



工程師必備第一工具

Git

ALAN 蔡孟玔

2015-05-30

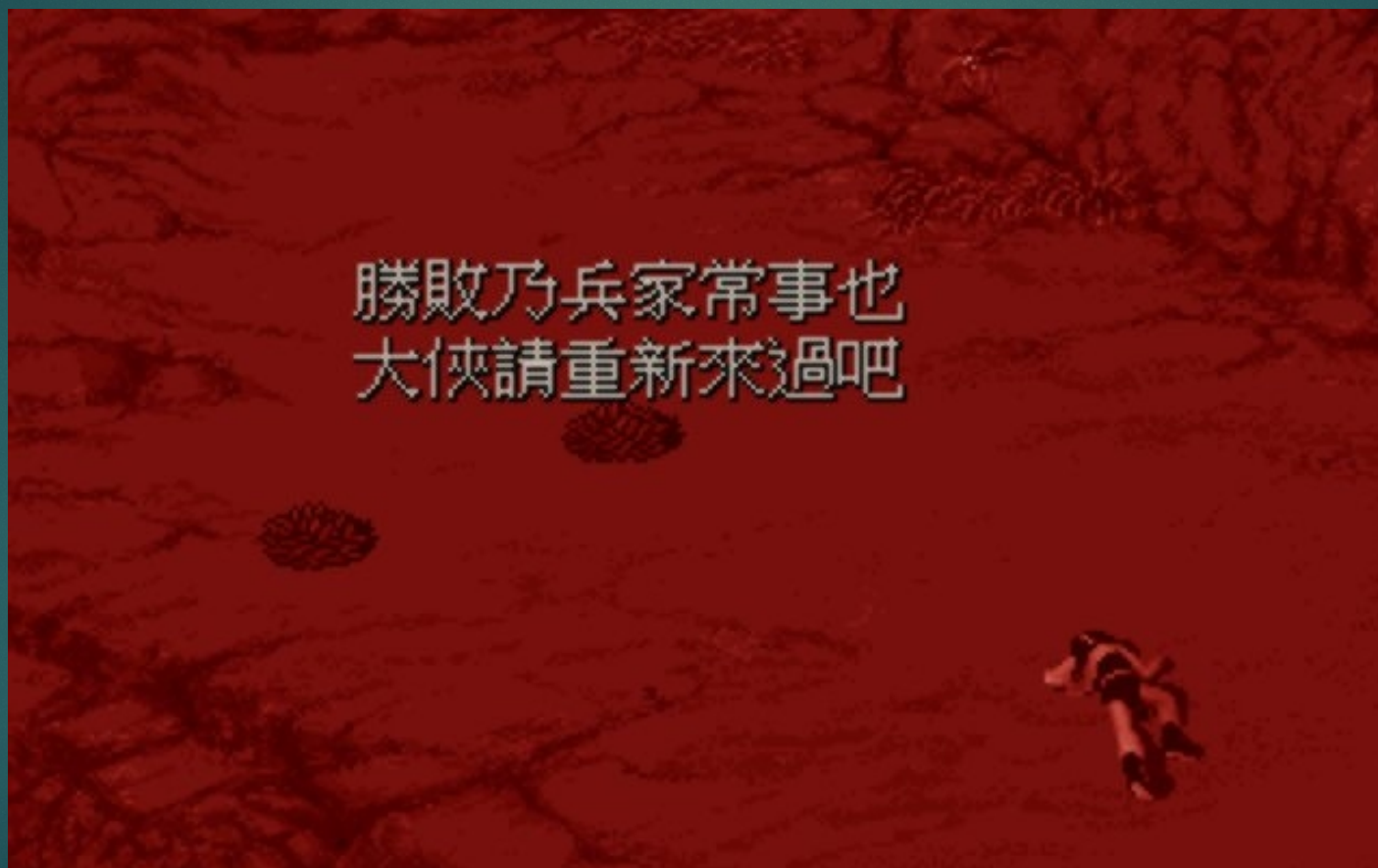
Agenda

- ▶ 什麼是 VCS (Version Control System 版控系統) ? 為什麼要學 ?
- ▶ Git 作為 VCS 的特別之處
- ▶ 安裝和設定 Git
- ▶ Git 上手介紹 – init、add、commit 等核心指令
- ▶ Git 的流程和概念
- ▶ Git 分支的做法
- ▶ Git 的 best practice

- ▶ 不會介紹
 - ▶ 遠端連線的部分

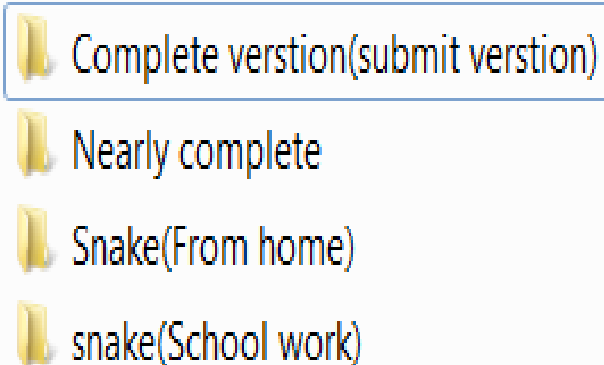
VCS 是什麼

什麼情況下才能夠重新來過？



VCS 的黑暗時代 – 沒有版控的時代

- ▶ 程式沒有那麼複雜
- ▶ 要保留版本就是：

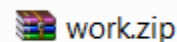


Complete verstion(submit verstion)

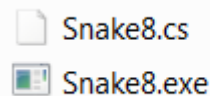
Nearly complete

Snake(From home)

snake(School work)

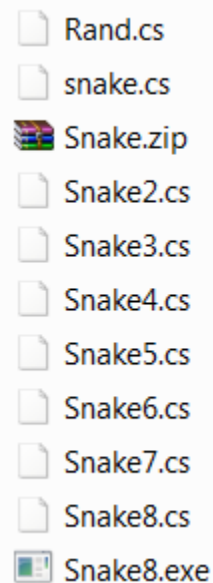


work.zip



Snake8.cs

Snake8.exe



Rand.cs

snake.cs

Snake.zip

Snake2.cs

Snake3.cs

Snake4.cs

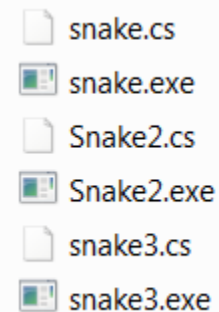
Snake5.cs

Snake6.cs

Snake7.cs

Snake8.cs

Snake8.exe



snake.cs

snake.exe

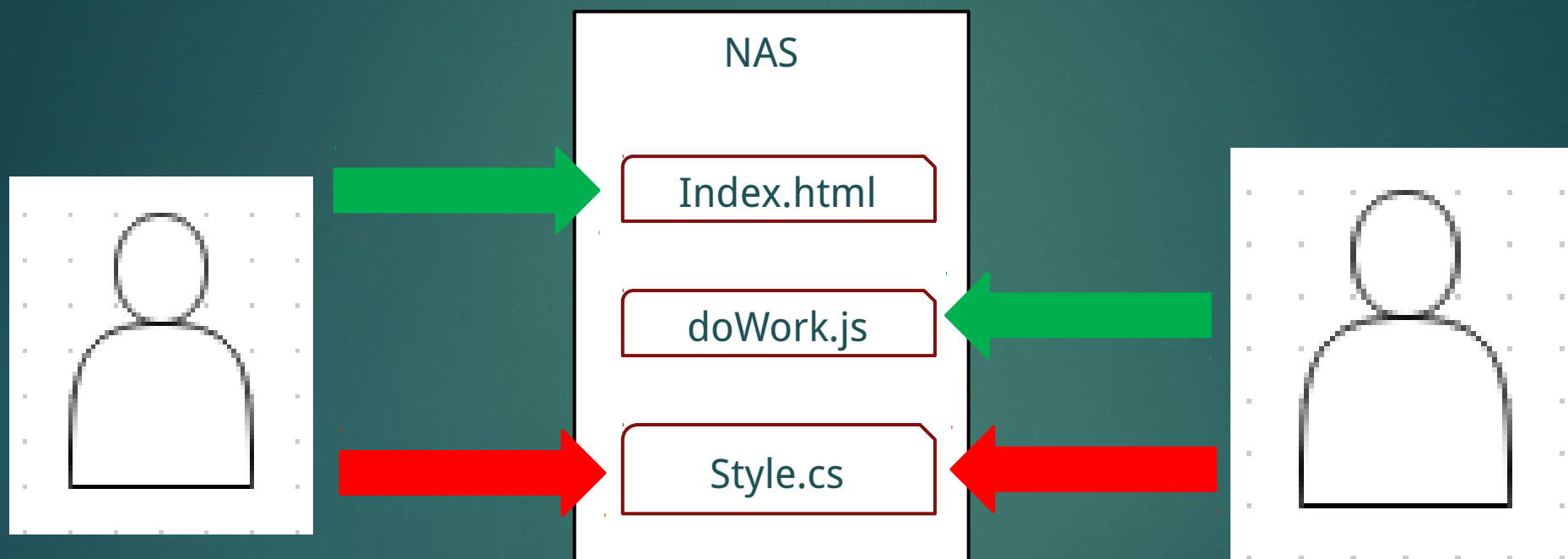
Snake2.cs

Snake2.exe

snake3.cs

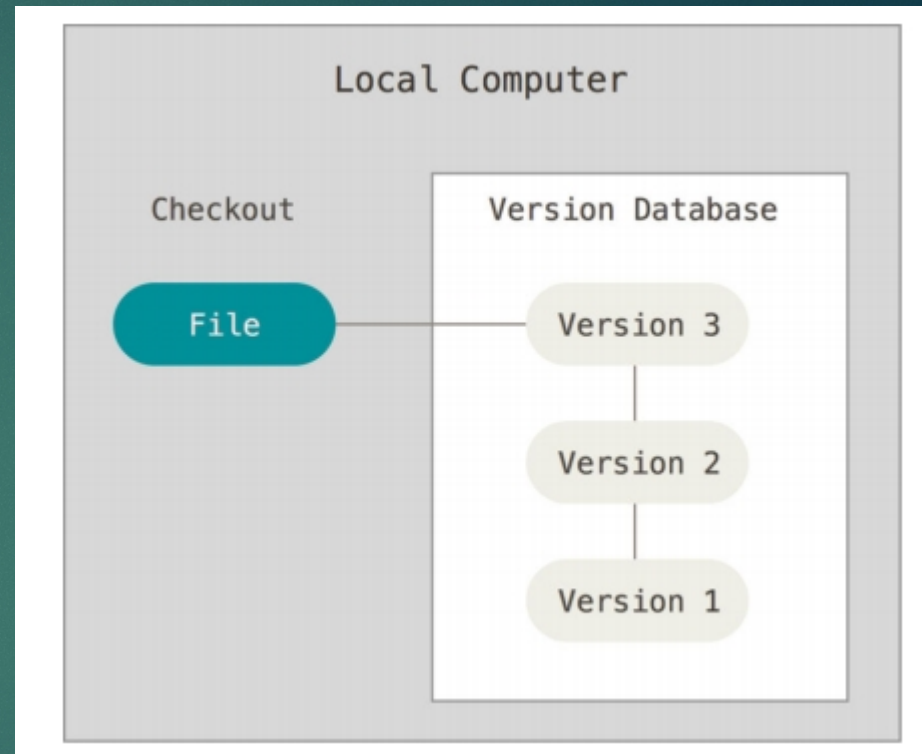
snake3.exe

沒有版控 就 沒法協同合作



第一代 – 一個人的時代

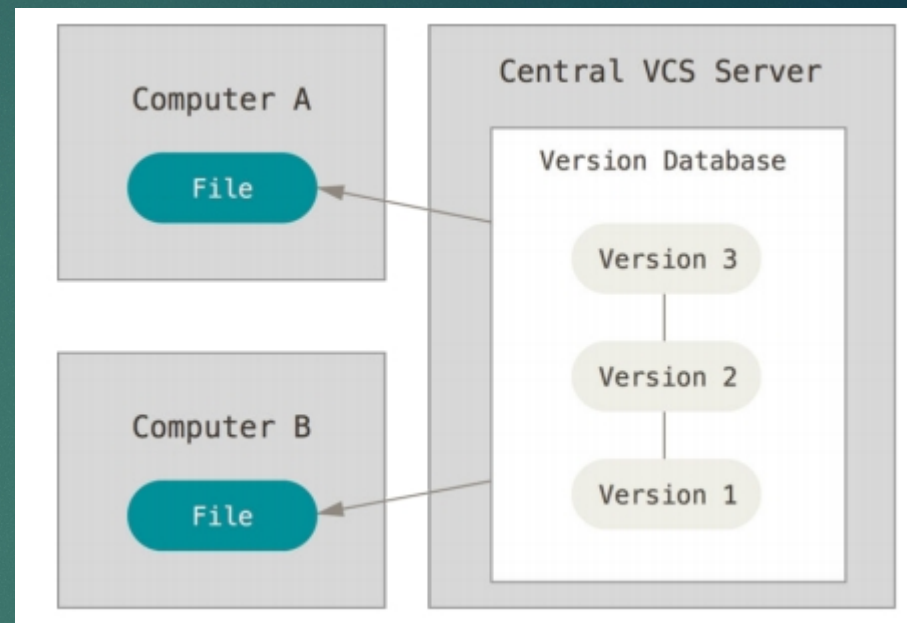
- ▶ 沒版控最大問題是，同時修改怎麼辦？
- ▶ 使用 Lock 技術
 - ▶ 同時只能一個人來
- ▶ 軟體都是由少數個人寫出來的
- ▶ 不能協同工作？
- ▶ 大型軟體怎麼辦？



來源： Pro Git P28 Figure1-1

第二代 – CVCS (Centralized) 集中式版控

- ▶ 有個中央 Server 記錄版本
 - ▶ 可以多人使用
- ▶ 第一代問題是容易雙方互等
 - ▶ 在 commit (提交) 的時候針對有重複的去做修正
- ▶ 典型程式有
 - ▶ Subversion (SVN), Team Foundation Server (TFS), CVS, SourceSafe,
- ▶ 但是
 - ▶ 沒網路要開發怎麼辦？
 - ▶ 不同團隊如何並行開發？
 - ▶ Server 掛掉怎麼辦？

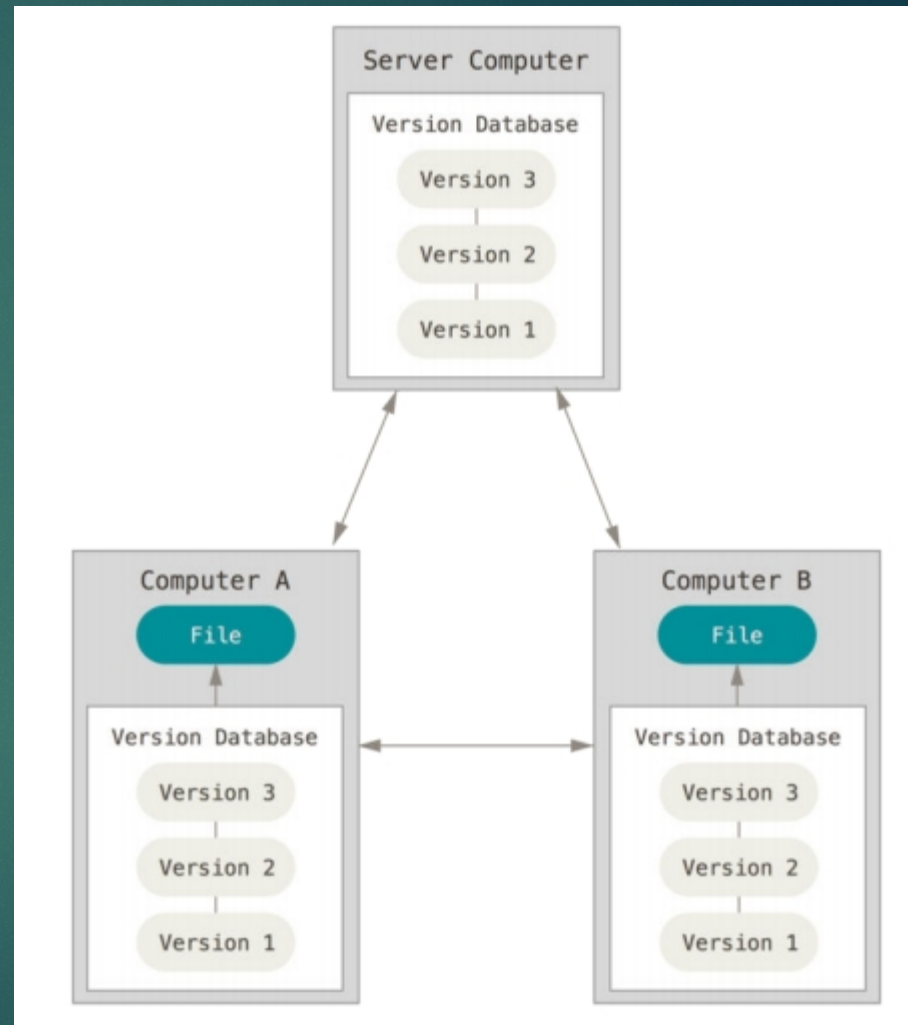


來源： Pro Git P29 Figure1-2

第三代 – D(Distributed)VCS – 分散式 版控系統

- ▶ 不再只有一個中央系統
 - ▶ 任何人裡面有記錄就是一個 Repository
 - ▶ 隨時可以 commit
 - ▶ merge before commit

- ▶ 代表就是：



來源： Pro Git P30 Figure1-3

不同時代的比較

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, SourceSafe, Subversion, Team Foundation Server
Third	Distributed	Changesets	Commit before merge	Bazaar, Git, Mercurial

http://ericsink.com/vcbe/html/history_of_version_control.html

版控為何是工程師第一工具？

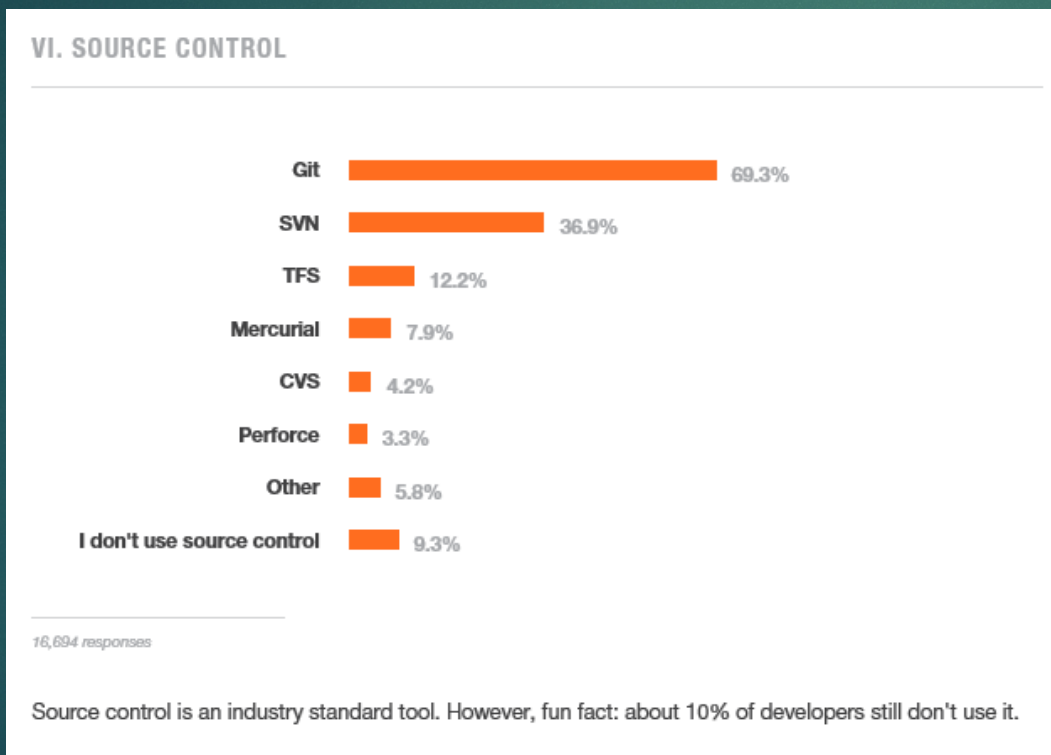
- ▶ 版本控制工具又稱為
 - ▶ VCS – Version Control System
 - ▶ SCM – Source Control Management
- ▶ 要開始開發第一件事情一定是問 git/svn 在那裡
- ▶ 可以隨時回溯到某一個時間點
- ▶ 通透 – 完全知道誰在什麼時間點做了什麼

為什麼選 GIT

- ▶ Git 是由 Linux 之父 Linus Torvalds 開發出來
 - ▶ 今年是 10 週年
 - ▶ <http://www.ithome.com.tw/news/95088>
- ▶ Git 的主要目的是用來管理大型軟體程式的版控
 - ▶ 處理大量資料
 - ▶ 速度快
 - ▶ 容易建立分支 – 分散工作
- ▶ 不需要任何其他裝備就可以開始使用

為什麼選 GIT 2

- ▶ 基本上大部分的 Open Source 專案都用 Git



<http://stackoverflow.com/research/developer-survey-2015#tech-sourcecontrol>

Github

- ▶ Github 在 2008 年成立
- ▶ 把 git 的協同工作能力展現到了極致
- ▶ 也是 Github 和 Git 相輔相成同時成長
- ▶ 裡面有一個 Student 優惠專案

Git 適合版控什麼

- ▶ 任何類型的專案
- ▶ 比較純文字類型的檔案
 - ▶ Word 就比較不適合
- ▶ 預設不適合像
 - ▶ 大型檔案
 - ▶ Binary
 - ▶ 圖片

準備使用 Git 的環境

安裝 Git

- ▶ Git 本身屬於 Command Line 程式
- ▶ 跨平台
 - ▶ Linux
 - ▶ Mac
 - ▶ Windows
- ▶ Gui 程式依照不同平台有不同

Windows 安裝

- ▶ Git 本身
 - ▶ 用 Cygwin 來安裝
 - ▶ 用 Msysgit
 - ▶ <https://msysgit.github.io/>
 - ▶ 除了 git 本身之外還有包含：
 - ▶ Git bash
 - ▶ Gui
 - ▶ 檔案總管 integration

Powershell 作為 cmd

- ▶ 在 Svn，厲害的人都用 cmd，在 Git 厲害的人都用 GUI
 - ▶ 設定 Git 為系統 Path 參數
 - ▶ 如果 Powershell 版本是 2.0（Windows 7），請升級到 3.0 以上
- ▶ Powershell 套件 posh-git（<https://github.com/dahlbyk/posh-git>）

```
Set-ExecutionPolicy RemoteSigned
```

```
(new-object Net.WebClient).DownloadString("http://psget.net/GetPsGet.ps1") |  
iex install-module posh-git
```

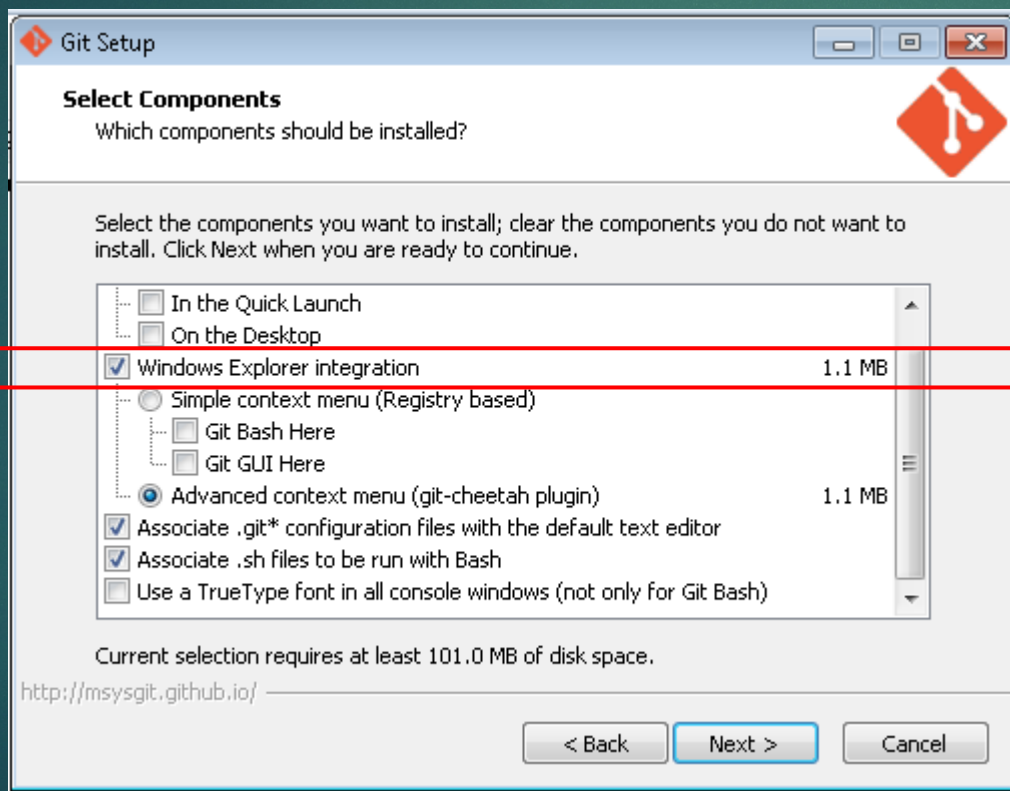

Windows 安裝 Gui

- ▶ TortoiseGit (俗稱 小烏龜, TortoiseSvn)
 - ▶ 支援 Windows
 - ▶ <https://code.google.com/p/tortoisegit/>
 - ▶ 建議 Diff 工具用 WinMerge (<http://winmerge.org/>)
- ▶ SourceTree
 - ▶ 支援 Windows 和 Mac
 - ▶ <https://www.atlassian.com/software/sourcetree/overview>
- ▶ GitHub For Windows
- ▶ 更多 Gui 工具 - <https://git-scm.com/downloads/guis>

Msysgit 安裝 1

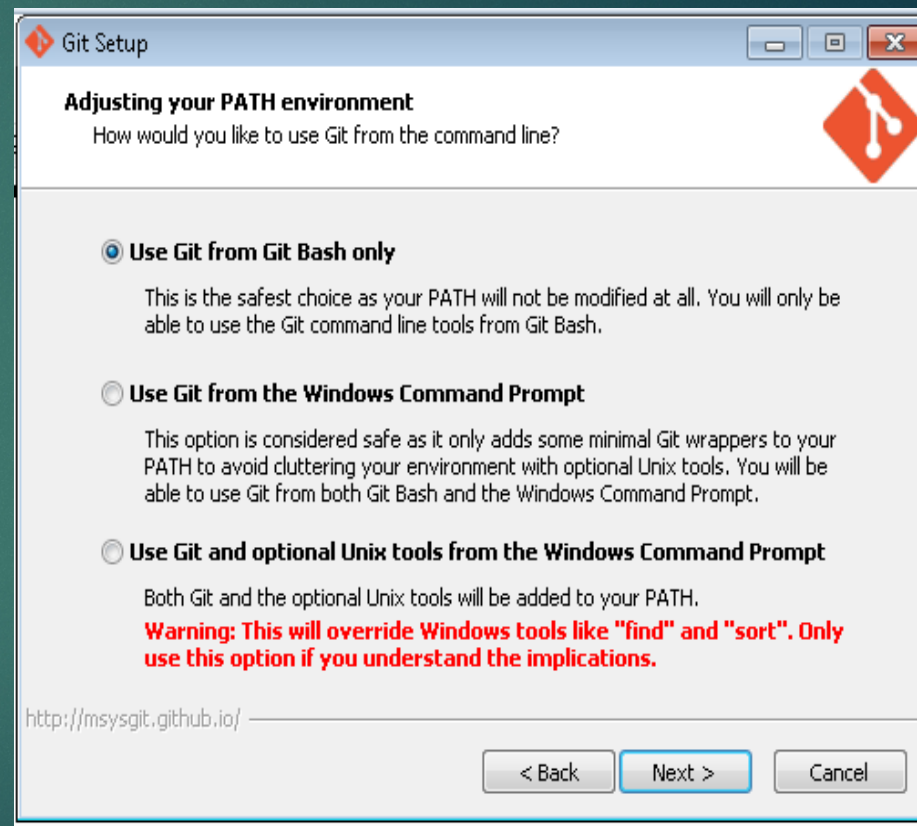
► Windows Explorer Integration

- 建議不要勾，因為其他 Gui 工具就有提供



Msysgit 安裝 2

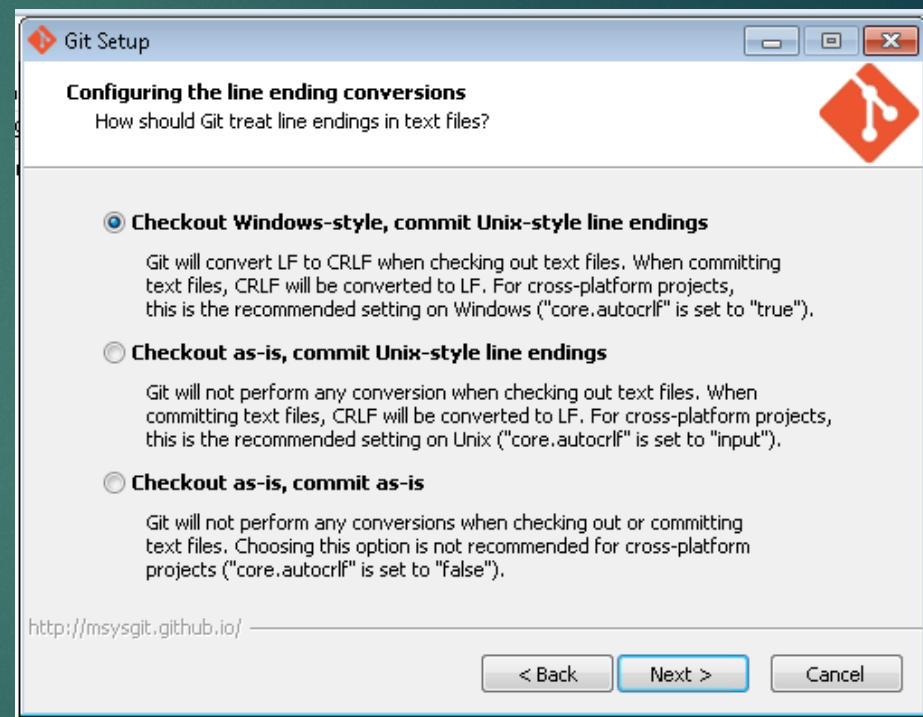
- ▶ Use Git from Git Bash only
 - ▶ 只有 Git Bash 能夠跑 Git
- ▶ Use Git from the Windows Command prompt
 - ▶ 在 Path 上面加 git - cmd 就可以執行 git
- ▶ Use Git and optional Unix tools from Windows Command Prompt
 - ▶ 在 Path 加上 git 和 安裝一些 Unix 工具



Msysgit 安裝 3

- ▶ Windows 斷行吃 CRLF 而 Unix 吃 LF
- ▶ 這邊是設定版控儲存的斷行方式
- ▶ 這個可以安裝完成之後透過 git config 設定

```
git config --system core.autocrlf"true"
```



開始使用 Git – 設定

設定檔 - 層級

- ▶ System 層級
 - ▶ Unix - /etc/gitconfig
 - ▶ Windows - {Git 安裝路徑}\etc\gitconfig
- ▶ 使用者層級
 - ▶ Unix - ~/.gitconfig 或者 ~/.config/git/config
 - ▶ Windows - %userprofile%\gitconfig
- ▶ 專案層級
 - ▶ 在專案下的 .git/config
- ▶ 權重：專案層級 > 使用者層級 > System 層級

設定設定檔 – git config

- ▶ System 層級

- ▶ git config --system { 參數 }

```
git config --system user.name "Alan Tsai"
```

- ▶ 使用者層級

- ▶ git config --global { 參數 }

```
git config --global user.name "Alan Tsai"
```

- ▶ 專案層級

- ▶ git config { 參數 }

```
git config user.name "Alan Tsai"
```

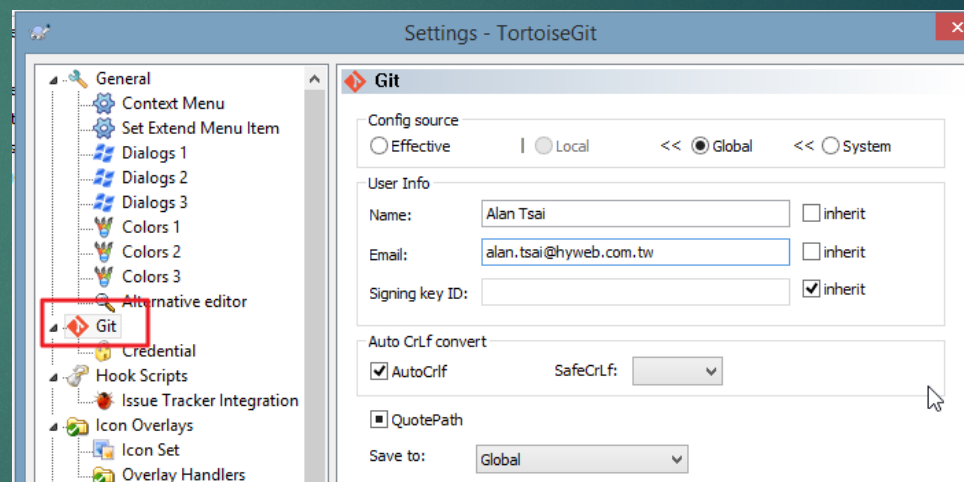
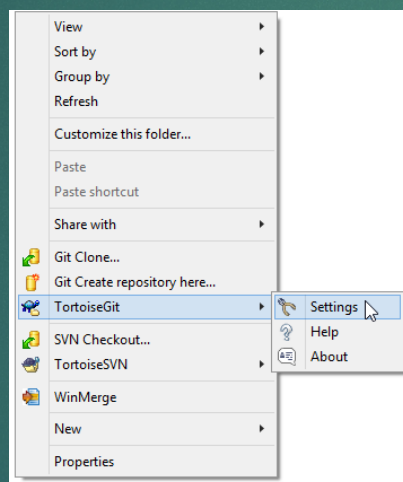

必設定參數

- ▶ 姓名和 Email 為必設欄位

- ▶ 透過 cmd

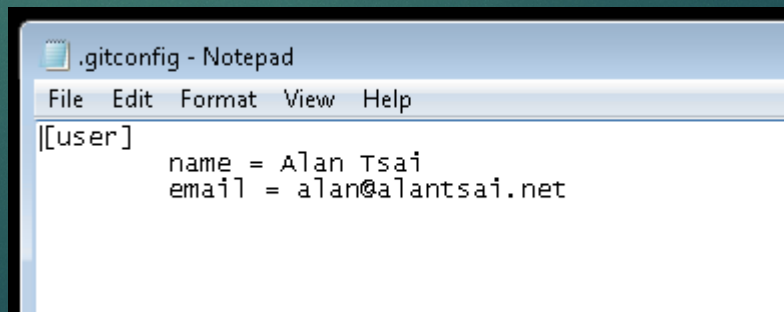
```
git config --global user.name "Alan Tsai"
git config --global user.email "alan@alantsai.net"
```

- ▶ 透過 TortoiseGit



- ▶ 透過直接改檔案

- ▶ 路徑 - %userprofile%\gitconfig



設定使用的編輯器

- ▶ 預設 git 使用 vim 來輸入 commit message
- ▶ 可以使用自己喜歡的編輯器
 - ▶ 範例 – 設定使用 notepad++

```
git config core.editor "C:/Program Files (x86)/Notepad++/notepad++.exe"
```


確認目前設定的參數

- ▶ 看所有的參數值：

```
git config --list
```

- ▶ 針對某一個參數值：

```
git config user.name
```

```
PS C:\Users\vagrant> git config user.name  
Alan Tsai
```

```
Administrator: Windows PowerShell  
PS C:\Users\vagrant> git config --list  
core.symlinks=false  
core.autocrlf=true  
color.diff=auto  
color.status=auto  
color.branch=auto  
color.interactive=true  
pack.packsizelimit=2g  
help.format=html  
http.sslcainfo=/bin/curl-ca-bundle.crt  
sendemail.smtpserver=/bin/msmtp.exe  
diff.astextplain.textconv=astextplain  
rebase.autosquash=true  
merge.tool=kdiff3  
mergetool.kdiff3.path=C:/Program Files/KDiff3/kdiff3.exe  
diff.guitool=kdiff3  
difftool.kdiff3.path=C:/Program Files/KDiff3/kdiff3.exe  
core.editor="C:/Program Files (x86)/GitExtensions/GitExtensions.exe" fi  
user.name=Alan Tsai  
user.email=alan@alantsai.net
```


Git 基礎使用說明

建立一個本機 repo

- ▶ 要使用 git 就要建立一個本機的 repository
 - ▶ 這個 repository 將會記錄每一次的版本變化

- ▶ 使用指令：

- ▶ 先移動到專案目錄，然後使用 git init

```
cd C:\Project\Git  
git init
```

- ▶ 使用 TortoiseGit
- ▶ 在資料夾會多出一個 .git 的隱藏資料夾
 - ▶ 所有的版控歷史都在這個資料夾裡面

加入一個檔案到版控

- ▶ 先在資料夾加一個叫做 *ReadMe.md* 的檔

- ▶ 下一個 git status 指令來看目前版控情況

git status

- ▶ 下 add 把檔案加到 staging

git add .

- ▶ 下 commit 把 staging 內容存到版控

git commit

```
Administrator: posh~git ~ git [master]
C:\project\git [master +1 ~0 -0 !]> git status
On branch master

Initial commit

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)
C:\project\git [master +1 ~0 -0 !]> _
```

```
C:\project\git [master +1 ~0 -0 !]> git add .
C:\project\git [master +1 ~0 -0 !]> git status
On branch master

Initial commit

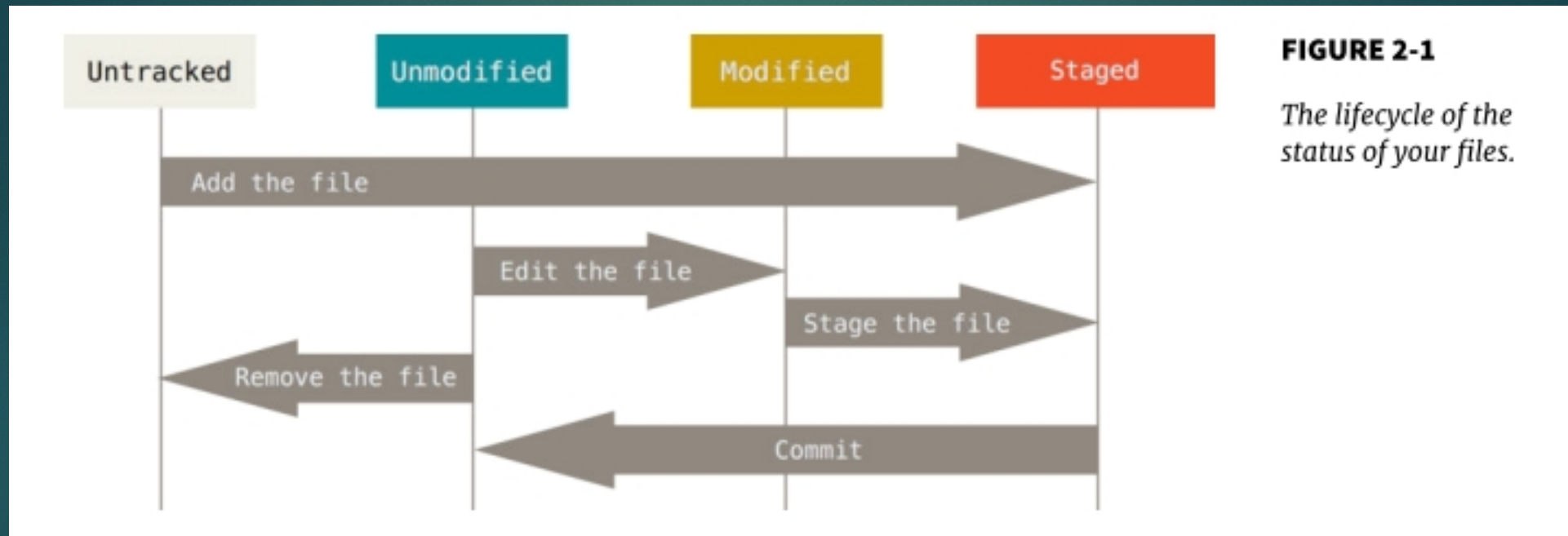
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   ReadMe.md

C:\project\git [master +1 ~0 -0 !]> _
```

```
C:\project\git [master +1 ~0 -0 !]> git commit
[master (root-commit) daea7f1] commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ReadMe.md
C:\project\git [master]> _
```

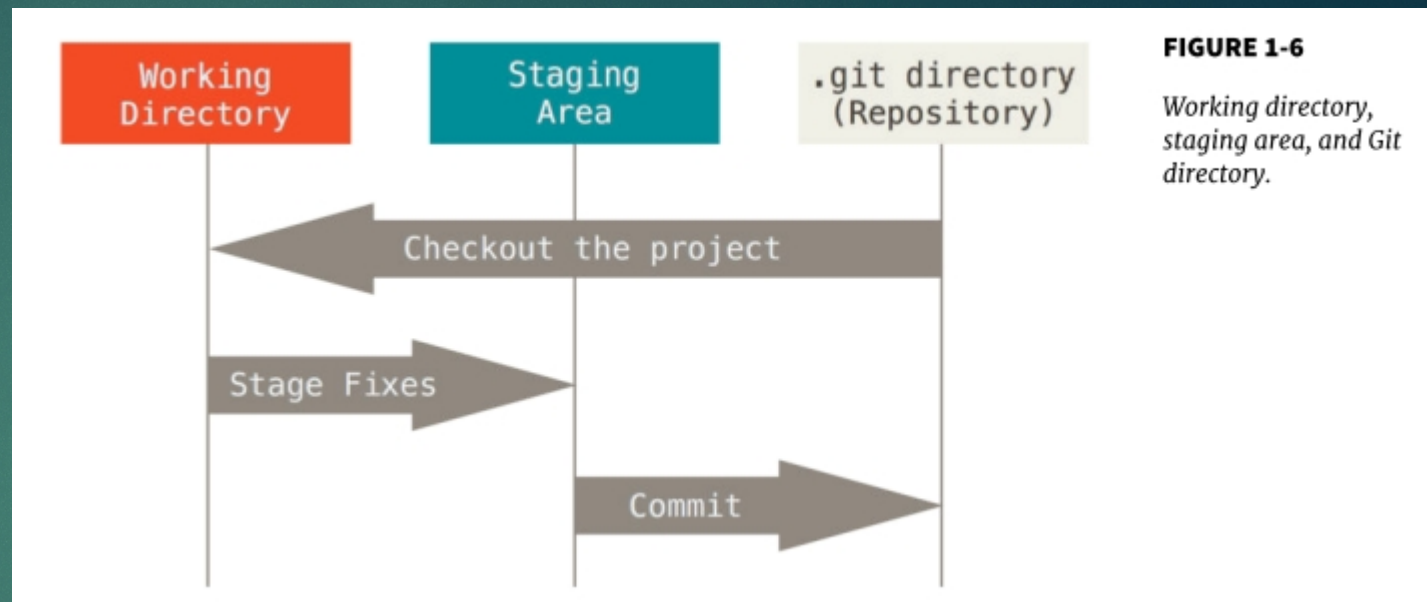

檔案的幾種狀態



- ▶ Git add 就是把檔案放到 Staging

關鍵字

- ▶ Repository（也稱作為“版控”）
 - ▶ 整個版控歷史記錄。
 - ▶ 就是 .git 資料夾
- ▶ Working Directory
 - ▶ 目前的工作資料夾
 - ▶ 任何修改尚未進入 Staging 或者 版控的修改
- ▶ Staging Area
 - ▶ 任何被 git add 過的檔案
 - ▶ 下次 git commit 會進入的版控的內容



看版本歷史記錄: git log

- ▶ 用 git log 指令可以看到歷史記錄

- ▶ 這個指令有很多參數

- ▶ 舉例:

git log

```
R:\jquery [master]> git log
commit efdd6f2b3eff7b211c88b1423d627e72e6dad858
Merge: e4acaac daf7a72
Author: Alan Tsai <alantsai2007@outlook.com>
Date: Sat May 30 08:54:19 2015 +0800

    Merge branch 'dev'

    Conflicts:
        .gitignore

commit e4acaacd558ff47ad8b2f13c487da36a0744d5be
Author: Alan Tsai <alantsai2007@outlook.com>
Date: Sat May 30 08:51:51 2015 +0800

    fff

commit daf7a725fed6a5b84689d9d8c495d917af36f9ff
Author: Alan Tsai <alantsai2007@outlook.com>
Date: Sat May 30 08:46:35 2015 +0800

    test

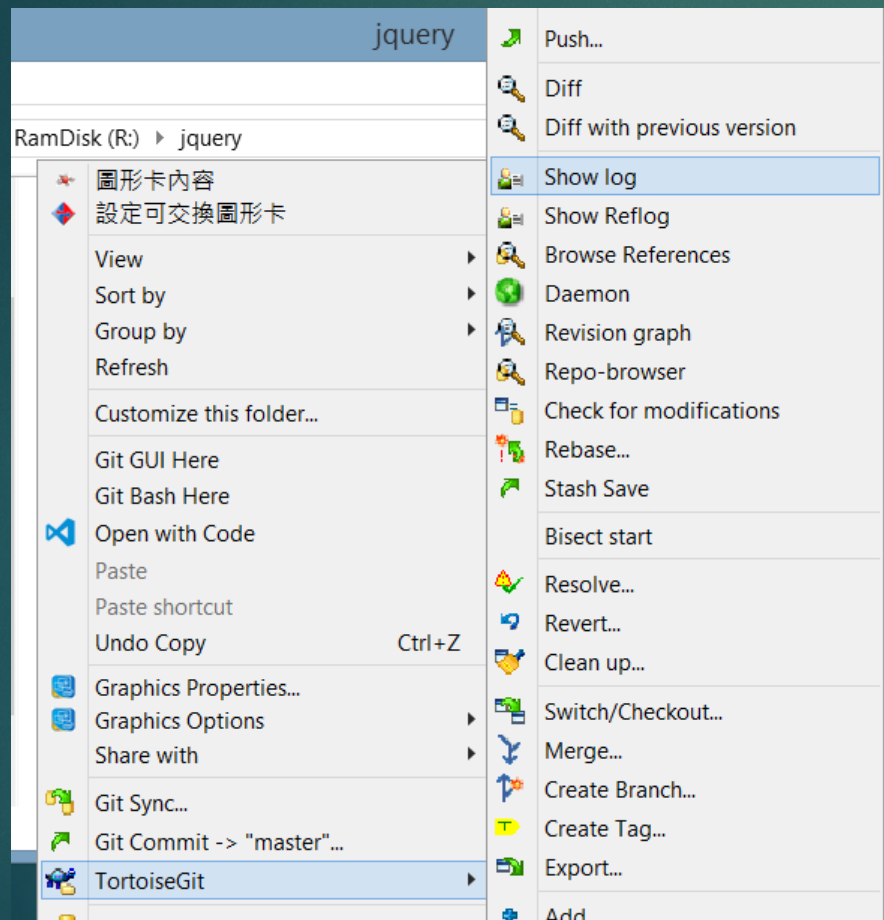
commit 6df669f0fb87cd9975a18bf6bbe3c3548afa4fee
Author: Oleg Gaidarenko <markelog@gmail.com>
Date: Wed May 20 18:09:46 2015 +0300

    Event: remove outdated originalEvent hack
```

git log --oneline --abbrev-commit --all --graph

```
R:\jquery [master]> git log --oneline --abbrev-commit --all --graph
* efdd6f2 Merge branch 'dev'
* |
* | * daf7a72 test
* | * e4acaac fff
* |
* |
* |
* 6df669f Event: remove outdated originalEvent hack
* 7475d5d Event: Remove fake originalEvent from jQuery.Event.simulate
* 3c92770 Docs: remove redundant instruction from the readme
```


用 TortoiseGit 的 Show log



<All Branches> From: 3/22/2006 To: 5/30/2015 Filter by Subject, M

Graph	SHA-1	Actions	Message
	00000000000000000000...		Working dir changes
	efdd6f2b3eff7b211c8...		master Merge branch 'dev'
	daf7a725fed6a5b84689d...		dev test
	e4acaacd558ff47ad8b2f1...		dev2 fff
	6df669f0fb87cd9975a18b...		origin/HEAD origin/master Event: remove outdated originalEvent hack
	7475d5debeb7c5315892...		Event: Remove fake originalEvent from jQuery.Event.simulate
	3c9277086742fe3a38a26...		Docs: remove redundant instruction from the readme
	a644101ed04d0beacea8...		Build: update requirejs dependency to 2.1.17
	0705be475092aede1edd...		Event: remove deprecated event aliases
	c074006a69db73a116dc...		Event: provide verbose comment for focus(in out) & rename support prop
	61e21a4eaf479406b6603...		Build: bower.json: remove moot `version` field
	2d715940b9b6fdeed005c...		Offset: account for scroll when calculating position

Git status – 取得目前狀況

► 提供有用諮詢供參考：

```
R:\test [master]> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    dsf.txt

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

        zcvzcvzv.txt

no changes added to commit (use "git add" and/or "git commit -a")
R:\test [master +1 ~0 -1 !]> _
```

```
R:\test [master +1 ~0 -0 | +0 ~0 -1]> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

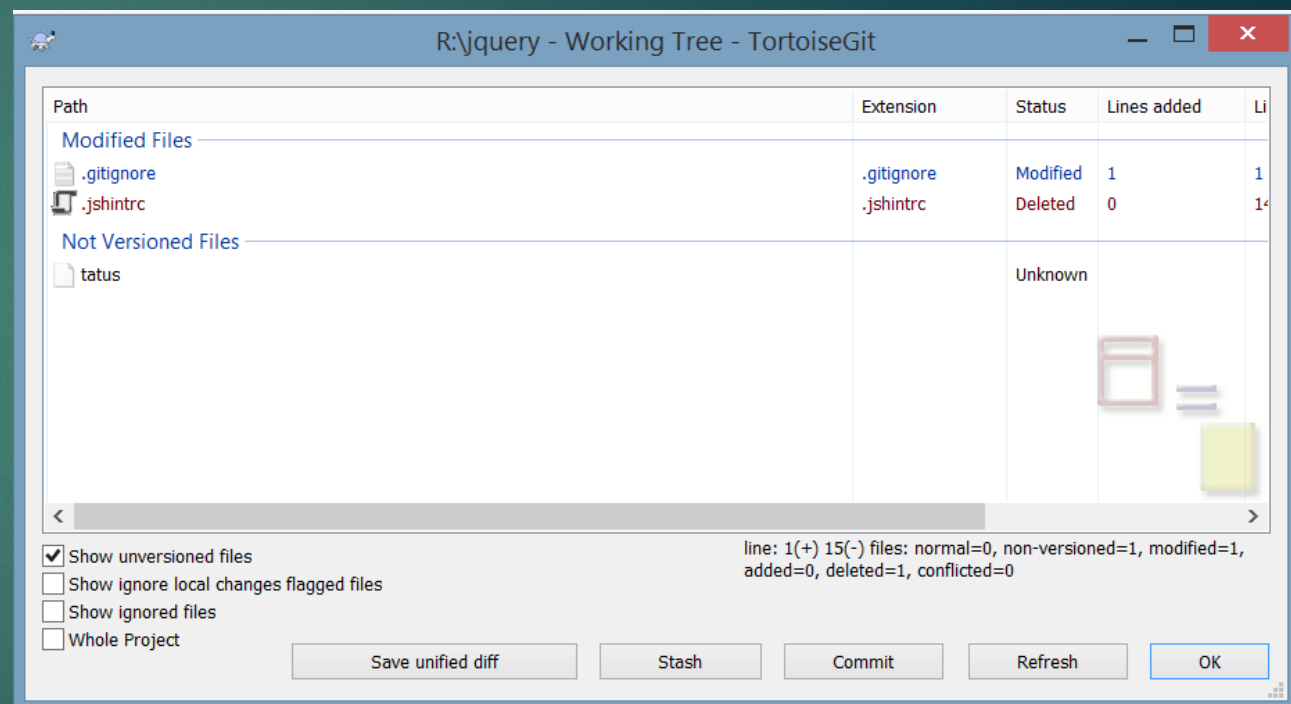
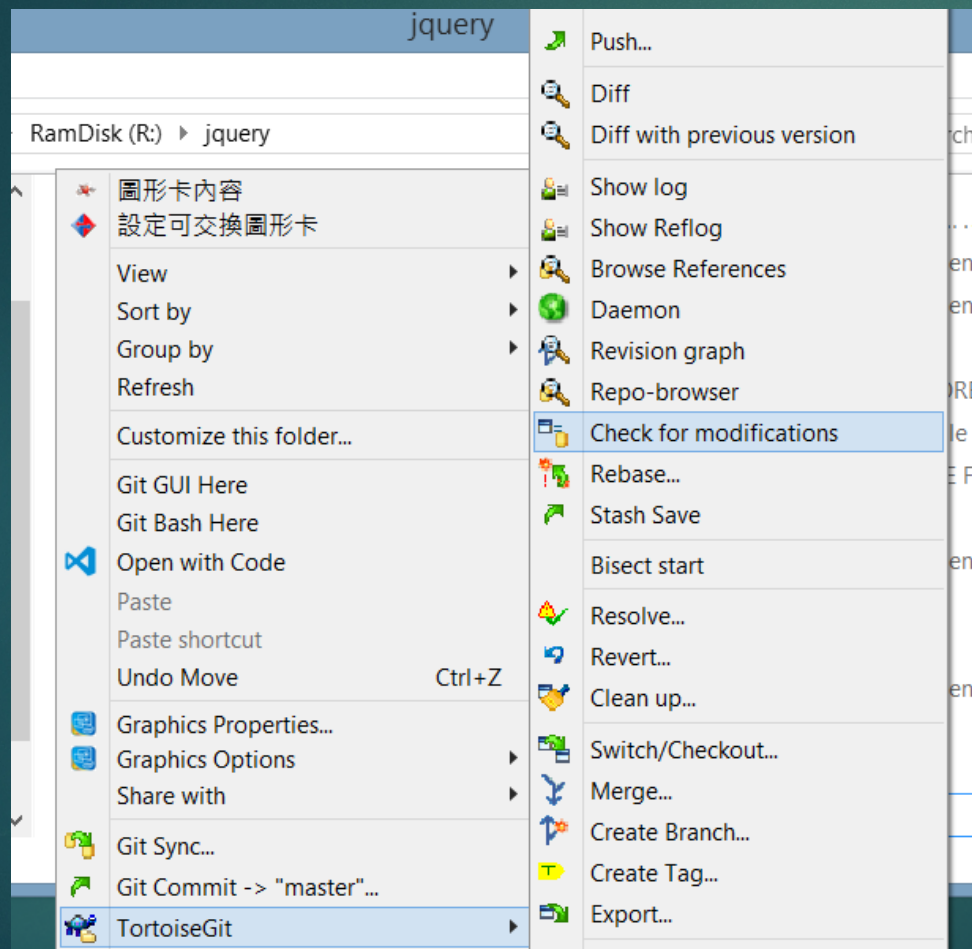
        new file:   zcvzcvzv.txt

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

        deleted:    dsf.txt

R:\test [master +1 ~0 -0 | +0 ~0 -1]>
```


TortoiseGit – Check Modification 看那些有修改



Git add 進階使用

- ▶ `git add --u`
 - ▶ 加入已經有存在 repo 裡面的檔案到 staging
- ▶ `git add -A` 或者 `git add --all`
 - ▶ 加入新增檔案和已經有存在 repo 裡面的檔案到 staging
- ▶ `git add -i`
 - ▶ 互動式加入檔案

設定忽略檔案 - .gitignore

- ▶ 並不是所有檔案都要進入版控
 - ▶ 原則上能夠重新產生出來都不要進去
 - ▶ 例如編譯過的 dll、exe
- ▶ 專案層級就是放在和 .git 同一個層級，建立 .gitignore 檔案即可
- ▶ 全域可以用

```
git config --global core.excludesfile "{ 路徑 }"
```

- ▶ gitignore 檔案可以參考：<https://github.com/github/gitignore>

儲存到版本庫 - Git commit

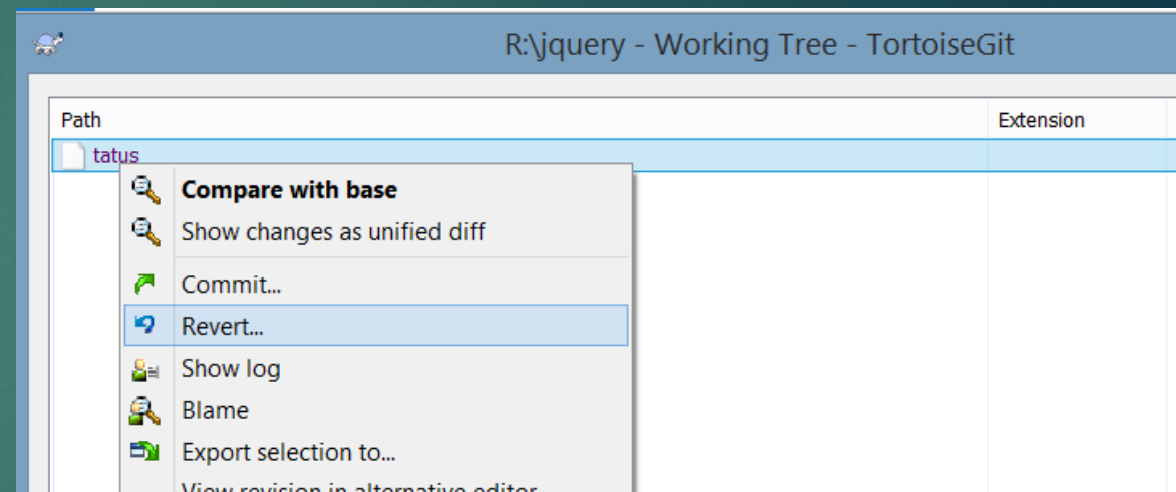
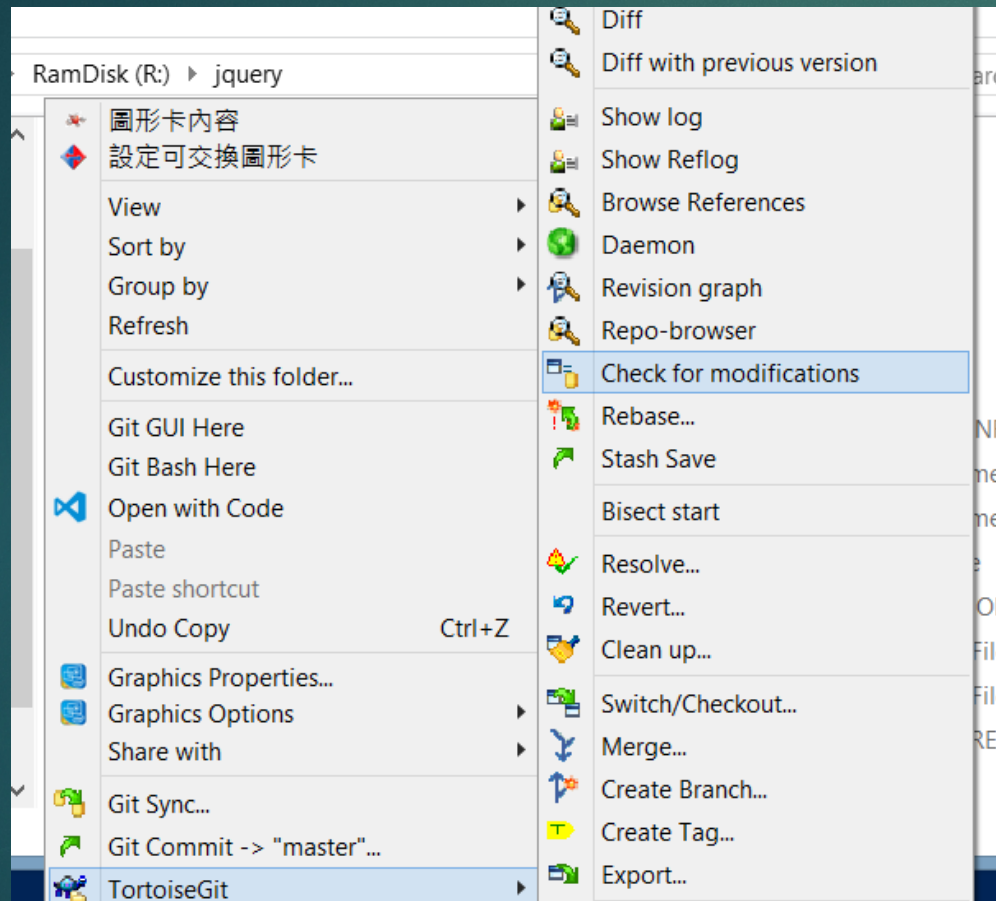
- ▶ Git commit – 把目前的 staging 結果記錄到版控
- ▶ git commit -a
 - ▶ git add -a + git commit
- ▶ Git commit –amend
 - ▶ 修改上一個版本
 - ▶ 例如：
 - ▶ 修改版本訊息
 - ▶ 增加或減少檔案

還原到最後一次儲存版本

- ▶ 可以使用指令 `git checkout` 來還原到目前 repo 版本
- ▶ 修改 ReadMe.md
- ▶ 還原修改

```
git checkout ReadMe.md
```


還原到最後一次儲存版本 - tortoiseGit



git reset HEAD { 檔案 }

- ▶ 把檔案從 staging 給退回到 working copy

```
git reset HEAD ReadMe.md
```


暫存功能 – git stash

- ▶ 功能做到一半臨時需要去處理別的怎麼辦？

- ▶ 暫存目前在 Working Directory 的任何修改

```
git stash
```

- ▶ 取回第一筆暫存並且把這個暫存刪掉

```
git stash pop
```

- ▶ 看目前有那些暫存

```
git stash list
```


git help 幫助指令

- ▶ 可以用 git help 來列出指令

```
git help
```

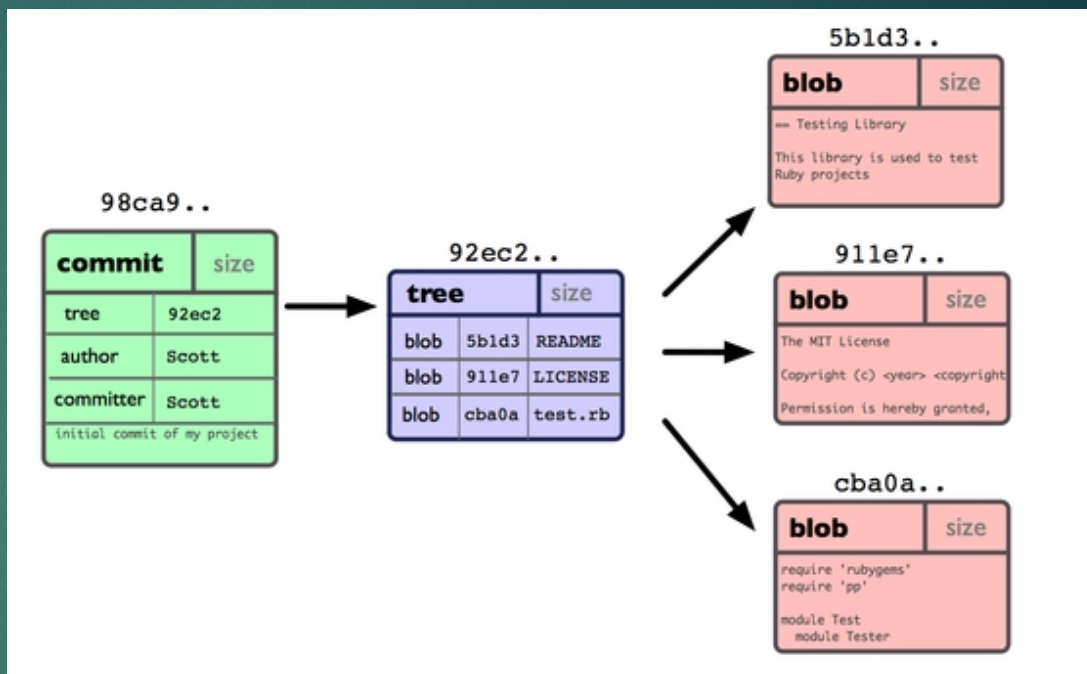
- ▶ 可以用 git { 指令 } -h 列出指令的幫助訊息

```
git add -h
```


Git 最強大的功能 - 分支

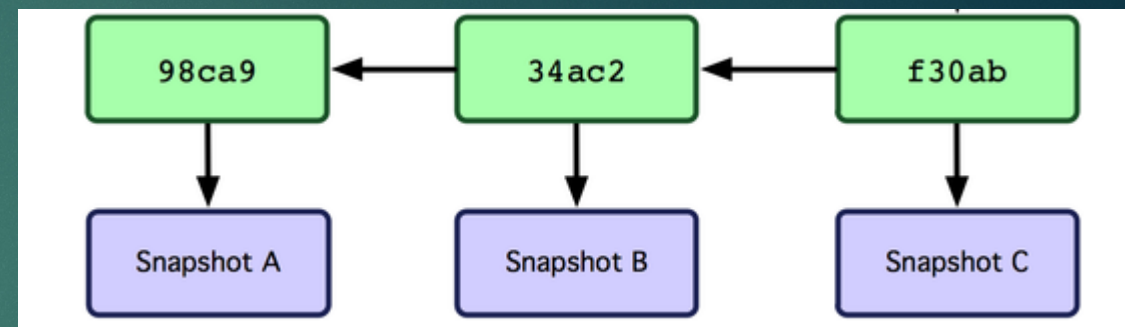
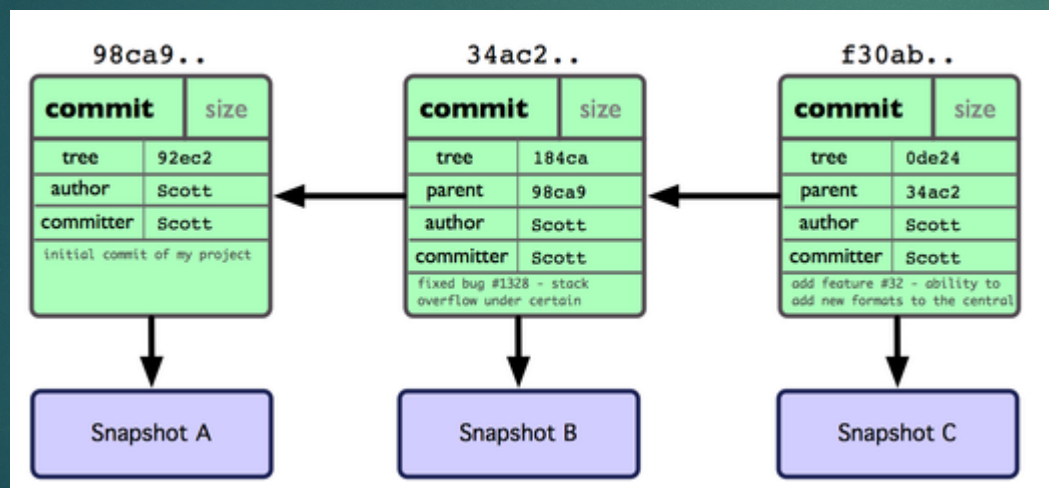
一個 commit 有什麼東西

- ▶ 最簡單來看
 - ▶ 修改的檔案內容
 - ▶ 這個 Commit 的上一個 Commit
 - ▶ 等等資訊建立出一個 Sha1 的代表



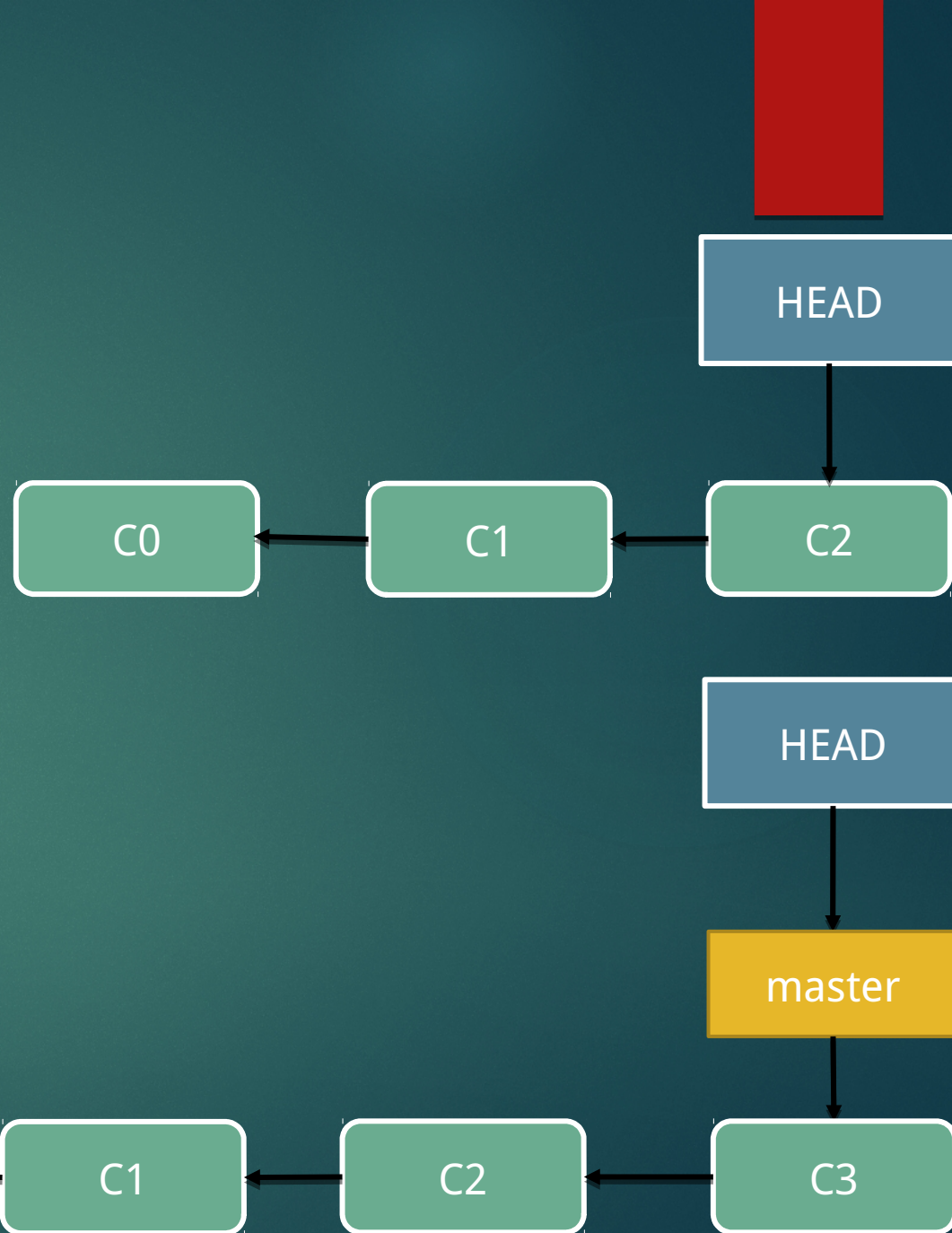
commit 之間如何關聯

- ▶ Git 的每一個 commit 都是屬於檔案當時的 snapshot
 - ▶ 而不是只記錄檔案的差異性
- ▶ 每一個 Commit 都會連某一個父節點。



HEAD 和 master

- ▶ HEAD 代表目前所在的 commit 是那個
- ▶ 如果這個時候做 git commit，會多出一個 C3 並且 HEAD 指向 C3
- ▶ HEAD 代表目前在那裡，但是缺少一個東西幫你一直指向目前最新
 - ▶ 因此所有 git 版控有個 master branch



什麼是分支？

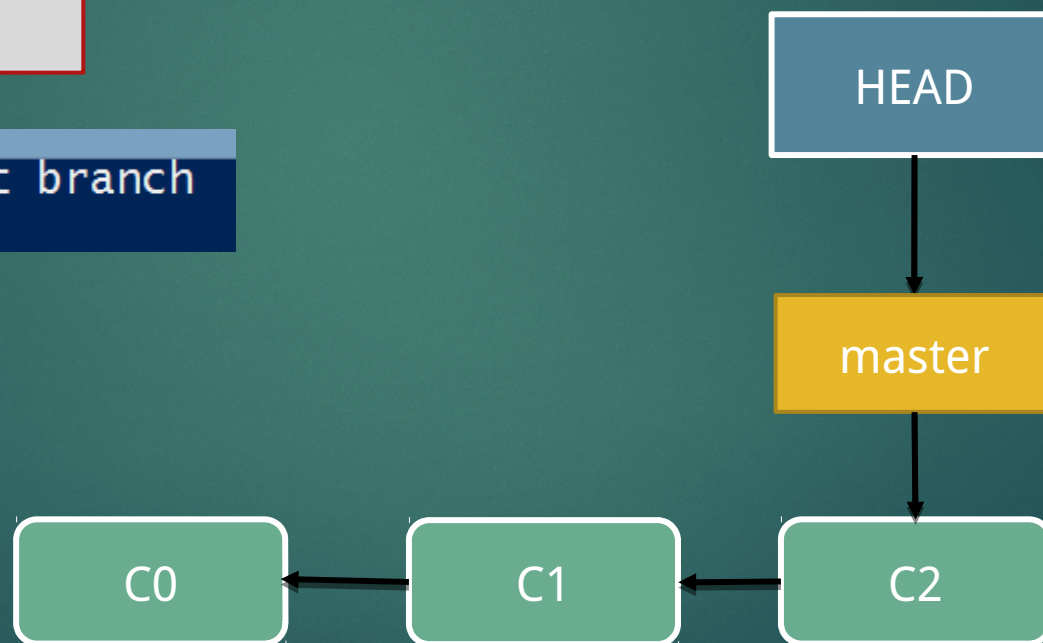
- ▶ 分支只是一個可移動的指標 (pointer)，指向某一個 commit 而已
 - ▶ 讓使用者不用輸入“落落長”的 sha1
- ▶ 任何 git repo 預設一定有一個 master 分支
- ▶ 只有當 HEAD（也就是目前）在某一個分支上，才能夠 commit

分支管理 - git branch

- ▶ 假設目前的樹是有 3 個 commit ,
並且只有預設的 branch :

```
git branch
```

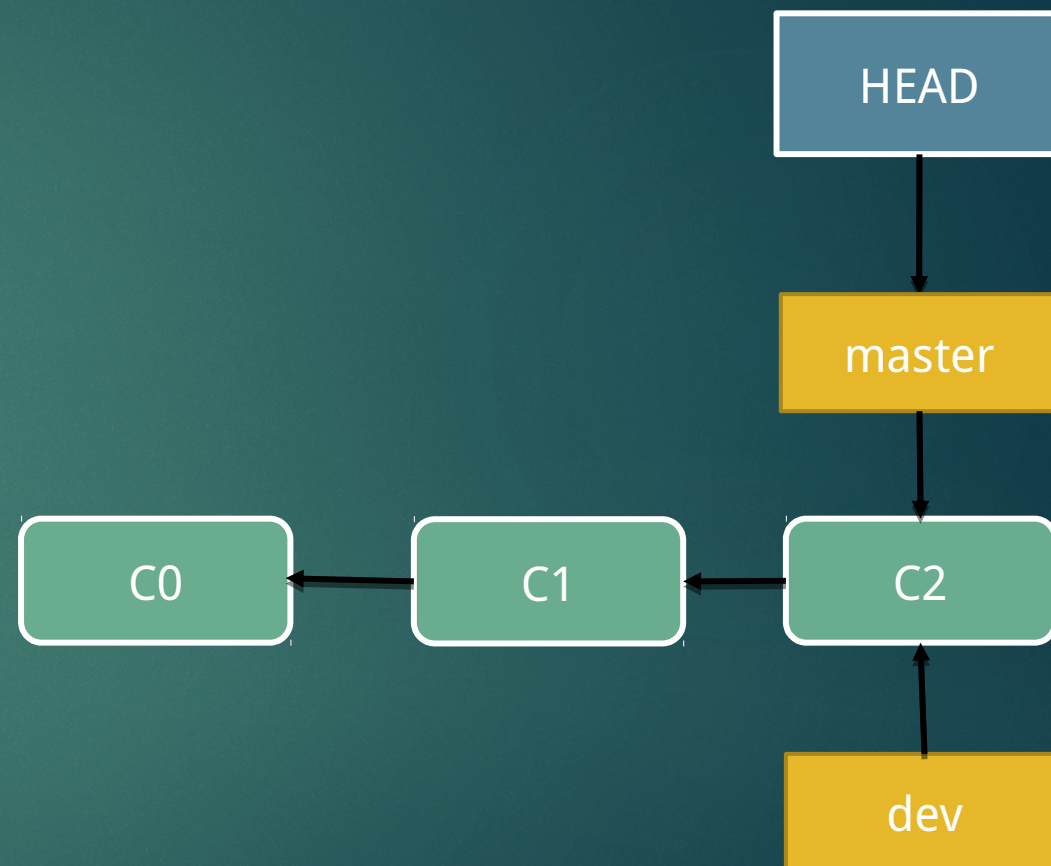
```
R:\test [master]> git branch  
* master
```



分支管理 - git branch {branch name}

```
git branch dev  
git branch
```

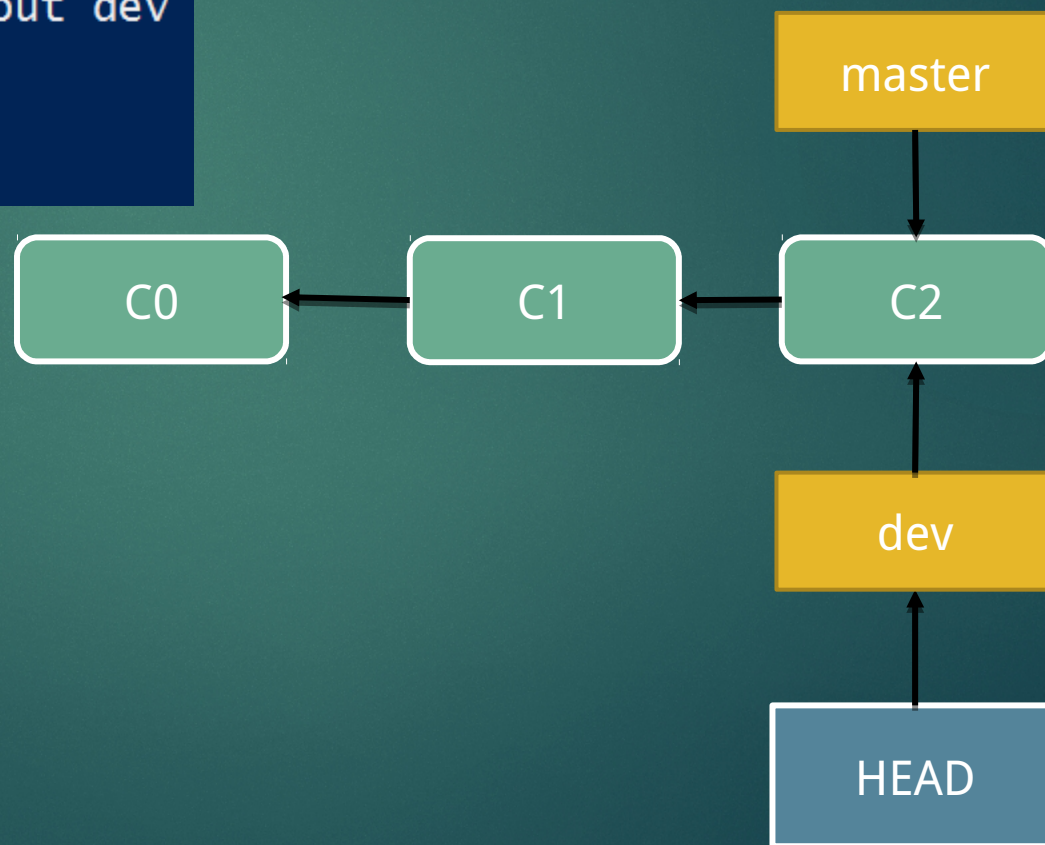
```
R:\test [master]> git branch dev  
R:\test [master]> git branch  
dev  
* master
```



分支管理 – git checkout 切换分支

```
git checkout dev
```

```
R:\test [master]> git checkout dev  
Switched to branch 'dev'  
R:\test [dev]> git branch  
* dev  
  master
```



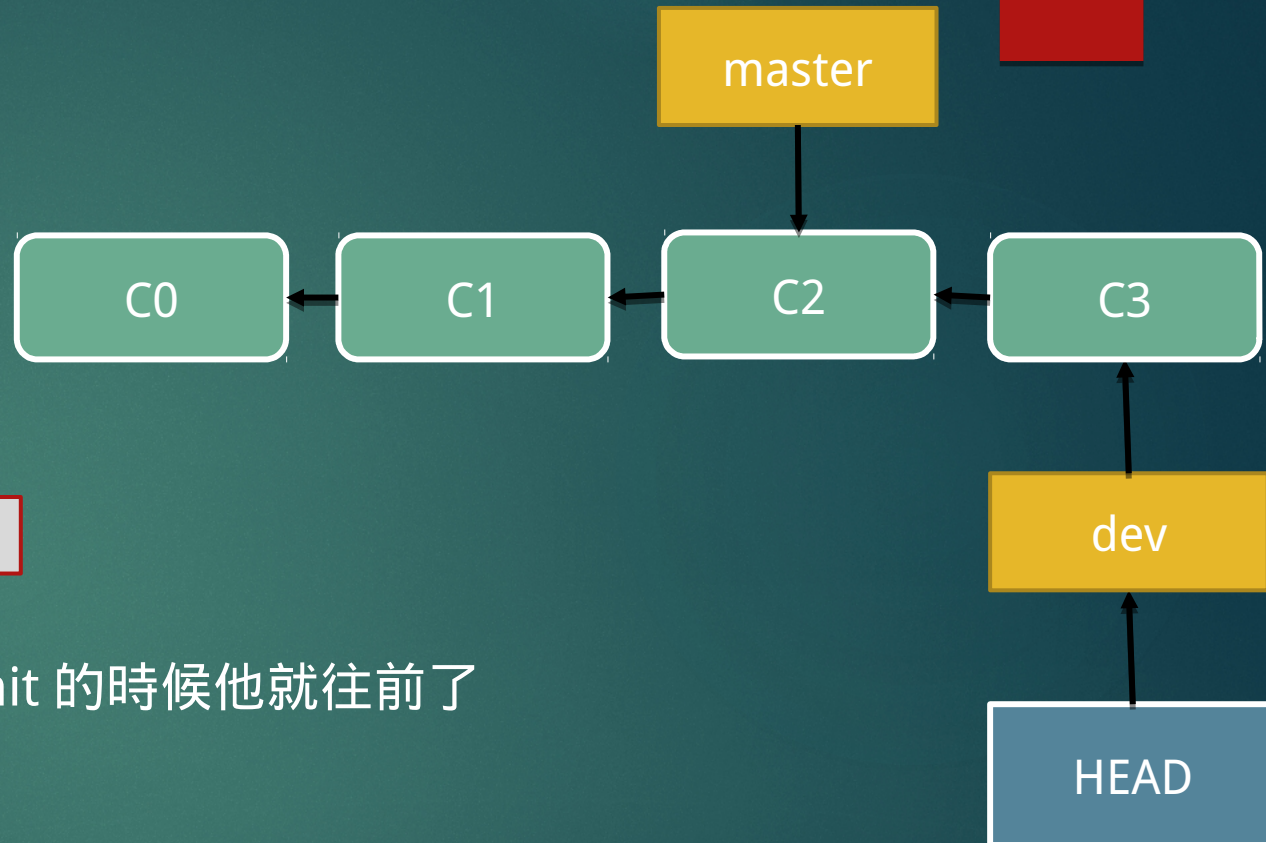
git commit

```
git add .  
git commit
```

▶ 或

```
git commit -a m "change readMe"
```

- ▶ 因為 HEAD 在 dev 分支，所以 commit 的時候他就往前了
- ▶ 而 master 不在目前 HEAD 之下所以不會動



什麼時候適合開分支？

- ▶ 分支主要目的是在不影響其他版本的情況下可以去做測試
- ▶ 所以只要是：
 - ▶ 開發新功能
 - ▶ 開發上面的測試
 - ▶ 修 bug
- ▶ 都可以考慮用分支來做
- ▶ 完成之後在合併想要的內容

git tag – 不可移動的分支

- ▶ 有時候會想要在某些 commit 上面標註一些事情
 - ▶ 版本資訊 – 例如 1.0
- ▶ 自己記 sha1 不方便，也沒法分享給所有人
 - ▶ 用 branch 怕別人移動
- ▶ 這就是 git tag 的用途
 - ▶ 針對某個 commit 設定標籤
 - ▶ 本質上就是一個不可移動的 branch

git tag

- ▶ 列出目前的 tag

```
git tag
```

- ▶ Tag 有兩種

- ▶ annotated tag

- ▶ 可以寫說明

```
git tag -a v1.0
```

- ▶ Lightweight tag

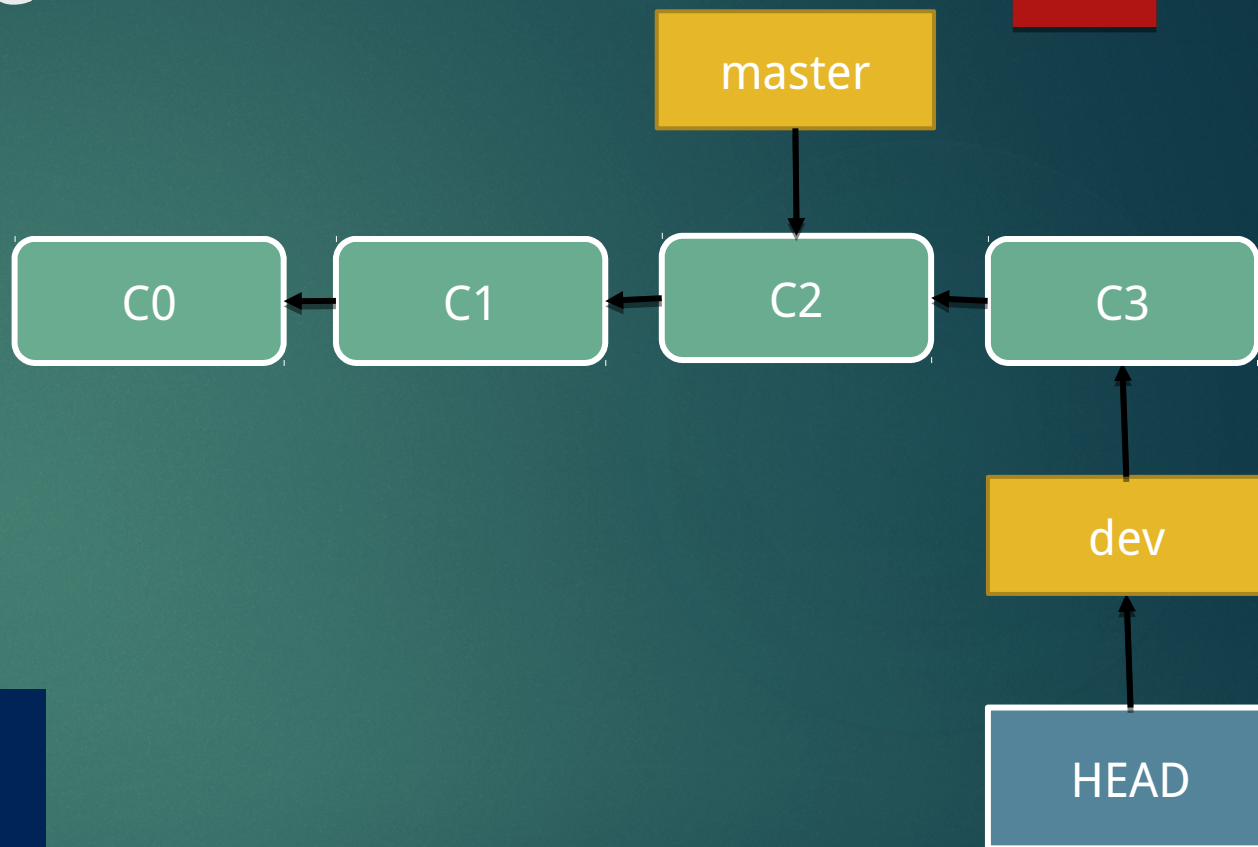
```
git tag v1.0
```

- ▶ 建議使用 annotated tag

合併分支 – git merge

- ▶ 假設開發完成了

- ▶ C3 很正常，準備讓 master 分支也有包含這個功能



```
git checkout master  
git merge dev  
git branch -d dev
```

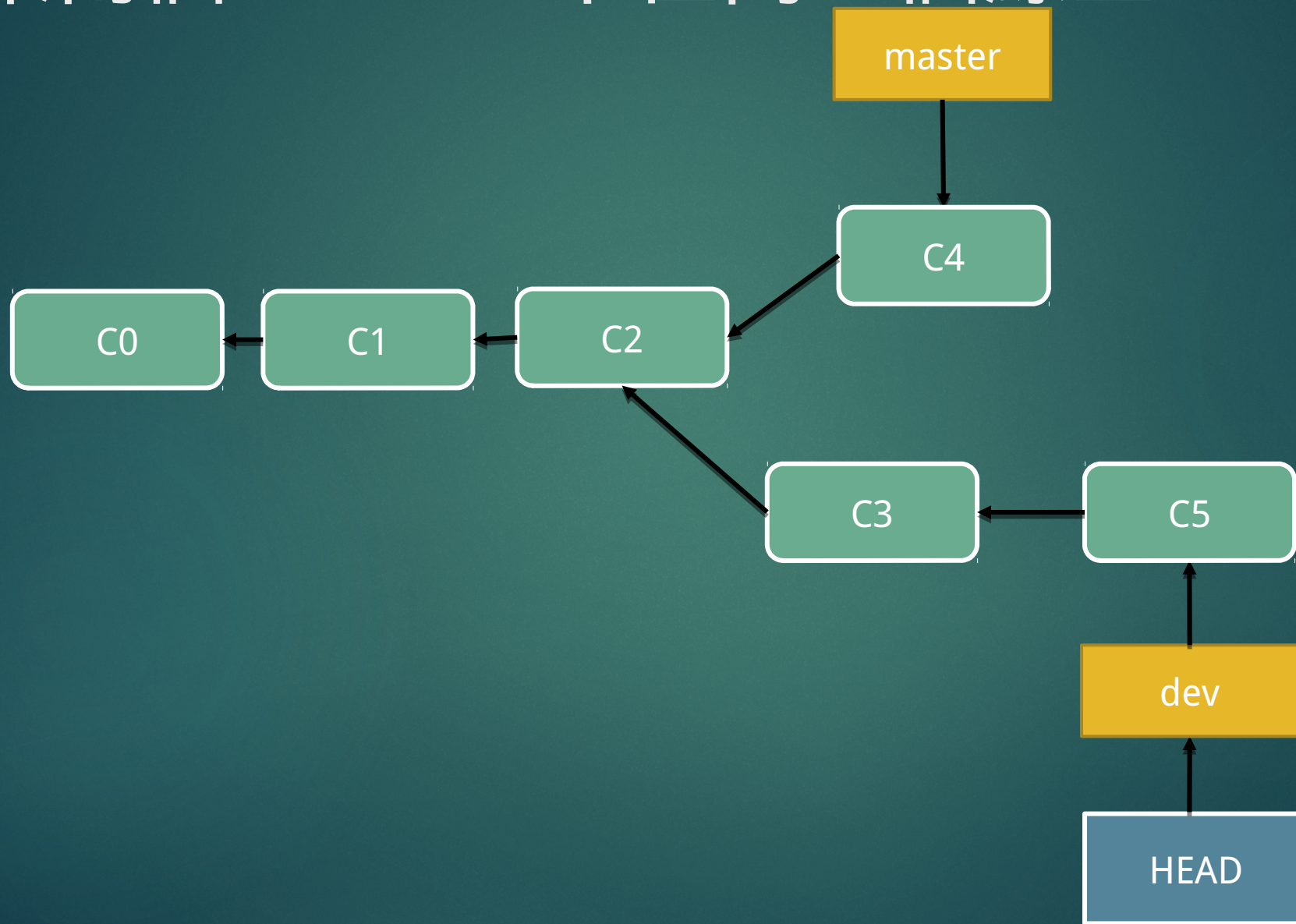
```
R:\test [dev]> git checkout master  
Switched to branch 'master'  
R:\test [master]> git merge dev  
Updating ed843d0..00b614c  
Fast-forward  
 dsf.txt | 1 +  
 1 file changed, 1 insertion(+)  
R:\test [master]> git branch -d dev  
Deleted branch dev (was 00b614c).
```


Fast Forward Merge

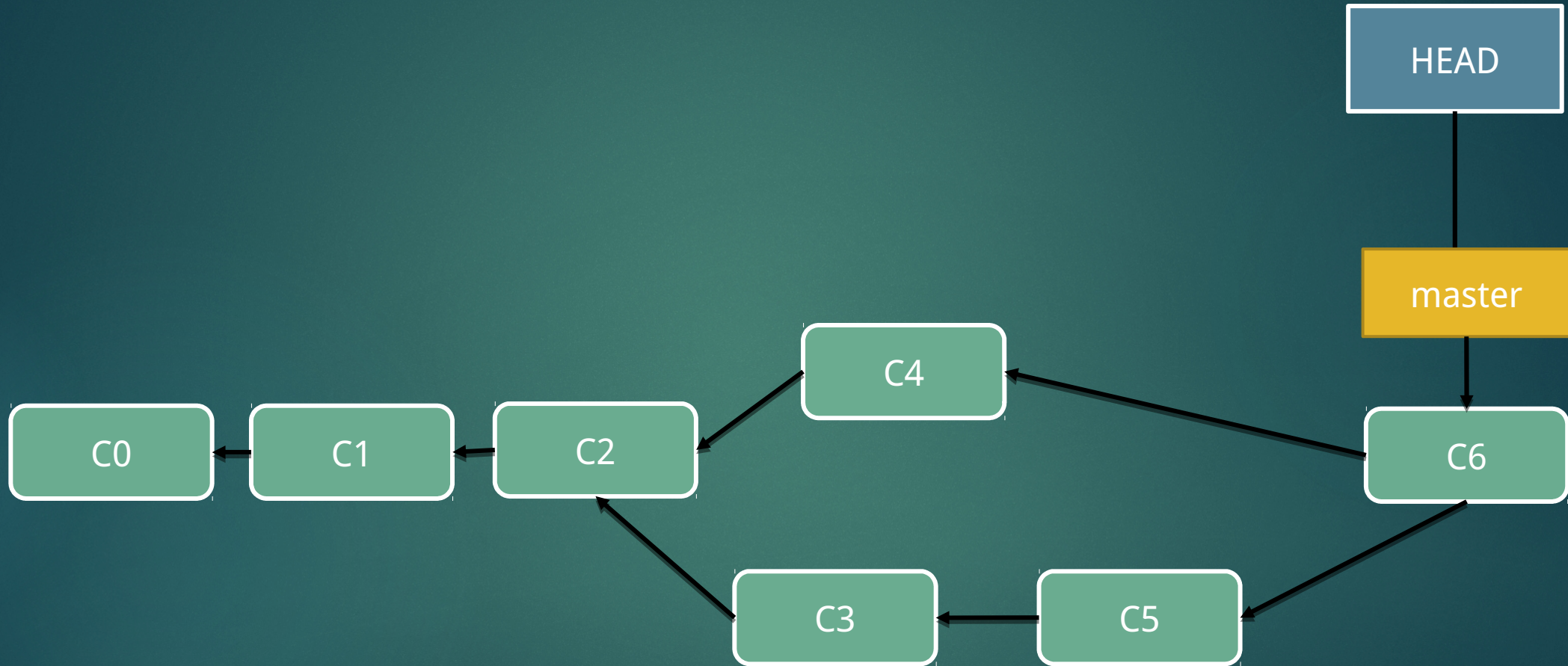
- ▶ 當 merge 發生，git 會去找兩個 branch 之間最後一個的共通 commit
 - ▶ 以我們的例子就是 C2
- ▶ 找到之後看看是否接下來都在同一個線上
 - ▶ 如果是就是所謂的 “Fast Forward Merge”



如果兩個 branch 不在同一個線上



Merge



衝突 (Conflict)

- ▶ 假設在 merge 的時候同個檔案，同個地方有被同時修改，會出現 conflict

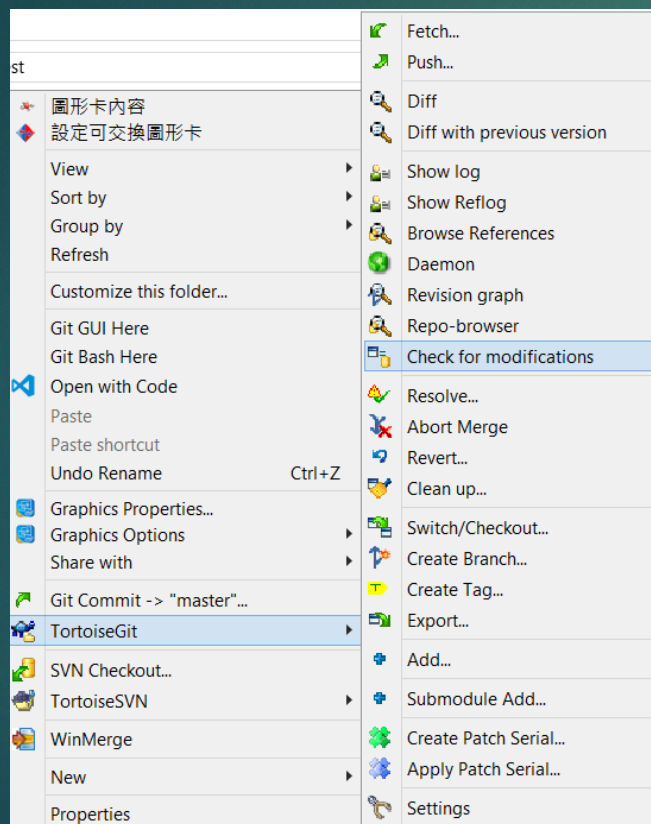
```
R:\test [master]> git merge dev
Auto-merging dsf.txt
CONFLICT (content): Merge conflict in dsf.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- ▶ 如果直接打開 conflict 檔案，會看到
 - ▶ 基本上 ===== 以上的就是那個版本有的內容
 - ▶ 以例子來說就是 HEAD 的內容是“版本 1”
 - ▶ ===== 之下是合併的版本內容
 - ▶ 例子就是 dev branch 的“版本 2”文字

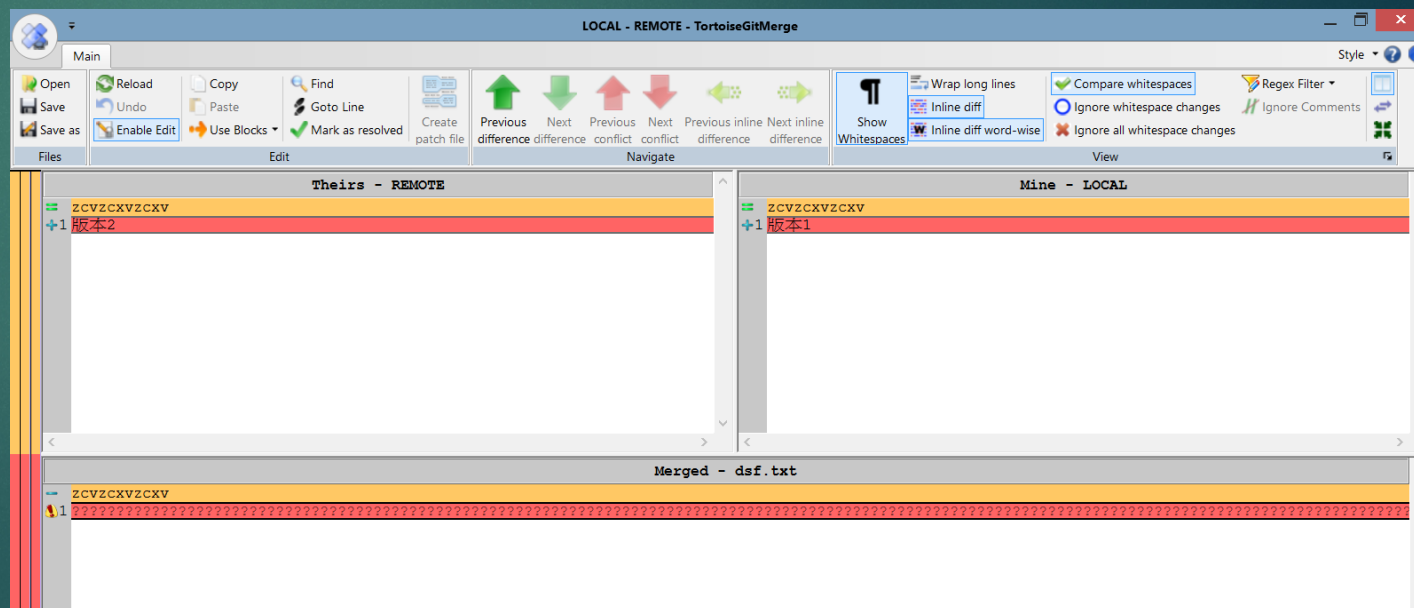
```
<<<<<<< HEAD
版本1
=====
版本2
>>>>>>> dev
|
```


解決衝突

► 建議直接使用工具



R:\test - Working Tree				
Path	Extension	Status	Li...	Li...
dsf.txt	.txt	Conflict	5	1



圖解方式了解 Git 指令

- ▶ <http://marklodato.github.io/visual-git-guide/index-en.html>
- ▶ <http://onlywei.github.io/explain-git-with-d3/#branch>



Best Practise

多久 commit 一次

- ▶ 請記住一個原則
- ▶ 每一個 commit 是 minimum working unit
 - ▶ 意思是最小可運作的單位
 - ▶ 換句話說，只要完成一個小的 task，就 commit
- ▶ 請不要一個 commit 裡面有 3~4 個**不同性質**功能的修改
- ▶ 建立好的 commit 歷史記錄對於未來很有幫助

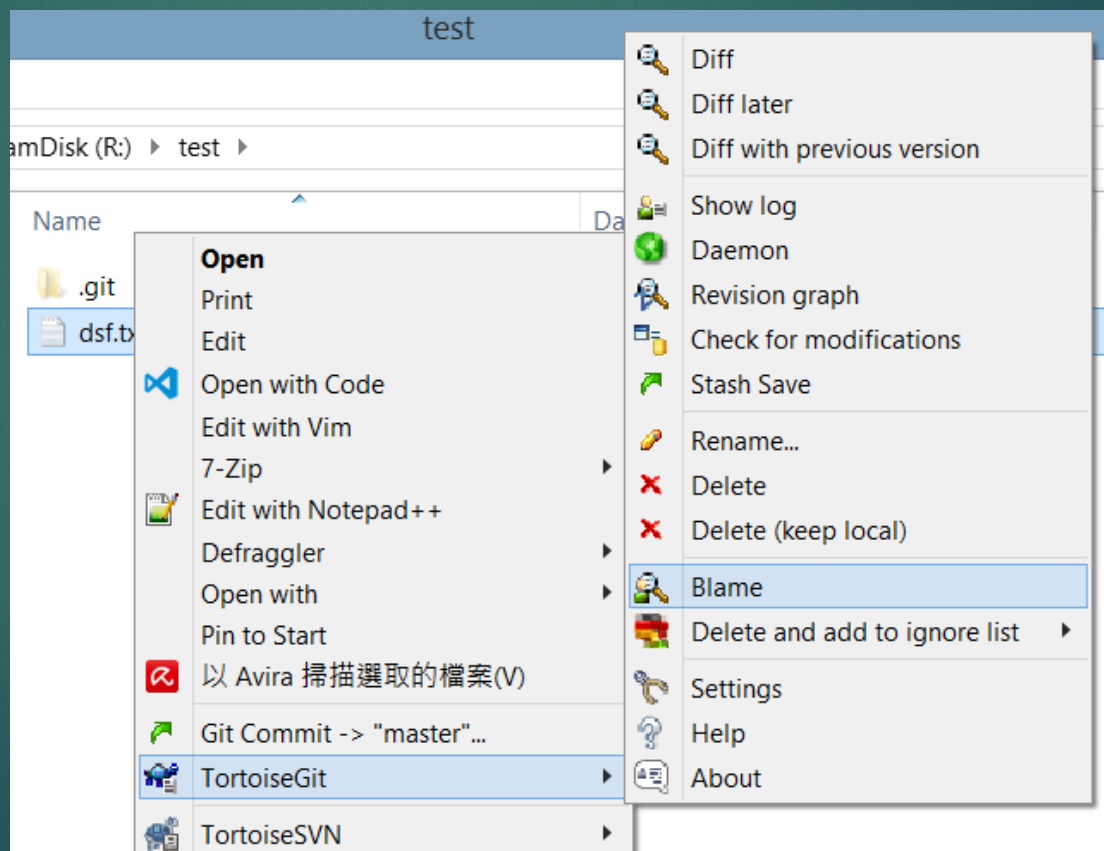
Commit Message 重要性

- ▶ 不管用什麼 CVS，請記住一定要寫
 - ▶ 因為太重要了，因此 git 逼你一定要寫
- ▶ 寫的好在 3 個月回來看你才容易知道什麼時間改了什麼，為什麼改
 - ▶ 在 debug 很有幫助
- ▶ 我個人習慣會在前面用先表示是什麼功能
 - ▶ 例如：[前台][首頁] 左邊 banner 跑版調整

兇手就是你 – git blame 的應用

- ▶ 有時候要針對當一個檔案每一行修改的時間和誰修改

- ▶ 請使用 git blame



快速取得要更新的檔案

- ▶ 有時候更新程式或者要提供別人有改過那些
- ▶ 可以利用比對 log 方式產生出有修改的內容

結語

總結

- ▶ 現在是最容易學習 VCS 的年代
 - ▶ 每一個工程師必備的第一技能
- ▶ Git 入門有門檻
 - ▶ 進去之後，再也不想用其他版控
- ▶ 透過今天對於 git 有深入了解
 - ▶ 針對本機使用不會有問題

下次相關內容

- ▶ 熟悉 local 環境使用 git 之後
 - ▶ 要連到遠端發佈到網上的 server，例如 github、bitbucket 等非常簡單
 - ▶ 只需要在熟悉幾個指令：pull、push、fetch
- ▶ Github 的特色和功能使用
 - ▶ 提供什麼讓他成為最夯的 online repository service
 - ▶ 目前對於學生有提供免費使用 micro account 的等級（一般是 7 美元 / 1 個月）
 - ▶ <https://education.github.com/pack>
- ▶ Git 在上遠端版控之前可以做的事情
 - ▶ 用 rebase 整理自己的版控歷史 – 改寫歷史也不是不可能
- ▶ 歡迎大家課餘時間在弄清楚 git 之後也可以自己研究看看如何和 github 等服務對接

相關資源

- ▶ Git 官網 <https://git-scm.com/>
 - ▶ Pro Git <https://git-scm.com/book/en/v2>
- ▶ Git 指令圖解說明
 - ▶ <http://marklodato.github.io/visual-git-guide/index-en.html>
- ▶ 線上 git 操作
 - ▶ <https://try.github.io/levels/1/challenges/1>
 - ▶ <http://onlywei.github.io/explain-git-with-d3/#branch>
 - ▶ <http://pcottle.github.io/learnGitBranching/>

相關資源 2

- ▶ 連猴子都能夠懂的 Git 入門指南
 - ▶ <http://backlogtool.com/git-guide/tw/>
- ▶ 30 天精通 Git 版本控管
 - ▶ 裡面有進入到 git 的細節
 - ▶ <https://github.com/alantsai/Learn-Git-in-30-days>

大家幫忙填一下問卷

- ▶ 網址：
- ▶ 主要收集一下大家今天的感受
- ▶ 方便對於下一次內容作調整



Q AND A

謝謝大家