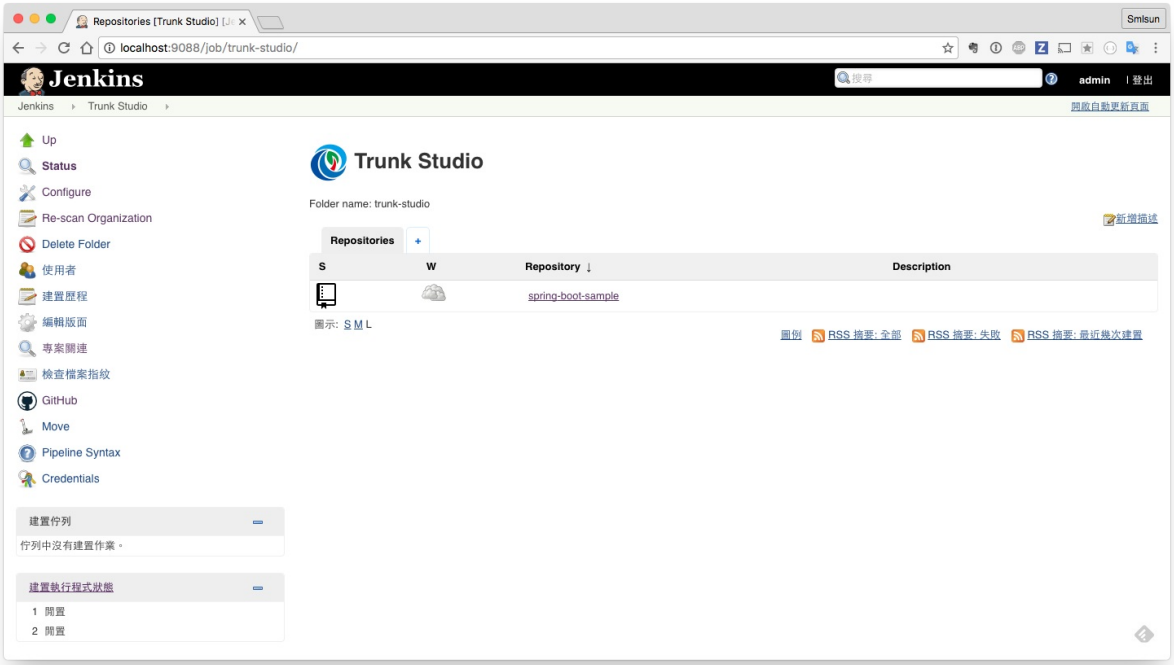
執行結果如下

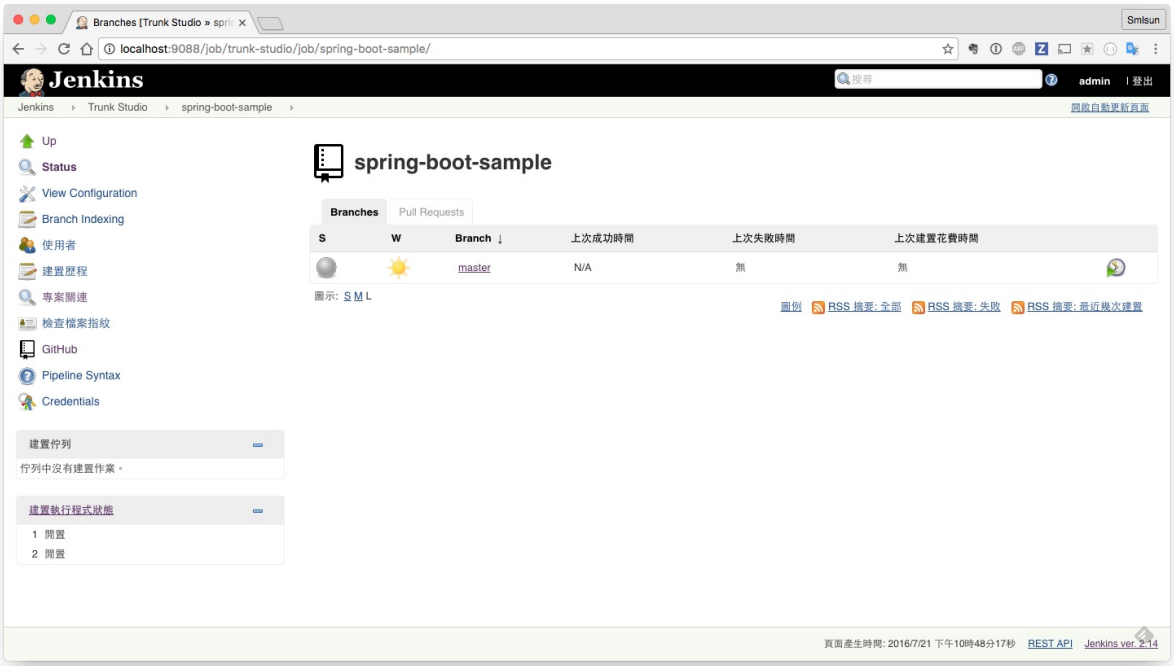


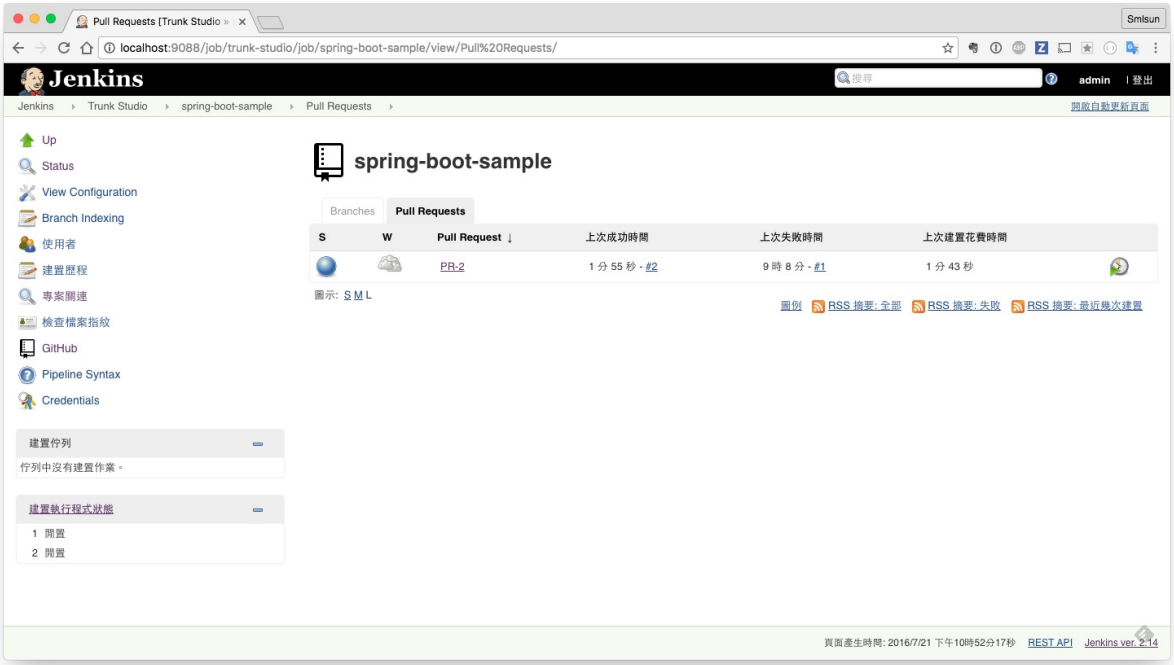
github-organization

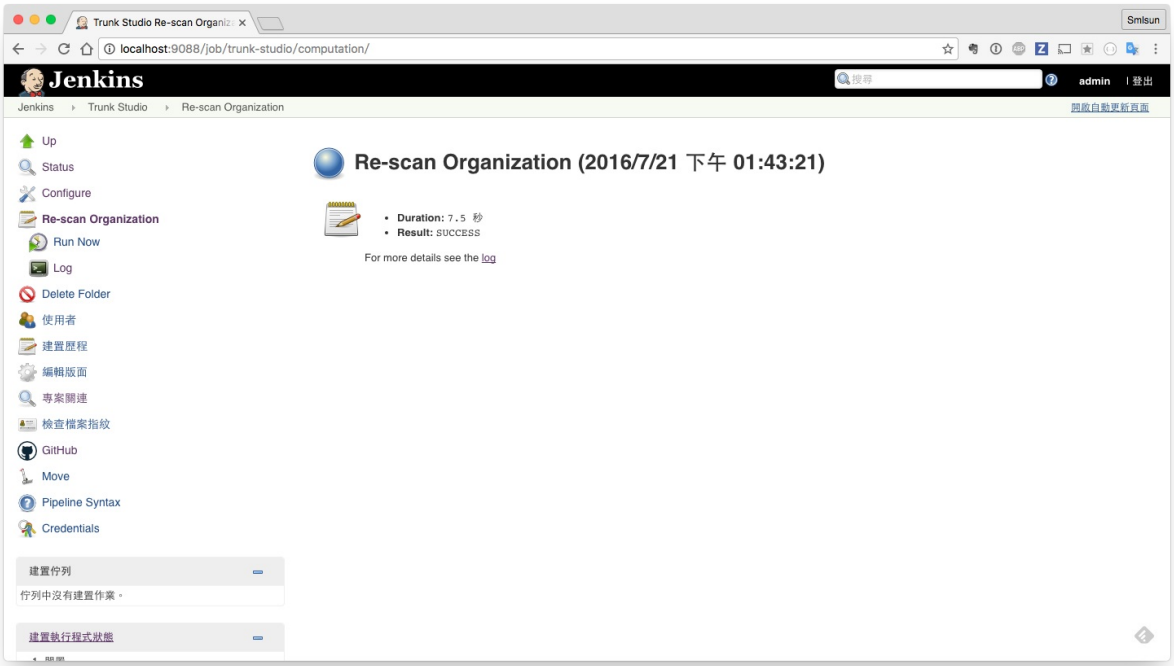


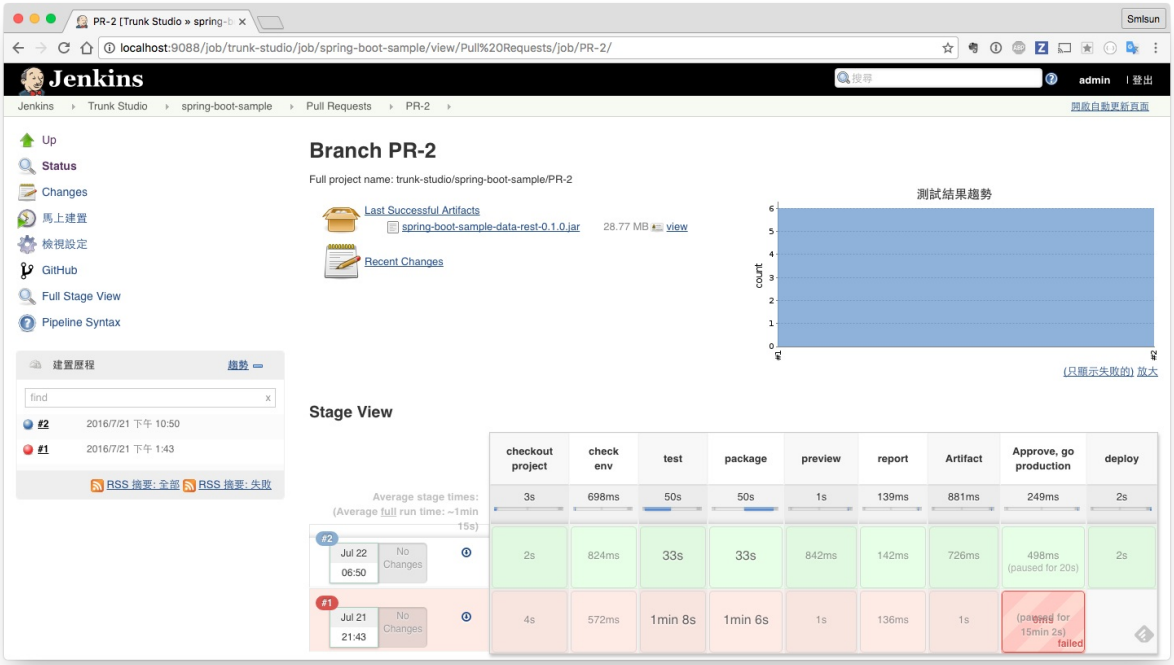


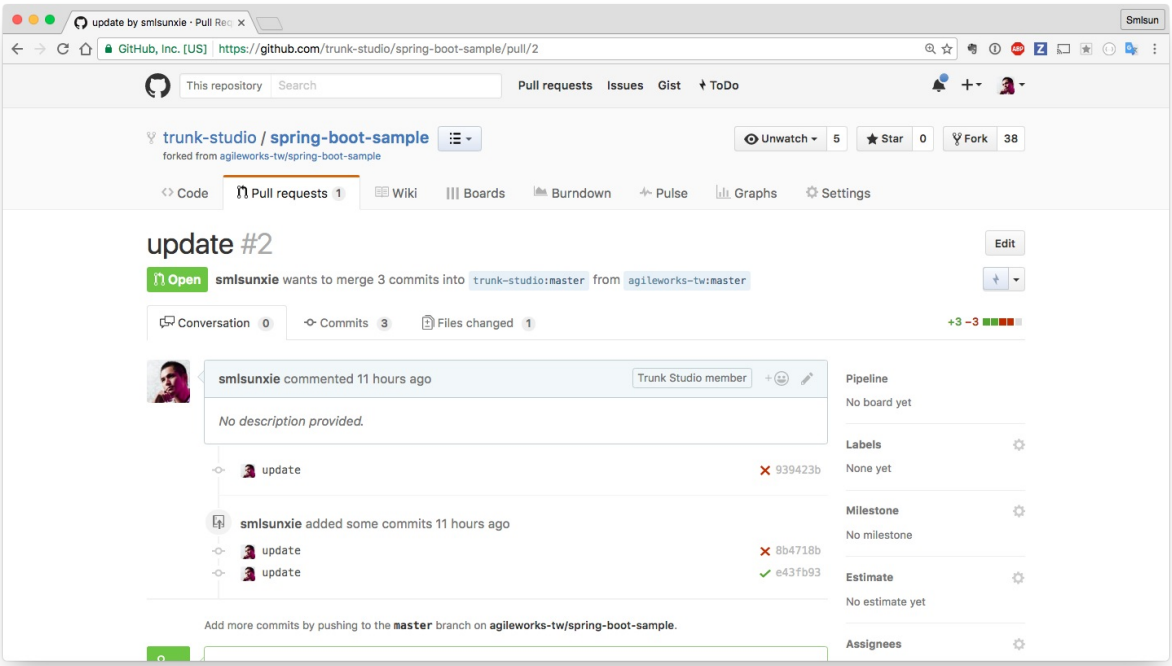

Repositories [Trunk Studio]
localhost:9088/job/trunk-studio/
Jenkins
admin
登出
Jenkins
Trunk Studio
開啟自動更新頁面
Up
Status
Configure
Re-scan Organization
Delete Folder
使用者
建置歷程
編輯版面
專案關連
檢查檔案指紋
GitHub
Move
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
1 閒置
2 閒置
Trunk Studio
Folder name: trunk-studio
新增描述
Repositories
S
W
Repository
Description
spring-boot-sample
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置


Branches [Trunk Studio » spri
localhost:9088/job/trunk-studio/job/spring-boot-sample/
Jenkins
admin
登出
Jenkins
Trunk Studio
spring-boot-sample
開啟自動更新頁面
Up
Status
View Configuration
Branch Indexing
使用者
建置歷程
專案關連
檢查檔案指紋
GitHub
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
1 閒置
2 閒置
spring-boot-sample
Branches
Pull Requests
S
W
Branch
上次成功時間
上次失敗時間
上次建置花費時間
master
N/A
無
無
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
頁面產生時間: 2016/7/21 下午10時48分17秒
REST API
Jenkins ver. 2.14

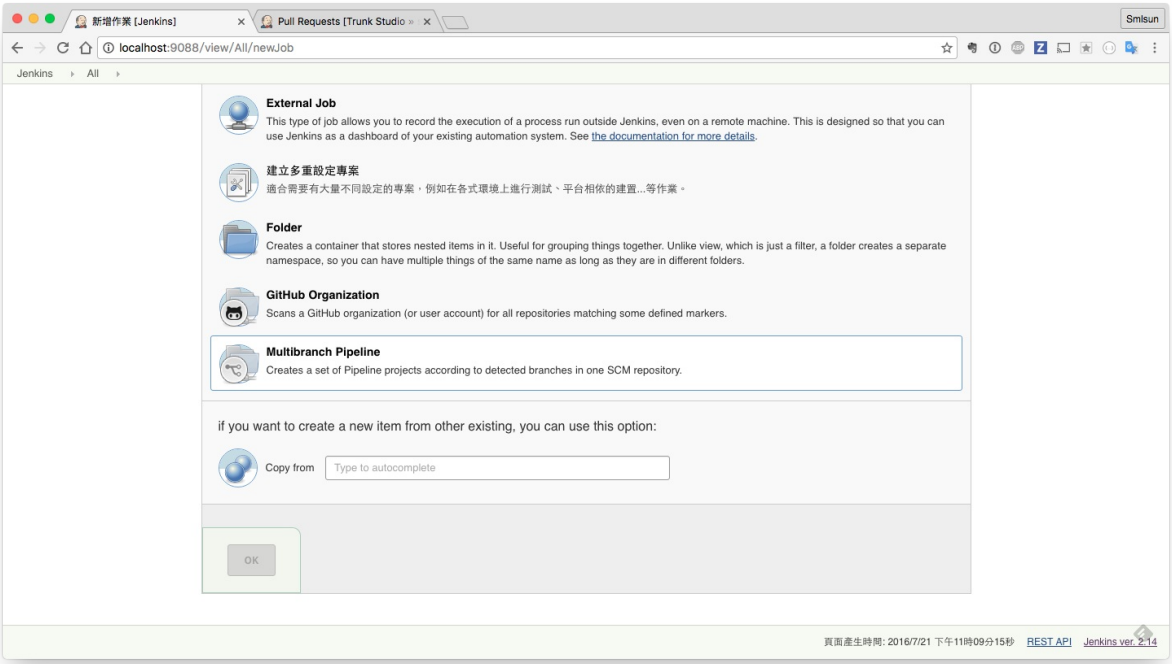

Pull Requests [Trunk Studio
localhost:9088/job/trunk-studio/job/spring-boot-sample/view/Pull%20Requests/
Jenkins
admin
Jenkins
Trunk Studio
spring-boot-sample
Pull Requests
Up
Status
View Configuration
Branch Indexing
使用者
建置歷程
專案關連
檢查檔案指紋
GitHub
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
1 閒置
2 閒置
spring-boot-sample
Branches
Pull Requests
S
W
Pull Request
上次成功時間
上次失敗時間
上次建置花費時間
PR-2
1 分 55 秒 - #2
9 時 8 分 - #1
1 分 43 秒
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
REST API
Jenkins ver. 2.14

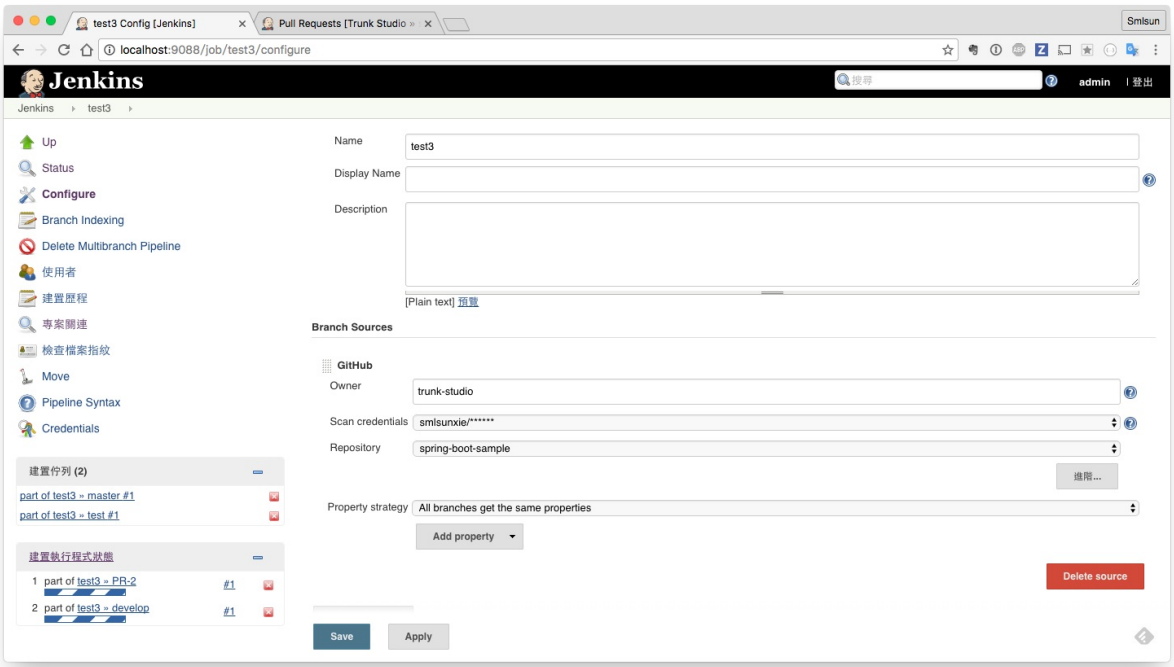

Trunk Studio Re-scan Organiz
localhost:9088/job/trunk-studio/computation/
Jenkins
admin
Jenkins
Trunk Studio
Re-scan Organization
Up
Status
Configure
Re-scan Organization
Run Now
Log
Delete Folder
使用者
建置歷程
編輯版面
專案關連
檢查檔案指紋
GitHub
Move
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
Re-scan Organization (2016/7/21 下午 01:43:21)
Duration: 7.5 秒
Result: SUCCESS
For more details see the log

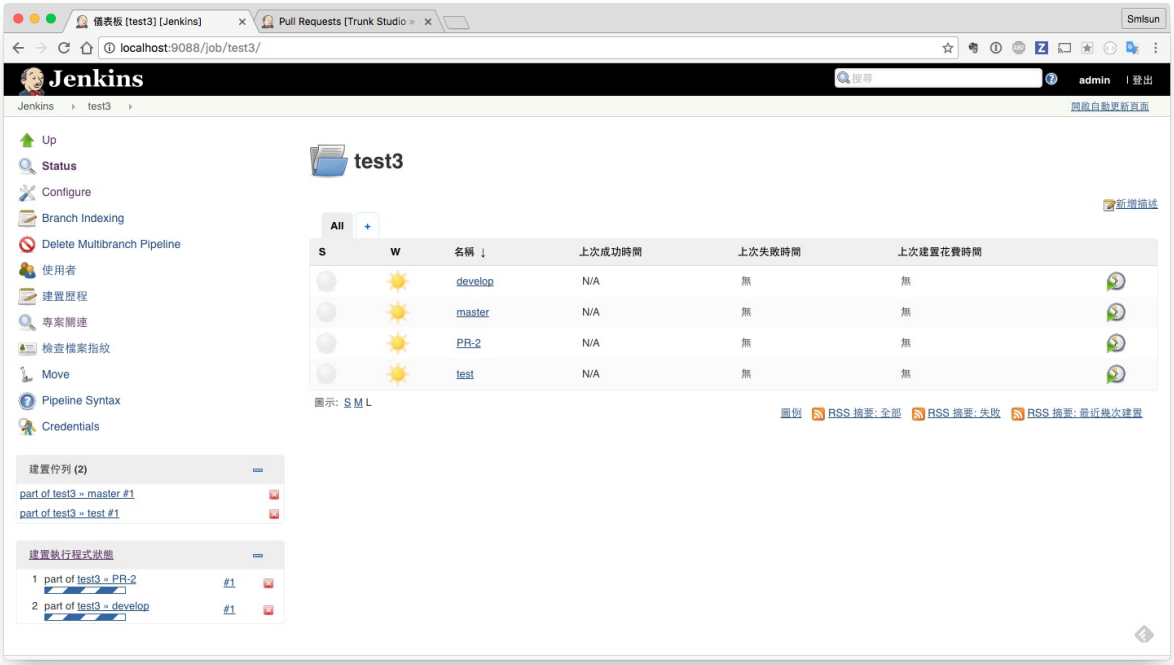

PR-2 [Trunk Studio » spring-
localhost:9088/job/trunk-studio/job/spring-boot-sample/view/Pull%20Requests/job/PR-2/
Jenkins
admin
Jenkins › Trunk Studio › spring-boot-sample › Pull Requests › PR-2
Up
Status
Changes
馬上建置
檢視設定
GitHub
Full Stage View
Pipeline Syntax
建置歷程
趨勢
#2 2016/7/21 下午 10:50
#1 2016/7/21 下午 1:43
RSS 摘要: 全部
RSS 摘要: 失敗
Branch PR-2
Full project name: trunk-studio/spring-boot-sample/PR-2
Last Successful Artifacts
spring-boot-sample-data-rest-0.1.0.jar 28.77 MB view
Recent Changes
測試結果趨勢
count
(只顯示失敗的) 放大
Stage View
checkout project
check env
test
package
preview
report
Artifact
Approve, go production
deploy
Average stage times:
(Average full run time: ~1min 15s)
3s
698ms
50s
50s
1s
139ms
881ms
249ms
2s
#2
Jul 22 06:50
No Changes
2s
824ms
33s
33s
842ms
142ms
726ms
498ms (paused for 20s)
2s
#1
Jul 21 21:43
No Changes
4s
572ms
1min 8s
1min 6s
1s
136ms
1s
failed

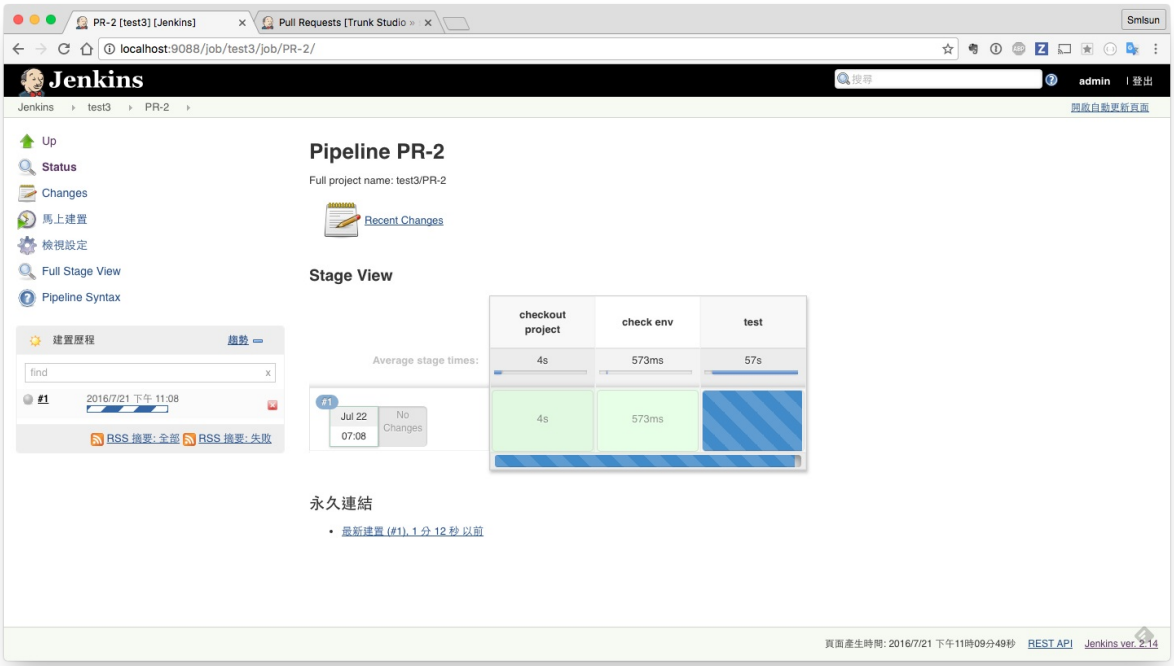

update by smlsunxie · Pull Re
GitHub, Inc. [US] https://github.com/trunk-studio/spring-boot-sample/pull/2
This repository
Search
Pull requests
Issues
Gist
ToDo
trunk-studio / spring-boot-sample
forked from agileworks-tw/spring-boot-sample
Unwatch 5
Star 0
Fork 38
Code
Pull requests 1
Wiki
Boards
Burndown
Pulse
Graphs
Settings
update #2
Edit
Open
smlsunxie wants to merge 3 commits into trunk-studio:master from agileworks-tw:master
Conversation 0
Commits 3
Files changed 1
+3 −3
smlsunxie commented 11 hours ago
Trunk Studio member
No description provided.
update
939423b
smlsunxie added some commits 11 hours ago
update
8b4718b
update
e43fb93
Add more commits by pushing to the master branch on agileworks-tw/spring-boot-sample.
Pipeline
No board yet
Labels
None yet
Milestone
No milestone
Estimate
No estimate yet
Assignees

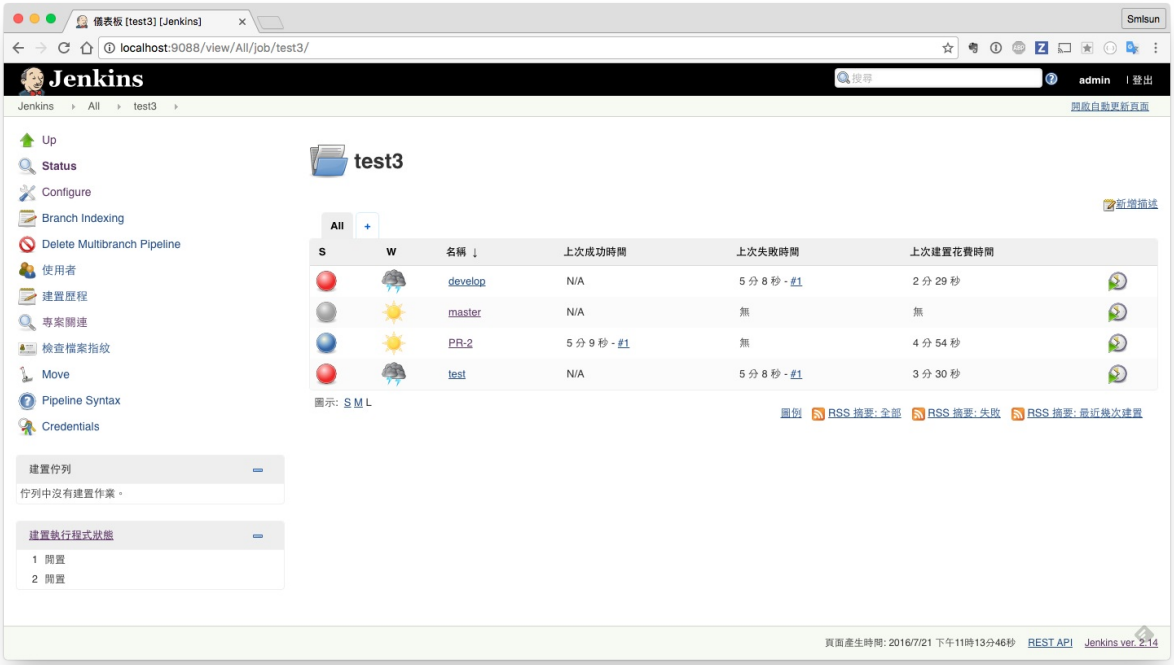
Multibranch Pipeline






儀表板 [test3] [Jenkins]
Pull Requests [Trunk Studio »
localhost:9088/job/test3/
Jenkins
admin
登出
Jenkins › test3 ›
開啟自動更新頁面
Up
Status
Configure
Branch Indexing
Delete Multibranch Pipeline
使用者
建置歷程
專案關連
檢查檔案指紋
Move
Pipeline Syntax
Credentials
test3
新增描述
All
S
W
名稱 ↓
上次成功時間
上次失敗時間
上次建置花費時間
develop
master
PR-2
test
N/A
無
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
建置佇列 (2)
part of test3 » master #1
part of test3 » test #1
建置執行程式狀態
1 part of test3 » PR-2 #1
2 part of test3 » develop #1


PR-2 [test3] [Jenkins]
Pull Requests [Trunk Studio »
localhost:9088/job/test3/job/PR-2/
Jenkins
admin
登出
Jenkins › test3 › PR-2 ›
開啟自動更新頁面
Up
Status
Changes
馬上建置
檢視設定
Full Stage View
Pipeline Syntax
Pipeline PR-2
Full project name: test3/PR-2
Recent Changes
Stage View
checkout project
check env
test
Average stage times:
4s
573ms
57s
#1
Jul 22
07:08
No Changes
建置歷程
趨勢
find
#1
2016/7/21 下午 11:08
RSS 摘要: 全部
RSS 摘要: 失敗
永久連結
最新建置 (#1), 1 分 12 秒 以前
REST API
Jenkins ver. 2.14


儀表板 [test3] [Jenkins]
localhost:9088/view/All/job/test3/
Jenkins
admin
登出
Jenkins › All › test3 ›
開啟自動更新頁面
Up
Status
Configure
Branch Indexing
Delete Multibranch Pipeline
使用者
建置歷程
專案關連
檢查檔案指紋
Move
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
1 閒置
2 閒置
test3
新增描述
All
S
W
名稱 ↓
上次成功時間
上次失敗時間
上次建置花費時間
develop
N/A
5 分 8 秒 - #1
2 分 29 秒
master
N/A
無
無
PR-2
5 分 9 秒 - #1
無
4 分 54 秒
test
N/A
5 分 8 秒 - #1
3 分 30 秒
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
頁面產生時間: 2016/7/21 下午11時13分46秒
REST API
Jenkins ver. 2.14

設置建置環境

Jenkins 有許多社群貢獻的第三方 Plugins：

- NodeJS Plugin
- Rvm

但是以筆者過去經驗來說，就遇過安裝 Node.js Plugin 就會影響 RVM Plugin 的運作，安裝過多的 Plugins 可能導致許多問題。

除非您已經瞭解這些 Plugins 的特性及可能帶來的副作用，否則並不建議一開始就安裝太多 Plugins。

事實上絕大多數的建置流程，都可以利用 Jenkins 內建的任務定義方式來完成；如果搭配 Docker 來用更可以大幅化簡開發環境的配置工作。

一般來說，我們建議直接使用執行 Jenkins 程式的使用者帳號，把相關所需的環境設置完成。

配置 Jenkins 使用者的環境

在 Ubuntu Linux 利用 `apt-get` 完成 Jenkins 安裝後，會自動產生名為 `jenkins` 的系統使用者帳號（user）及群組（group）。

`/etc/passwd`

```
jenkins:x:117:125:Jenkins,,,:/var/lib/jenkins:/bin/bash
```

`/etc/group`

```
jenkins:x:125:
```

在 Jenkins 定義的任務，預設會使用 `jenkins` 使用者的權限進行操作，因此並非所有的檔案皆可任意讀寫或執行。

檔案權限的觀念在 Linux 系統中很重要，你在自己或其他帳號的家目錄中安裝的程式或放置的檔案，並不一定能夠被 Jenkins 的任務使用。

如果你希望幫 `jenkins` 帳號增加一些專屬的配置，需要在終端機中使用 `sudo su jenkins` 可以切換成 `jenkins` 使用者的身份進行各項操作，該 user 的家目錄（Home Directory）位置是 `/var/lib/jenkins` 。

不使用 plugin 來安裝相關 library 的好處

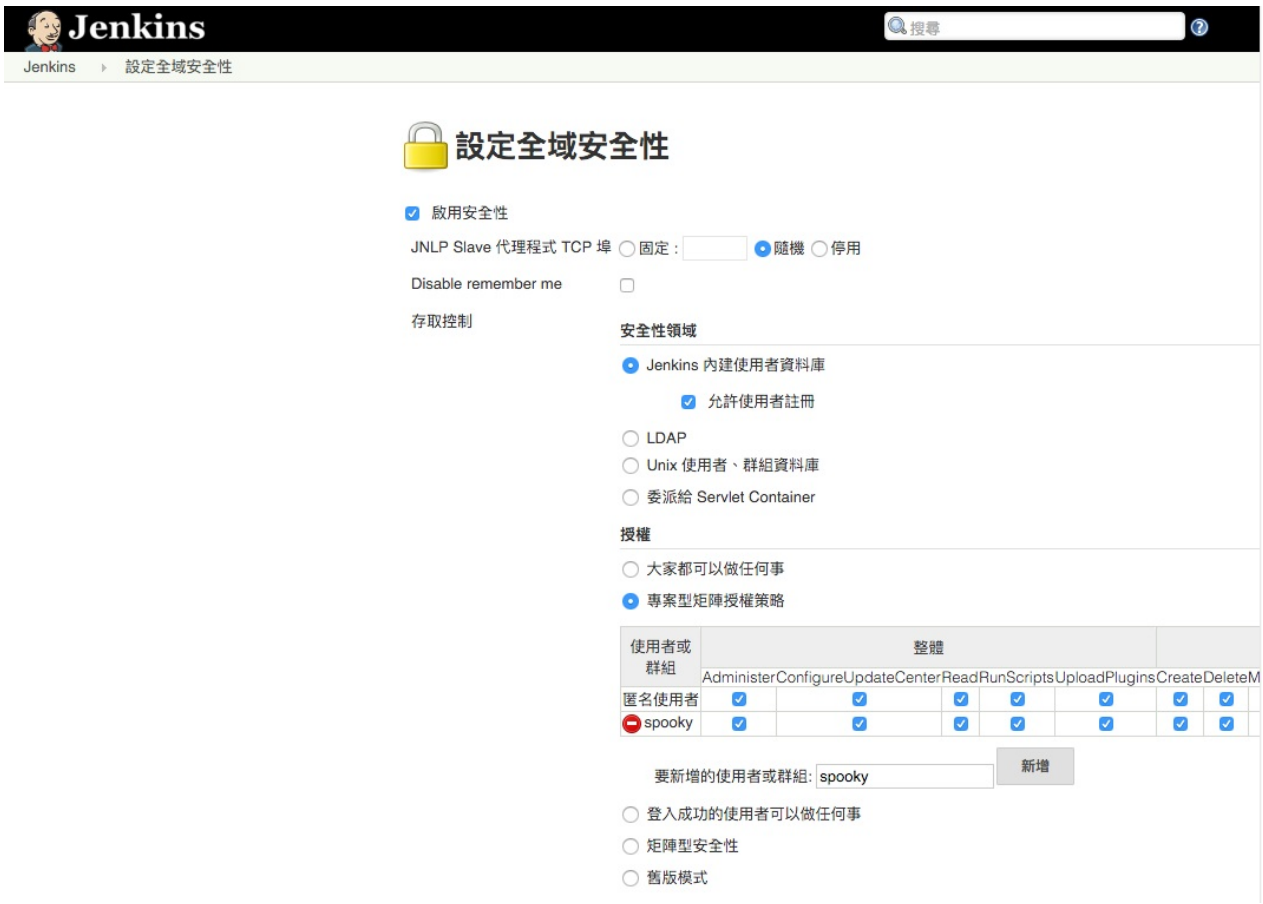
除了 task 的執行之外，另外一個原因是，在除錯的過程中讓 jenkins CI 跟 jenkins user 同樣環境設置方式將會有利於加速確認環境是否正確或是有異常時可以直接用 jenkins user 進行 debug

這件事非常重要，因為若都只能用 task 確認，每次確認結果將會比較慢，執行上會比較有挫折感。

相同的，若我們需要新增新的處理，也可以進行直接用 jenkins user 進行，將會更直接的把自動化需求完成，一旦確認沒有問題 理論上再把相關指令讓 jenkins CI 執行就會正確！

設置安全性

設置畫面



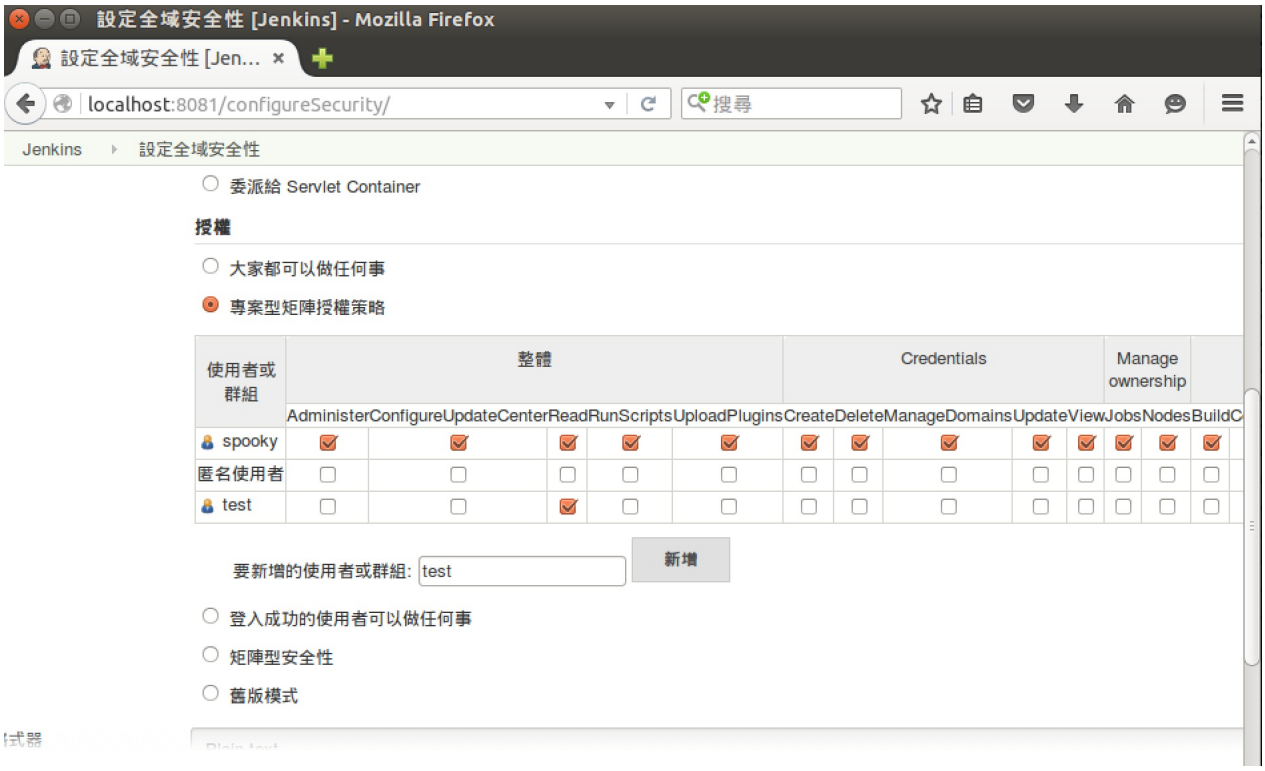
專案型矩陣授權策略

使用時機

希望可以控管特定 task 可被特定人員存取，例如：production deploy 時，只有 DevOps 人員可以進行 release task 的執行

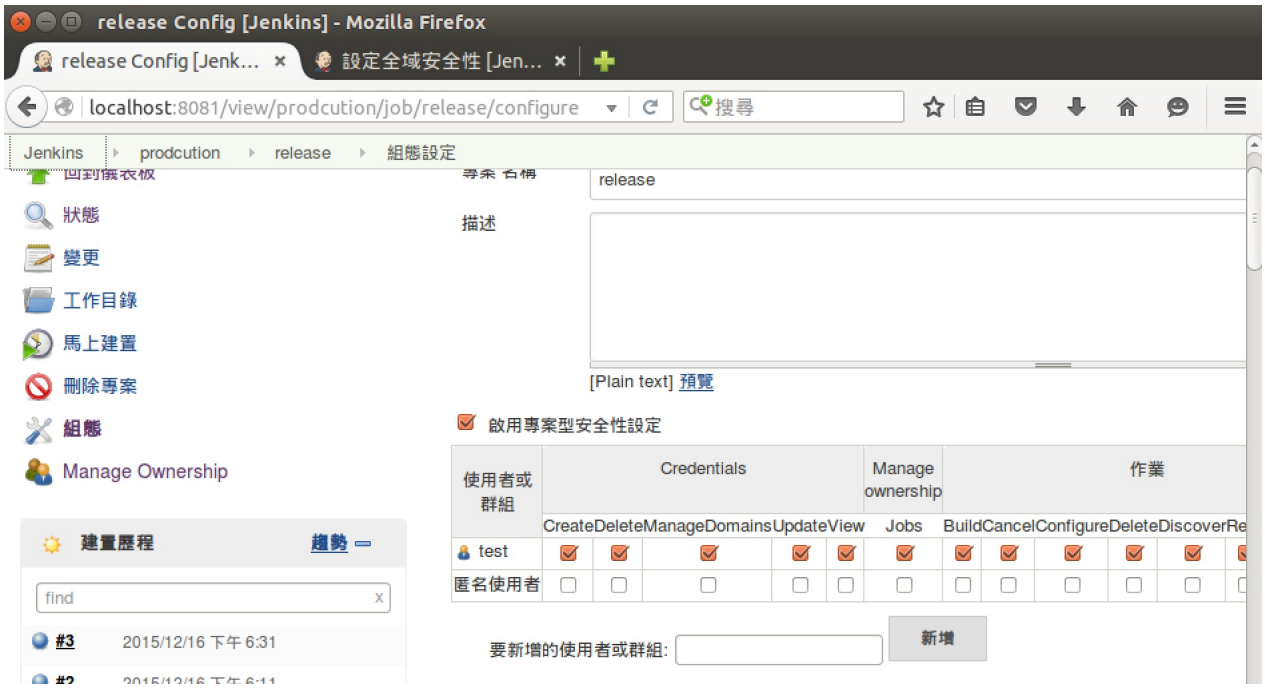
設置方式

在 `Jenkins > 設定全域安全性` 使用 `專案型矩陣授權策略` 並且在新增人員至少要有整體 `read` 權限，如下圖：



需要注意的是，若使用者有設置 `整體 Administer` 權限，預設將可以看到所有的 task。

接著，我們可以進入特定 task 進行權限設置，如下圖：



最後再使用 test 這個 user 進行登入後，將只能看到有設置可以存取的 task 被顯示出來

時區設置

在 `/etc/default/jenkins` 加入下面設定：

`JAVA_ARGS="-Dorg.apache.commons.jelly.tags.fmt.timeZone=Asia/Taipei" -`

參考：http://blog.blackbing.net/post/2014/03/07/jenkins-notes

Master/slave

節點套件安裝

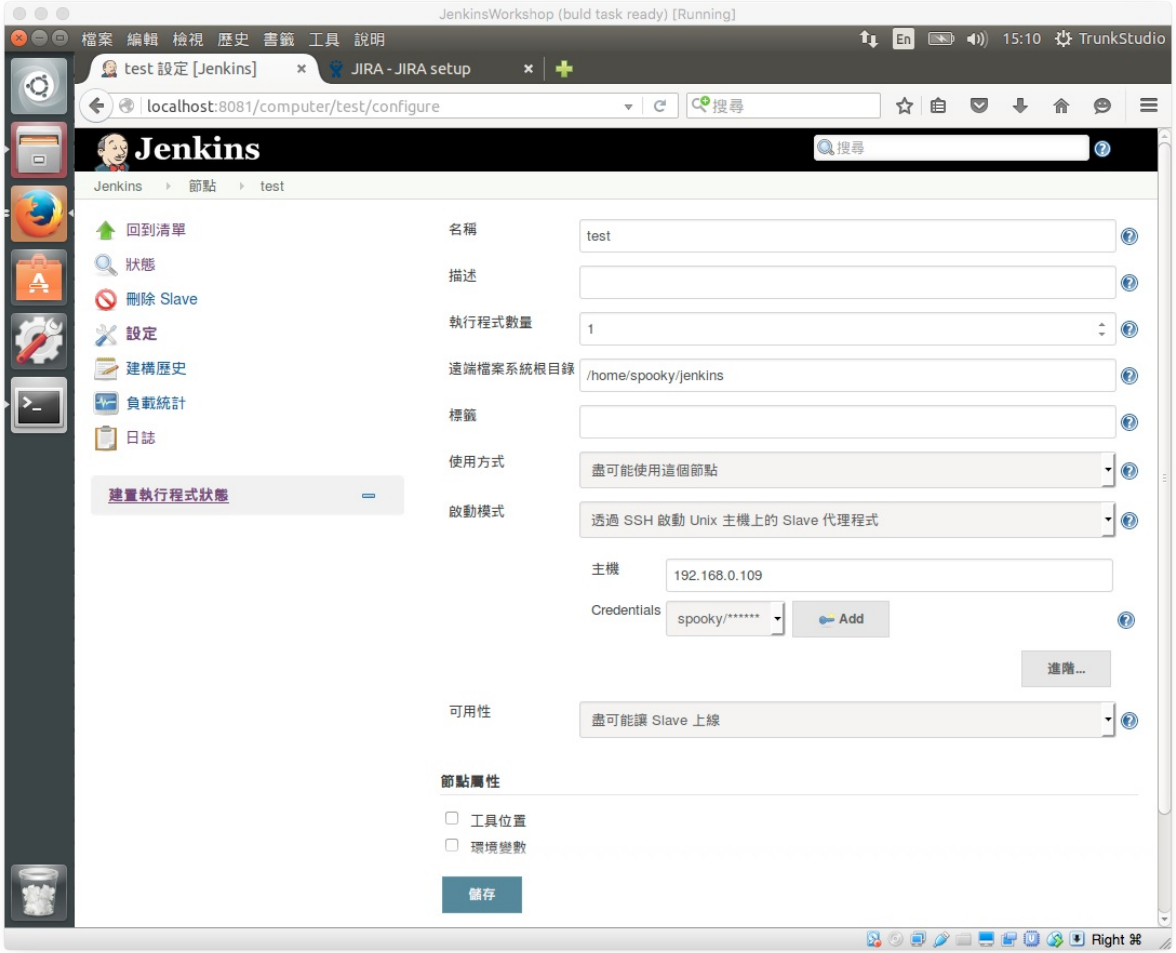
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install default-jre
```

節點使用者設定 ssh

新增你要用來執行 slave 的 user，e.g. slave

參考 ssh 進行設定

jenkins slave 設置



slave 無網路情形

可以使用 `在 Master 上執行指令啟動 Slave`

參考 jenkins 內的說明

> 啟動 Slave 代理程式的指令，可以控制 Slave 電腦並與 Master 溝通。Jenkins 假設執行的程式會在正式的 Slave 機器上啟動 slave.jar。
>
> 可以由 http://${jenkinsServerHost}/jnlpJars/slave.jar 下載 slave.jar。
>
> 簡單一點就像 "ssh 主機名稱 java -jar ~/bin/slave.jar"。 但是，一般會建議您在 Slave 上面寫一個小 Shell Script，控制 Java 及 slave.jar 的位置， 也能設定 PATH 這類節點間不盡相同的環境變數。就像:
>
> # !/bin/sh exec java -jar ~/bin/slave.jar
>
> 您可以使用任何指令執行 Slave 機器上的程式，例如 RSH。 只要最後程式的 stdin 及 stdout 被連到 "java -jar ~/bin/slave.jar" 就好。
>
> 大型部署環境下，可以考慮從掛載 NFS 的共通位置中載入 slave.jar，就不用每次升級 Jenkins 時還要同步更新每部機器上的這個檔案。
>
> 設定成 "ssh -v 主機名稱" 可以幫助您處理連線問題。

Docker

在雲端運算時代，硬體的管理不再是問題，但是軟體配置和管理的問題仍在，而 Docker 的出現能夠幫助開發者更有效率地解決問題。

接下來一系列的課程中，我們將有效率地幫助您 Docker 從入門到進階的學習。

認識 Docker

課程長度：10分鐘

難易度：入門

參考資料

- Docker —— 從入門到實踐
- What is Docker?

What is Docker

Docker 是一個開源專案，誕生於 2013 年初，最初是 dotCloud 公司內部的一個業餘專案。Docker 的基礎是 Linux 容器（LXC）等技術，目標是實作輕量級的作業系統虛擬化解決方案。Docker 自從開放源碼後受到廣泛的關注與討論，以至於 dotCloud 公司都改名為 Docker Inc.。

- Released as open source in March 2013
- Join the Linux Foundation in June 2013 1
- Apache 2.0 License
- Written in Go

Docker Project (GitHub) - https://github.com/docker/docker

在 LXC 的基礎上 Docker 進行了進一步的封裝，讓使用者不需要去關心容器的管理，使得操作更為簡便。使用者操作 Docker 的容器就像操作一個快速輕量級的虛擬機一樣簡單。

Docker 受到業界大廠的擁戴：

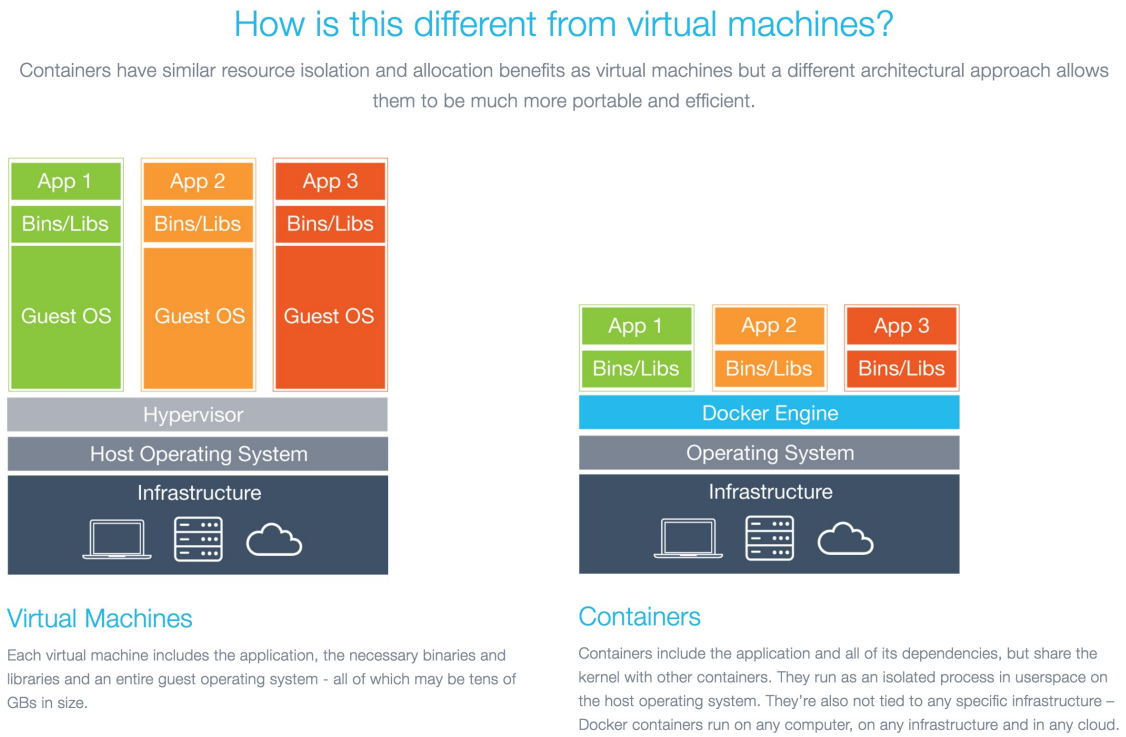
- RedHat 已經在其 RHEL6.5 中集中支援 Docker。
- Amazon、Google 及 Microsoft 也在自家 PaaS 產品支援 Docker 服務。

Service Providers
rackspace
amazon webservices
IBM
SOFTLAYER
openstack
OPENSHIFT
Microsoft Azure
stackato
Baidu 百度
Yandex
Google
StackDock
Quay.io
linode
Gilliam
DEIS
tutum
ORCHARD
domino
VOXOZ
clever cloud
Operating Systems
redhat
docker
CoreOS
ubuntu
debian
fedora
CentOS
Configuration Management
puppet labs
Chef
Jenkins
SALTSTACK
DotCi
Capistrano
Packer
ANSIBLE
Big Data
OpenCore
hadoop
syncsort
Dev Tools
Shippable
Fig
VAGRANT
heroku
wercker
drone.io
runnable
NODECHECKER
docker
Service Discovery
Eureka
etcd
SkyDNS
Official Repositories
MySQL
mongoDB
Ruby
Java
WordPress
redis
NGINX
node
ubuntu
debian
CentOS
Orchestration
MESOS
Shipper
CONSUL
VNS3
Marathon
fleet
ATOMIC
System Integrators
OpDemand
Wiredcraft
relevance lab
IZURI
SHADOWSOFT
Flux7
Dev9
MomentumSI
InfoSiftr

圖片來源

Docker 與虛擬機器比較

下面的圖片比較了 Docker 和傳統虛擬化方式的不同之處，可見容器是在作業系統層面上實作虛擬化，直接使用本地主機的作業系統，而傳統方式則是在硬體層面實作。



作為一種新興的虛擬化方式，Docker 跟傳統的虛擬化方式相比具有眾多的優勢。

首先，Docker 容器的啟動可以在秒級實作，這相比傳統的虛擬機方式要快得多。 其次，Docker 對系統資源的使用率很高，一台主機上可以同時執行數千個 Docker 容器。

容器除了執行其中應用外，基本不消耗額外的系統資源，使得應用的效能很高，同時系統資源消耗更少。傳統虛擬機方式執行 10 個不同的應用就要啟動 10 個虛擬機，而 Docker 只需要啟動 10 個隔離的應用即可。

Why Docker

具體說來，Docker 在以下幾個方面具有較大的優勢：

1. 更快速的交付和部署
2. 更有效率的虛擬化
3. 更輕鬆的遷移和擴展
4. 更簡單的管理

更快速的交付和部署

對開發和維運（DevOps）人員來說，最希望的就是一次建立或設定，可以在任意地方正常執行。

開發者可以使用一個標準的映像檔來建立一套開發容器，開發完成之後，維運人員可以直接使用這個容器來部署程式碼。 Docker 可以快速建立容器，快速迭代應用程式，並讓整個過程全程可見，使團隊中的其他成員更容易理解應用程式是如何建立和工作的。 Docker 容器很輕很快！容器的啟動時間是秒級的，大量地節約開發、測試、部署的時間。

更有效率的虛擬化

Docker 容器的執行不需要額外的虛擬化支援，它是核心層級的虛擬化，因此可以實作更高的效能和效率。

更輕鬆的遷移和擴展

Docker 容器幾乎可以在任意的平台上執行，包括實體機器、虛擬機、公有雲、私有雲、個人電腦、伺服器等。 這種兼容性可以讓使用者把一個應用程式從一個平台直接遷移到另外一個。

更簡單的管理

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分發和更新，從而實作自動化並且有效率的管理。

Docker Image

Docker 映像檔就是一個唯讀的模板。

例如：一個映像檔可以包含一個完整的 ubuntu 作業系統環境，裡面僅安裝了 Apache 或使用者需要的其它應用程式。

映像檔可以用來建立 Docker 容器。

Docker 提供了一個很簡單的機制來建立映像檔或者更新現有的映像檔，使用者甚至可以直接從其他人那裡下載一個已經做好的映像檔來直接使用。

Docker Container

Docker 利用容器來執行應用。

容器是從映像檔建立的執行實例。它可以被啟動、開始、停止、刪除。每個容器都是相互隔離的、保證安全的平台。

可以把容器看做是一個簡易版的 Linux 環境（包括root使用者權限、程式空間、使用者空間和網路空間等）和在其中執行的應用程式。

*註：映像檔是唯讀的，容器在啟動的時候建立一層可寫層作為最上層。

Docker 倉庫

倉庫是集中存放映像檔檔案的場所。有時候會把倉庫和倉庫註冊伺服器（Registry）混為一談，並不嚴格區分。實際上，倉庫註冊伺服器上往往存放著多個倉庫，每個倉庫中又包含了多個映像檔，每個映像檔有不同的標籤（tag）。

倉庫分為公開倉庫（Public）和私有倉庫（Private）兩種形式。

最大的公開倉庫是 Docker Hub，存放了數量龐大的映像檔供使用者下載。大陸的公開倉庫包括 Docker Pool 等，可以提供大陸使用者更穩定快速的存取。

當然，使用者也可以在本地網路內建立一個私有倉庫。

當使用者建立了自己的映像檔之後就可以使用 push 命令將它上傳到公有或者私有倉庫，這樣下次在另外一台機器上使用這個映像檔時候，只需要從倉庫上 pull 下來就可以了。

*註：Docker 倉庫的概念跟 Git 類似，註冊伺服器可以理解為 GitHub 這樣的託管服務。

Registry

註冊伺服器（registry server）是提供 Docker Repository 服務的主機，

- Docker Hub

  Docker 官方提供的註冊伺服器。

- DockerPool

  中國大陸的 Docker 社群建立的註冊伺服器。

當然，不是所有的 Docker Image 都適合發佈到公開的註冊伺服器，因此也可以建立私有、具存取權限控管的 Registry，例如透過雲端運算服務：

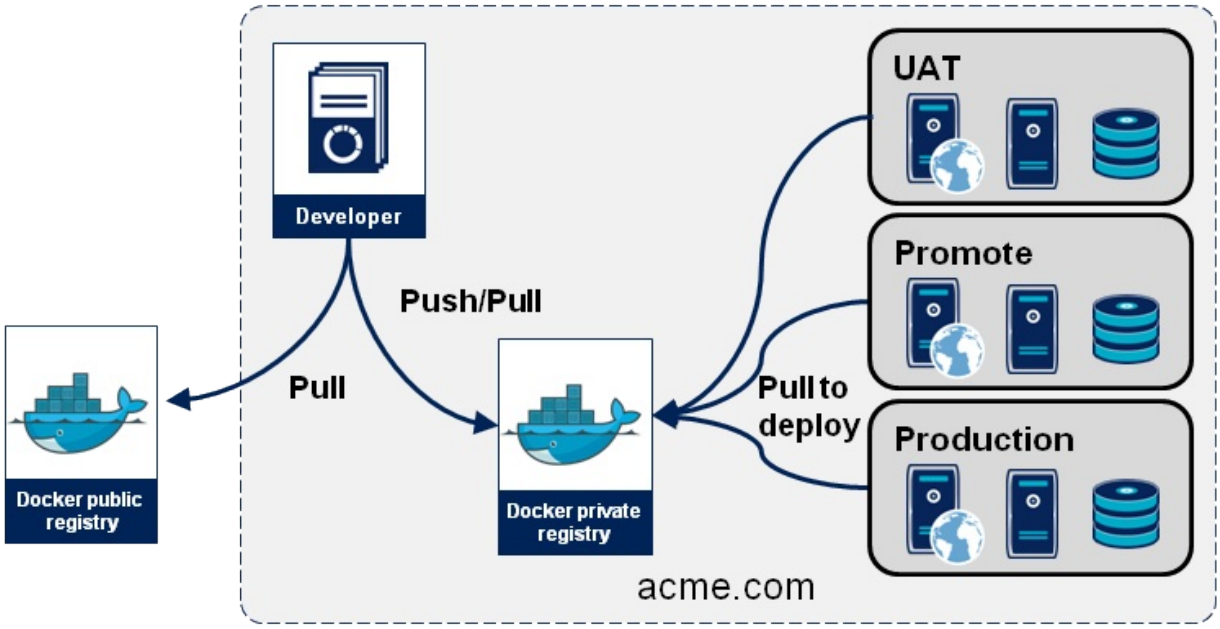
- Amazon EC2 Container Registry

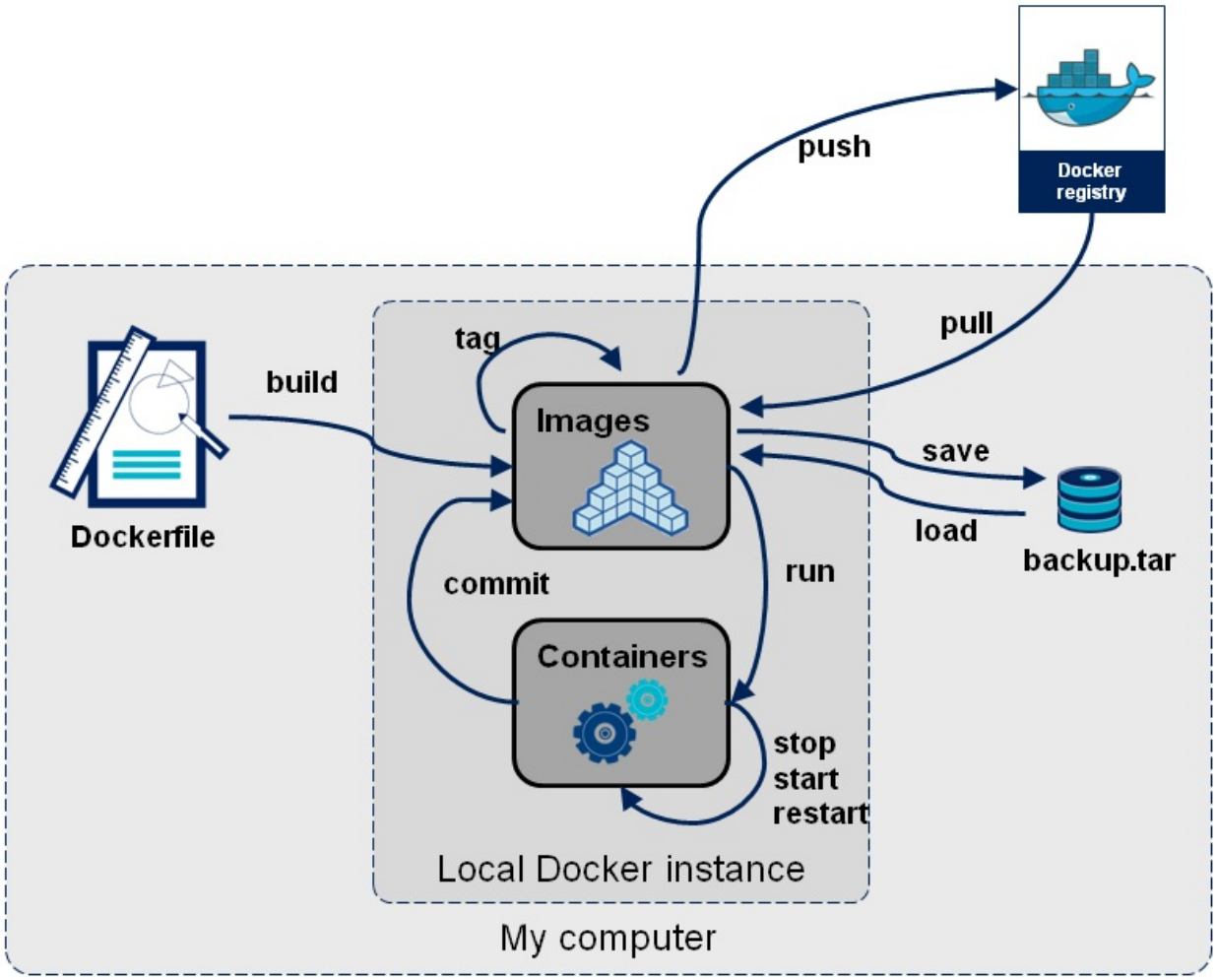
或是在私有伺服器主機架設。

Distribution - The Docker toolset to pack, ship, store, and deliver content.

https://github.com/docker/distribution

指令圖解


Developer
Push/Pull
Pull
Docker public registry
Docker private registry
Pull to deploy
UAT
Promote
Production
acme.com


push
Docker registry
pull
tag
build
Images
save
load
backup.tar
Dockerfile
commit
run
Containers
stop
start
restart
Local Docker instance
My computer

Lab 101: Docker Image

課程長度：15分鐘

難易度：基礎

映像檔（Image）是 Docker 的重要基礎概念，也是執行容器（Container）所必備。

- 映像檔類似 Ubuntu Live CD / USB 的 ISO。

映像檔與容器之間的關係？

- 如果把映像檔視為物件導向的類別（class），容器則可視為物件（object）。
- 啟動容器需要本機已存在所需映像檔。

映像檔從哪裡來？

- 來自 Docker 的倉庫（Repository）。
- 如果本機不存在所需映像檔，Docker 會嘗試從預設的倉庫（例如：Docker Hub）下載映像檔。

本單元將練習以下的 Docker 指令：

1. 搜尋映像檔

   `search`

2. 取得映像檔

   `pull`

3. 列出所有映像檔

`images`

4. 檢視映像檔資訊

   `inspect`

5. 備份與載入

   `save` , `load`

6. 刪除映像檔

   `rmi`

指令求助

查詢 Docker 的指令清單。

```
docker help
```

查詢指定的 Docker 指令說明及用法。

```
docker help pull
```

搜尋映像檔

使用 `docker search` 指令搜尋 Ubuntu Linux 相關的映像檔。

```
docker search ubuntu
```

搜尋結果的資料筆數非常多，如何篩選出我們需要的映像檔？

通常我們會從 STARS 數量評比來判斷映像檔的熱門程度，人氣指數愈高通常代表：由官方發行或功能完善且有持續更新維護。用 `-s 20` 篩選 STARS 數量大於 20 的映像檔：

```
docker search -s 20 ubuntu
```

執行結果：

```
NAME                        DESCRIPTION
              STARS     OFFICIAL   AUTOMATED
ubuntu                      Ubuntu is a Debian-based Linux ope
rating s...   3675      [OK]
ubuntu-upstart              Upstart is an event-based replacem
ent for ...   61        [OK]
torusware/speedus-ubuntu    Always updated official Ubuntu doc
ker imag...   25                    [OK]
ubuntu-debootstrap          debootstrap --variant=minbase --co
mponents...   24        [OK]
rastasheep/ubuntu-sshd      Dockerized SSH service, built on t
op of of...   24                    [OK]
```

若要讓 search 敘述完整呈現可加上面下面的 Option：

```
--no-trunc         Don't truncate output
```

取得映像檔

使用 `docker pull` 指令從 Docker Hub 下載映像檔。

例如取得最新的 Ubuntu Linux 作業系統映像檔，執行：

```
docker pull ubuntu
```

終端機將會出現下面訊息：

```
Using default tag: latest
latest: Pulling from library/ubuntu

d38575f188e0: Pull complete
b04ea90f261c: Pull complete
40dc9cd44ffa: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:4bc45eecd925cb8806294d0e8bc5c1cfd1149c6dd8a5ef
2165b6c02eabb8821f
Status: Downloaded newer image for ubuntu:latest
```

實際上這條指令省略了兩個參數內容：

1. Registry（註冊伺服器）
2. Tag（版本標記）

加上 Tag 版本標記，明確指定取得 `ubuntu:14.04` （目前的最新版本）的映像檔。

```
docker pull ubuntu:14.04
```

以下指令的執行結果與 `docker pull ubuntu` 相同。

```
# 加上 Tag 指定最新版
docker pull ubuntu:latest

# 加上 Registry，指定以 Docker Hub 註冊伺服器作為來源
docker pull registry.hub.docker.com/ubuntu

# 同時指定 Registry 與 Tag
docker pull registry.hub.docker.com/ubuntu:latest

# 指定以 DockerPool 註冊伺服器作為來源
docker pull dl.dockerpool.com:5000/ubuntu
```

備註：除了 Docker Hub 或 DockerPool 也可以搭配自行架設的私有註冊伺服器（Registry Server）。

更新映像檔

如果使用 `latest` 版本標記取得映像檔，日後可以重新執行 `docker pull` 檢查更新。

例如當 `ubuntu` 發佈更新後執行：

```
docker pull ubuntu
```

或

```
docker pull ubuntu:latest
```

就可以更新為最新版本。

列出映像檔

執行 `docker images` 指令列出本機上已有的映像檔：

```
REPOSITORY                    TAG                 IMAGE ID
         CREATED             SIZE
ubuntu                        latest              a1e4ed2ac65
b        8 hours ago         188 MB
ubuntu                        14.04               07c86167cdc
4        4 weeks ago         188 MB
```

- REPOSITORY

  代表映像檔所在的倉庫（Repository）名稱。

- TAG

  版本標記（若是在 `git pull` 時沒有指定版本，就會顯示 `latest` 代表預設的最新版本）。

- IMAGE ID

  映像檔的唯一識別編號（檔案的 SHA-1 Hash 編碼）。

檢視映像檔資訊

先執行 `docker images` 查看映像檔：

```
REPOSITORY                    TAG                 IMAGE ID
         CREATED             SIZE
ubuntu                        latest              a1e4ed2ac65
b        8 hours ago         188 MB
ubuntu                        14.04               07c86167cdc
4        4 weeks ago         188 MB
```

依據上面的清單內容，如果我們想檢視 `ubuntu:14.04` 的詳細資訊，可以執行 `docker inspect` ：

```
docker inspect ubuntu:14.04
```

或是指定 IMAGE ID：

```
docker inspect a1e4ed2ac65b
```

執行 `inspect` 的輸出結果為 JSON 格式，在 Console 中可以搭配 `less` 方便捲動檢視。

備份與載入映像檔

執行 `docker save` 指令將映像檔備份為 `ubuntu-latest.tar` ：

```
docker save -o ubuntu-latest.tar ubuntu:latest
```

如此一來，我們就能將 `ubuntu-latest.tar` 透過 USB 隨身碟等方式複製到其他電腦。

取得 `ubuntu-latest.tar` 的電腦，即可用 `docker load` 指令載入映像檔。

```
docker load --i ubuntu-latest.tar
```

如果遇到需要在沒有網路的環境下離線作業，或是不想透過網路傳輸體積大的映像檔， `docker save` 與 `save load` 提供一個簡單實用的方式。

刪除映像檔

先執行 `docker images` 查看映像檔：

```
REPOSITORY                    TAG                 IMAGE ID
         CREATED             SIZE
ubuntu                        latest              a1e4ed2ac65
b        8 hours ago         188 MB
ubuntu                        14.04               07c86167cdc
4        4 weeks ago         188 MB
```

依據上面的清單內容，如果我們想刪除 `ubuntu:14.04` 的映像檔，可以執行 `docker rmi` ：

```
docker rmi ubuntu:14.04
```

刪除的動作有一個前提，該映像檔必須沒有運行的容器（Container），否則就會產生錯誤訊息。然而，如果確定要放棄 Container（可能導致容器資料流失），可加上 `-f` 參數強制刪除映像檔。

```
docker rmi -f ubuntu:14.04
```

實務上我們應避免直接強制刪除映像檔，正確的做法是，先刪除映像檔相關的所有容器，然後才刪除映像檔。

Lab 102: Docker Container

課程長度：15分鐘

難易度：基礎

容器（Container）是映像檔（Image）的執行實例（Instance），容器帶有可讀寫的資料存取空間，讓容器的作業系統與應用程式在運行時，可以保存額外的狀態與資料。

本單元將練習以下的 Docker 指令：

`create`, `start`, `stop`, `ps`, `run`, `exec`, `attach`, `rm`

建立容器

先確認本機已有 `ubuntu:latest` 映像檔。

```
docker pull ubuntu
```

執行 `docker create` 指令新建一個容器。

```
docker create -it ubuntu:latest
```

為什麼需要 `-it` 參數呢？實際上這是 `-i` 與 `-t` 兩個參數的縮寫。

- `-i`

  Keep STDIN open even if not attached

- `-t`

  Allocate a pseudo-TTY

使用 `docker ps` 檢視所有容器的狀態。

```
docker ps -a
```

```
CONTAINER ID      IMAGE             COMMAND
2c06ac467b38      ubuntu:latest     "/bin/bash"
```

因為 `docker create` 指令建立的容器預設為停止狀態，所以加上參數 `-a` 才能顯示包含已停止的容器。

從上面的執行結果，我們可以得知容器的 `CONTAINER ID` ，利用 ID 來啟動（或停止）容器。

啟動容器：

```
docker start 2c0
```

通常我們對容器進行操作時，為了讓指令輸入方便，並不會使用完整的 `CONTAINER ID` ，只要輸入前三碼字母即可。

容器啟動後，再以 `docker ps` 查詢執行中的容器，可以看到 STATUS 的變化。

```
docker ps
```

停止容器，使用 `docker stop` 指令。

```
docker stop 2c0
```

執行容器

使用 `docker run` 可以啟動一個容器，並執行自訂的 Command Line 指令。

```
docker run ubuntu /bin/echo Hello, Docker!
```

執行結果會輸出：

```
Hello, Docker!
```

實際上 `echo` 指令是在 Ubuntu Linux 作業系統容器中執行。

多試試其他 Ubuntu Linux 的指令，例如 `uname` ：

```
docker run ubuntu /bin/uname -a
```

輸出結果：

```
Linux 5d45430a0a54 4.2.0-27-generic #32~14.04.1-Ubuntu SMP Fr
i Jan 22 15:32:26 UTC
2016 x86_64 x86_64 x86_64 GNU/Linux
```

Docker Container 提供一個完全隔離的環境給 Ubuntu Linux，有別於傳統的虛擬機器，Docker 的容器十分輕量、隨時可以新建或刪除。

執行 Bash Shell

執行：

```
docker run -it ubuntu:14.04 /bin/bash
```

在 Container 的 Shell 試試一些常用的 Linux 指令：

- `ls`
- `ps`
- `pwd`
- `dmesg`
- `uname`

要結束 Shell 只要執行：

```
exit
```

退出後可以用 `docker ps -a` 觀察容器的狀態，可以發現每一次 Docker 執行完指令後，都會回到停止狀態。

背景執行

讓一個長時間執行的指令在背景運作，可以利用 `-d` 參數。

```
docker run -d -it localhost:5000/training-webapp ping google.
com
```

參數說明：

- `-d` 表示讓程式在後台以背景狀態（Daemonized）的形式執行。

透過 `docker ps` 查詢容器的狀態：

```
CONTAINER ID        IMAGE                          COMMAND
374fb78dab41        localhost:5000/training-webapp
   "ping google.com"
```

要如何檢視 `ping google.com` 指令的輸出呢？可以透過 `docker logs` 指令：

```
docker logs 374
```

如果想要持續顯示最新的輸出，也可以加上 `-f` 參數（結束用 Ctrl + C 組合鍵）：

```
docker logs -f 374
```

進入容器

正在背景狀態執行的容器，可以利用 `docker exec` 執行新的 Bash Shell。

```
docker exec -ti 374 /bin/bash
```

或是執行指令，例如 `ps aux`：

```
docker exec -ti 374 ps aux
```

輸出結果：

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   T
IME COMMAND
root         1  0.0  0.2  12832  2048 ?        Ss+  11:01   0
:00 ping google.com
root        26  0.0  0.2  15572  2176 ?        Rs+  11:01   0
:00 ps aux
```

也可以利用 `docker attach` 指令，切換至容器的終端機。

```
docker attach 374
```

如果有多位使用者操作同一個容器，通常執行 `exec` 會比 `attach` 合適，因為 `attach` 會讓每個使用者共用同一個終端機畫面，因此造成互相牽制影響。

移除容器

列出所有容器（包含停止狀態的容器）。

```
docker ps -a
```

輸出如下：

```
CONTAINER ID        IMAGE               COMMAND
270ea7ee8b06        ubuntu              "/bin/bash"
```

如果要移除 `270ea7ee8b06` 這個容器，可以執行 `docker rm` 指令：

```
docker rm 270
```

假如容器還在運行中會出現錯誤，加上 `-f` 參數可以強制移除容器。

```
docker rm -f 270ea7ee8b06
```

如果同時要移除（或停止）所有容器，可以搭配以下 Shell 指令進行：

先停止所有容器。

```
docker stop $(docker ps -a -q)
```

再移除所有容器。

```
docker rm $(docker ps -a -q)
```

private registry server

若有需要自行架設 private registry server 可以參考下列網址

docker/distribution

- 舊版 private registry server 1.0 之 github repository 網址為 https://github.com/docker/docker-registry
- 新版 private registry server 2.0 之 github repository 網址為 https://github.com/docker/distribution

會有新舊之分，主要是舊版是用 python 實作，新版則改用 go 來實作，以便改善安全性及效能。

對 docker 而言架設一個 registry server 就是啟動一個 container，相關操作練習步驟如下。

啟動 private registry server

```
docker run -d -p 5000:5000 --restart=always --name registry
registry:2
```

從 docker hub/public registry server 取得 ubuntu image

```
docker pull ubuntu
```

變更 ubuntu tag 為 private registry image

```
docker tag ubuntu localhost:5000/ubuntu
```

將 image push 到 private registry server

```
docker push localhost:5000/ubuntu
```

再次從 private registry server pull image

```
docker pull localhost:5000/ubuntu
```

列出 private registry images list

```
curl -X GET http://localhost:5000/v2/_catalog
```

也可以透過網址

```
http://localhost:5000/v2/_catalog
```

public Docker hub

目前 Docker 官方有個 public Docker hub，裡面有數以萬計的 docker image，除了可以直接使用，也可以參考建置步驟

在開始使用之前，可以先進行登入

登入 Docker hub

官方提供了 `docker login` 指令，需要三個參數分別是

- `-e` : email
- `-p` : password
- `-u` : user id

範例指令如下

```
docker login -e test@gmail.com -p testpass -u testuser
```

如此將可以完成登入，相關登入資訊會放在使用者 home 目錄底下之 `.dockercfg` ，如此就可以不需要再重覆進行登入

search images

透過 docker search 找到合適的 image 以便使用，如透過下列指令進行搜尋

`docker search ubuntu`

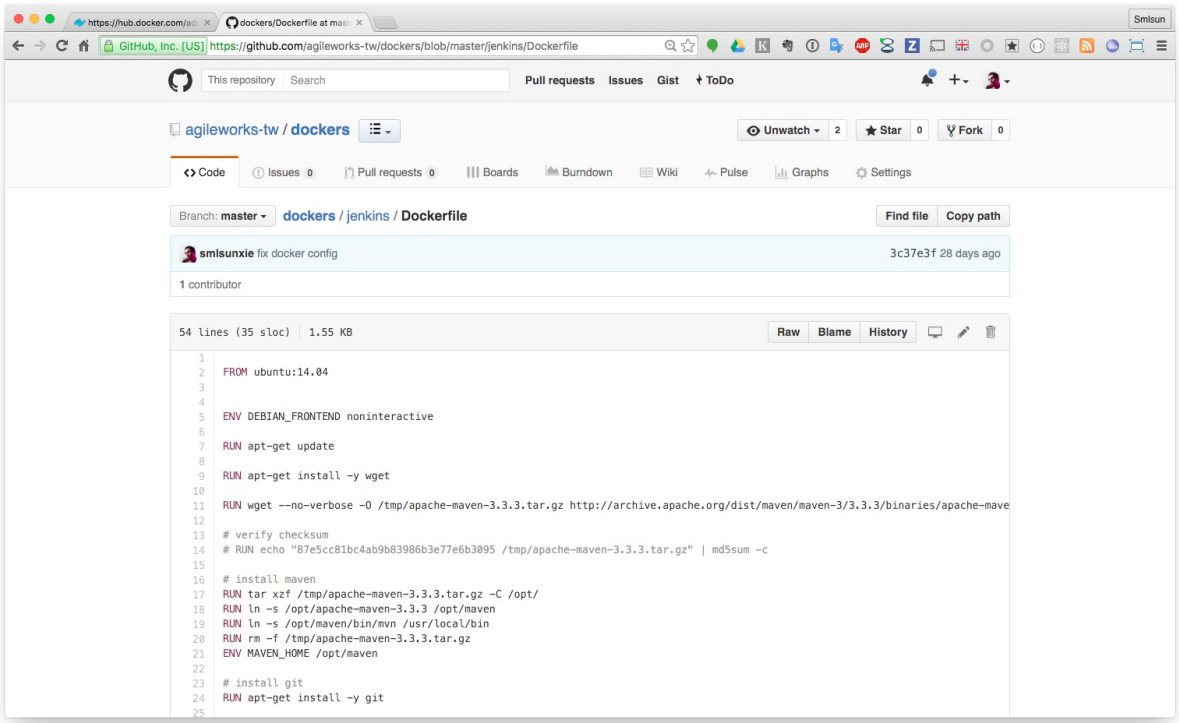
將輸出下列資訊

```
NAME                                    DESCRIPTION
                        STARS      OFFICIAL   AUTOMATED
ubuntu                                  Ubuntu is a Debian-based Li
nux operating s...      3583       [OK]
ubuntu-upstart                          Upstart is an event-based r
eplacement for ...      60         [OK]
torusware/speedus-ubuntu                Always updated official Ubu
ntu docker imag...      25                 [OK]
ubuntu-debootstrap                      debootstrap --variant=minba
se --components...      24         [OK]
rastasheep/ubuntu-sshd                  Dockerized SSH service, bui
lt on top of of...      23                 [OK]
neurodebian                             NeuroDebian provides neuros
cience research...      20         [OK]
```

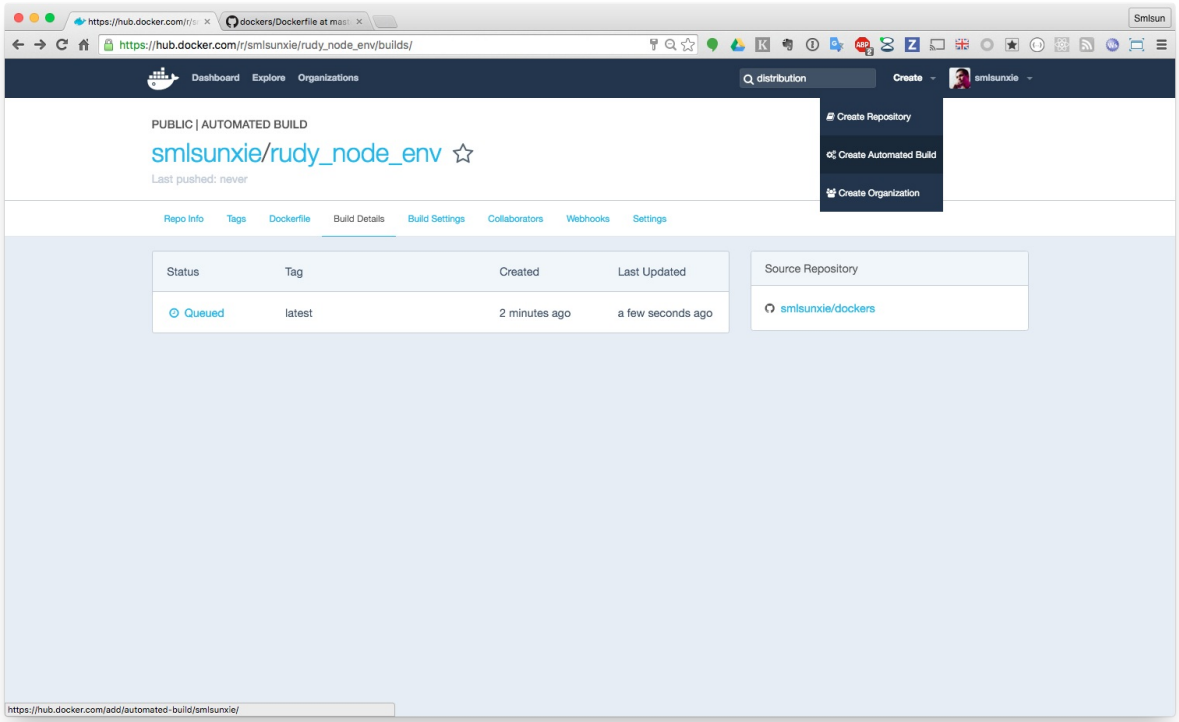
auto build

自動建置範例 dockerfile

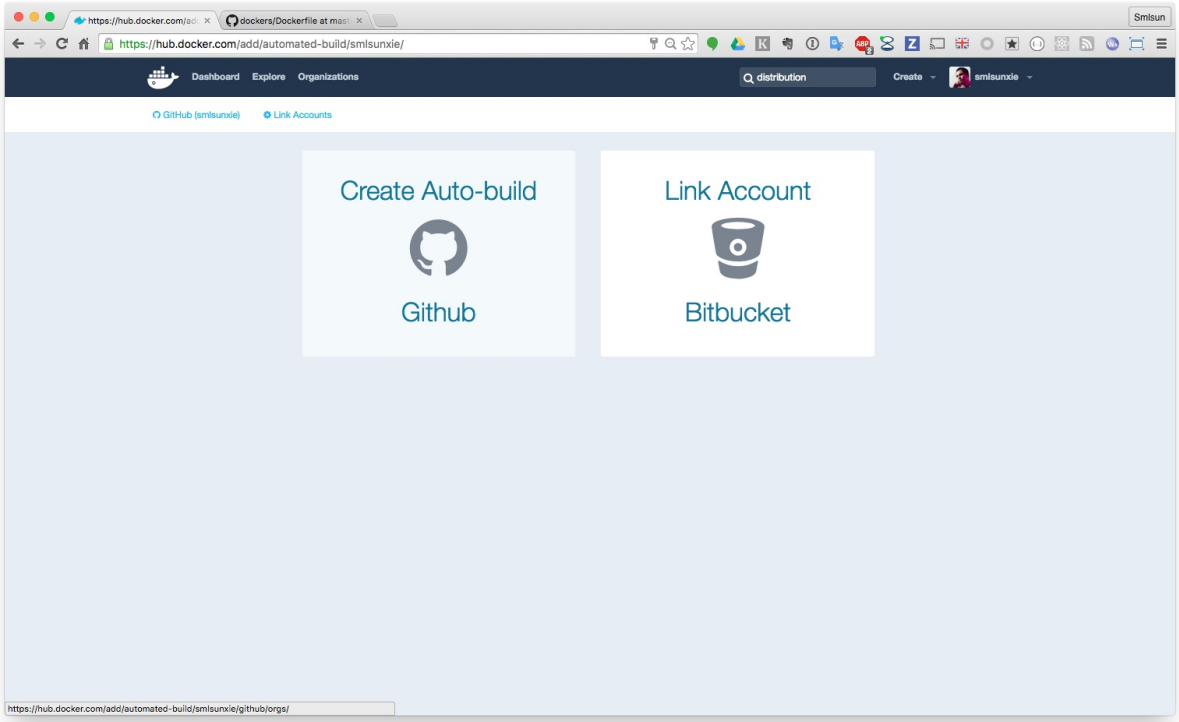
https://github.com/smlsunxie/dockers/blob/master/rudy_node/Dockerfile



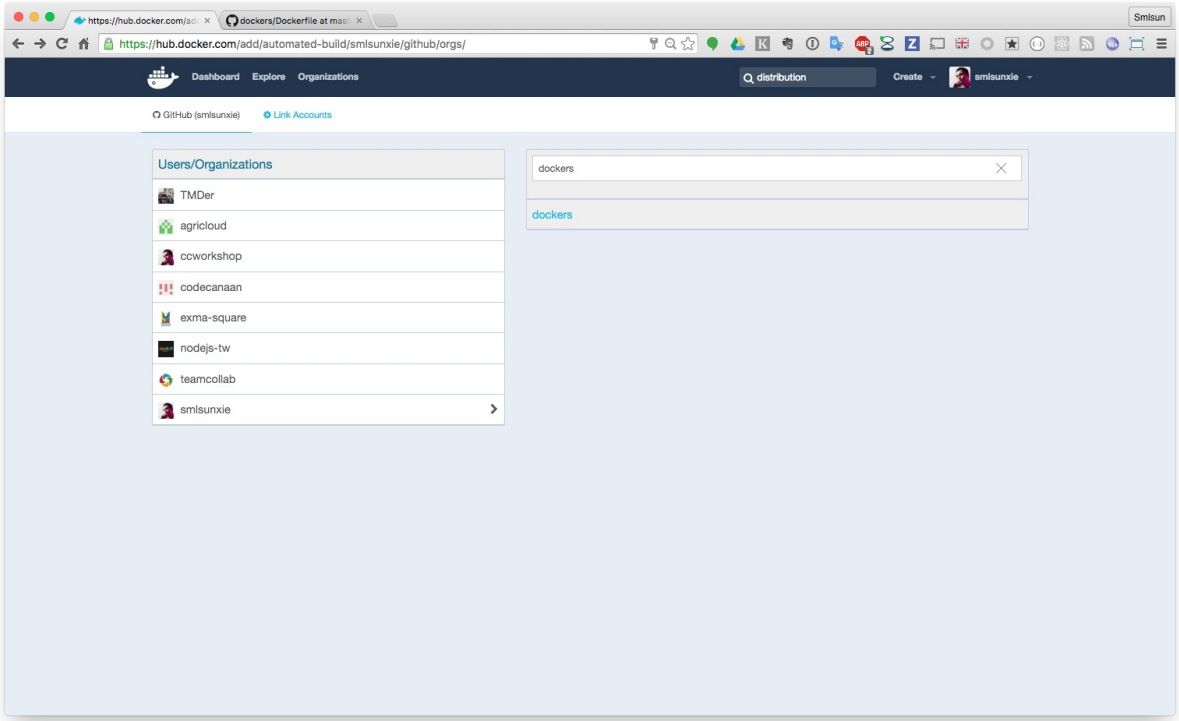
操作步驟



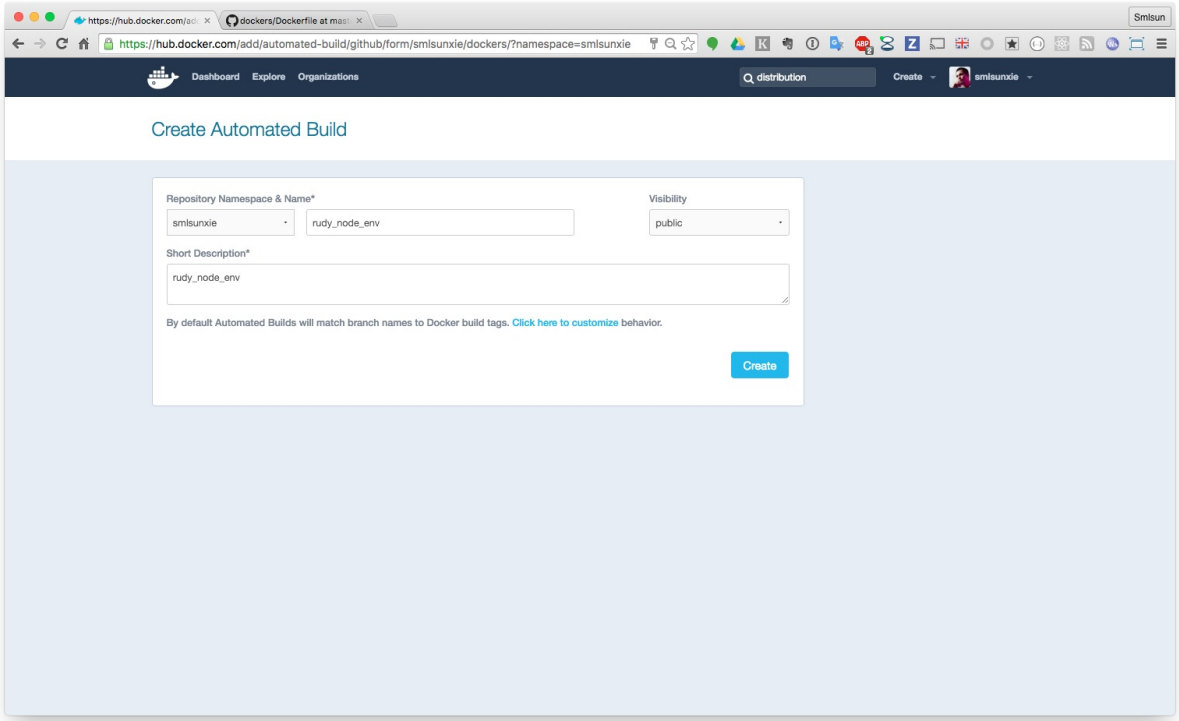
選擇 Create Automated Builds



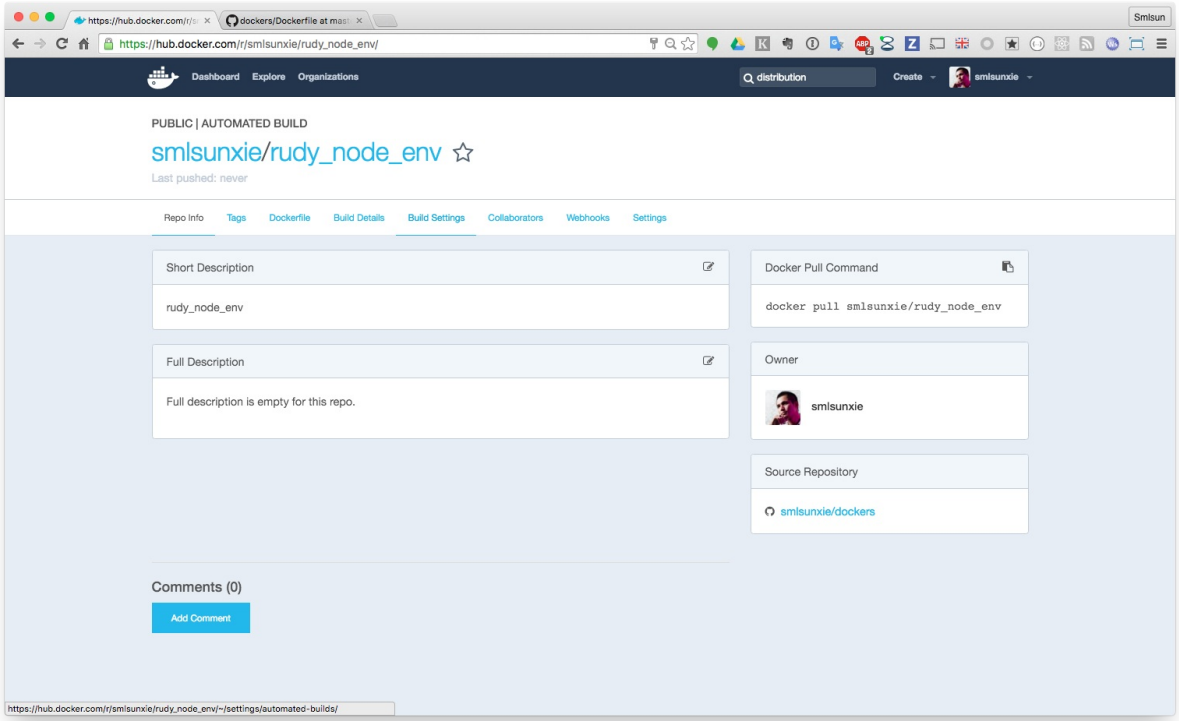
選擇要進行 auto build 的目標網站



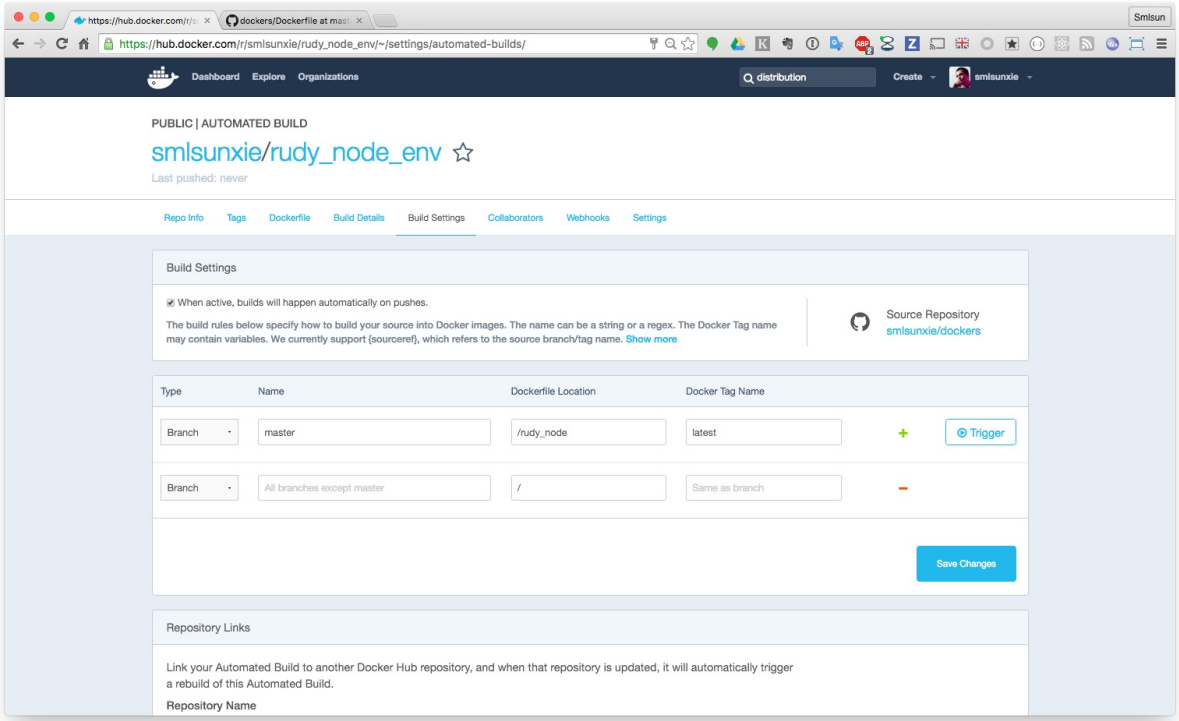
選擇要進行 auto build 之帳號以及 repository



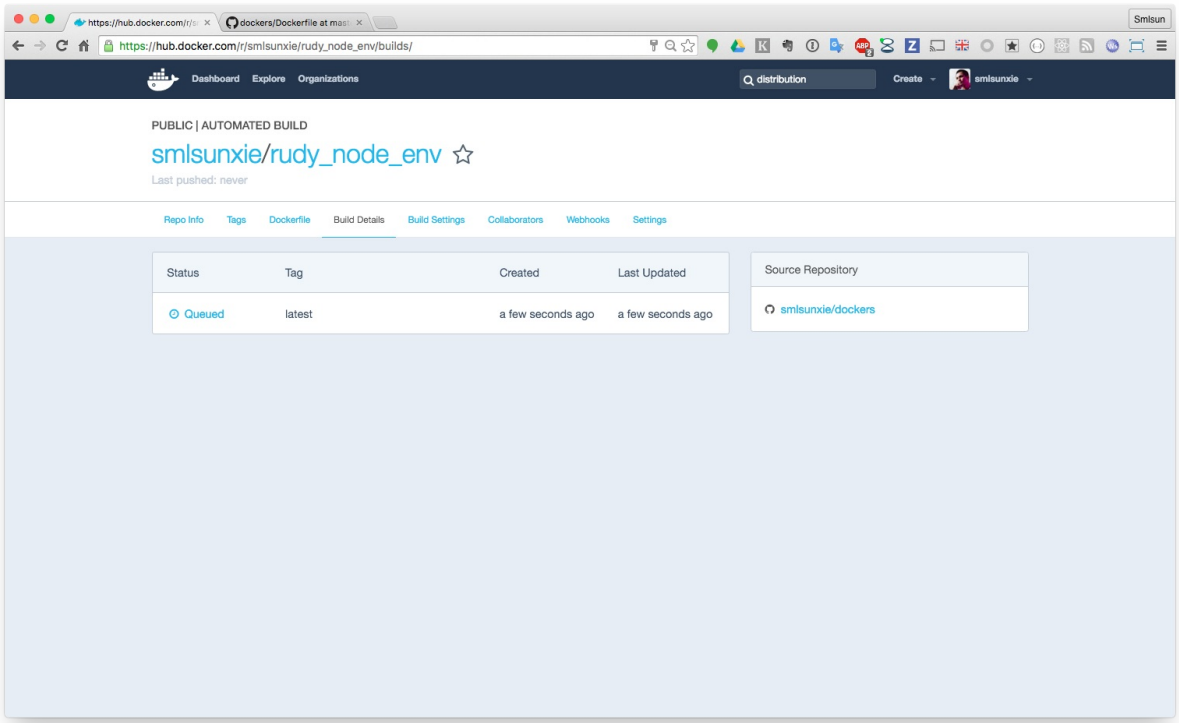
定義要在 docker hub 建立 repository 相關資訊



建立 docker hub 之 repository 完成後，進行 build setting



設定如圖，Dockerfile location 需要定義到目錄位置，設定完成後，執行 save 並且點選 trigger



接著進入到 build Details 可以看到目前正在等待建置，如此即完成 auto build

CMD 與 ENTRYPOINT 的差別

下列內容來源：論docker中 CMD 與 ENTRYPOINT 的區別

Dockerfile裏有 CMD 與 ENTRYPOINT 兩個功能咋看起來很相似的指令，開始的時候覺得兩個互用沒什麼所謂，但其實並非如此。

CMD

> The main purpose of a CMD is to provide defaults for an executing container.

CMD在容器運行的時候提供一些命令及參數，用法如下：

type 1

`CMD ["executable","param1","param2"](exec form, this is the preferred form)`

運行一個可執行的文件並提供參數。

type 2

`CMD ["param1","param2"](as default parameters to ENTRYPOINT)`

為ENTRYPOINT指定參數。

type 3

`CMD command param1 param2 (shell form)`

(shell form)：是以”/bin/sh -c”的方法執行的命令。

ENTRYPOINT

字面意思是進入點，而它的功能也恰如其意。

> An ENTRYPOINT allows you to configure a container that will run as an executable.

它可以讓你的容器功能表現得像一個可執行程序一樣。

容器功能表現得像一個可執行程序一樣，這是什麼意思呢？

使用下面的 ENTRYPOINT 建置 image：

`ENTRYPOINT ["/bin/echo"]`

docker build 出來的鏡像以後的容器功能就像一個 /bin/echo

比如我 build 出來的鏡像名稱叫 imageecho，那麼我可以這樣用它：

docker run -it imageecho “this is a test”

`ENTRYPOINT ["/bin/cat"]`

build 出來的鏡像你可以這樣運行 (假設名為 st)：

`docker run -it st /etc/fstab`

這樣相當： /bin/cat /etc/fstab 這個命令的作用。運行之後就輸出 `/etc/fstab` 裏的內容。

ENTRYPOINT有兩種寫法：

type 1

ENTRYPOINT "executable", "param1", "param2"

type 2

ENTRYPOINT command param1 param2 (shell form)

你也可以在 docker run 命令時使用 --entrypoint 指定（但是只能用 type 1）。

CMD 可以為 ENTRYPOINT 提供參數，ENTRYPOINT 本身也可以包含參數， 但是你可以把那些可能需要變動的參數寫到 CMD 裏而把那些不需要變動的參數寫到 ENTRYPOINT 裏面例如：

```
FROM ubuntu:14.10
ENTRYPOINT ["top", "-b"]
CMD ["-c"]
```

把可能需要變動的參數寫到 CMD 裏面。然後你可以在 docker run 裏指定參數，這樣 CMD 裏的參數(這裏是 -c) 就會被覆蓋掉而 ENTRYPOINT 裏的不被覆蓋。

volume

volume 屬於 docker 很重要的一環，除了運行相關程式之外，檔案操作與存放是專案在運行時不可或缺的

docker volume 提供下列指令

```
Usage:    docker volume [OPTIONS] [COMMAND]

Manage Docker volumes

Commands:
  create                    Create a volume
  inspect                   Return low-level information on a
volume
  ls                        List volumes
  rm                        Remove a volume
```

以往需要透過 docker 的運行來進行 volume 來進行，現在我們可以透過 docker volume 直接定義

docker volume create and inspect

create

```
Usage:    docker volume create [OPTIONS]

Create a volume

  -d, --driver=local    Specify volume driver name
  --help                Print usage
  --label=[]            Set metadata for a volume
  --name                Specify volume name
  -o, --opt=map[]       Set driver specific options
```

建立第一個 volume

`docker volume create --name mysql-data`

建立完成後我們可以透過 inspect 來確認 host 的檔案位置

inspect

```
Usage:    docker inspect [OPTIONS] CONTAINER|IMAGE [CONTAINER
|IMAGE...]

Return low-level information on a container or image

  -f, --format       Format the output using the given go tem
plate
  --help             Print usage
  -s, --size         Display total file sizes if the type is
container
  --type             Return JSON for specified type, (e.g ima
ge or container)
```

執行 `docker volume inspect mysql-data`

將會看到下列輸出

```
[
    {
        "Name": "mysql-data",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/mysql-data/_da
ta",
        "Labels": {}
    }
]
```

就可以直接透過 host 的 filesystem 進行存取

執行 mysql docker

```
docker run -d -p "3306:3306" \
-e MYSQL_ADMIN_PASS="pass" \
-e MYSQL_USER_DB="testdb" \
-e CREATE_MYSQL_BASIC_USER_AND_DB=true \
-v mysql-data:/var/lib/mysql/ \
dgraziotin/mysql
```

接著我們可以執行 `sudo ls /var/lib/docker/volumes/mysql-data/_data`

可以看到 db 資料已產出在 host 對應的 folder

```
ib_logfile0  ib_logfile1  ibdata1  mysql  performance_schema
 testdb
```

直接掛載 host 指定之 path

建立 test folder

`mkdir test && ls -al test`

輸出訊息為

```
total 8
drwxrwxr-x 2 user user 4096 Jun  1 09:21 .
drwxr-xr-x 9 user user 4096 Jun  1 09:21 ..
```

使用 -v 進行 volume

```
docker run --rm \
-v `pwd`/test:/root/test \
ubuntu \
touch /root/test/readme.md
```

執行

`ls -al test`

確認輸出產出檔案

```
total 8
drwxrwxr-x 2 user user 4096 Jun  1 09:24 .
drwxr-xr-x 9 user user 4096 Jun  1 09:21 ..
-rw-r--r-- 1 root root    0 Jun  1 09:24 readme.md
```

在 docker 內所產生或操作的檔案，將直接同步到 host 的資料夾

從 docker 進行 volume

與直接 create 不同，將檔案實體存在 docker 內，所以無法在 host 直接存取檔案

run docker 並給予別名

```
docker run \
--name sample-data \
-v /root/test \
ubuntu
```

使用 volumes-from 並建立一個檔案

```
docker run --rm \
--volumes-from sample-data \
ubuntu \
touch /root/test/readme.md
```

再次確認該檔案確實存在

```
docker run --rm \
--volumes-from sample-data \
ubuntu \
ls -al /root/test
```

輸出訊息如下

```
total 8
drwxr-xr-x 2 root root 4096 Jun  1 09:13 .
drwx------ 3 root root 4096 Jun  1 09:18 ..
-rw-r--r-- 1 root root    0 Jun  1 09:13 readme.md
```

link container

container 在使用使很多時候需要跟多個 container 進行溝通，舉例來說，在 docker 的使用上，很多時候我們會將前端跟後端如 DB 分為兩個 container。

啟動 DB container

```
docker run -d --name db localhost:5000/training-postgres
```

啟動 web container 並且透過 link 與 DB 進行溝通

```
docker run --name web --link db:db localhost:5000/training-webapp ping db
```

一旦有 link 的 option 定義，就可以直接透過 container name 進行存取，上述指令執行後將出現

```
PING db (172.17.0.2) 56(84) bytes of data.
64 bytes from db (172.17.0.2): icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from db (172.17.0.2): icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from db (172.17.0.2): icmp_seq=3 ttl=64 time=0.060 ms
```

代表 ip 是正常可以存取的

透過下面指令，我們可以查詢到哪些 container 是 link 在一起的

```
docker inspect -f "{{ .HostConfig.Links }}" web
```

將輸出

```
[/db:/web/db]
```

我們可以再進一步驗證，拿掉 `--link db:db`

```
docker rm -f web
docker run --name web localhost:5000/training-webapp ping db
```

此時輸出的訊息為

```
ping: unknown host db
```

expose

用於 container 跟 container 之間的 port 的溝通，也就是說定義 expose 的 port 號只能被其他 container 存取，不能被 host 也就是外部存取

啟動 mysql 作為 server

```
docker run -d \
--name mysql-server \
-e MYSQL_ADMIN_PASS="pass" \
--expose 3306 \
dgraziotin/mysql
```

執行 `docker ps`

將會看到下面結果，可以看到 ports 為 `3306/tcp`

```
CONTAINER ID        IMAGE               COMMAND             C
REATED              STATUS              PORTS               NA
MES
5af9391204b2        dgraziotin/mysql    "/run.sh"           4
 seconds ago        Up 3 seconds        3306/tcp            my
sql-server
```

啟動 mysql 作為 client

```
docker run -it --rm \
--name mysql-client \
--link mysql-server \
dgraziotin/mysql \
mysql -uadmin -h mysql-server -ppass
```

將會看到 mysql 已正確連接

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.47-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All r
ights reserved.

Oracle is a registered trademark of Oracle Corporation and/or
 its
affiliates. Other names may be trademarks of their respective
owners.
```

network create

透過 network create 並且在運行 container 時，透過指定 network 也可以讓 container 進行溝通

建立 default bridge

`docker network create -d bridge test_net`

運行 docker 加入 network option

啟動 DB container

`docker run -d --net test_net --name db localhost:5000/training-postgres`

`docker network inspect test_net`

```
 "Containers": {
     "0a7013db1d5cf56797312eed7fe7894a62d57f7476c4777f5ab4922e
 38c8aa2a": {
         "Name": "db",
         "EndpointID": "32e4cd2a6ad0feb1af567e1ca6fbc3f5300c7c
 6c1745ec1754cab6a17858b0f9",
         "MacAddress": "02:42:ac:15:00:02",
         "IPv4Address": "172.21.0.2/16",
         "IPv6Address": ""
     }
 },
```

若如下沒有加 net option 預設為 default 如下

`docker run --name web localhost:5000/training-webapp ping 172.21.0.2`

將沒辦法 ping 到，若同樣加上 --net test_net

`docker run --name web --net test_net localhost:5000/training-webapp ping 172.21.0.2`

將顯示

```
PING 172.21.0.2 (172.21.0.2) 56(84) bytes of data.
64 bytes from 172.21.0.2: icmp_seq=1 ttl=64 time=0.064 ms
64 bytes from 172.21.0.2: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 172.21.0.2: icmp_seq=3 ttl=64 time=0.131 ms
64 bytes from 172.21.0.2: icmp_seq=4 ttl=64 time=0.070 ms
```

表示連線正常

port

port 與 expose 有所不同，port 設置主要是要讓 container 與 host 進行溝通

下述的設置，將可以讓 container 內的 80 port 可以從 host 8888 port 進行存取

```
docker run --name nginx -d -p 8888:80 nginx
```

一旦運行完成後可以執行下述指令驗證

```
curl http://localhost:8888
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
 installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

表示已正確存取 nginx 的 welcome page

若不指令 host port 的話

```
docker run --name nginx -d -p 80 nginx
```

docker 將會自行指定 random port 以便 host 可以存取，透過 docker ps 來看一下

```
CONTAINER ID        IMAGE               COMMAND
    CREATED             STATUS              PORTS
                NAMES
f614261df0b1        nginx               "nginx -g 'daemon off
"   2 seconds ago       Up 1 seconds        443/tcp, 0.0.0.0:
32776->80/tcp   nginx
```

將可以透過 `http://localhost:32776` 存取 nginx

透過 docker-compose 使用 network

範例專案： https://github.com/agileworks-tw/docker-basic-sample.git

docker-compose

```
version: '2'
services:
  web:
    image: hiromasaono/curl
    networks:
      - back-tier

  nginx:
    image: nginx
    networks:
      - back-tier

networks:
  back-tier:
    driver: bridge
  front-tier:
    driver: bridge
```

上面的 docker-compose 我們可以先透過下列指令

```
docker-compose up nginx
```

接著

```
docker-compose run web /bin/bash
```

進入 docker container 內之後執行

```
ping nginx
```

將會顯示

```
64 bytes from network_nginx_1.network_back-tier (172.21.0.2):
 icmp_seq=1 ttl=64 time=0.108 ms
64 bytes from network_nginx_1.network_back-tier (172.21.0.2):
 icmp_seq=2 ttl=64 time=0.134 ms
64 bytes from network_nginx_1.network_back-tier (172.21.0.2):
 icmp_seq=3 ttl=64 time=0.099 ms
```

則表示網路是可以通的，若我們把上面的 docker-compose 內容改為

```
version: '2'
services:
  web:
    image: hiromasaono/curl
    networks:
      - front-tier

  nginx:
    image: nginx
    networks:
      - back-tier

networks:
  back-tier:
    driver: bridge
  front-tier:
    driver: bridge
```

此時進行上面的指令，將會出現

```
ping: unknown host nginx
```

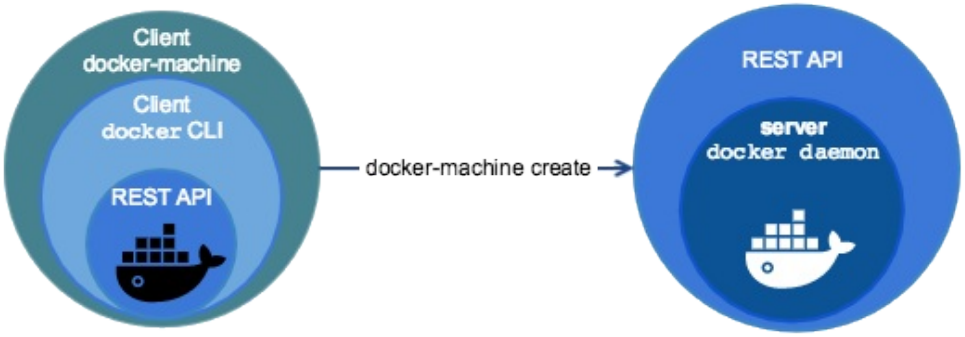
使用 docker network 可以很方便地進行環境建置，即使是不同的 container 也可以很方便的組合

docker-machine

所謂的 Docker Machine 簡單來說，就是一個可以讓你快速在不同平台上建立 Docker engine 的 command tool。

以往，如果你要建立一個 Docker engine，必須先 login 對應的平台（不管是 Cloud，如：AWS、Azure, 或自己的電腦、VM），進行各種設定。Docker Machine 提供了一個單一介面，讓你可以透過這個介面直接去控制各種不同 Host 上面的 Docker。

也就是說，如果你在很多平台上面都有 Docker, 現在可以不必分別登入，可以直接從 Docker Machine 這個介面去統一管理，並直接指定某個平台上的機器執行 Docker 命令。Docker Machine 讓你可以快速在任何平台上建立伺服器並安裝好 Docker, 完成與 Client 端溝通的設定。它建立的 Docker 主機預設就是安全的，伺服器與 client 端的連接都有進行加密和授權。



上圖上半段，可以看到 docker-machine 的運作原理，透過 docker-machine 對 client 的 docker CLI 操作 REST API 來進行遠端的 docker engine 建立。

而實際上就像上述所說明，透過 docker-machine 我們就可以如上圖下方，操作多台 docker engine 進行 container 之運行。

install machine

On OS X or Linux

```
$ curl -L https://github.com/docker/machine/releases/download
/v0.7.0/docker-machine-`uname -s`-`uname -m` >/usr/local/bin/
docker-machine && \
  chmod +x /usr/local/bin/docker-machine
```

On Windows with git bash

```
$ if [[ ! -d "$HOME/bin" ]]; then mkdir -p "$HOME/bin"; fi &&
 \
curl -L https://github.com/docker/machine/releases/download/v
0.7.0/docker-machine-Windows-x86_64.exe > "$HOME/bin/docker-m
achine.exe" && \
chmod +x "$HOME/bin/docker-machine.exe"
```

download page

https://github.com/docker/machine/releases/

Downloads

docker-machine-Darwin-x86_64	38.9 MB
docker-machine-Linux-x86_64	34.6 MB
docker-machine-Windows-i386.exe	27.1 MB
docker-machine-Windows-x86_64.exe	34.8 MB
Source code (zip)	
Source code (tar.gz)	

machine driver

官方支援的 driver 連結如下：

https://docs.docker.com/machine/drivers/

- Amazon Web Services
- Microsoft Azure
- Digital Ocean
- Exoscale
- Google Compute Engine
- Generic
- Microsoft Hyper-V
- OpenStack
- Rackspace
- IBM Softlayer
- Oracle VirtualBox
- VMware vCloud Air
- VMware Fusion
- VMware vSphere

大部分主流的 cloud provider 都有支援可以令你快速進行 docker Engine 的建立

透過 Docker Machine 建立 Docker Engine

VirtualBox

建立 virtualBox 的 Docker Engine 是最簡單的，執行下列指令即可

```
docker-machine create -d virtualbox dev
```

如此就會在你的 virtualBox 建立一 dev 的虛擬機

DigitalOcean

建立 DigitalOcean 之 docker Machine 首先必須要有 Access Token，我們可以把屬於你的 Token 透過環境變數載入，Docker 將會預設讀取對應的變數名稱，範例如下

```
export DIGITALOCEAN_ACCESS_TOKEN=123456
```

接著就可以指定 Driver 以及你所希望對應的 image

```
docker-machine create \
--driver digitalocean \
--digitalocean-image ubuntu-15-10-x64 \
digitalocean-dev
```

Docker Machine env

為了能夠操作多台 Docker Engine，Machine 會把相關的資訊存在你的 `$HOME/.docker/machine` 底下，若需要讀取時，你可以透過

`docker-machine env`

來進行存取。

首先，透過 create 一 docker Engine 透過 driver 為 virtualBox

`docker-machine create -d virtualbox test`

`docker-machine create -d virtualbox dev`

建立好之後，我們可以透過 `env` 指令來查看相關的資訊

`docker-machine env test`

將可以看到類似下面的訊息

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:2376"
export DOCKER_CERT_PATH="/Users/spooky/.docker/machine/machin
es/test"
export DOCKER_MACHINE_NAME="test"
# Run this command to configure your shell:
# eval $(docker-machine env test)
```

若是查看 `dev`

`docker-machine env dev`

將可以看到

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/spooky/.docker/machine/machin
es/dev"
export DOCKER_MACHINE_NAME="dev"
# Run this command to configure your shell:
# eval $(docker-machine env dev)
```

不同的 Docker Engine 其對應的資訊將會有所不同

可以看到最後的註解

```
# Run this command to configure your shell:
# eval $(docker-machine env test)
```

我們可以透過下面指令切換到 test

`eval $(docker-machine env test)`

也可以透過下面指令切換到 dev

`eval $(docker-machine env dev)`

透過指令的執行把相關 environment 載入，當你需要管理多台 Docker Engine 時，不只是本地的 VM 遠端的 cloud 也可以透過類似的方式進行 Docker 的管理跟操作，將會是很方便的 tool。

列出所有 docker-machine 相關資訊

跟 docker image 或是 container 一樣，若要查詢相關資訊，也可以透過 docker-machine 所提供的 `ls` 來檢視目前管理的 docker engine 有哪些。

假設我們透過 docker-machine 建立兩個 virtualBox 之 Docker engine

`docker-machine create -d virtualbox test`

`docker-machine create -d virtualbox dev`

並且透過 `env` 載入環境變數進行 docker 操作，透過下列指令

`eval $(docker-machine env dev)`

接著執行

`docker-machine ls`

將會看到下面輸出

```
NAME        ACTIVE   DRIVER       STATE     URL
           SWARM   DOCKER    ERRORS
dev         *        virtualbox   Running   tcp://192.168.99.1
00:2376            v1.10.2
test        -        virtualbox   Running   tcp://192.168.99.1
01:2376            v1.10.3
```

其中可以看到

- ACTIVE: 打上 `*` 表示正在操作的 Docker engine
- STATE: Docker engine 狀態
- URL: engine 所在 ip
- SWARM: 是否為 SWARM master or node，後續章節會再行介紹
- DOCKER: 建立 Docker 時所安裝的 Docker 版本

使用 docker-machine 連結已存在之遠端 docker engine

假設你已經有個 remote 的 ubuntu server，在該 server 上已經有安裝 docker engine，一般來說你需要管理 docker 時，可能就需要透過 ssh 連上 server 之後進行 docker 的操作。

透過 docker-machine 之 generic driver 將可以已讓你在你的電腦集中管理已存在的 ubuntu server。

ubuntu user 必須是 sudo 並且使用 sudo 時不需要 password

若你不是用 root 帳號，可以透過下列指令，讓你的 linux user 符合條件

```
sudo visudo
```

```
yourname ALL=(ALL) NOPASSWD: ALL
```

輸入 Ctrl-X 離開，你的 user 進行 sudo 時將不再需要打密碼，如此才可進行 docker-machine 建立

建立 docker-machine

```
docker-machine create \
--driver generic \
--generic-ip-address=11.22.33.44 \
--generic-ssh-key=~/.ssh/id_rsa \
ubuntuvm
```

載入環境變數進行 Docker engine 操作

```
eval "$(docker-machine env ubuntuvm)"
```

如此一來你就可以在你的電腦操做已存在之 remote docker engine

docker-machine 升級

Docker 目前處在快速發展的當下，其版本更新非常快速，很多新特性也一直不停的推陳出新，很多時候我們要進行升級時，需要連線到 host 主機才能進行，但假設一開始就用 docker-machine 在進行 Docker Host 的管理，我們將可以很方便的進行升級。

假設我們已經有兩台 docker host 透過 ls 查出下面資訊

```
➜  ~ docker-machine ls
NAME        ACTIVE   DRIVER       STATE     URL
           SWARM   DOCKER    ERRORS
dev         *        virtualbox   Running   tcp://192.168.99.1
00:2376            v1.10.2
test        -        virtualbox   Running   tcp://192.168.99.1
01:2376            v1.10.3
```

可以看到其中 dev 是屬於較舊的版本，我們將可以透過下面指令進行更新

`docker-machine upgrade dev`

將會看到下面訊息的輸出

```
Waiting for SSH to be available...
Detecting the provisioner...
Upgrading docker...
Stopping machine to do the upgrade...
Upgrading machine "dev"...
Copying /Users/spooky/.docker/machine/cache/boot2docker.iso t
o /Users/spooky/.docker/machine/machines/dev/boot2docker.iso.
..
Starting machine back up...
(dev) Check network to re-create if needed...
(dev) Waiting for an IP...
Restarting docker...
```

再次查詢 docker-machine 狀態

```
➜  ~ docker-machine ls
NAME        ACTIVE   DRIVER       STATE     URL
          SWARM   DOCKER    ERRORS
dev         *        virtualbox   Running   tcp://192.168.99.1
00:2376           v1.10.3
test        -        virtualbox   Running   tcp://192.168.99.1
01:2376           v1.10.3
```

可以看到 docker 已被更新到最新的版本。

有些時候 docker 會有些問題，不妨升級看看，說不定官方已經解決。

docker-machine 移除

docker-machine 可以建立，當然也可以移除，同樣的我們先建立一 docker-machine

`docker-machine create -d virtualbox test`

查詢 docker-machine 狀態

```
➜  ~ docker-machine ls
NAME        ACTIVE   DRIVER       STATE     URL
           SWARM   DOCKER    ERRORS
test         *        virtualbox   Running   tcp://192.168.99.
100:2376           v1.10.3
```

進行 docker-machine 刪除

`docker-machine rm test`

執行結果如下

```
About to remove test
Are you sure? (y/n): Y
Successfully removed test
```

java 開發環境安裝

在一般標準開發環境下，我們需要安裝 Java 相關的 library

安裝 java 8 & maven

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
sudo apt-get install maven
```

選擇 java 8 作為預設 java version

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

確認 java 及 maven 安裝完成

```
java -version
maven -v
```

java 開發環境安裝

在一般標準開發環境下，我們需要安裝 Java 相關的 library，其安裝指令如下

```
apt-get install software-properties-common
add-apt-repository ppa:openjdk-r/ppa -y
apt-get update
apt-get install openjdk-8-jdk
apt-get install maven
```

若要自行建置 docker 的 java 運行環境，我們需要先取得 base image，我們使用 `ubuntu:latest`

pull ubuntu 作為 base image

`docker pull ubuntu:latest`

一但執行完成後，我們可透過 run 進入 ubuntu container

run ubuntu with TTY

`docker run -it ubuntu:latest /bin/bash`

將會看到類似，ID 會因為每個人而有所不同，此 ID 將會在安裝完成後進行後續操作識別用

`root@94fe61e92bbb:/#`

表示目是已經可以對 ubuntu docker container 進行操作，接著我們可以執行 java 相關運行環境的安裝

```
apt-get install software-properties-common
add-apt-repository ppa:openjdk-r/ppa -y
apt-get update
apt-get install openjdk-8-jdk
apt-get install maven
```

安裝完成後，透過下列指令確認安裝正確

`java -version`

正確將輸出下面內容

```
openjdk version "1.8.0_72-internal"
OpenJDK Runtime Environment (build 1.8.0_72-internal-b15)
OpenJDK 64-Bit Server VM (build 25.72-b15, mixed mode)
```

`mvn -v`

正確將輸出下面內容

```
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.8.0_72-internal, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "4.1.19-boot2docker", arch: "amd64
", family: "unix"
```

將 Container commit 為 java8-maven3 image

首先退出 ubuntu Container

```
root@94fe61e92bbb:/# exit
```

一旦退出後 container `94fe61e92bbb` 將會立即停止，透過

`docker ps -a`

查看目前所有 Container 的狀態，如下

```
CONTAINER ID        IMAGE               COMMAND
    CREATED             STATUS                          PORTS
            NAMES
94fe61e92bbb        ubuntu:latest       "/bin/bash"
    20 minutes ago      Exited (130) 8 seconds ago
            admiring_elion
```

接著我們可以把 container `94fe61e92bbb` commit 為新的 image

`docker commit 94fe61e92bbb trunk/java8-maven3`

透過

`docker images`

查看是否有剛剛 commit 完成的 images，預期將看到下面內容

```
REPOSITORY                    TAG                 IMAGE ID
         CREATED             SIZE
trunk/java8-maven3            latest              2c957733181
9        8 seconds ago       732.4 MB
```

表示已經完成開發環境的建置

確認 docker image 可以正確運行

透過下面指令確認 docker 之 java 及 maven 版本

`docker run --rm trunk/java8-maven3 java -version`

```
openjdk version "1.8.0_72-internal"
OpenJDK Runtime Environment (build 1.8.0_72-internal-b15)
OpenJDK 64-Bit Server VM (build 25.72-b15, mixed mode)
```

```
docker run --rm trunk/java8-maven3 mvn -v
```

```
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.8.0_72-internal, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "4.1.19-boot2docker", arch: "amd64
", family: "unix"
```

如此就完成 docker 之 java 環境建置。

java 開發環境安裝

在一般標準開發環境下，我們需要安裝 Java 相關的 library，其安裝指令如下

```
apt-get install software-properties-common
add-apt-repository ppa:openjdk-r/ppa -y
apt-get update
apt-get install openjdk-8-jdk
apt-get install maven
```

若要自行建置 docker 的 java 運行環境透過 Dockerfile，我們需要先使用 base image 為 `ubuntu:latest`

使用 Dockerfile 時，一旦開始進行建置，Docker 會把 Dockerfile 所在位置底下所有的檔案載入以便建置過程中使用，為了加速建置程序，一般來說一個 Dockerfile 請放置於獨立的資料夾

建立 Dockerfile 存放之資料夾

```
mkdir java8-maven3-build
cd java8-maven3-build
```

建立 Dockerfile

```
cat > Dockerfile <<EOL
FROM ubuntu:latest

RUN apt-get install -y software-properties-common
RUN add-apt-repository ppa:openjdk-r/ppa -y
RUN apt-get update
RUN apt-get install -y openjdk-8-jdk
RUN apt-get install -y maven
EOL
```

進行 Dockerfile build

```
docker build -t trunk/java8-maven3-build .
```

確認 Dockerfile build 完成

`docker images`

查看是否有剛剛 commit 完成的 images，預期將看到下面內容

```
REPOSITORY                    TAG                 IMAGE ID
         CREATED             SIZE
trunk/java8-maven3-build      latest              b8ce33c94dc
c        8 seconds ago       737.1 MB
```

表示已經完成開發環境的建置

確認 docker image 可以正確運行

透過下面指令確認 docker 之 java 及 maven 版本

`docker run --rm trunk/java8-maven3-build java -version`

```
openjdk version "1.8.0_72-internal"
OpenJDK Runtime Environment (build 1.8.0_72-internal-b15)
OpenJDK 64-Bit Server VM (build 25.72-b15, mixed mode)
```

```
docker run --rm trunk/java8-maven3-build mvn -v
```

```
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.8.0_72-internal, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "4.1.19-boot2docker", arch: "amd64
", family: "unix"
```

如此就完成 docker 之 java 環境建置。

範例專案以 Java 為例

為了可以將 Docker 或 Jenkins 實際應用在日常開發或是部署上，需要先了解一個專案的基本結構，此章節將說明在開發中一個標準的專案須有哪些環節需要注意，此章節開始前，假設已經完成 Java 8 以及 Maven 3 的安裝。

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製飯粒專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

建置專案

透過下列指令將會把 maven pom.xml 中定義的套件進行安裝

```
mvn install
```

測試專案

透過下列指令，將會運行自動測試。

```
mvn test
```

運行專案

透過下列指令，將會運行 development mode 作為開發中方確認結果。

```
mvn spring-boot:run
```

透過下列網址

`http://localhost:8000/api`

可以存取範例專案運行結果

打包專案

`mvn package`

執行完，將會看到訊息

```
[INFO] Building jar:
/Users/spooky/projects/agileWorks/spring-boot-sample/target/s
pring-boot-sample-data-rest-0.1.0.jar
```

production run

```
java -jar \
/Users/spooky/projects/agileWorks/spring-boot-sample/target/s
pring-boot-sample-data-rest-0.1.0.jar
```

透過下列網址

`http://localhost:8000/api`

可以存取 production mode 範例專案運行結果。

java 範例專案運行測試

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案套件安裝

運行指令如下：

```
docker run \
--name sample-java-install \
-v `pwd`:/app \
-v /root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn install
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中的 `/app` 路徑
2. `$HOME/.m2:/root/.m2`：將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

如此我們就可以讓每次建置都使用同一個 docker volume container，注意這邊

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn install
```

上述指令執行第二次時，maven 就不會再次 download 相關 jar 檔，`sample-java-data` container 也可以作為後續建置所需的 cache 來源。

`-w` or `--workdir`

為執行 CMD 所在檔案路徑，也就是說 `mvn install` 將會在 container 中的 `/app` 執行，而因為 `-v` 已將範例專案掛載到 `/app` 就像在 host 執行指令一樣。

執行安裝指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn install` 則為此次建置要使用的 command 將此專案所需的套件進行安裝。

java 範例專案運行測試

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn test
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中

的 `/app` 路徑

2. `$HOME/.m2:/root/.m2` : 將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn test
```

相關說明可以參考 java project install

執行測試指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn test` 將會運行預先寫好的 test case，完成專案測試運行。

java 範例專案運行運行

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
-p 8000:8000 \
maven:3-jdk-8 \
mvn spring-boot:run
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中的 `/app` 路徑
2. `$HOME/.m2:/root/.m2`：將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
-p 8000:8000 \
maven:3-jdk-8 \
mvn spring-boot:run
```

相關說明可以參考 java project install

`-p` or `--publish`

對於 docker container 而言，預設 container 是不對外開放 port 的，我們可以根據我們運行上的需要開放必要的 port。

`-p 8000:8000` 代表的意思就是將 container 內的 8000 publish 到 host 8000 port 令我們可以存取 service。

執行專案運行

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn spring-boot:run` 將會將專案運行，透過下列網址進行存取

http://localhost:8000/api

java 範例專案打包

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn package
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中

的 `/app` 路徑

2. `$HOME/.m2:/root/.m2` : 將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn package
```

相關說明可以參考 java project install

執行打包指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn package` 將會專案打包程序，執行完成後，將會看到下列訊息

`Building jar: /app/target/spring-boot-sample-data-rest-0.1.0.jar`

代表建置完成。

run and up

run

使用 run 時，預設將沒辦法讓共享的 port 可以被 expose 出去

也就是說假若妳用 run 啟動了一個 web 將沒有辦法讓啟動完成的 docker 可以被外部連結

但使用 docker compose 用於在 jenkins 自動化測試時，在 docker 內運行的錯誤將可以被正確回傳 exit code

如此一來，就可以正確反應 test 狀態。

不過運行 run 不能 expose 的問題已被解決，目前可以使用：

1. 希望開啟特定 service 的 port 但不想被掛到 host: service-ports

   EX: docker-compose run --service-ports mysql web

2. 要將特定 port 掛到 host 可以使用： --publish or -p

   docker-compose run --publish 1337:1337 web

up

基本上就會將在 docker compose 中有開啟的 port 還有掛載到 host 的 port，

以及對應 links 相關的 Containers 一次啟動完成。

其他有用指令

extends

可以繼承於某一個特定的 docker-compose-yml

ex:

```
extends:
  file: common.yml
  service: webapp
```

Variable substitution

與 extends 類似，不過可以透過 env 變數進行相關參數替換

ex:

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```

製作 docker-compose 之 dev mode service

難易度：進階

練習說明

在之前的章節中，我們可以透過

```
docker run -it ubuntu:latest /bin/bash
```

來進入 docker container

透過 volume 來進行相關檔案掛載，如：

```
docker run -v `pwd`:/app -v /root/.m2 -w /app maven:3-jdk-8 m
vn install
```

透過上述兩個指令的組合可以完成透過 docker 來進行 java 語言環境進行軟體開發。

針對範例專案，請完成透過 docker-compose.yml 關於 dev mode 的定義，也就是說可以進入 docker 中 java 語言環境進行軟體開發。

需要完成以下需求

1. 使用 `maven:3-jdk-8` image
2. 將範例專案相關檔案掛載到 container 之 `/app` 資料夾
3. 使用 `volumes-from` 或使用 `volume` 將 `~/.m2` maven cache 掛載到 container，container 內 mavne cache 路徑為 `/root/.m2`
4. workdir 為 `/app`
5. 將 container port 8000 與 host 8000 port 進行 mapping

6. 執行指令為 `/bin/bash`
7. 透過 `docker-compose run --service-ports dev` 來進行運作
8. 完成透過 docker 來進行 java 語言環境進行軟體開發

請將上述步驟轉換為 docker-compose 的相關處理。

java 範例專案運行測試

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案套件安裝

運行指令如下：

```
docker run \
--name sample-java-install \
-v `pwd`:/app \
-v /root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn install
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中的 `/app` 路徑
2. `$HOME/.m2:/root/.m2`：將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

如此我們就可以讓每次建置都使用同一個 docker volume container，注意這邊

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn install
```

上述指令執行第二次時，maven 就不會再次 download 相關 jar 檔，`sample-java-data` container 也可以作為後續建置所需的 cache 來源。

`-w` or `--workdir`

為執行 CMD 所在檔案路徑，也就是說 `mvn install` 將會在 container 中的 `/app` 執行，而因為 `-v` 已將範例專案掛載到 `/app` 就像在 host 執行指令一樣。

執行安裝指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn install` 則為此次建置要使用的 command 將此專案所需的套件進行安裝。

透過 docker compose 運行

將 docker command 轉換為 docker-compose

以下面的指令為例

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn install
```

轉換為 docker-compose.yml 其檔案內容如下：

```
data:
  image: ubuntu
  volumes:
    - /root/.m2

install:
  image: maven:3-jdk-8
  volumes:
    - ./:/app
  volumes_from:
    - data
  working_dir: /app
  command: "mvn install"
```

透過 docker-compose 運行專案安裝

`docker-compose run install`

`volumes_from` 允許格式如下

```
volumes_from:
 - service_name
 - service_name:ro
 - container:container_name
 - container:container_name:rw
```

因為 `volumes_from` 的定義當我們運行 `install` 的 service 時，docker-compose 會連帶的將 `data` service 一併帶起

java 範例專案運行測試

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn test
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中

的 `/app` 路徑
2. `$HOME/.m2:/root/.m2` : 將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn test
```

相關說明可以參考 java project install

執行測試指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn test` 將會運行預先寫好的 test case，完成專案測試運行。

透過 docker compose 運行

將 docker command 轉換為 docker-compose

以下面的指令為例

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn test
```

轉換為 docker-compose.yml 其檔案內容如下：

```
data:
  image: ubuntu
  volumes:
    - /root/.m2

test:
  image: maven:3-jdk-8
  volumes:
    - ./:/app
  volumes_from:
    - data
  working_dir: /app
  command: "mvn test"
```

透過 docker-compose 運行專案安裝

`docker-compose run test`

java 範例專案運行運行

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
-p 8000:8000 \
maven:3-jdk-8 \
mvn spring-boot:run
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中的 `/app` 路徑
2. `$HOME/.m2:/root/.m2`：將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
-p 8000:8000 \
mvn spring-boot:run
```

相關說明可以參考 java project install

`-p` or `--publish`

對於 docker container 而言，預設 container 是不對外開放 port 的，我們可以根據我們運行上的需要開放必要的 port。

`-p 8000:8000` 代表的意思就是將 container 內的 8000 publish 到 host 8000 port 令我們可以存取 service。

執行專案運行

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn spring-boot:run` 將會將專案運行，透過下列網址進行存取

http://localhost:8000/api

透過 docker compose 運行

將 docker command 轉換為 docker-compose

以下面的指令為例

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
-p 8000:8000 \
mvn spring-boot:run
```

轉換為 docker-compose.yml 其檔案內容如下：

```
data:
  image: ubuntu
  volumes:
    - /root/.m2

server:
  image: maven:3-jdk-8
  volumes:
    - ./:/app
  volumes_from:
    - data
  working_dir: /app
  ports:
   - "8000:8000"
  command: "mvn spring-boot:run"
```

透過 docker-compose 運行專案安裝

```
docker-compose up server
```

這邊使用 up 主要是因為要讓 port 可以 publish 所以使用 up 關鍵字

若使用 run 預設 port 是 unpublish 的。

我們一樣可以透過下列網址來進行驗證。

http://localhost:8000/api

java 範例專案打包

範例 github repository

https://github.com/agileworks-tw/spring-boot-sample

clone repository

透過下面指令，將會複製範例專案

```
git clone git@github.com:agileworks-tw/spring-boot-sample.git
```

pull java 8 and maven 3 docker image

```
docker pull maven:3-jdk-8
```

透過 docker image 進行專案測試

運行指令如下：

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn package
```

`-v` or `--volume`

使用了兩次：

1. `${pwd}:/app`：將目前專案所在目錄掛載到 docker container 中

的 `/app` 路徑
2. `$HOME/.m2:/root/.m2` : 將 docker 內的 maven cache 掛載到 host 以加快後續的建置

另外一個方式就是使用 docker volume container

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn package
```

相關說明可以參考 java project install

執行打包指令

最後使用 `maven:3-jdk-8` image 則為官方提供的 maven 環境。

`mvn package` 將會專案打包程序，執行完成後，將會看到下列訊息

`Building jar: /app/target/spring-boot-sample-data-rest-0.1.0.jar`

代表建置完成。

透過 docker compose 運行

將 docker command 轉換為 docker-compose

以下面的指令為例

```
docker run \
--name sample-java-data \
-v /root/.m2 \
ubuntu
```

```
docker run --rm \
-v `pwd`:/app \
--volumes-from sample-java-data \
-w /app \
maven:3-jdk-8 \
mvn package
```

轉換為 docker-compose.yml 其檔案內容如下：

```
data:
  image: ubuntu
  volumes:
    - /root/.m2

package:
  image: maven:3-jdk-8
  volumes:
    - ./:/app
  volumes_from:
    - data
  working_dir: /app
  command: "mvn package"
```

透過 docker-compose 運行專案安裝

`docker-compose run package`

建立 production mode image

建立 Dockerfile

```
FROM java:8-jdk
COPY target/spring-boot-sample-data-rest-0.1.0.jar /app/
EXPOSE 8000
WORKDIR /app
CMD /bin/bash -c 'java -jar spring-boot-sample-data-rest-0.1.
0.jar'
```

因為是 production mode 我們只需要最後打包完成的檔案，所以儘可能精簡執行所需的套件

執行建置指令

需要注意的是，一旦運行 `docker build` ，docker 檔案的存取只能局限於 docker build 運行的所在位置。

因為 build 的指令是直接使用 `spring-boot-sample-data-rest-0.1.0.jar` 所以在 build 之前我們可以先運行 package 指令

```
docker run --rm \
-v `pwd`:/app \
-v $HOME/.m2:/root/.m2 \
-w /app \
maven:3-jdk-8 \
mvn package
```

如此就會產生對應的 jar 檔，接著我們就可以進行 production image 的建置

```
docker build -t agileworks/spring_prod .
```

運行 production image

```
docker run -p 8000:8000 agileworks/spring_prod
```

docker production 發布流程練習

難易度：進階

練習說明

在之前的章節已經完成相關的 docker 說明，

需要完成以下需求

1. 透過 docker login 進行登入
2. 透過 docker-compose 進行專案建置
3. 透過 docker build 進行 production image 建置
4. 透過 docker push 將建立好的 docker 發佈到 docker hub

請將上述步驟相關指令完成並且正確執行。

使用 production image 運行專案進行 preview

難易度：進階

練習說明

完成範例專案之 production image 建置後，我們需要能夠使用該 image 進行實際運作

需要完成以下需求：

1. 透過 docker run 運行建置好的 production image
2. 將 container 內的 port 8000 與 host port 8000 mapping 並且開放
3. 在運行時給予別名 preview
4. 確保下次使用 docker run 時，是新建一個 docker container

請將上述步驟相關指令完成並且正確執行。

jenkins 建立 production deploy task

難易度：進階

練習說明

使用 jenkins 建立範例專案之 production deploy task

需要完成以下需求：

task shell script

目的：完成 production image 建立，並且發佈到 docker hub

1. 透過 docker login 進行登入
2. 透過 docker-compose 進行專案建置
3. 透過 docker build 進行 production image 建置
4. 透過 docker push 將建立好的 docker 發佈到 docker hub

透過 ssh 連線到 production 機器運行建置好的 image

1. 使用 ssh 連到 localhost（模擬連線到遠端機器）
2. 透過 docker pull 取得更新完成的 production image
3. 將 container 內的 port 8800 與 host port 8800 mapping 並且開放
4. 在運行時給予別名 prod
5. 確保下次使用 docker run 時，是新建一個 docker container 重新運作
6. 加上 restart always 令主機重開時能夠自動重啟 docker container

請將上述步驟相關指令完成並且正確執行。

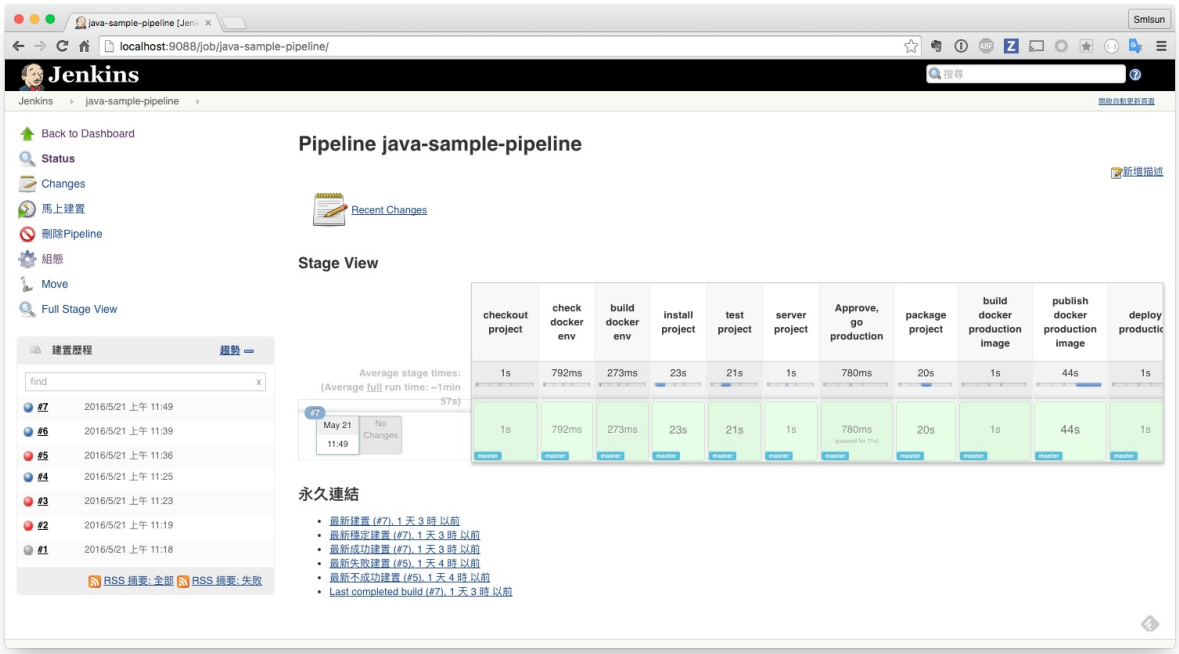
使用 Pipeline 透過 docker 進行建置

我們可以把前面章節所完成的相關 task 濃縮為單一個 Jenkinsfile

```
node {

  stage 'checkout project'
  git url: 'https://github.com/agileworks-tw/spring-boot-samp
le.git'

  stage 'check docker env'

  sh "docker -v"
  sh "docker-compose -v"
  sh "docker ps"

  stage 'build docker env'
  sh "make build-docker-env"

  stage 'test project'
  sh "docker-compose run --rm test"

  stage 'server project'
  sh "docker-compose up -d server"

  try{
    stage 'Approve, go production'
    def url = 'http://localhost:8000/'
    input message: "Does staging at $url look good? ", ok: "D
eploy to production"
  }finally{
    sh "docker-compose stop server"
  }

  stage 'package project'
  sh "docker-compose run --rm package"
```

```
    stage 'build docker production image'
    sh "make build-docker-prod-image"

    stage 'publish docker production image'
    sh "docker push localhost:5000/java_sample_prod"

    stage 'deploy production'
    sh "make deploy-production-local"

}
```

執行結果如下



vt-x amd-v 異常造成 VM 無法開啟

在執行 VM 時出現下列訊行

> 電腦問題(0022) - VirtualBox 錯誤訊息 VT-x/AMD-V 硬體加速在您的系統不可用

我們可以透過下面步驟來進行排除

BIOS 設定

Intel

在 Intel 電腦的 BIOS 中，(有可能是在 Advanced，或者 CPU Configuration 裡面) 找一個名稱如 Intel Virtual Technology 或 Virtualization 之類的選項，如果原設定是 Disabled，則將它改為 Enabled 之後再儲存，離開 BIOS 重開機就可以了。

AMD

在 AMD 電腦的 BIOS 中，(有可能是在 Advanced，或者 CPU/System Configuration 裡面) 找一個名稱是 SVM Mode 或 Virtualization Technology 或 Virtualization 之類的去改為 Enabled，儲存之後離開 BIOS，再重開機就可以了。

參考資料

https://notepad-of-steven.blogspot.tw/2015/08/0022-virtualbox-vt-xamd-v.html

ubuntu docker install

```
sudo apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys
 58118E89F3A912897C070ADBF76221572C52609D
sudo add-apt-repository "deb https://apt.dockerproject.org/re
po ubuntu-$(lsb_release -s -c) main"
sudo apt-get update
sudo apt-get install docker-engine
```

設定 user 可以執行 docker

```
sudo groupadd docker
sudo usermod -a -G docker {username}
sudo service docker restart
```

確認安裝是否完成

```
docker -v
docker ps
docker run hello-world
```

注意事項

如果還是不行，可以重開 ubuntu 再次[確認安裝是否完成]。

docker-compose install

下述指令請用 `root` 執行

```
curl -L https://github.com/docker/compose/releases/download/1
.6.2/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/do
cker-compose
chmod +x /usr/local/bin/docker-compose
```

docker-machine install

下述指令請用 `root` 執行

```
curl -L https://github.com/docker/machine/releases/download/v
0.5.0/docker-machine_linux-amd64.zip >machine.zip && \
unzip machine.zip && \
rm machine.zip && \
mv docker-machine* /usr/local/bin
```

OSX or Windows install docker

安裝 docker-tools

docker-tools: https://www.docker.com/toolbox

安裝 docker-tools 將會自動安裝 Docker，docker-machine，docker-compose

透過 docker-machine 建立 docker engine vm

```
docker-machine create -d virtualbox dev
```

啟動 docker-machine

```
docker-machine start dev
```

預期結果

```
Starting VM...
Started machines may have new IP addresses. You may need to r
e-run the `docker-machine env` command.
```

透過 docker-machine 連結 docker engine vm

執行下列指令完成連結

```
eval $(docker-machine env default)
```

啟動 bash 時自動連結 docker engine vm

執行下列指令將連結

```
eval $(docker-machine env default)
```

確認 docker 運作正常

若要確認是否正常運行，可以執行下列指令：

```
docker ps
```

正常將出現

```
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

若出現下列訊息

```
Get http:///var/run/docker.sock/v1.20/containers/json: dial u
nix /var/run/docker.sock: no such file or directory.
* Are you trying to connect to a TLS-enabled daemon without T
LS?
* Is your docker daemon up and running?
```

就有可能需要執行 `$(docker-machine env default)` 來進行連接。

若再不行，請

1. 確定 Virtual Box 是否最新 ，5.0.x 以上
2. 刪除 Virtual Box 管理介面的 default VM
3. 重新執行 Docker Quickstart Terminal

因為是使用 VM 的關係，若網路斷斷續續也會造成 Docker 網路不穩定，所以若有網路連不上的情形可以再重啟 VM

參考資料

- 《Docker —— 從入門到實踐》正體中文版