Table of Contents

Introduction

講師：劉艾霖 (alincode)

> 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

VM

- 安裝 VirtualBox

Cloud 9

```
帳號：user
密碼：password
```

打開 IDE

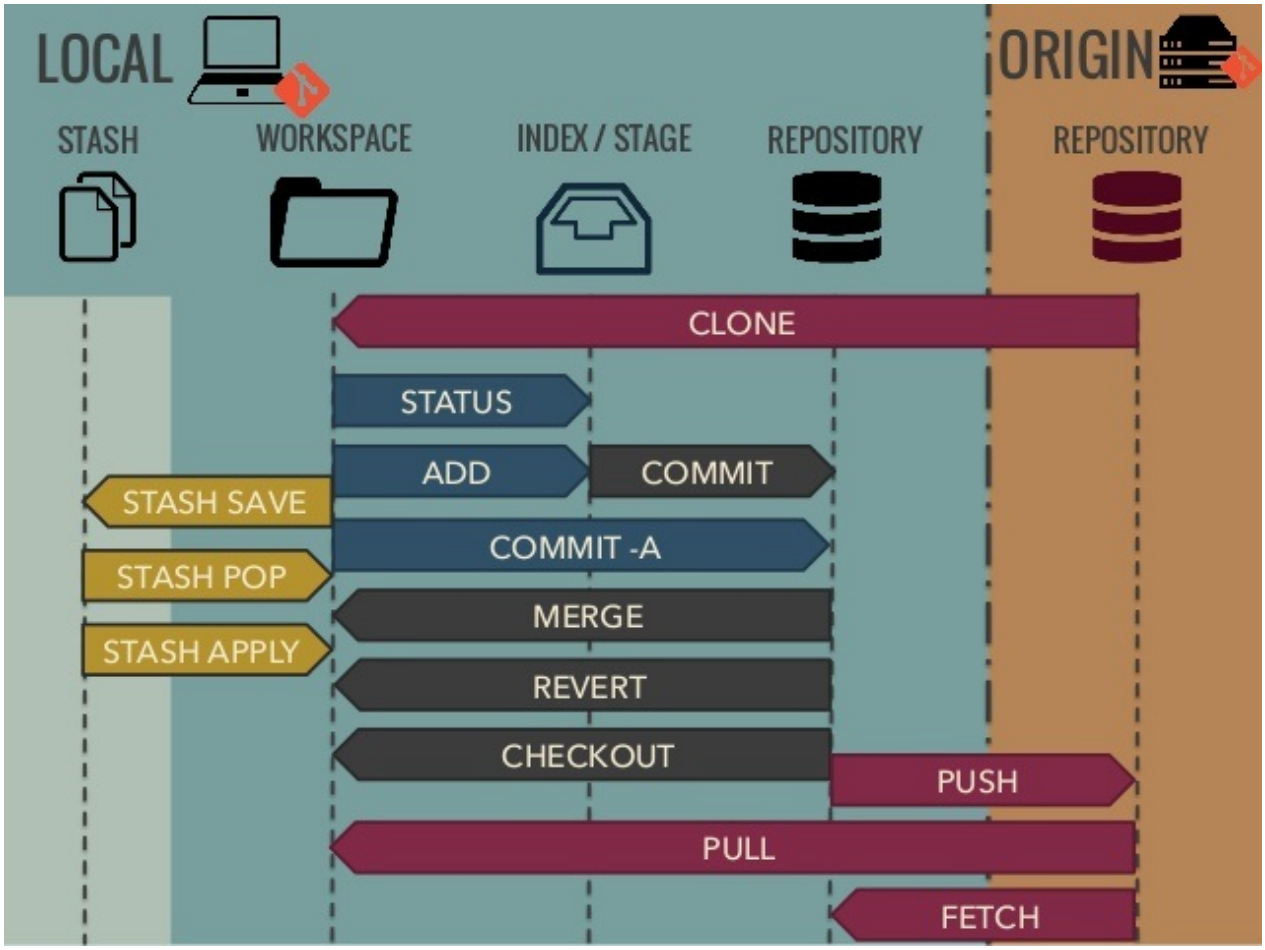
http://localhost:9083

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

- 可以在離線狀態 commit
- 非常快速
- 數不清的新功能
- 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。

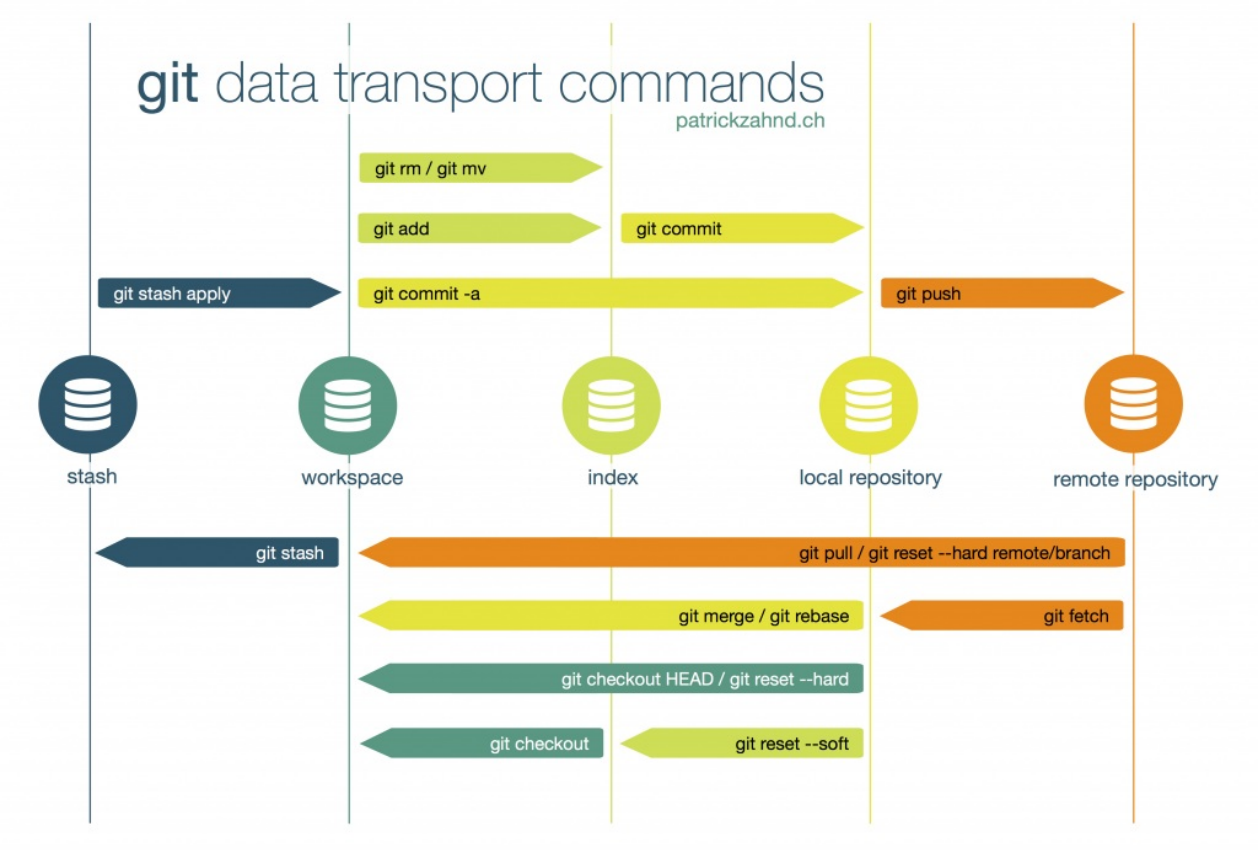
SVN

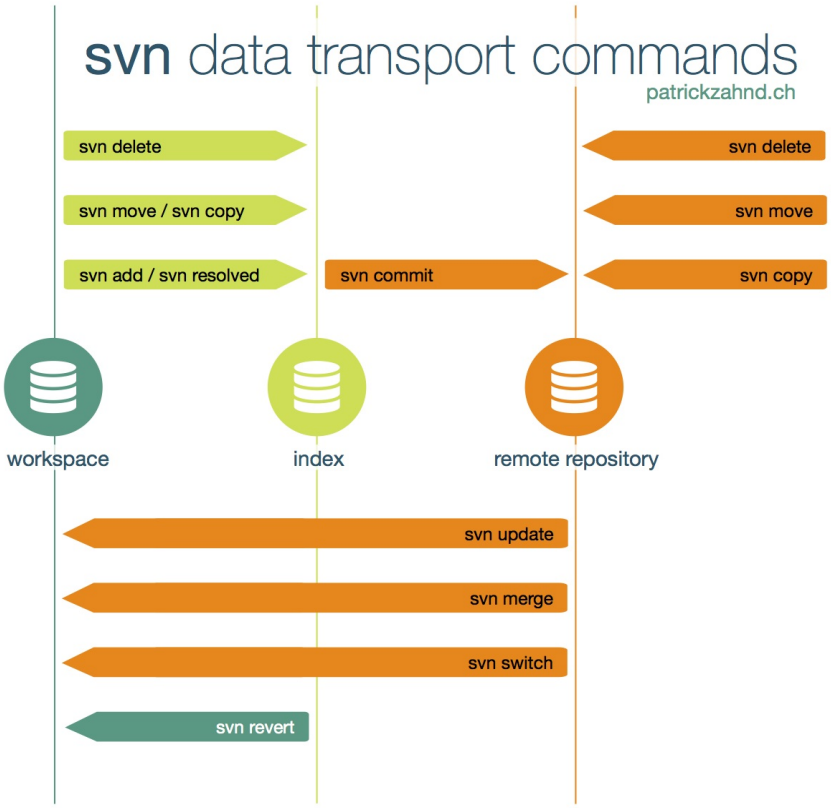
- 不能在離線狀態 commit
- 只要 commit 多的時候，查看 log 容易 timeout
- 很容易發生衝突，甚至只要檔案重新命名，SVN 就認不出來了。
- Squashed commit 壓縮成只有一個 merge-commit，不會有被合併的 log。

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」。

transport


git data transport commands
patrickzahnd.ch
git rm / git mv
git add
git commit
git stash apply
git commit -a
git push
stash
workspace
index
local repository
remote repository
git stash
git pull / git reset --hard remote/branch
git merge / git rebase
git fetch
git checkout HEAD / git reset --hard
git checkout
git reset --soft


svn data transport commands
patrickzahnd.ch
svn delete
svn delete
svn move / svn copy
svn move
svn add / svn resolved
svn commit
svn copy
workspace
index
remote repository
svn update
svn merge
svn switch
svn revert

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

注意事項

- --global 參數比需要緊接著 git config

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.co "checkout"	設定 checkout 暱稱
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：init

情境： 1. 建立一個裸容器

1. 透過 `mkdir init-bare.git` 指令，新增一個資料夾叫 `init-bare.git`
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

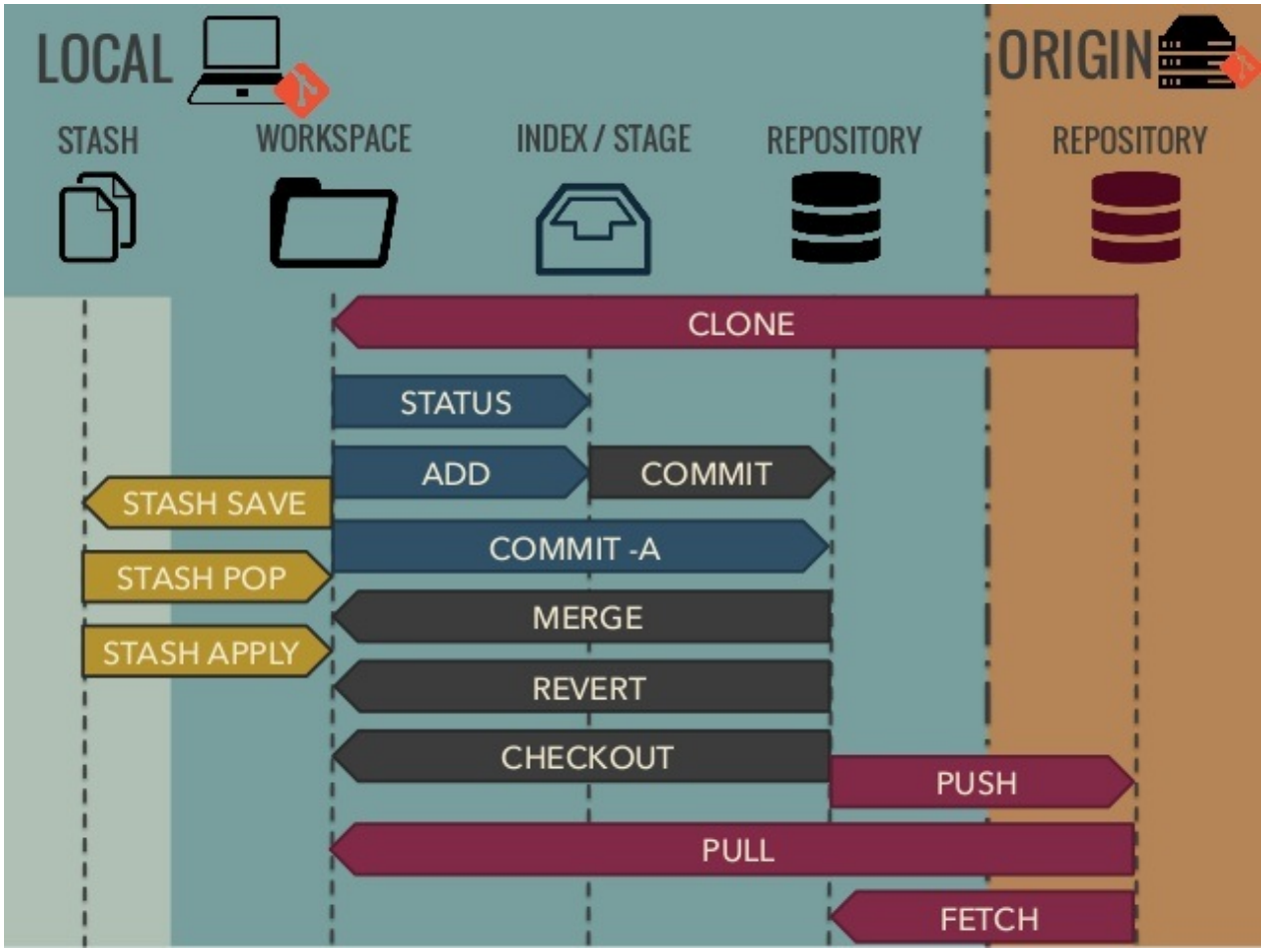
find .git

```
.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



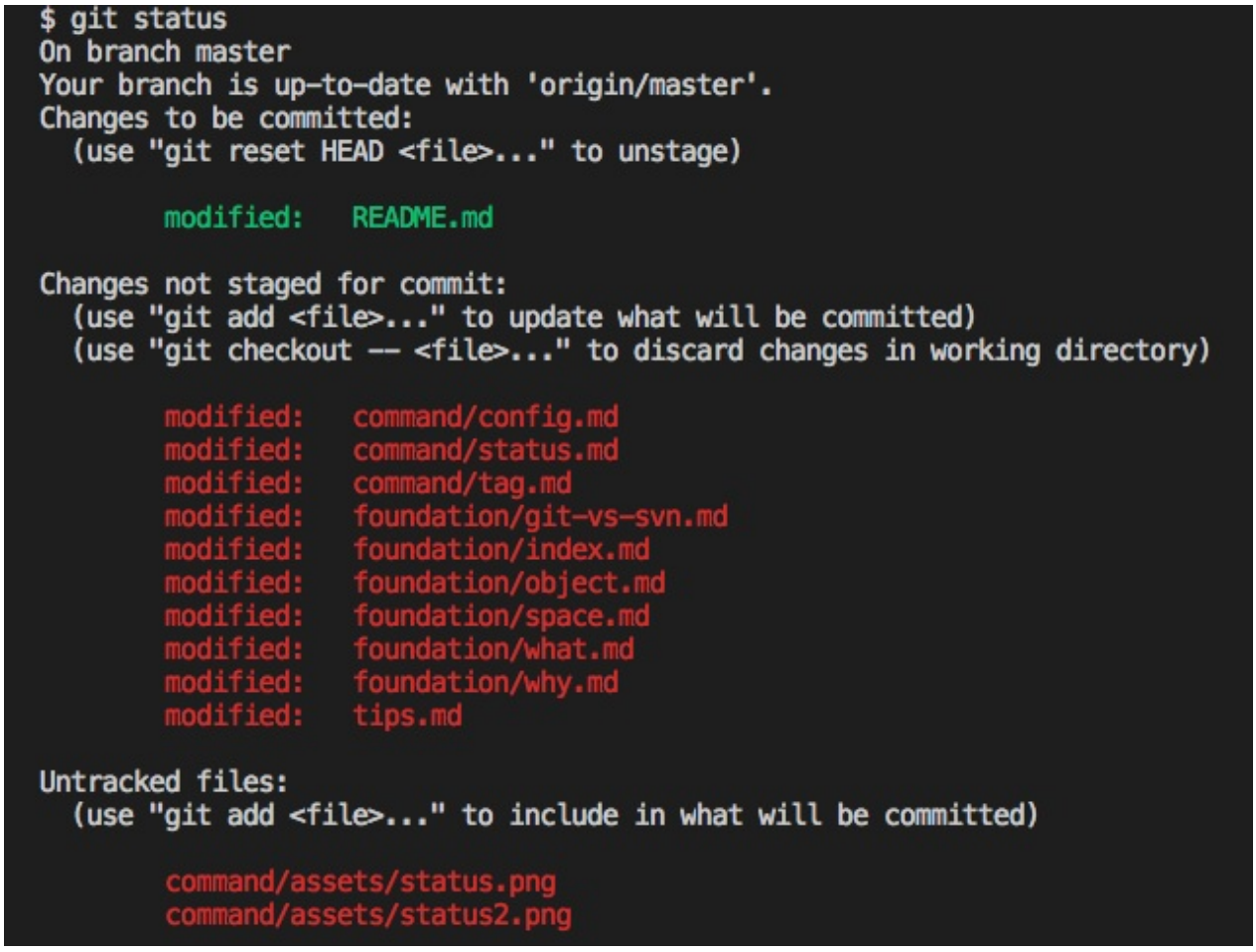
檔案階段	檔案所在位置	說明
已修改	Working Directory(目前的目錄)	經過修改的檔案
已暫存	Staging area(暫存區)	要提交的變動清單
已提交	Repository(容器)	已提交的檔案及變動記錄

git status 指令

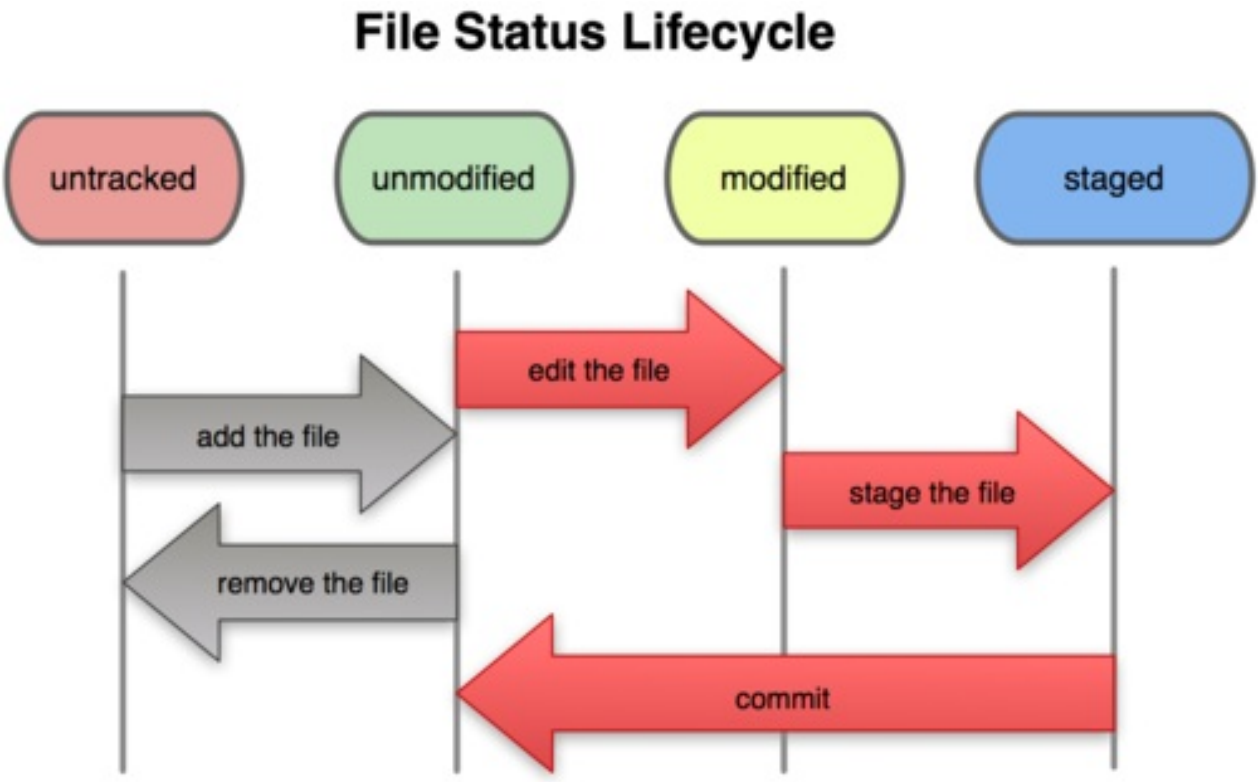
使用情境

- 觀看檔案的狀態

狀態



- 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
```

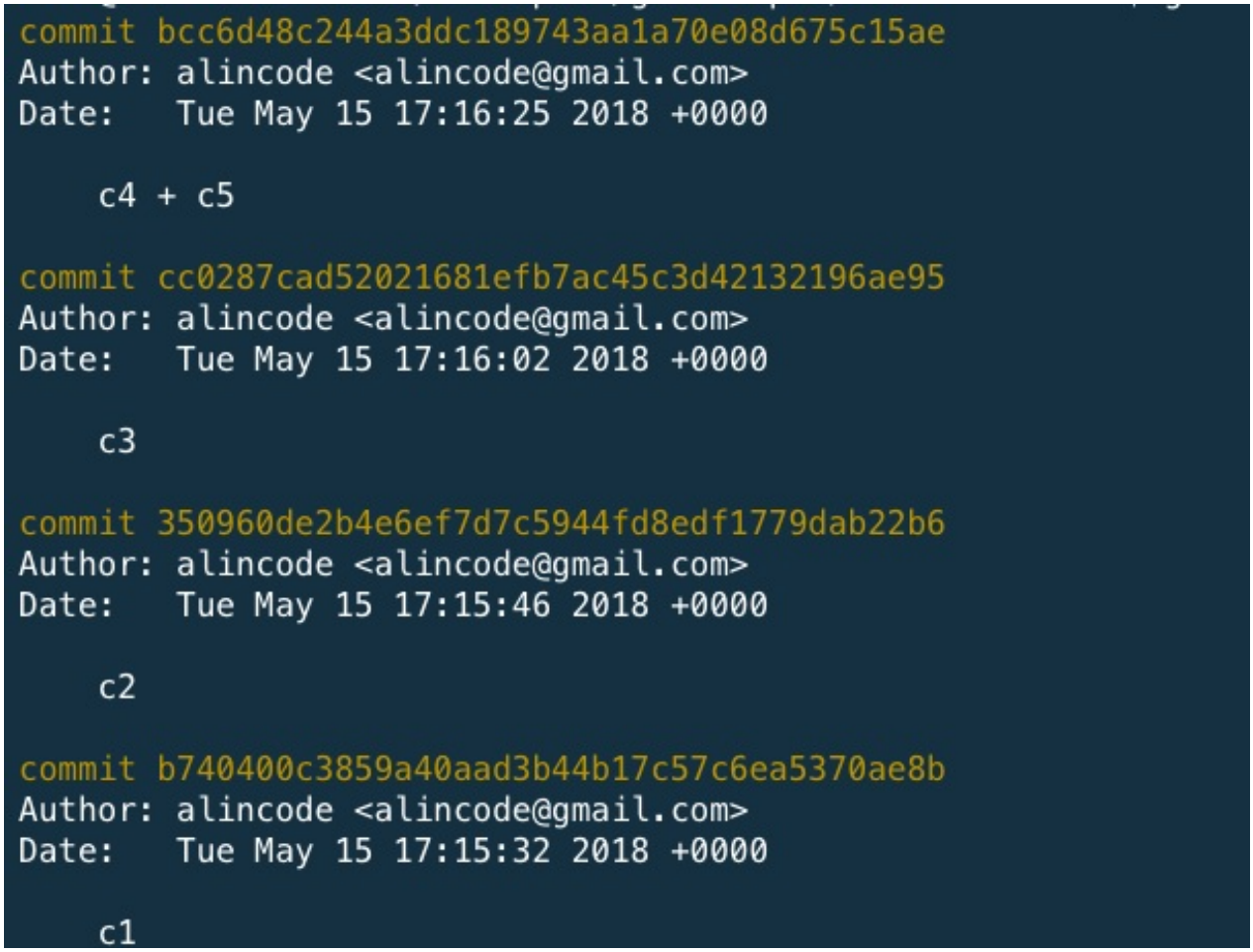
```
    -m, --message <message>
                          commit message
    -c, --reedit-message <commit>
                          reuse and edit message from specifi
ed commit
    -C, --reuse-message <commit>
                          reuse message from specified commit
    --fixup <commit>      use 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：新增兩個送交紀錄

1. 透過 `mkdir commit` 指令，新增一個資料夾叫 `commit`
2. 透過 `git init` 指令，進行專案初始化。
3. 透過 `echo "Hello World" >> 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 >> README.md` 指令，編輯 `README.md`
7. 透過 `git add .` 指令，將所有更變加入 index 中
8. 透過 `git commit -m 'update readme'` 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


commit bcc6d48c244a3ddc189743aa1a70e08d675c15ae
Author: alincode <alincode@gmail.com>
Date: Tue May 15 17:16:25 2018 +0000
c4 + c5
commit cc0287cad52021681efb7ac45c3d42132196ae95
Author: alincode <alincode@gmail.com>
Date: Tue May 15 17:16:02 2018 +0000
c3
commit 350960de2b4e6ef7d7c5944fd8edf1779dab22b6
Author: alincode <alincode@gmail.com>
Date: Tue May 15 17:15:46 2018 +0000
c2
commit b740400c3859a40aad3b44b17c57c6ea5370ae8b
Author: alincode <alincode@gmail.com>
Date: Tue May 15 17:15:32 2018 +0000
c1

```
commit bcc6d48c244a3ddc189743aa1a70e08d675c15ae
Author: alincode <alincode@gmail.com>
Date:   Tue May 15 17:16:25 2018 +0000

    c4 + c5

diff --git a/4.md b/4.md
new file mode 100644
index 0000000..b8626c4
--- /dev/null
+++ b/4.md
@@ -0,0 +1 @@
+4
diff --git a/5.md b/5.md
new file mode 100644
index 0000000..7ed6ff8
--- /dev/null
+++ b/5.md
@@ -0,0 +1 @@
+5
diff --git a/README.md b/README.md
new file mode 100644
index 0000000..d00491f
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+1
```

使用情境

- 查看送交內容 / 更變

常用指令範例

範例	說明
git show README.md	
git log [HEAD]	查看 commit 歷史紀錄
git log -2	查看最後兩筆 commit 歷史紀錄
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	列出包含該檔案變動的提交(包含改名前)
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
git log --since="1 weeks ago"	一週內的 log

- –graph：排列出 commit 節點的演進圖
- –oneline：最精簡的方式顯示．
- –all：顯示所有分支的commit紀錄。
- –decorate：表示要標示分支的名稱。

git log \^[commitA] # A 之後的提交(不列出 A 之前的提交，不含 A) git log -[num] # 最近[num]筆的提交紀錄

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：rm

情境：誤新增了一筆不該放入 repo 的檔案，需要被移除。

1. 透過 `touch system.log` ，新增一筆 log。
2. 透過 `git add .` ，將更變建立索引。
3. 透過 `git commit -m 'add log'` ，新增 commit 紀錄。
4. 透過 `git rm system.log` 指令，刪除 system.log 檔案
5. 透過 `git commit -m 'remove log'` 指令，將刪除檔案的紀錄送交給 repo。
6. 透過 `touch .gitignore` 指令，新增 .gitignore 檔案。
7. 編輯 .gitignore 檔案，加入 *.log
8. 透過 `git add .` ，將更變建立索引。
9. 透過 `git commit -m 'add .gitignore file'` ，新增 commit 紀錄。

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
```

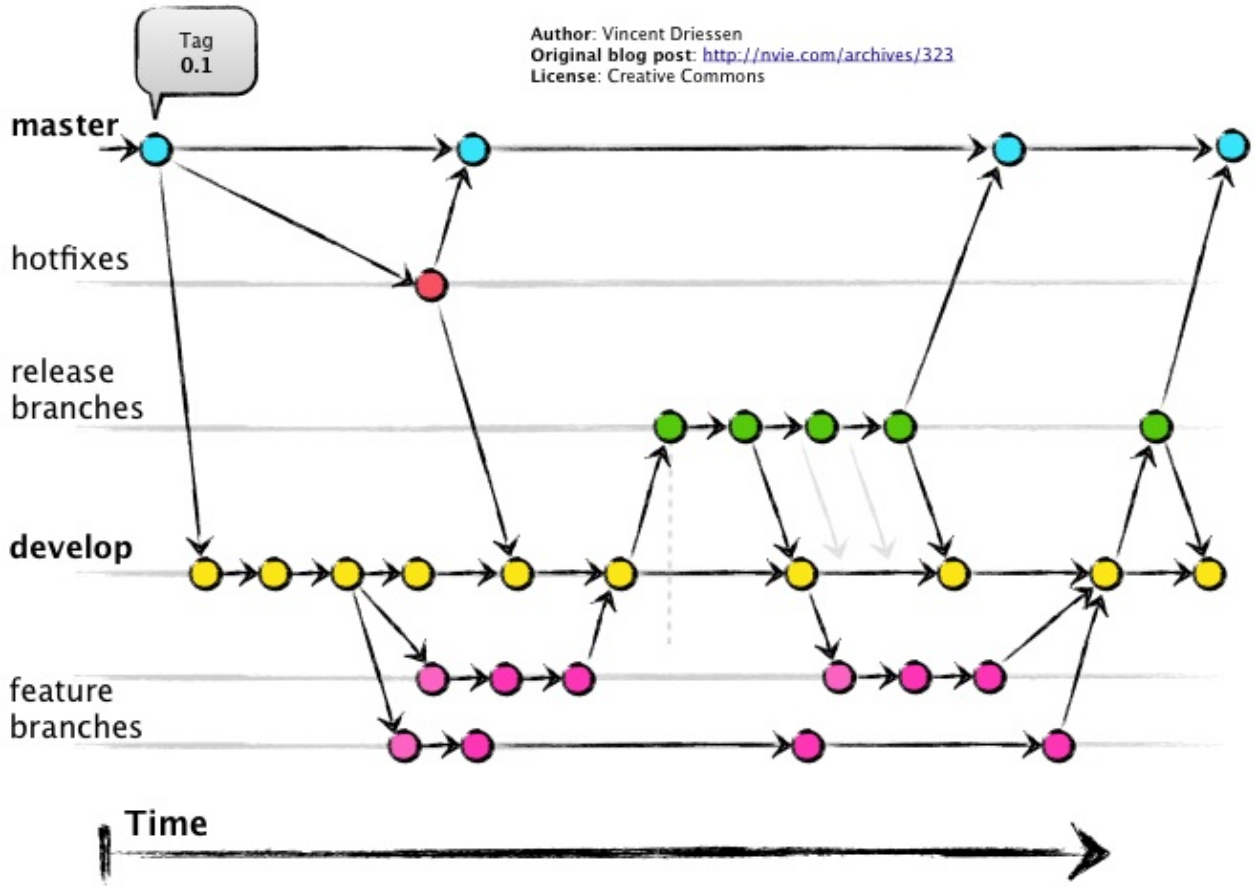
Git Flow 整體概念

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 repo 但不同分支，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。



建議分支命名

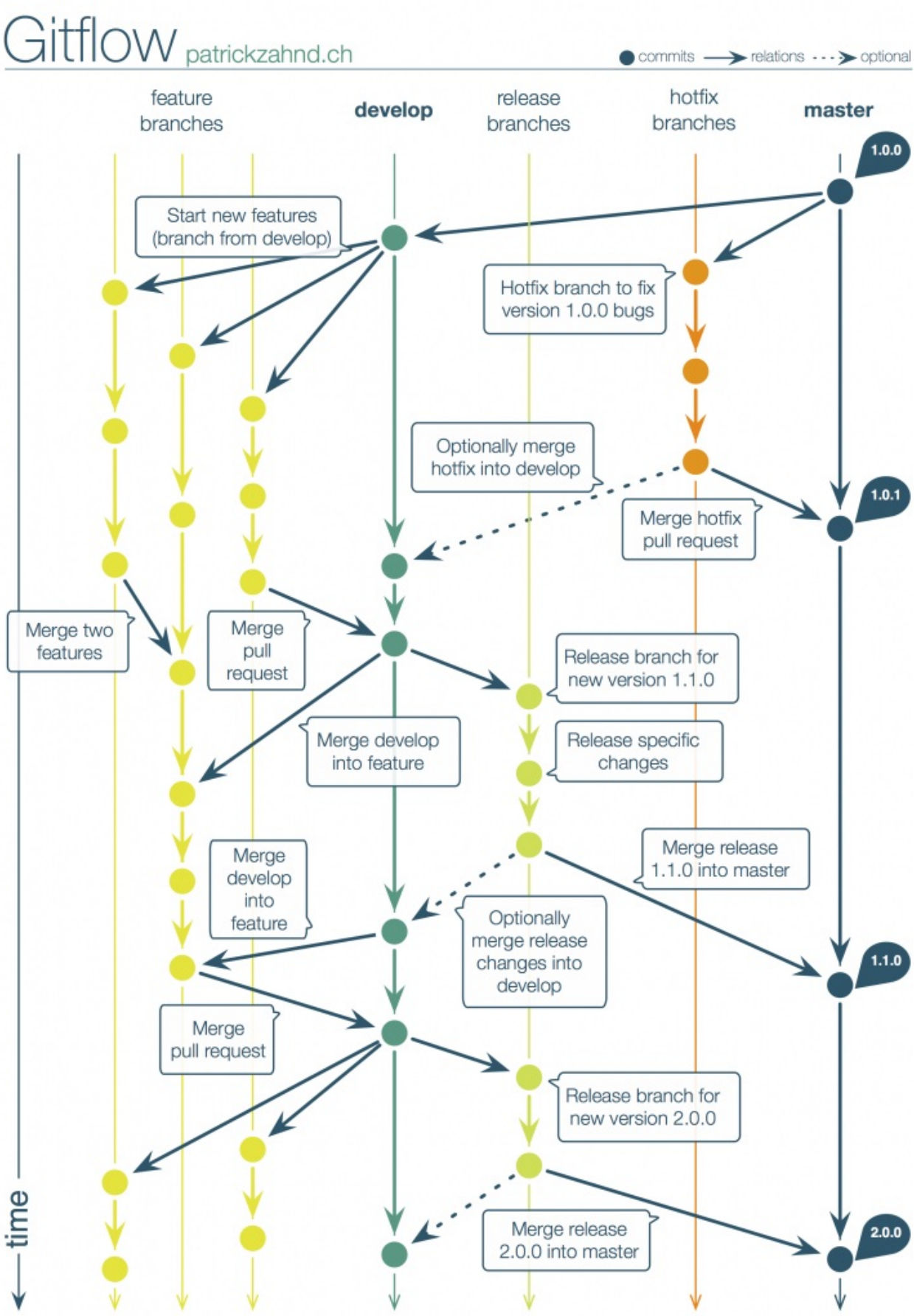
```
hotfix/issue-1
feature/issue-1
```

主要 branch

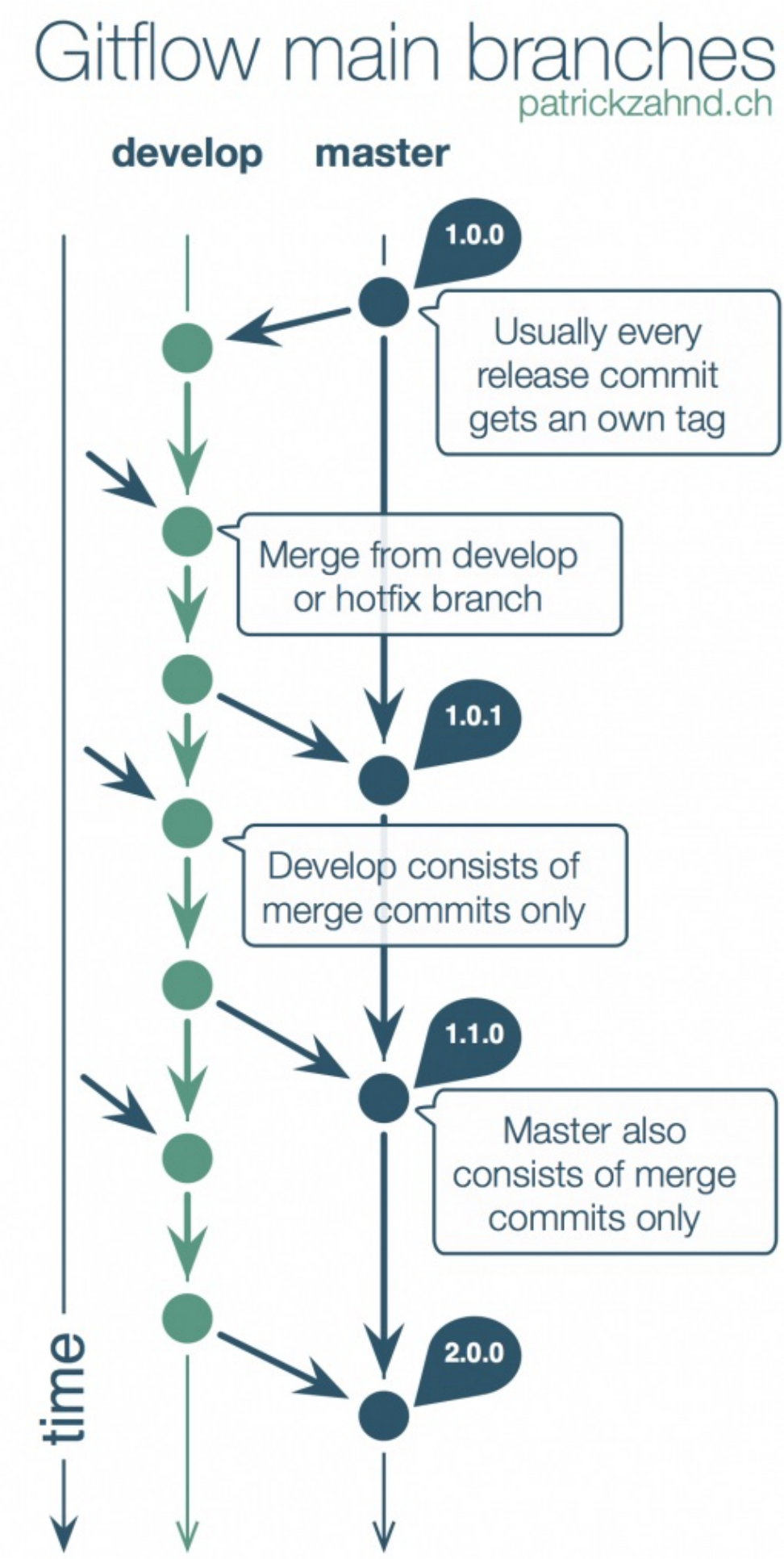
- master: 主要版本，只接受 hotfix 和 release 的 merge
- develop: 所有 feature 開發都從這分支出去，完成後 merge 回來

支援 branch

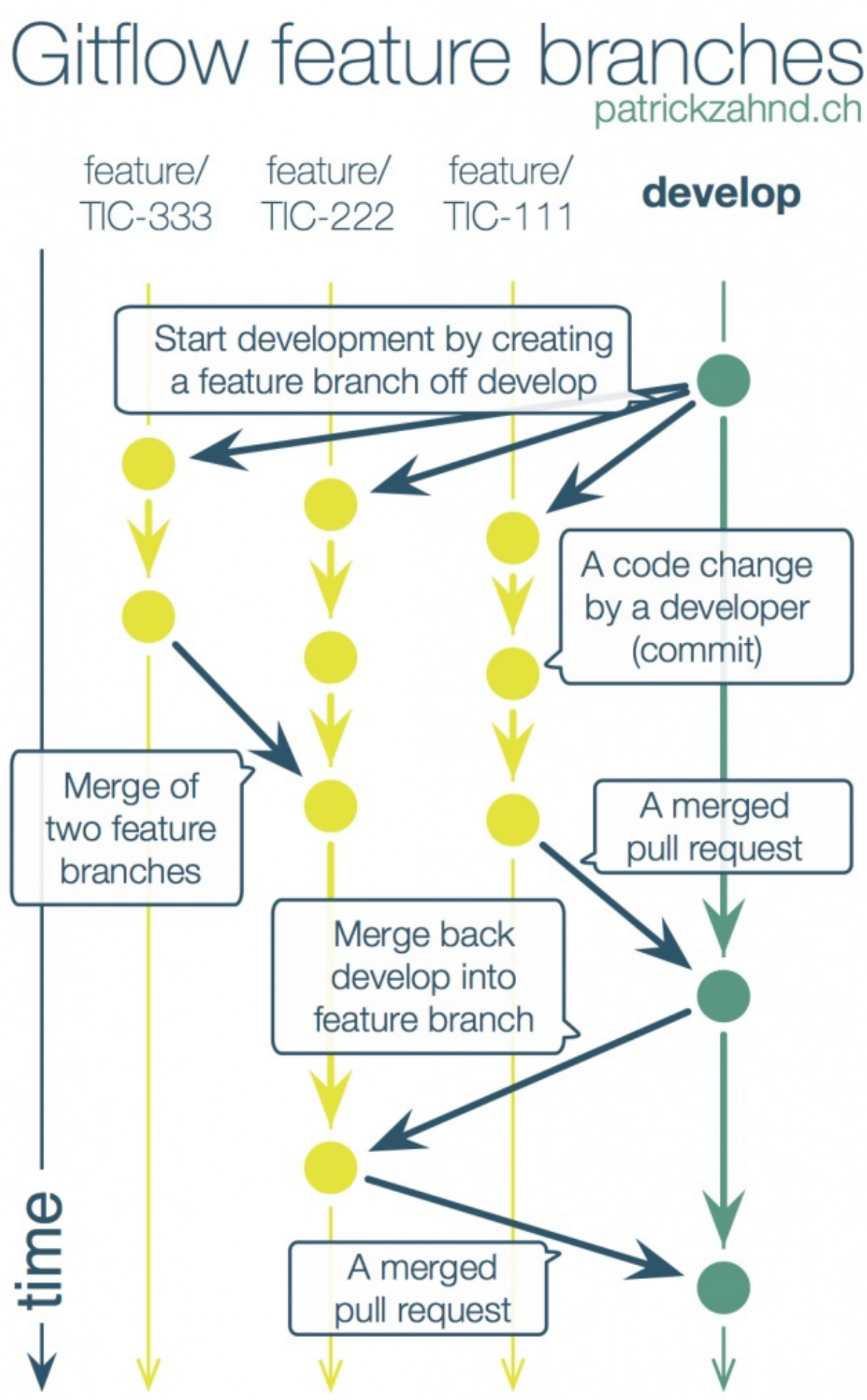
- feature branches：從 develop 分支出來，當功能開發修改完成後 merge 回 develop
- release branches：從 develop 分支出來，是準備釋出的版本，只修改版本號與 bug，完成後 merge 回 develop 與 master，並在 master 標上主版本號或次版本號的 tag (major 版號)
- hotfix branches：從 master 分支出來，主要是處理已釋出版本需要立即修改的錯誤，完成後 merge 回 develop 與 master，並在 master 標上修訂版本號的 tag。


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
1.0.1
Merge hotfix pull request
Merge two features
Merge pull request
Release branch for new version 1.1.0
Merge develop into feature
Release specific changes
Merge develop into feature
Merge release 1.1.0 into master
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

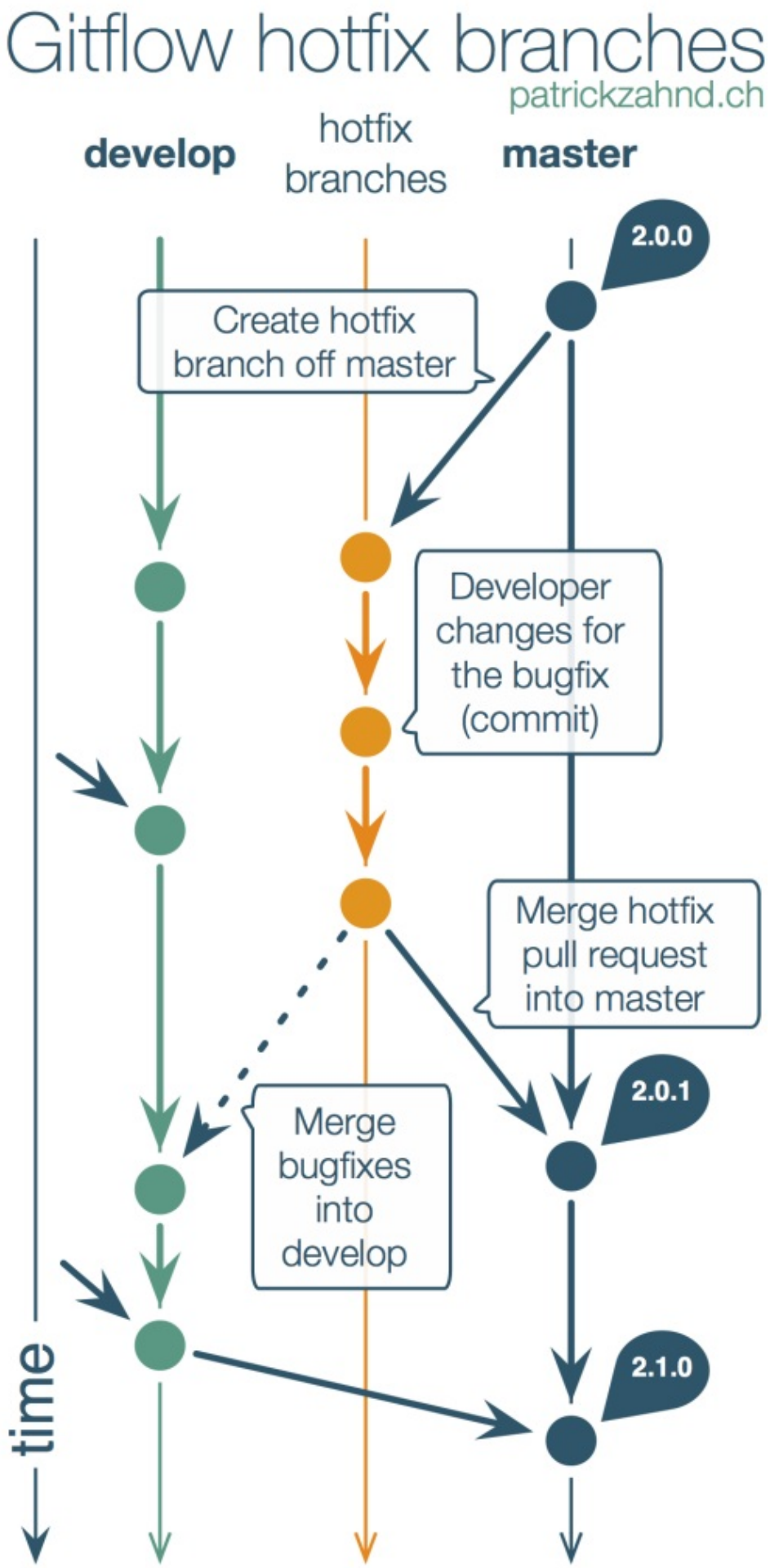
Master 分支


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

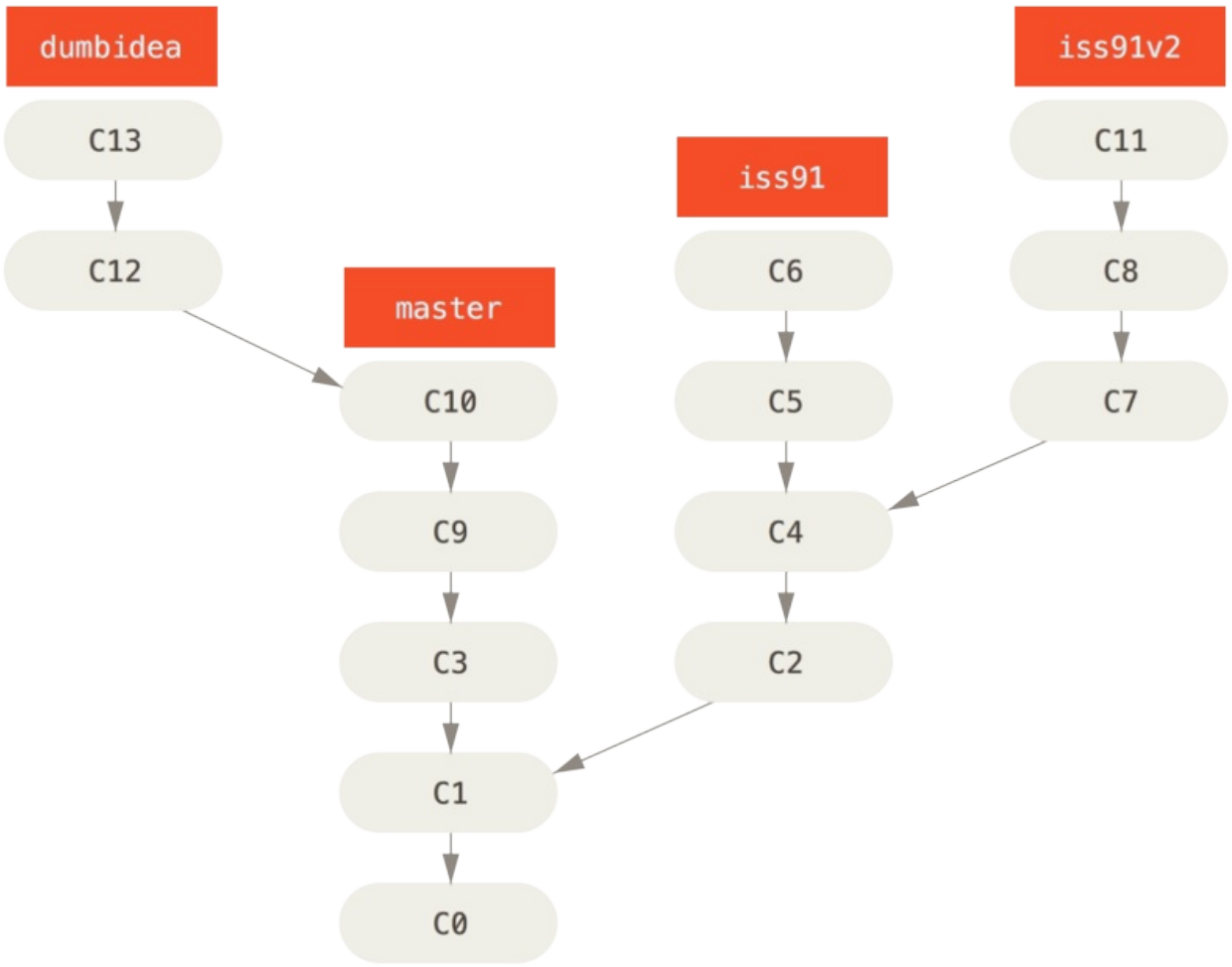
Feature 分支


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



圖片來源 - git-scm.com

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
- 透過 `git checkout -- README.md` ，還原尚未 commit 的最新的變動

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：commit

情境：還原尚未 commit 的最新的變動

1. 編輯 README.md
2. 透過 `git checkout -- README.md` 指令，還原修改的內容

練習題：branch

情境：新增一個分支

Step1

```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'

sh start.sh
```

Step2

1. 使用 `git branch hotfix/issue-1` 指令來建立 `hotfix/issue-1` 分支
2. 使用 `git checkout hotfix/issue-1` 指令來切換至 `hotfix/issue-1` 分支
3. 新增一個叫 `hotfix1.md` 檔案 `git add hotfix1.md`
4. 透過 `git commit -m 'h1'` 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

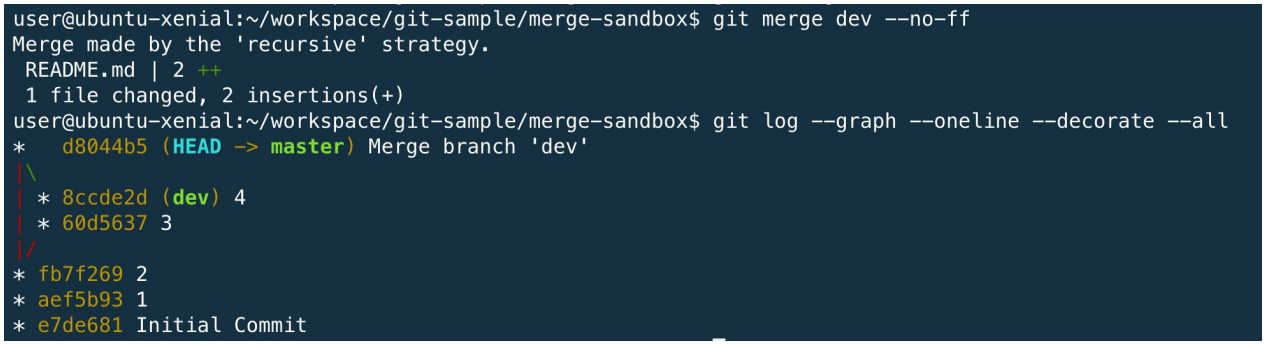
```
* 1ea846c (HEAD -> dev) 4
* 096a417 3
* 3c20c25 (master) 2
* 72d8959 1
* 337f2b6 Initial Commit
```

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
```



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

注意事項

- 只能用在未發佈的送交紀錄

使用情境

- 將新增並已加入索引的檔案，還原到沒加入索引之前。
- 還原到過去某一個狀態，重新 commit。

常用範例

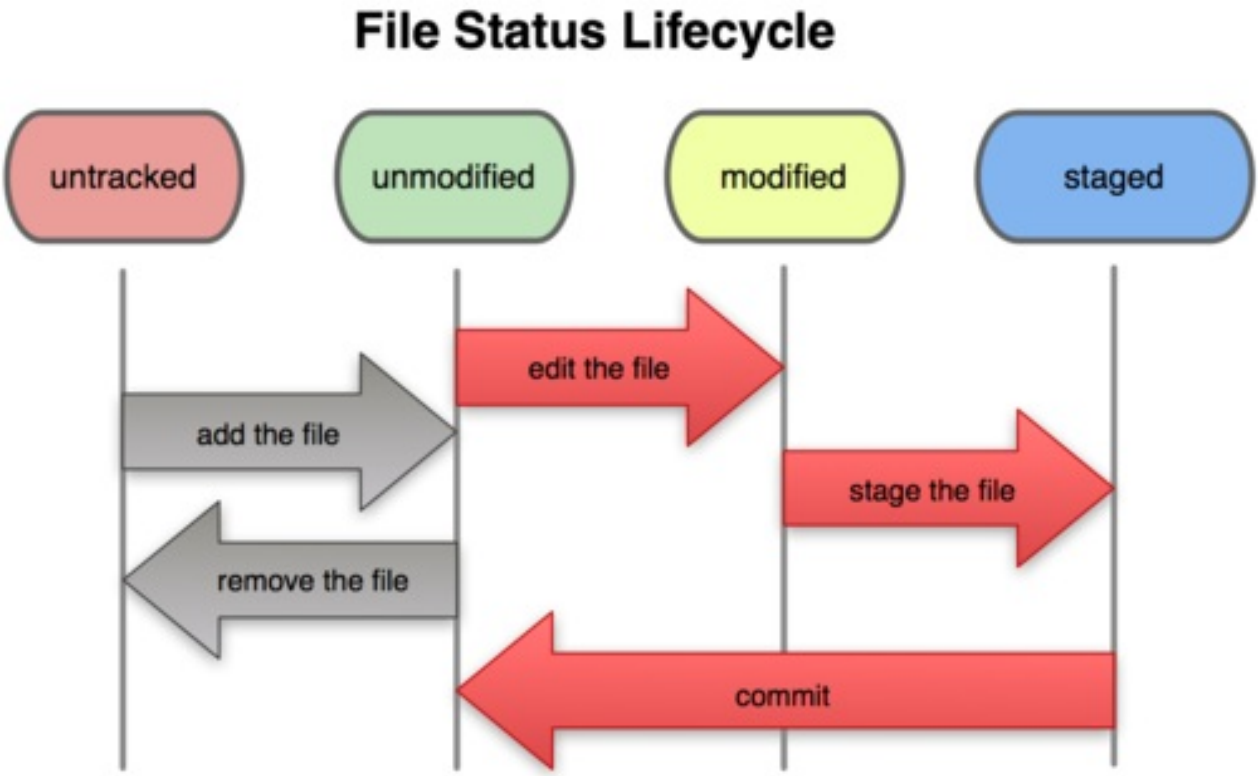
範例	說明
git reset --sort HEAD	
git reset —-hard	回覆到沒修改的狀態

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
```

練習題：reset

情境：將新增並已加入索引的檔案，還原到沒加入索引之前



```
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git status
On branch master
nothing to commit, working directory clean
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ echo "readme" >> README.md
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git add .
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README.md

user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git reset HEAD README.md
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$
```

1. 新增一個 README2.md
2. 透過 `git add README2.md` 指令，將修改的內容加入索引。
3. 透過 `git status` 指令，查看目前檔案狀態。
4. 透過 `git reset HEAD README2.md` 指令，將檔案變成 Unstaged (移除索引)。

情境：重新 commit，並讓不該存在的檔案消失

```
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git log --oneline
babb10e c5
2c37f34 c4
3464ddf c3
f9d53a8 c2
1050142 c1
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git reset 3464ddf
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git log --oneline
3464ddf c3
f9d53a8 c2
1050142 c1
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        4.txt
        5.txt

nothing added to commit but untracked files present (use "git add" to track)
user@ubuntu-xenial:~/workspace/git-sample/reset-sandbox$
```

1. 新增 1 個 commit 紀錄
2. 編輯 m1.md
3. 加入 hello 文字
4. 並 commit 進去
5. 透過 `reset --hard HEAD^` 指令，還原到前一個 commit 狀態，並把上一次的變更清除。

git revert 指令

新增還原的送交紀錄

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
```

練習題

情境：還原某一個已經發佈的 commit

```
echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> 1.md && git add . && git commit -m 'c1'
echo "2" >> 2.md && git add . && git commit -m 'c2'
echo "3" >> 3.md && git add . && git commit -m 'c3'
git reflog
```

output:

```
3f1e27b HEAD@{0}: commit: c3
e15d462 HEAD@{1}: commit: c2
d9968dd HEAD@{2}: commit: c1
106cbc4 HEAD@{3}: commit (initial): init
```

```
git revert HEAD@{1}
git log --oneline
```

git rebase 指令

- 重新指定位置 (整形)
- 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。

注意事項

- 僅適用於還沒公開發佈的送交紀錄 (須謹慎使用)

使用情境

- 重新指定送交的基礎位置
- 將歷史紀錄進行重新排序、編輯、移除、壓縮、拆散送交

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。

選擇用 merge 還是 rebase?

需要保留樹狀記錄就用 merge，反之則用 rebase。

常用範例

範例	說明
git rebase --onto	指定要從哪裡開始
git rebase master	
git rebase –abort	取消 rebase
git rebase –continue	解決衝突後，繼續rebase
git rebase -skip	忽略 rebase 的 commit
git rebase -i HEAD^^	互動模式 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。
- –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。
- -skip：忽略一個原本要rebase的commit

互動模式

```
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log mes
sage
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
```

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
```

rebase 練習題

1. 情境：重新指定送交的基礎位置

Step1

```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'

git checkout -b develop
echo "1" >> d1.md && git add . && git commit -m 'd1'
echo "2" >> d2.md && git add . && git commit -m 'd2'

git checkout master
echo "3" >> m3.md && git add . && git commit -m 'm3'

sh start.sh
```

Step2

```
git checkout develop
git rebase master
```

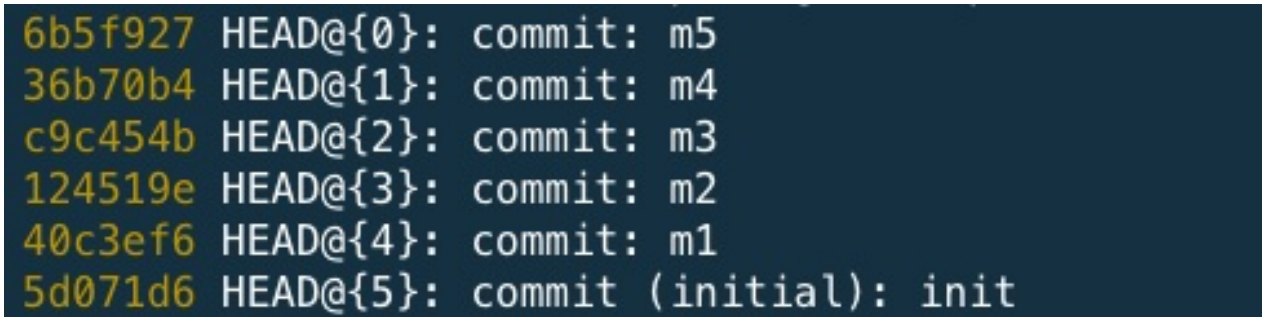
Step3

```
git checkout master
git merge develop
```

2. 情境：合併 commit

Step1

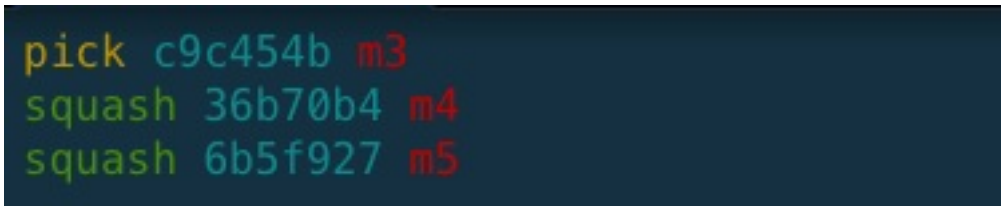
```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'
echo "3" >> m3.md && git add . && git commit -m 'm3'
echo "4" >> m4.md && git add . && git commit -m 'm4'
echo "5" >> m5.md && git add . && git commit -m 'm5'

sh start.sh
```



Step2

```
git rebase -i HEAD~3
// or
git rebase -i 124519e
```

Step3



Step4

```
# This is a combination of 3 commits.
# The first commit's message is:
m3

# This is the 2nd commit message:

m4

# This is the 3rd commit message:

m5

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed May 16 14:27:26 2018 +0000
#
# interactive rebase in progress; onto 124519e
# Last commands done (3 commands done):
#    squash 36b70b4 m4
#    squash 6b5f927 m5
# No commands remaining.
# You are currently editing a commit while rebasing branch 'master' on '124519e'.
#
# Changes to be committed:
#       new file:   m3.md
#       new file:   m4.md
#       new file:   m5.md
```

Step5

```
# This is a combination of 3 commits.
# The first commit's message is:
m3 + m4 + m5

# This is the 2nd commit message:


# This is the 3rd commit message:


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed May 16 14:36:25 2018 +0000
#
# interactive rebase in progress; onto ff48ff6
# Last commands done (3 commands done):
#    squash d34c16e m4
#    squash 36bce0a m5
# No commands remaining.
# You are currently editing a commit while rebasing branch 'master' on 'ff48ff6'.
#
# Changes to be committed:
#       new file:   m3.md
#       new file:   m4.md
#       new file:   m5.md
```

3. 情境：指定要從哪裡開始

Step1

```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'
echo "3" >> m3.md && git add . && git commit -m 'm3'
git checkout -b develop
echo "1" >> d1.md && git add . && git commit -m 'd1'
git checkout -b feature/issue-1
echo "1" >> f1.md && git add . && git commit -m 'f1'
echo "2" >> f2.md && git add . && git commit -m 'f2'
git checkout develop
echo "2" >> d2.md && git add . && git commit -m 'd2'
echo "3" >> d3.md && git add . && git commit -m 'd3'
echo "4" >> d4.md && git add . && git commit -m 'd4'

sh start.sh
```

Step2

```
git checkout feature/issue-1
git rebase --onto b93be68 8a42167
```

Step3

```
git rebase develop
```

4. 情境：兩個分支交疊在一起

- feature/issue-1 是從 develop 開出的分支
- develop 是從 master 開出的分支

Step1

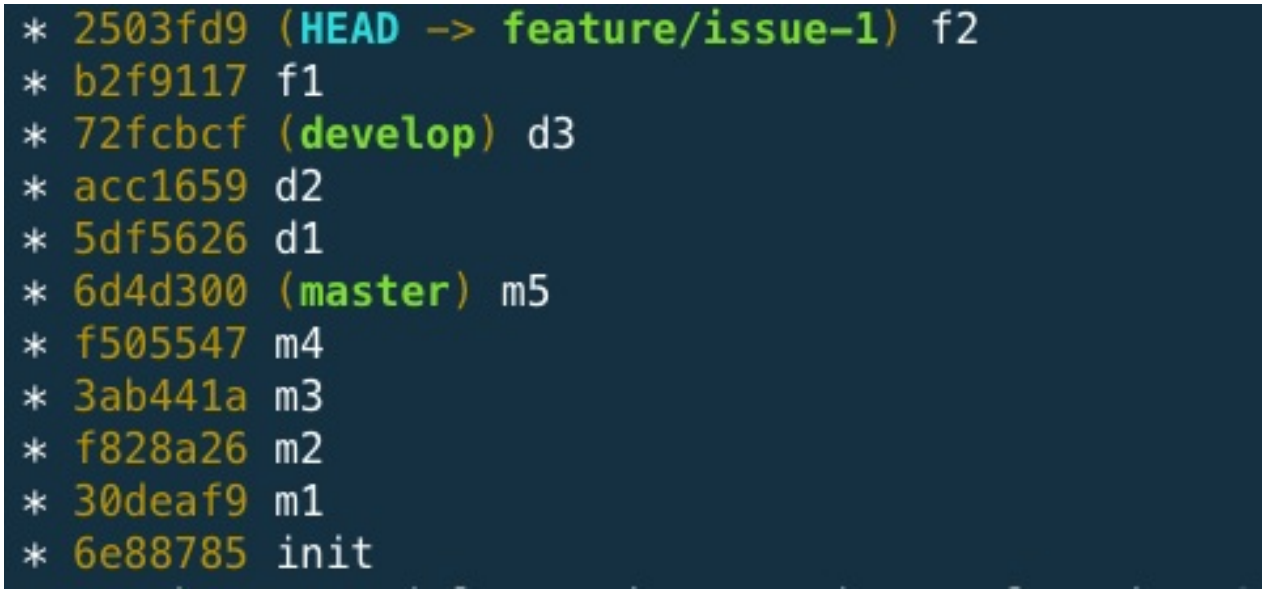
```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'
echo "3" >> m3.md && git add . && git commit -m 'm3'
git checkout -b develop
echo "1" >> d1.md && git add . && git commit -m 'd1'
echo "2" >> d2.md && git add . && git commit -m 'd2'
git checkout -b feature/issue-1
echo "1" >> f1.md && git add . && git commit -m 'f1'
echo "2" >> f2.md && git add . && git commit -m 'f2'
git checkout develop
echo "3" >> d3.md && git add . && git commit -m 'd3'
git checkout master
echo "4" >> m4.md && git add . && git commit -m 'm4'
echo "5" >> m5.md && git add . && git commit -m 'm5'

sh start.sh
```

Step2

```
git checkout develop
git rebase master
```

Step3

```
git checkout feature/issue-1
git rebase develop
```



5. 情境：悲劇現場

Step1

- init 有 1 個
- m 有 3 個
- d 有 4 個
- f 有 2 個

```
vi start.sh

echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
echo "2" >> m2.md && git add . && git commit -m 'm2'
echo "3" >> m3.md && git add . && git commit -m 'm3'
git checkout -b develop
echo "1" >> d1.md && git add . && git commit -m 'd1'
git checkout -b feature/issue-1
echo "1" >> f1.md && git add . && git commit -m 'f1'
echo "2" >> f2.md && git add . && git commit -m 'f2'
git checkout develop
echo "2" >> d2.md && git add . && git commit -m 'd2'
echo "3" >> d3.md && git add . && git commit -m 'd3'
echo "4" >> d4.md && git add . && git commit -m 'd4'

sh start.sh
```

```
* ae1e023 (HEAD -> develop) d4
* bac0cfe d3
* 2c49b7f d2
| * 8c193eb (feature/issue-1) f2
| * 237bbb4 f1
|/
* 6b193ca d1
* 13e64db (master) m3
* 9074c5f m2
* fd0d378 m1
* 27f06bc init
```

Step2：參數位置弄顛倒了

```
git rebase --onto 6b193ca bac0cfe
```

```
* c759b18 (HEAD -> develop) d4
| * 8c193eb (feature/issue-1) f2
| * 237bbb4 f1
|/
* 6b193ca d1
* 13e64db (master) m3
* 9074c5f m2
* fd0d378 m1
* 27f06bc init
```

6. rebase 衝突

```
echo "Hello World" >> README.md && git add . && git commit -m
 'init'
echo "1" >> m1.md && git add . && git commit -m 'm1'
git branch hotfix/issue-1
vi m1.md

git add . && git commit -m 'hotfix update'
git checkout master

vi m1.md
git add . && git commit -m 'master update'
git checkout hotfix/issue-1
git rebase master
git diff
vi m1.md
git diff
git add .
git commit
git rebase --continue
```

Reset vs Revert

Reset

- 用在還未發佈的 commit
- 直接修改舊的送交歷史紀錄，不會多增加一個新的節點。

Revert

- 用在還未發佈的 commit
- 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。

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**到另一個容器的分支。
- 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

使用情境

- 指定遠端 repo 位置

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

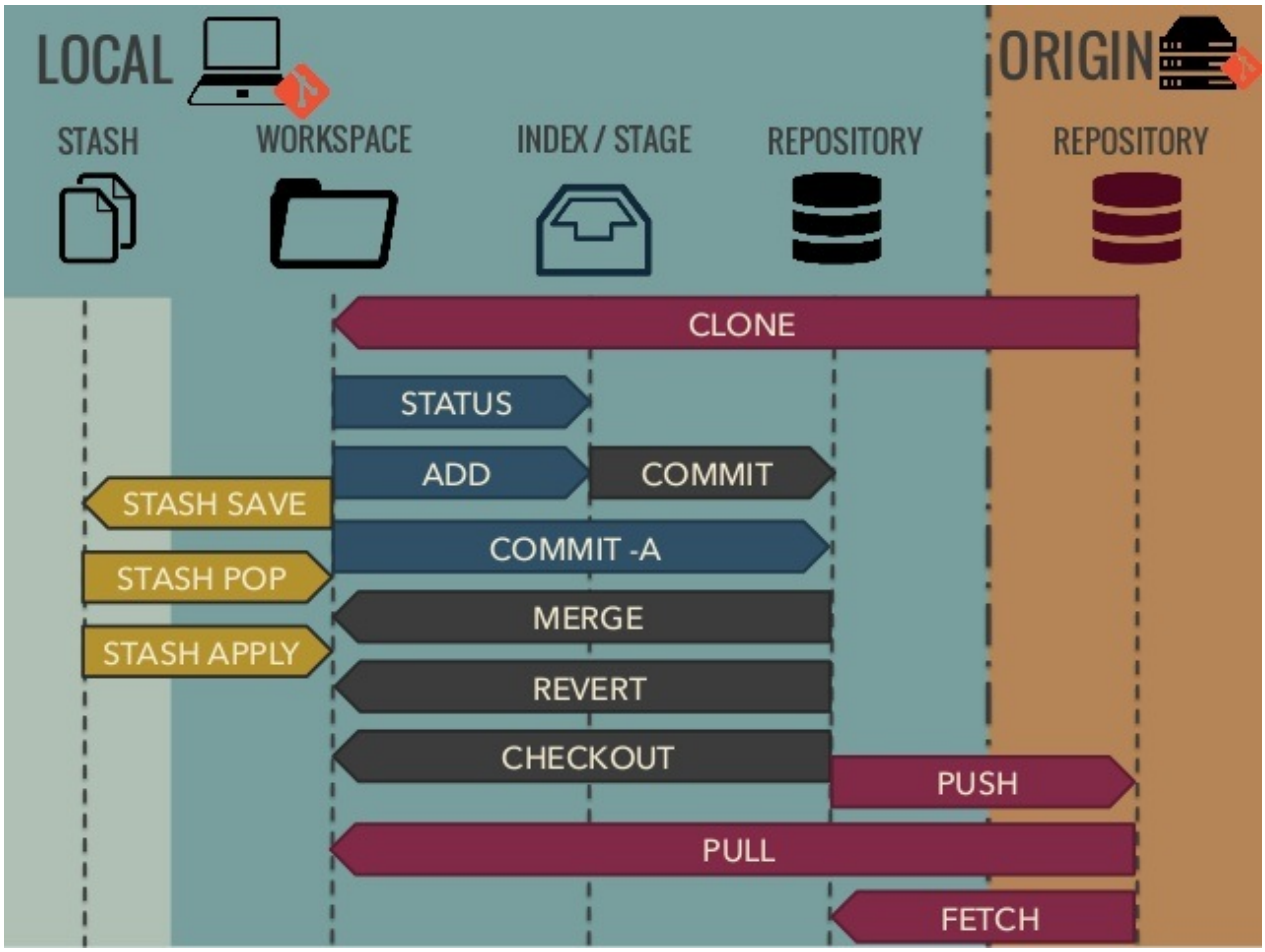
語法結構

```
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 repo 的最新歷史記錄。取得的送交紀錄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 repo 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：push

情境：發佈送交

1. `git pull`
2. `git push`

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

命名

```
[主版本號].[次版本號].[修訂版本號]
```

```
v0.0.1
// or
v0.0.1
```

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

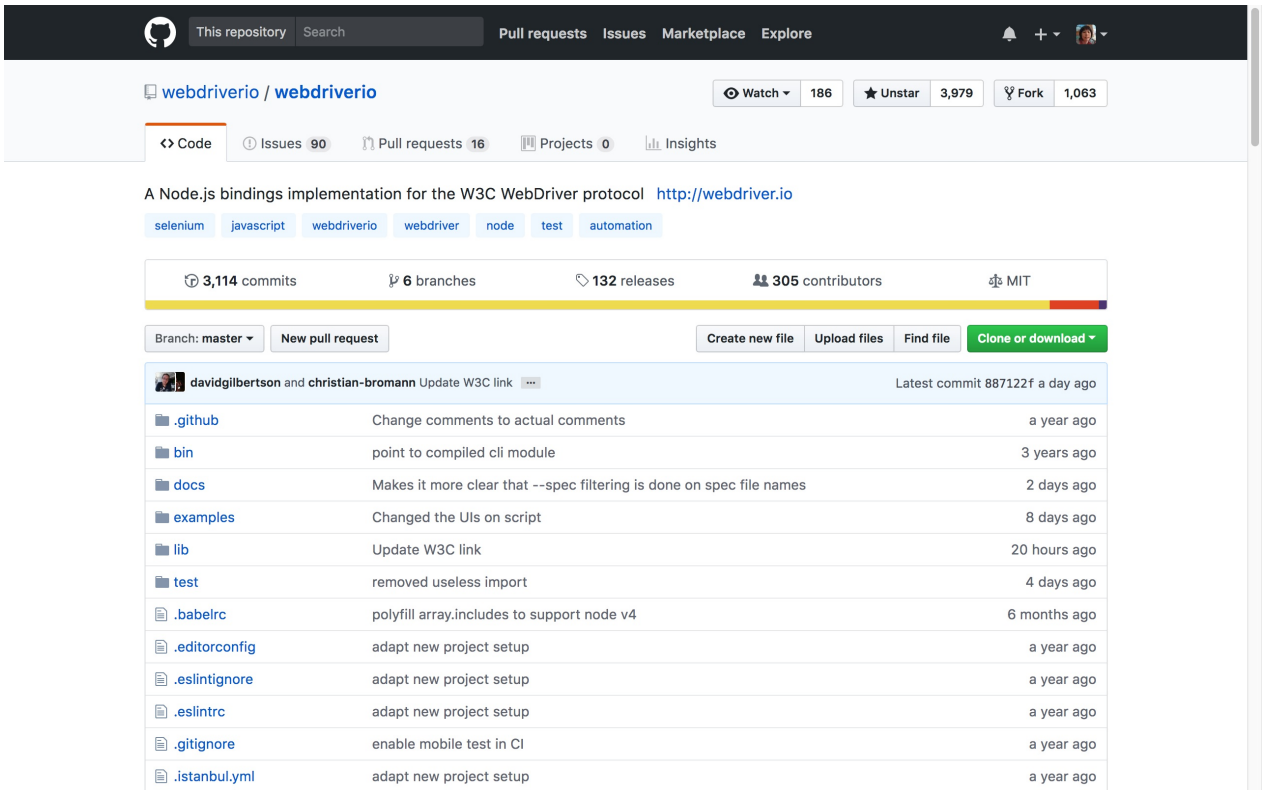
```
usage: git tag [-a | -s | -u <key-id>] [-f] [-m <msg> | -F <f
ile>] <tagname> [<head>]
   or: git tag -d <tagname>...
   or: git tag -l [-n[<num>]] [--contains <commit>] [--points
-at <object>]
                [--format=<format>] [--[no-]merged [<commit>]
] [<pattern>...]
   or: git tag -v <tagname>...

    -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

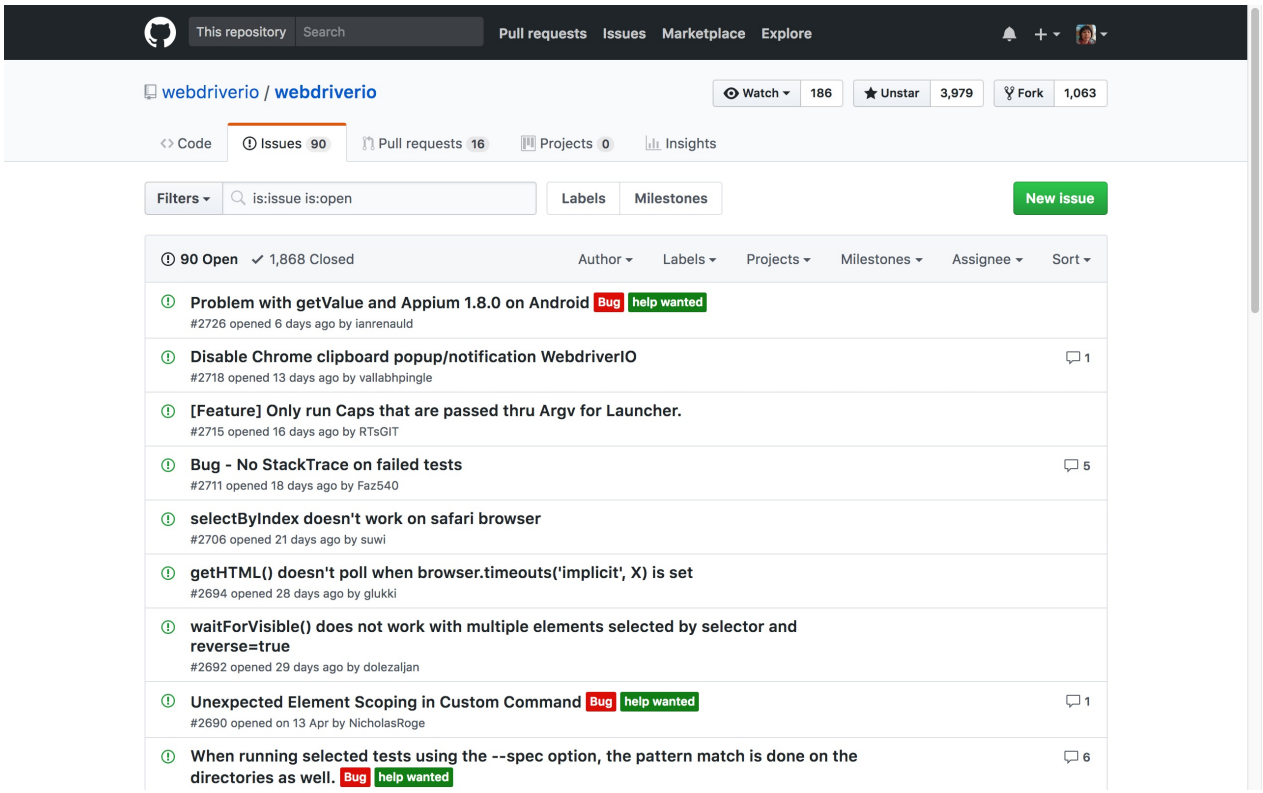
例如像 facebook 按「讚」

Fork

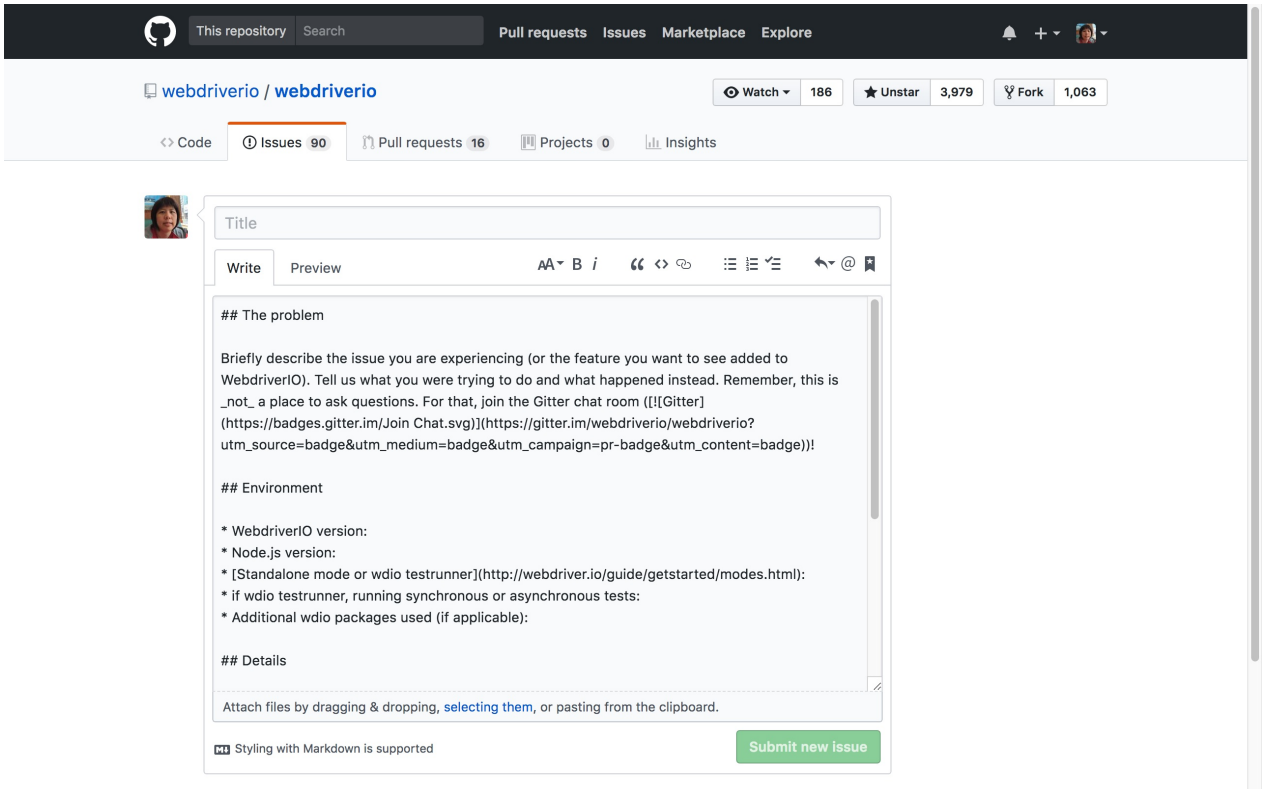
fork 是發生在「整個容器」的階層上，branch 是發生在「單一容器」中。

Github issue

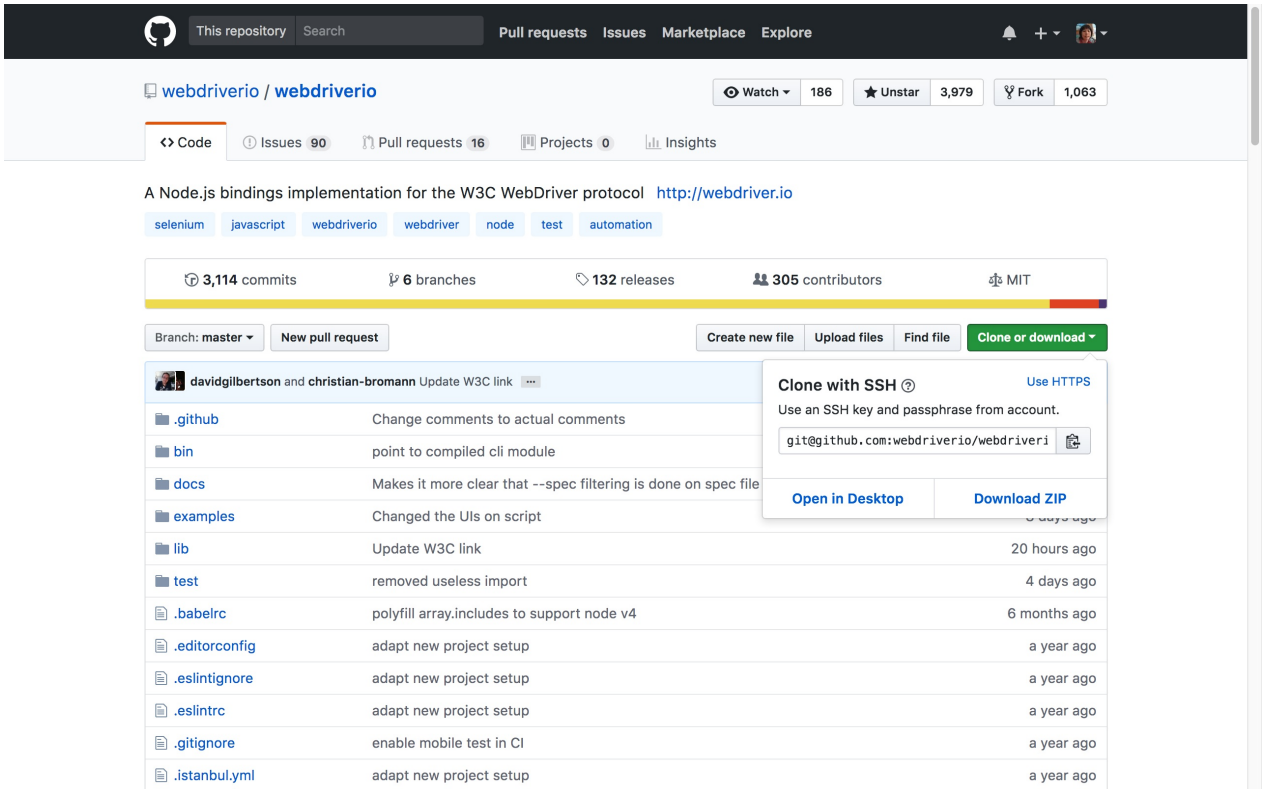
查看 issue 清單



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

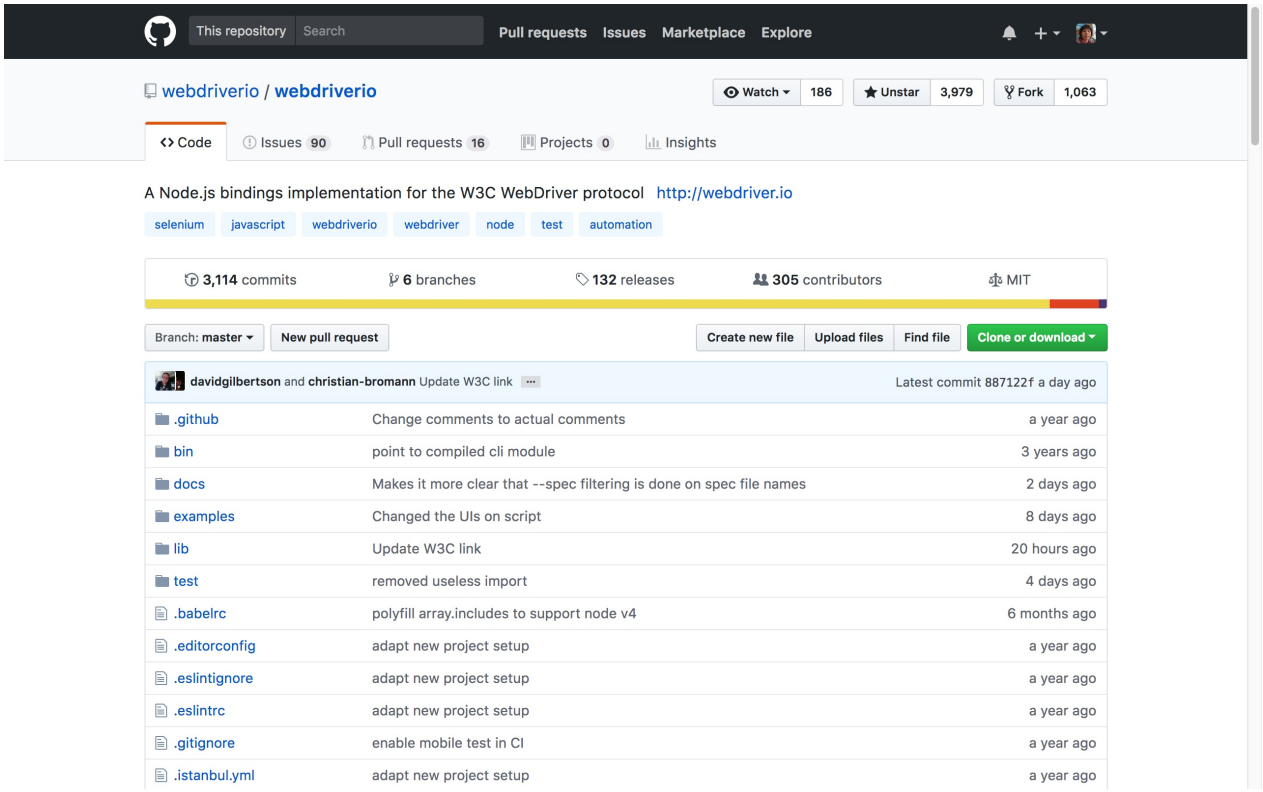
`git remote -v`

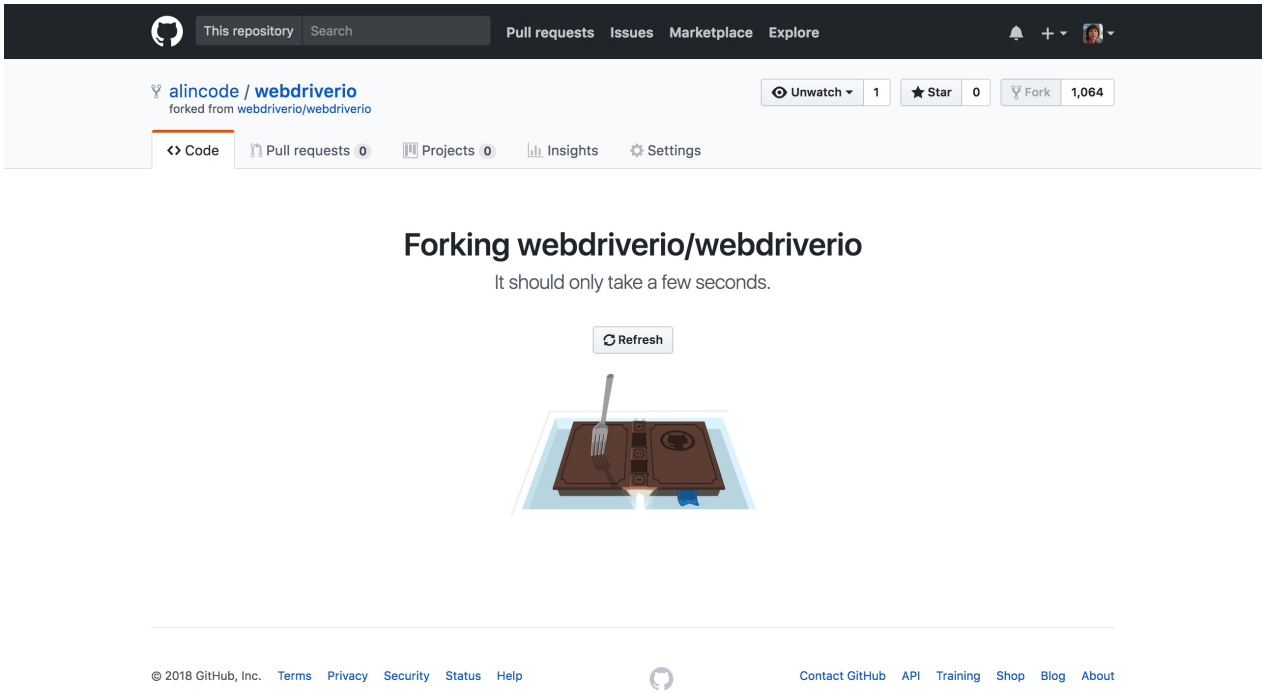
```
origin  https://github.com/your-username/forked-repository.git (fetch)
origin  https://github.com/your-username/forked-repository.git (push)
upstream    https://github.com/original-owner-username/original-repository.git (fetch)
upstream    https://github.com/original-owner-username/original-repository.git (push)
```

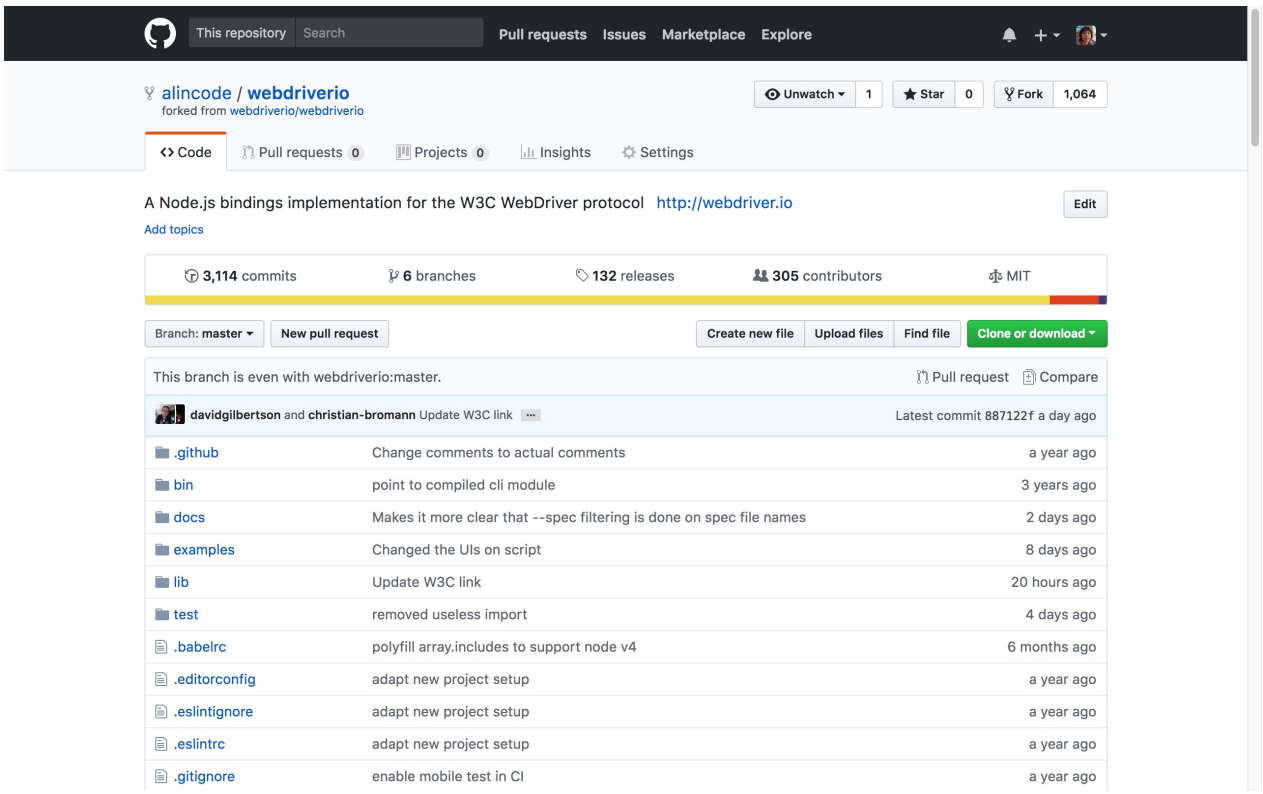
```
git fetch upstream
git checkout master
git merge upstream/master
```

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
webdriverio / webdriverio
Watch 186
Unstar 3,979
Fork 1,063
Code
Issues 90
Pull requests 16
Projects 0
Insights
A Node.js bindings implementation for the W3C WebDriver protocol http://webdriver.io
selenium
javascript
webdriverio
webdriver
node
test
automation
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
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
.istanbul.yml adapt new project setup a year ago


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

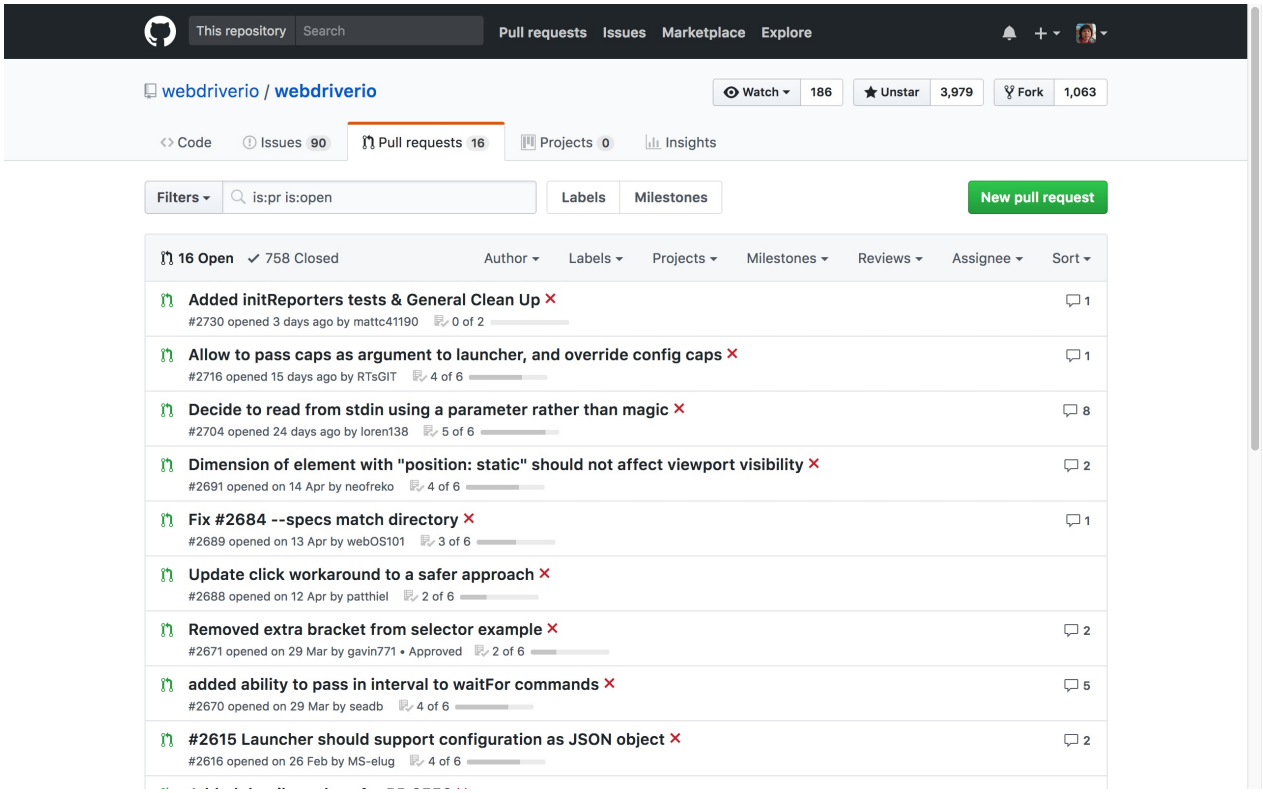

This repository
Search
Pull requests
Issues
Marketplace
Explore
alincode / webdriverio
forked from webdriverio/webdriverio
Unwatch 1
Star 0
Fork 1,064
Code
Pull requests 0
Projects 0
Insights
Settings
A Node.js bindings implementation for the W3C WebDriver protocol http://webdriver.io
Edit
Add topics
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

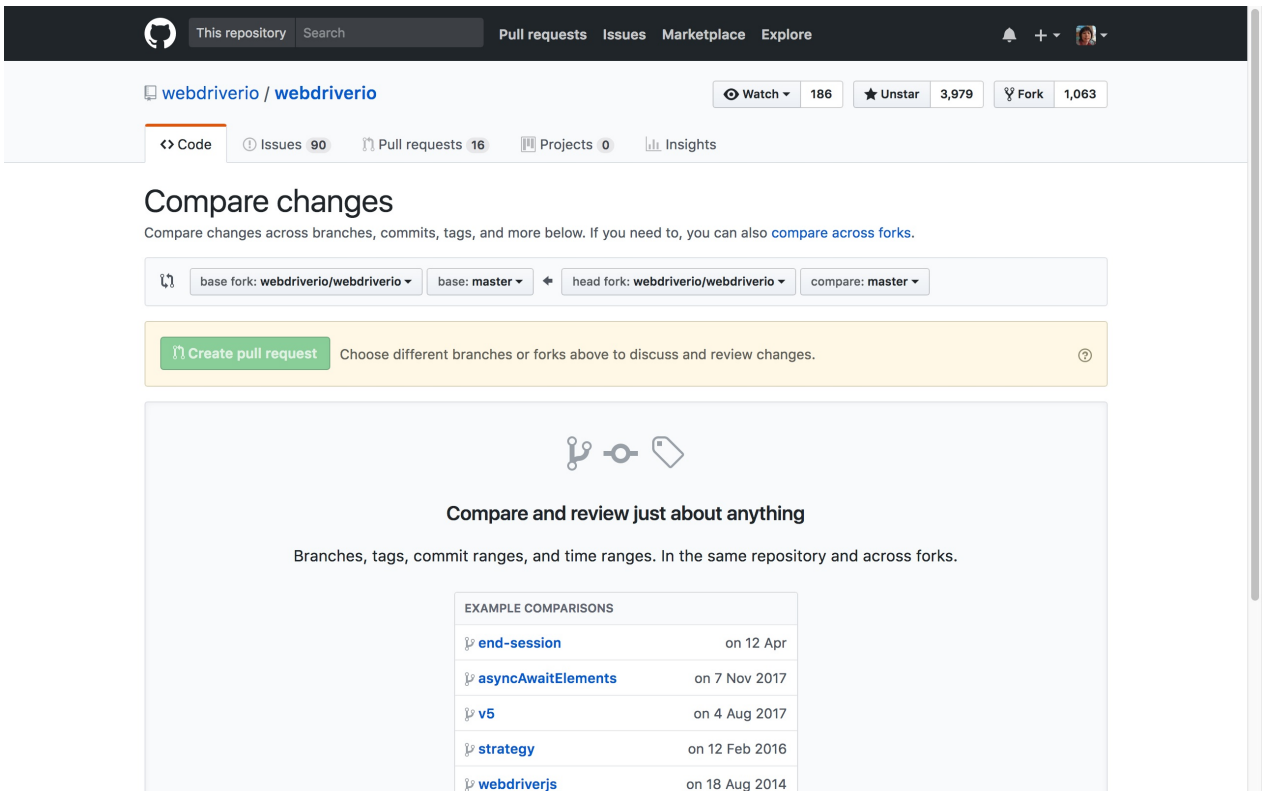
Pull Request 是一種通知機制。你修改了他人的代碼，將你的修改通知原來的作者，希望他合併你的修改，這就是 Pull Request。

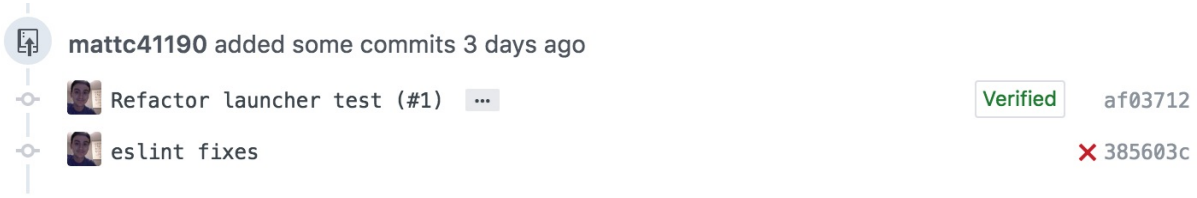
向 Github 託管的某一項目中的某一分支提出合併請求。項目管理者則可以在 github 的 web 界面上合併來自不同分支的代碼，解決合併衝突，做代碼審查或對代碼進行評論。

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 reflog 指令

追蹤變更軌跡

注意事項

這些更變軌跡紀錄並不會被同步到遠端 repo，僅限於本地端。

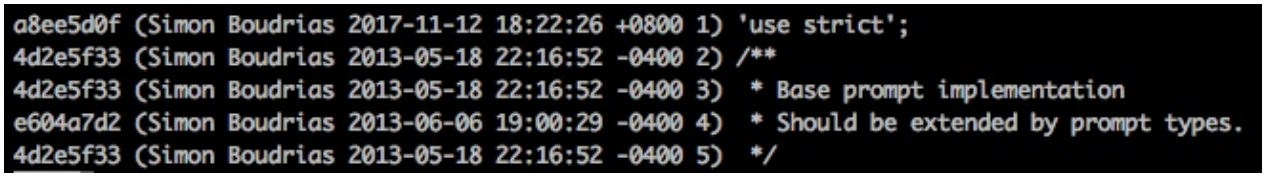
常用指令範例

範例	說明
git config --global gc.reflogExpire "never"	
git config --global gc.reflogExpireUnreachable "never"	
git config --global gc.reflogExpire "7 days"	
git config --global gc.reflogExpireUnreachable "7 days"	
git reflog delete HEAD@{2}	刪除更變軌跡
git reflog	列表更變軌跡

語法結構

```
usage: git reflog [ show | expire | delete | exists ]
```

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

- 快速內容有指定的關鍵字的檔案位置

常用範例

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

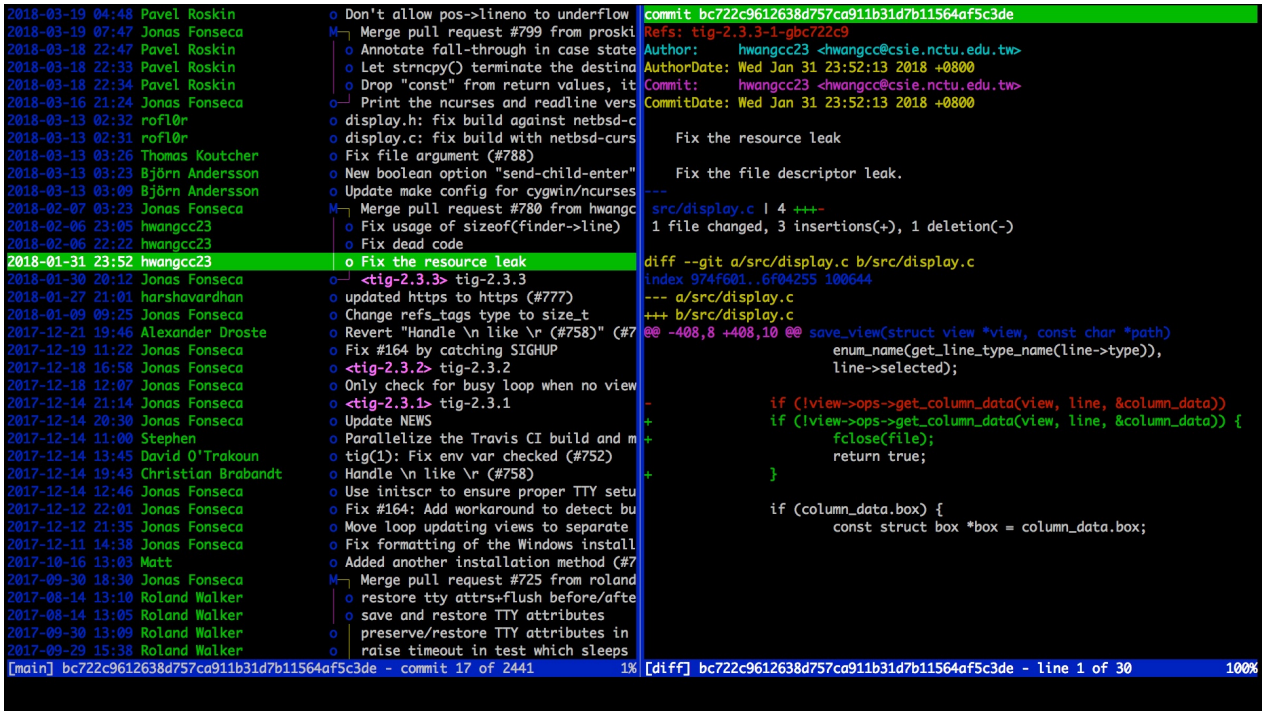
語法結構

```
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

```
# 新增一個空的資料夾
mkdir docs && cd docs
# 放一個名稱為 .gitkeeper 的檔案
touch .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