

# 第三周 UIView 萬花筒

申潤五 Danny Shen  
shenfive@gmail.com

今天的目標

Swift 的迴圈使用

Swift 的物件導向初步

深入了解 UIViewController

學習使用 Git 作版本控制

更多 UI 元件

Icon 與 登陸頁面

使用 Timer 作多工

# Loop

在許多時候，我們要處理不只一筆資料，而是一串資料，或者更多，顯然一個變數是放不下的，此時我們就會使用陣列 (Array)或其他集合資料來存放。我們會依實際需要，來選擇合適的資料種類。

但處理這類資料，就是同樣的方式重覆做很多次，直到所有的資料處理完成為止，這種動作，叫做迴圈 (Loop)，除了處理資料以外，迴圈也用在許多其他的地方。

Swift 只有兩種 Loop，for in 和 while



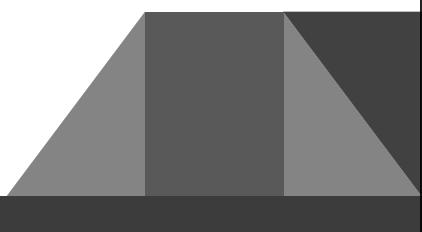
## for in

大多數的狀況，會配合集合物件 Array，Dictionary 等使用

但我們也經常臨時產生一個數字【範圍(Range)】來執行固定次數的

例如 1 ~ 10 的數字範圍來完成

目前的 Swift 版本沒有 for，只有 for in



```
// 常見的用法
var friends = ["Eddy", "Gary", "Jimmy"]
for friend in friends {
    print("Hi, \(friend)!")
}
```

//使用範圍的用法

```
for i in 1...10{
    print(i)
}
```

// 也可以反過來用

```
for i in (1...10).reversed(){
    print(i)
}
```

//更複雜的改用 stride

```
for i in stride(from: 0, to: 50, by: 5){
    print(i)
}
```

## while

while 是只看條件成立就會執行，不會分配變數，也不需配合資料去執行

分為前置條件與後置條件兩種

前置可能一次都不會執行

若是後置條件，就至少會執行一次

```
// 前置範例
var num = 0
while num < 10 {
    print("\(num)")
    num += 1
}

//後置範例
var m = 1
repeat {
    m += m
} while m < 100

print(m)
//印出 128
```

## Class

類別 (Class) 是一個事物的藍圖與工廠，用來設計一個東西的特性/功能與生產實際的物件。

我們之前已經用過 class 了，例如 UIViewController 就是 class

物件導向程式設計中，class 就是程式，而使用 class 產出的東西放在記憶體中，就是 Instance

class 中寫的 function 就叫 方法 (method)

class 中使用的變數，就叫屬性 (property)

首先，我們來了解一下【物件】(Object)。在物件導向的程式開發中，所有的東西都是物件。顯示器是物件，鍵盤是物件，藍芽是物件，影片播放器是物件，連控制這些東西的程式，也是物件。

在程式設計時，我們不會用到【物件】而是用到用來描述物件的程式，叫做【類別】(Class, 類)。類別是我們實際會寫的程式，也就是說，【類別】是【物件】的說明書，是設計圖，用來說明物件的一切細節，其中包括所有的細節，必需無微不至，巨細靡遺。有了類別之後，我們就可以根製類別，建立物件，而建立出來的東西，就叫做【實體】(instance, 實例, 實體)。

【實體】會依據【類別】創造出來，再把它放到記憶體中，也就是變數中，再給予實體一些指令，實體就會依據類別來執行。

相信讀者聽到這兒，多半覺得很抽象，也不易了解，這是一定的，不過沒有關係，物件導向最好的學習方法，就是由實作當中去理解，等寫程式經過一段日子後，再回頭來思考什麼是物件導向，也就是說，如果你腦中卡關不理解，不要在糾纏什麼是物件導向，但我們至少要了解【物件】【類別】【實體】這三者之間的關係，這是物件導向的第一步，我們在寫程式時，都是在撰寫類別，而在撰寫時，經常需要用到體的操作。

類別通常就是程式本身，也就是一種物件的說明書，或者是一份藍圖，用來製作物件的依據。其內容就是要正確的描述我們要建立的物件有那些功能或特性。

## class 的特性

繼承性 (Inheritance) 是指，在某種情況下，一個類別會有「子類別」。子類別比原本的類別（稱為父類別）要更加具體化。這而的【父】也許叫【師父】會比較容易體會，【師父】會的學生基本上都會，是吧？

封裝性 (Encapsulation) 的物件導向程式設計隱藏了某一方的具體執行步驟，取而代之的是通過訊息傳遞機制傳送訊息給它。封裝是通過限制只有特定類別的物件可以存取這一特定類別的成員，而它們通常利用介面實作訊息的傳入傳出。

## 多型

多型 ( Polymorphism ) 是指由繼承而產生的相關的不同的類別，其物件對同一訊息會做出不同的回應

## 抽象性

抽象 ( Abstraction ) 是簡化複雜的現實問題的途徑，它可以為具體問題找到最恰當的類別定義，並且可以在最恰當的繼承級別解釋問題

## Struct/Enum

相對於其他語言，struct 是一個相當強大的資料型式，如同 class 一樣可以有屬性，方法，封裝等特性，但只缺一個繼承的特性。在其他有 struct 的語言中，struct 一般只能拿來組織邏輯相關的 data，使用上沒什麼疑義。除了無法被繼承以外，和 class 的使用方式幾乎完全一樣。之前提過的 enum 其實也和 struct 類似，只是屬性定義的方法略有不同外，和 struct 是完全一樣的東西

事實上，我們之前用的 Int，Double 等，都是一種 struct

但造成 struct/enum 與 class 不同了，除了沒有繼承之外，更重的的是記憶體管理的不同

## Class, Struct and Enum 的比較表

		Copy by	inheritance	static variable	instance variable	static method	instance method
Class	Reference	✓	✓ (class variable)	✓	✓ (class method)	✓	
Struct	Value	✗	✓	✓	✓	✓	
Enum	Value	✗	✓	✗	✓	✓	

## value type 和 reference type

value type 在指定到另一個變數時，是複製完整的資料，使用不同的記憶體

所以在另一個變數改變其屬性時，不會影響到原來的變數

reference type 在指定到另一個變數時，是複製 reference 位置，也就是其實他們是使用同一個記憶體

所以在另一個變數改變其屬性時，原來的變數會跟著改變

```
class refClass{
    var per1 = 0
}

struct valStruct{
    var per1 = 0
}

var someReftype = refClass()
var anotherReftype = someReftype
anotherReftype.per1 = 2
print(someReftype.per1)

var someValtype = valStruct()
var anotherValtype = someReftype
anotherValtype.per1 = 2
print(someValtype.per1)

//更進一步查一下記憶體位置
ObjectIdentifier(someReftype)
ObjectIdentifier(anotherReftype)
```

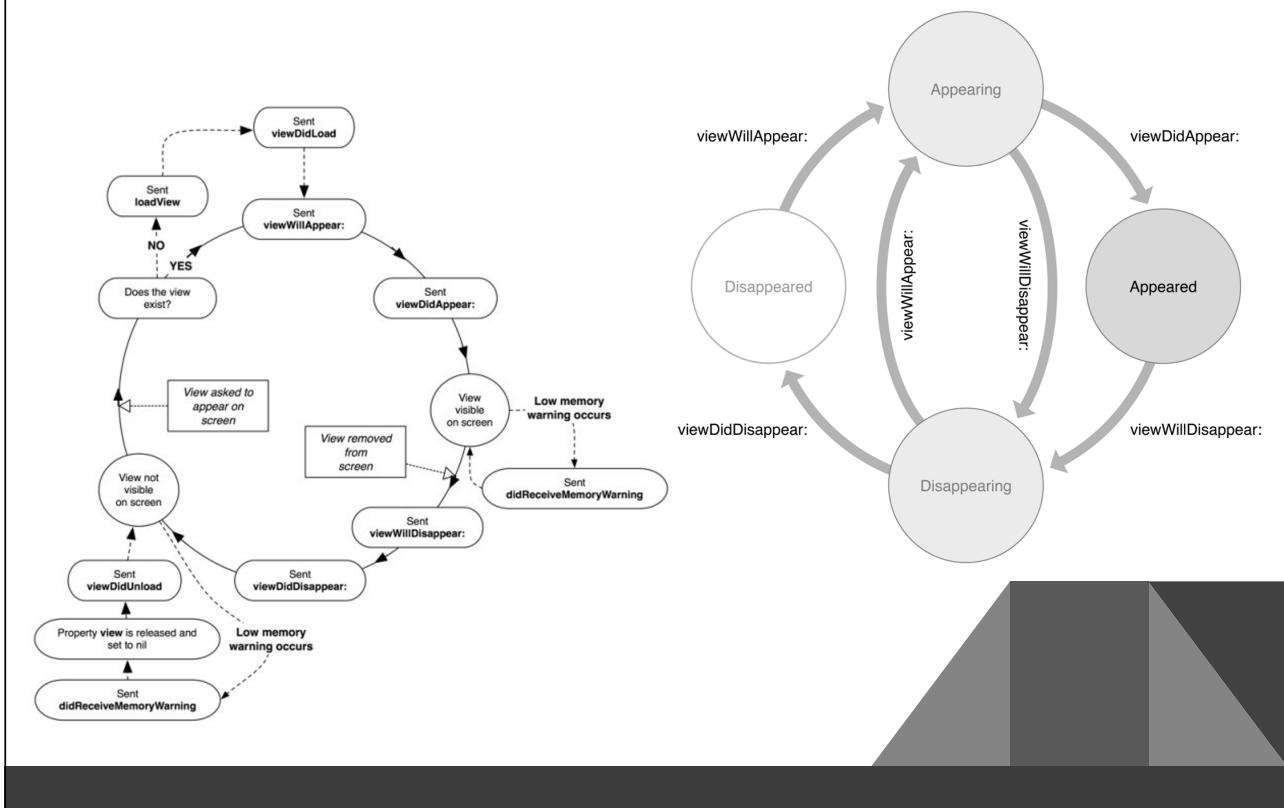
## Apple 說 Swift 是【協定導向】程式語言POP

如果你有開發過其他語言，也許會覺得 Swift 有點怪，但這的確是 Swift 特別之處

<https://developer.apple.com/videos/play/wwdc2015/408/?time=868>

之後我們討論到 協定時，再討論這個問題

# UIViewController 生命周期



ViewController的生命週期  
中各方法執行順序如下：

init—>  
loadView—>  
viewDidLoad—>  
viewWillAppear—>  
viewDidAppear—>  
viewWillDisappear—>  
viewDidDisappear—>  
viewWillUnload->  
viewDidUnload—>  
dealloc

# UIViewController 生命周期事件覆寫

class 的 override 就是改變師父的做法，或改良師父的做法，UIViewController 的生命周期事件，你只能改良，不可以改變，否則就很容易產生錯誤，因為程式庫在各生命周期中已經做了許多事情。

所以 override 後，第一件事，就是呼叫 super (師父) 的相同方法，照做一次，然後再做自己要做的事情

建一個空白 APP，並在第一個 VC 加入以下程式，並加入一些頁面切換的UI，看看有什麼事會發生

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        print("viewDidLoad")
    }
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("viewDidAppear")
    }
    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)
        print("viewDidDisappear")
    }
    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        print("viewWillDisappear")
    }
    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)
        print("viewDidDisappear")
    }
}
```

# Timer

Timer 是 Swift 的一種工具，可以定時執行一些工做  
是最簡單的一種多執行緒程式  
呼叫的片段可以使用選擇器 `#selector` 來執行  
實做一個可暫停的計數器



```
//在 ViewController 加一個 UILabel 拉成 IBOutlet 名稱 timeCount  
//在 ViewController 加一個 Timer 與 計數用變數
```

```
var timeCountNumber = 0  
var timer:Timer!
```

```
//在畫面將出現時建立 Timer
```

```
timer = Timer.scheduledTimer(timeInterval: 1, target: self, selector:  
#selector(setCounter), userInfo: nil, repeats: true)
```

```
//在畫面將見時，停止 Timer  
timer.invalidate()
```

並加上合適的 UI

## 再談 UIView 各項屬性

我們看的到的一切基本上都是 UIView 的子類，所以大多數特性都類似

bounds, frame, center 用來控制位置

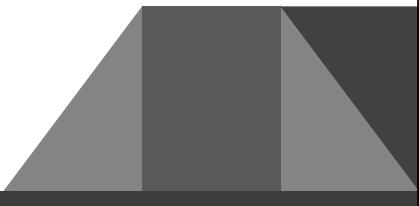
subviews 回傳子 UIView 的陣列

transform 旋轉的位置

alpha,backgorundColor 透明度與背景

isHidden 是否隱藏

layer 渲染層



```
// 定時旋轉程式
class ViewController: UIViewController {
    var counter = 0.0
    var timer:Timer!
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        var newView = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
        newView.center = self.view.center
        newView.backgroundColor = UIColor.red
        newView.transform = CGAffineTransform(rotationAngle: 0)
        view.addSubview(newView)
        timer = Timer.scheduledTimer(withTimeInterval: 0.05, repeats: true, block: { (timer) in
            self.rotateView(targateView: newView)
        })
    }

    @objc func rotateView(targateView:UIView){
        let angle = counter * M_PI / 180
        targateView.transform = CGAffineTransform(rotationAngle: CGFloat(angle))
        counter += 4
        print(counter)
    }
}

//圓角 UIView
newView.layer.cornerRadius = 15
newView.clipsToBounds = true
```

# Segment

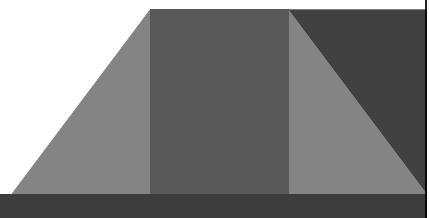
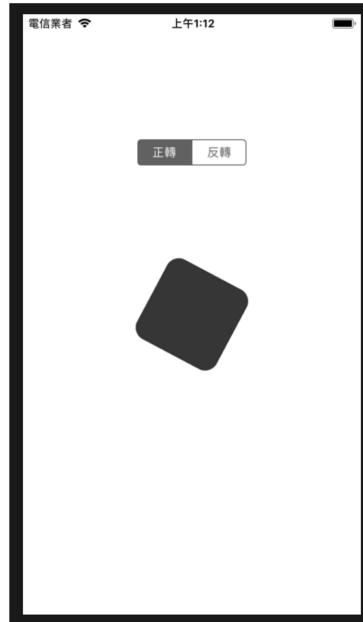
一種常見的選擇器

單選選單，必選選單，必需有預設值

可設 Outlet 與 Action

可設 每個 Section 的各項參數

使用 .selectedSegmentIndex 取得目前選項



//建一個 Segment , 設好UI

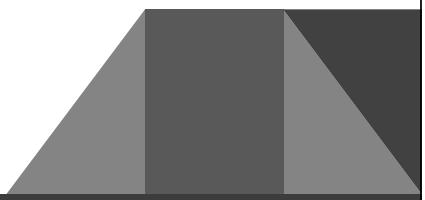
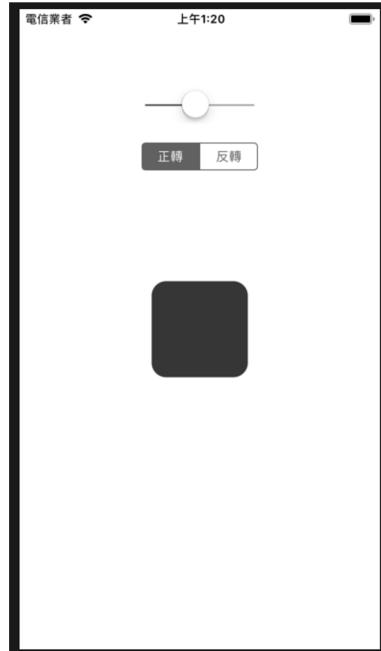
```
@objc func rotateView(targateView:UIView){
    let angle = counter * M_PI / 180
    targateView.transform = CGAffineTransform(rotationAngle:
    CGFloat(angle))
    if directionSegment.selectedSegmentIndex == 0 {
        counter += 4
    }else{
        counter -= 4
    }
}
```

# Slider

數值設定器

必需有最大值/最小值/預設值/目前值

回傳格式為 Float 必要時需型別轉換



//同前例，建立可調速的動畫

```
@objc func rotateView(targetView:UIView){

    let angle = counter * M_PI / 180
    targetView.transform = CGAffineTransform(rotationAngle: CGFloat(angle))

    if directionSegment.selectedSegmentIndex == 0 {
        counter += Double(speedSlider.value)
    }else{
        counter -= Double(speedSlider.value)
    }
}
```

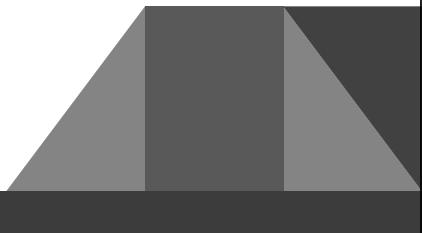
## 其他常用元件

Text Field

Switch

Stepper

Text View



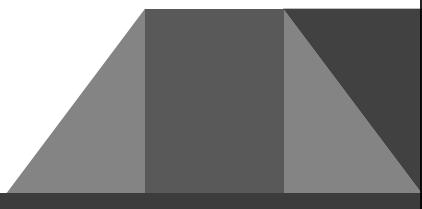
## Git

Git 是目前使用最廣的原始碼管理工具

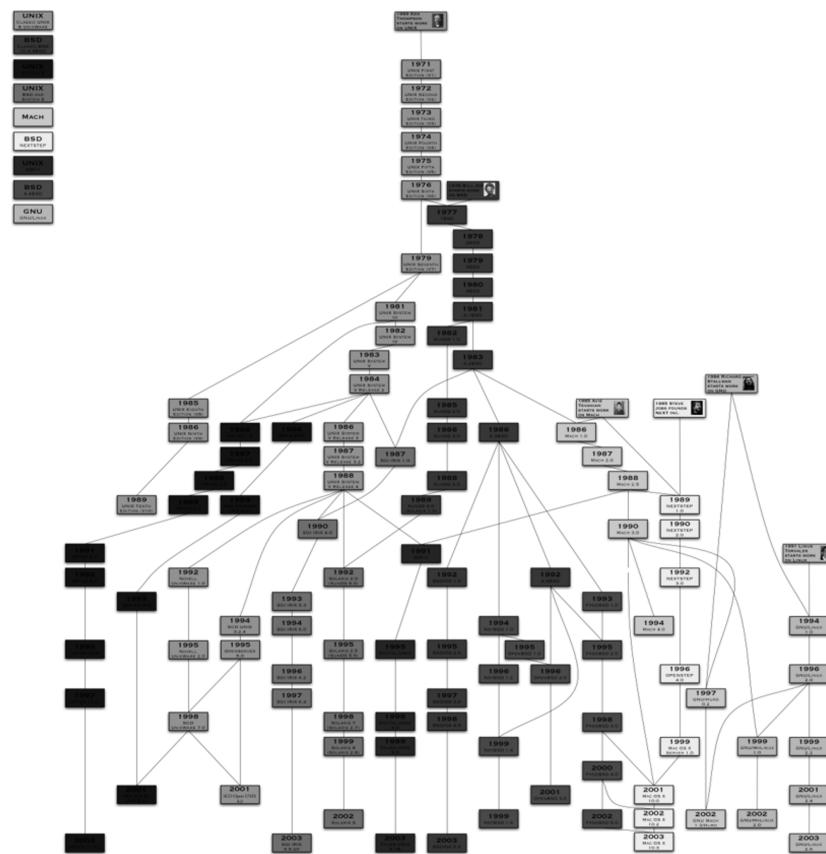
版本控制 ( Revision control ) 是維護工程藍圖的標準作法，能追蹤工程藍圖從誕生一直到定案的過程。此外，版本控制也是一種軟體工程技巧，藉此能在軟體開發的過程中，確保由不同人所編輯的同一程式檔案都得到同步。

若你是一個以工程師為目標的人，或以資訊開發者自居，你在未來十年內應徵開發工作，若面試者是資訊背景主管，第一個面對的題目可能是

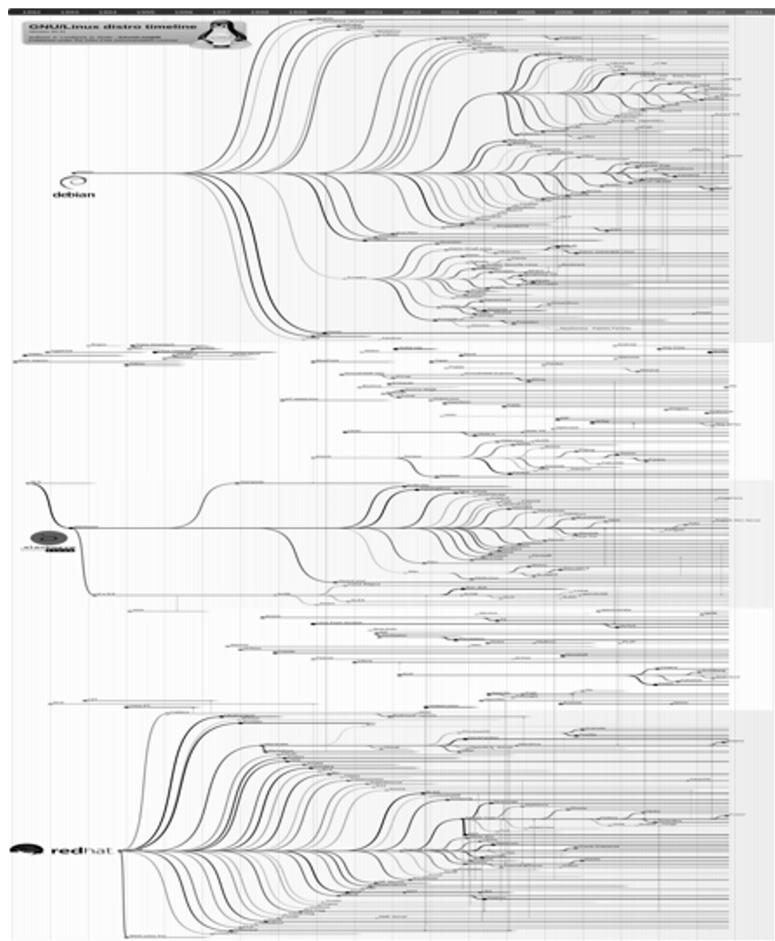
【你在 GitHub 上，有什麼專案？】



## UNIX 的 版本控制



這個樣子



## 為什麼用 Git

分散 & 集中都可用, 即使網路不通, 也可使用, 不一定需要伺服器

世上最多開源碼社群, GitHub 使用的方法

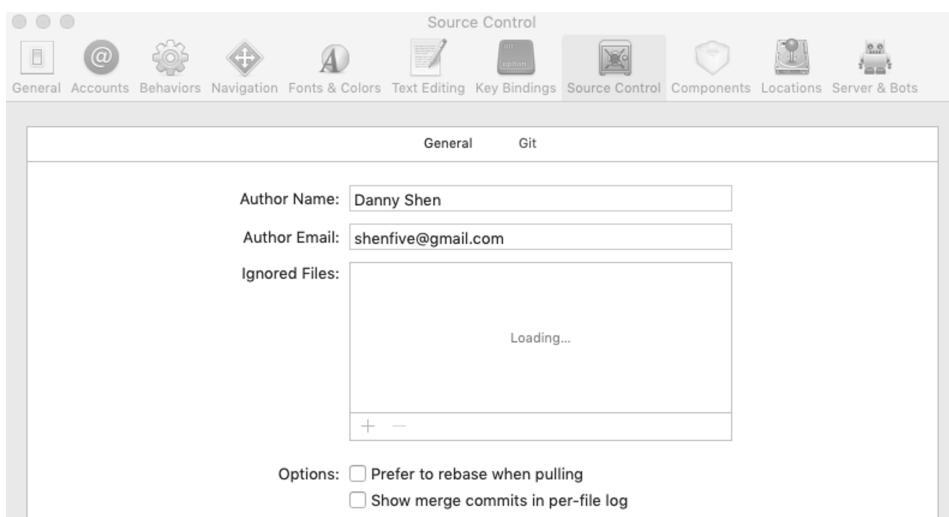
Linux 的創立者(Linus Torvalds), 所創立的方法

目前大多數 IDE 都支援的方法

mac 應都有安裝了, 如果任何原因沒有安裝, 請至 <https://git-scm.com/downloads> 下載安裝

## Xcode 設定

至少要有一個代號, 和一個電子郵件, 這只是記錄, 不是驗證或通知



# Git 入門版

寫到一定的時候，就要 Commit

不管改了什麼，都可以與上次 Commit 作比較

必要時，可以恢復到某次的 Commit

Source Control -> Commit 並按下說明，就可以了

The screenshot shows the Xcode interface with a project named 'CS193P\_L1'. In the left sidebar, there are files like 'CS193P\_L1', 'CS193P\_L1App', 'ContentView', 'Assets', 'Preview Content', 'Preview Assets', and 'NewFile'. The 'NewFile' file is selected and has a status indicator 'M' next to it. The main editor window displays the following Swift code:

```
1 //  
2 // Newfile.swift  
3 // CS193P_L1  
4 //  
5 // Created by 申潤五 on 2022/1/4.  
6 //  
7 import Foundation  
8  
9
```

A red arrow points from the text '有修改過白檔案' (Has modified white document) to the status bar at the top of the Xcode window, which shows 'CS193P\_L1 <- My Mac (Designed for iPad)'.

其實 XCode  
訊息早已告知那些檔案作了什麼修改

# 固定忽略 gitignore

gitignore[檔名]

萬用字元 \*.xxxx

資料匣

不設不會怎樣, 設了非常專業 , 不是每個東西都要版本控制的 ,  
最常用來處理不必要的流量, 如編輯過程中測試用的執行檔之  
類的

<https://github.com/github/gitignore>

接下來.....

應該深入研究 Git 的指令 , 原理 , 方法 , 雲端等.....

但本課程並不是 Git 教程 , 我們就下載一個傻瓜軟體  
SourceTree 來用

至少要學會切換 Header

最好是學會開 Branch

在開發的生涯會陸續補學完 Git 的強大功能

# 有關 APP Icon 的製作

你先要有一堆各式不同的圖檔，Apple 說的

<https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/>

然後放到 Assets 中的 Icon

完成！

先到這，或自己找張圖至少  
512 \* 512  
<https://www.flaticon.com/>

再到這  
<https://appicon.co/>

完成



## 有關 LaunchScreen.storyboard

載入中畫面，幾乎所有的 App 都會看到 半秒到數秒間  
可以載入，設定 Autolayout 等所有程式或 StoryBoard 可做的事

程式載入完成後，會自動消失，所以也不要太麻煩的事  
通常只是作一些畫面自適應或本地化等事情

