

• = = = •

# 嵌入式智慧影像分析與實境界面

## Fall 2018

• = = = •

INSTRUCTOR : YEN-LIN CHEN(陳彥霖), PH.D.

PROFESSOR

DEPT. COMPUTER SCIENCE AND INFORMATION

ENGINEERING

NATIONAL TAIPEI UNIVERSITY OF TECHNOLOGY

• = •

# Lecture 1 – 嵌入式系統開發與實驗平台簡介

• = •

## 本課程討論群組

- 本課程課後討論FB社群：
- <https://www.facebook.com/groups/381460865722688/>

# 本課程作業/報告上傳

- 位址：**北科i學園**
- 請各位同學先將分組的名單上傳至FB社團中，接著我們會將你們分組，你們可以查看你們各自的組別。
- 作業上傳格式，以組別命名，壓縮為**zip**，例如：組別X.zip
- 其zip裡要包含如下資料夾
  - 程式碼 //存放程式碼
  - 報告 //存放報告 (PDF檔)
- 請依照此規則上傳，否則會造成助教改作業的困擾，成績有問題要自行負責
- 若有其他有關此作業的備註訊息要跟TA告知，可在根目錄附上文字檔 (Readme.txt)，並留下你的E-Mail

# 嵌入式處理器簡介

# ARM架構

- 在處理核心當中，以ARM架構最為熱門，然而，ARM架構與一般x86有什麼不一樣？其最主要的差異在於，指令集，Reduced Instruction Set Computing(RISC)以及Complex Instruction Set Computing(CISC)。
  - **指令集**：用來命令處理核心該進行哪些處理或是運算所用，而指令集的差異在於，一道指令可以完成的事情。

## 指令集架構

- 接著，如何區分這些指令集？一個依據為，「暫存器」的使用。因為資料要透過CPU進行運算，需要先存放在暫存器當中，接著讓CPU讀取、運算，而兩個指令集架構最大的差異在那？
  - CISC：一道指令可以執行多種運算處理。
  - RISC：一道指令，執行一種運算處理。
- 那一種指令集比較花費「暫存器」？RISC，為什麼？因為RISC在設計初期，有pipeline的機制讓CPU隨時保持在Busy的狀態，不會等待指令進來。但是pipeline需要事先把指令以及資料都存放在暫存器當中，所以造就了RISC需要較多的暫存器存放資料，但是效能將會較好。

## 處理核心架構問題

- CISC：處理核心設計較為複雜，原因在於一道指令要能進行多種資料處理，需要硬體的支援。
- RISC：處理核心設計較為簡約，因為一個指令僅有一個動作，所以硬體設計並沒有如CISC複雜。

## 穿戴式裝置的抉擇

- 為什麼現今穿戴式裝置普遍使用RISC架構，也就是ARM架構的解決方案？「省電」，為什麼RISC架構會比CISC架構省電？因為「硬體架構」，由於CISC的設計是透過增加硬體的設計來讓CPU可以同時處理多種運算，造就了需要更多的電力來驅動CPU，然而RISC的硬體架構較為簡單，所以需要的電力，相對較低，這也是為什麼RISC普遍被用於行動裝置的主要原因。

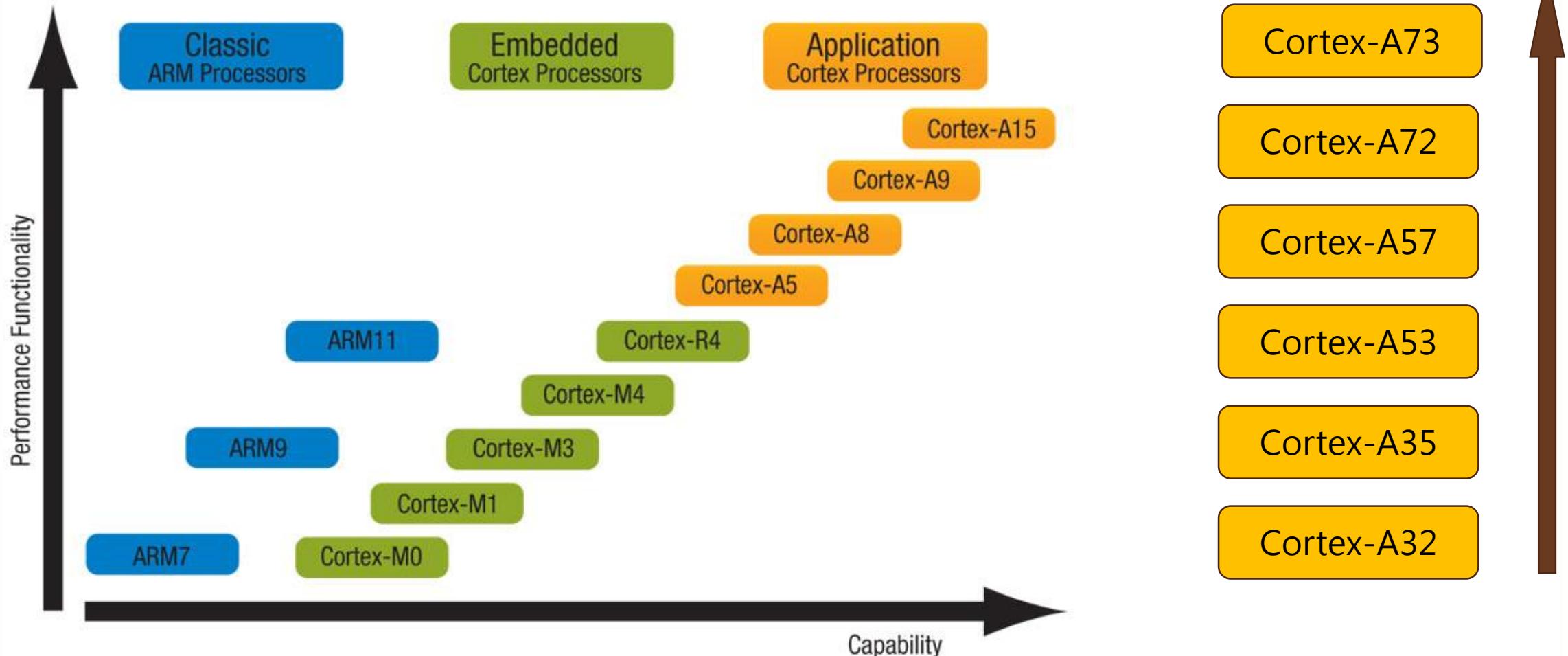
# ARM架構的演進

- 在ARM架構的演進主要分為：
  - Von Neumann Architecture
  - Harvard Architecture
- 這兩個架構當中，主要的區別在記憶體架構，
  - 集中存放
  - Data與Instructions分別存放。
- 效能考量，Harvard Architecture可以進行指令預讀取，進而提升執行效率。

# ARM經營模式

- ARM，現階段為一家提供「CPU」設計架構的公司，提供架構設計作為參考設計，並授權給其他CPU生產公司進行參考、修改。例如：
  - Qualcomm
  - Texas Instruments
  - NVIDIA
- 讓各個公司可以根據這些基礎架構下，建立符合自家產品的定位。例如：
  - Qualcomm：整合Baseband processor
  - Texas Instruments：整合DSP
  - NVIDIA：整合繪圖晶片

# ARM 架構演進



Ref: "Appropriately ARMing the Embedded World"  
<http://www rtcmagazine com/articles/view/101895>

# ARM 架構定位

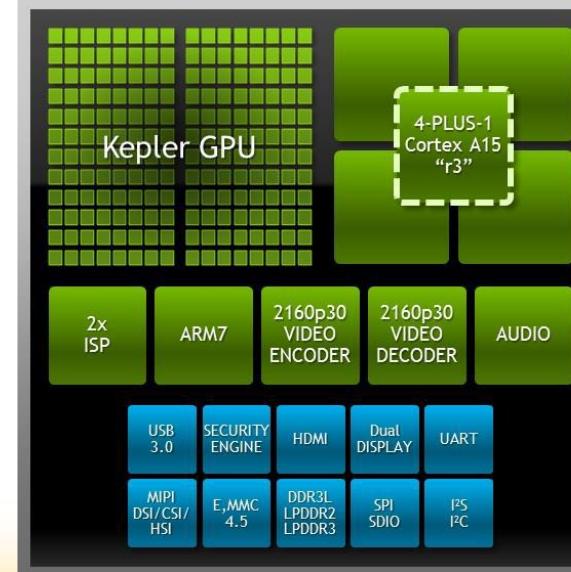
- Cortex-A系列：Application應用，例如，現今常見的手機，大多屬於Cortex-A系列架構。
- Cortex-M系列：Microcontroller應用，常應用於工業控制的用途。
- Cortex-R系列：Real-time應用，例如：硬碟控制、引擎管理、Baseband即時處理器核心...等。

# Jetson TK1平台簡介

# NVIDIA Jetson TK1

## 嵌入式平台

- Tegra K1 SOC
  - NVIDIA Kepler GPU，含 192 個 CUDA 核心 (0.85 GHz)
  - NVIDIA 4-Plus-1™ 四核心 ARM® Cortex-A15 CPU (2.3GHz)
- 2 GB 記憶體 搭配 64 位元寬度
- 16 GB eMMC

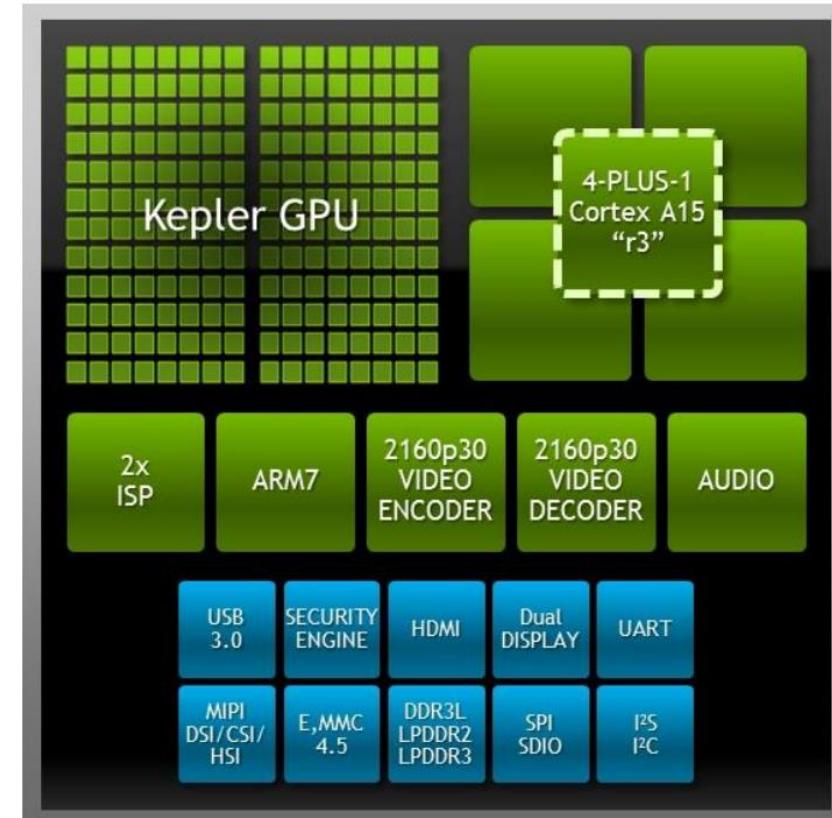


# Tegra K1 Mobile Processor(1/2)

- Processor — Nvidia Tegra K1 (4x Cortex-A15 cores @ up to 2.3GHz with 192-core Mobile Kepler GPU and power management core)
- Memory:
  - 2GB to 4GB x16 RAM (64-bit width)
  - 16GB eMMC 4.51 flash
  - 4MB SPI boot flash
  - SD/MMC slot
- Display/imaging:
  - HDMI port
  - DP and LVDS signals via expansion ports
  - Touch SPI 1×4 via expansion ports
  - 1×1 CSI-2 camera I/O via expansion ports
- Networking — Gigabit Ethernet port (Realtek RTL8111GS)
- Other I/O:
  - USB 3.0 port
  - Micro-USB 2.0 (OTG)
  - RS232 port
  - SATA port (type A)
  - JTAG connector
  - Audio line-out, mic-in (Realtek ALC5639)
  - GPIOs, UARTs, HSIC, and I2C via expansion ports
- Expansion — half-sized mini-PCIe slot; 2x expansion connectors (2mm pitch, 2×25 female )
- Dimensions — 5.0 x 5.0 inches (127 x 127mm)
- Power — 12V DC
- Operating system — Linux with 3.10.24 kernel image, bootloader, Nvidia drivers, and flashing utilities; requires host PC running Ubuntu 9.04 or higher

# Tegra K1 Mobile Processor(2/2)

- Applications include:
  - Automotive Infotainment
    - In-dash navigation
  - Consumer
    - PND
    - PMP
    - Digital Video Camera
  - Medical
    - Patient monitoring
    - Portable ultrasound
  - Industrial
    - Point of sale
  - Embedded applications
    - Smart white goods



# Do you really know the difference?

- <Question> What is the difference between different operation system?
  - <Answer>
    - High-level OS:
      - Win Mobile: rich driver support and applications but expensive
      - WinCE: rich driver support and applications but expensive
      - Linux: free of charge and rich driver support but not well integrated
    - Low-level OS
      - RTOS: small and fast response but limited driver support
      - MS .NET MF: free of charge and well integrated for Windows OS
    - Non OS
      - C code: small and fast response and easy to development
    - Others
      - DSP/BIOS: TI developed simple OS specific for DSP only
      - CSL: TI developed drivers for DSP only

# 於VM上架設Linux

# 於VMWare上安裝Ubuntu 14.04 LTS

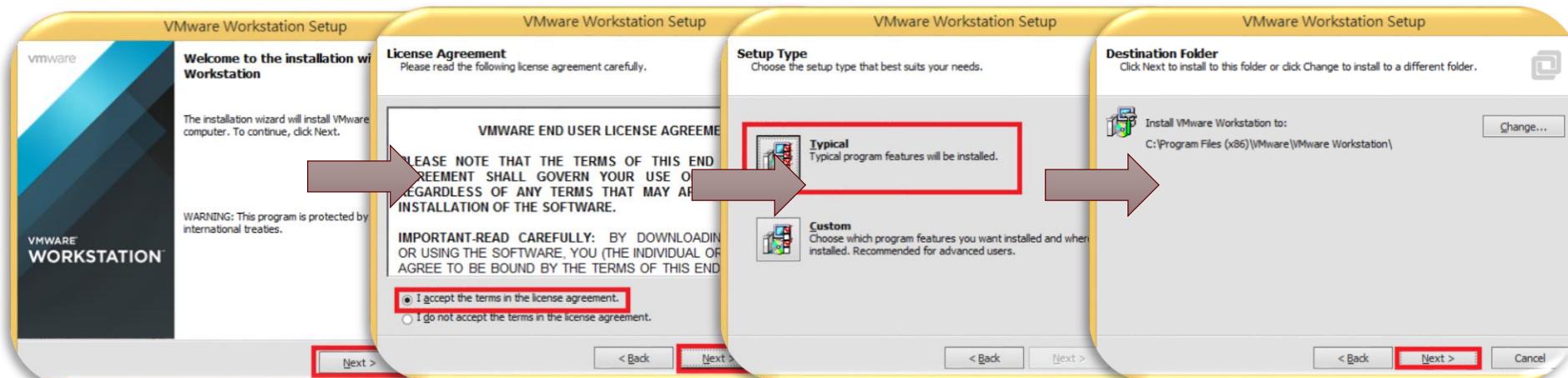
- 為了能夠在Windows下同時使用Linux及Windows，故必須利用VMWare建立Virtual Machine，並在Virtual Machine上安裝Linux，以利往後建立嵌入式系統開發環境。
  - 介紹如何透過VMWare建立Virtual Machine及安裝Linux (Ubuntu)。

# 於VMWare上安裝Ubuntu 14.04 LTS

- 安裝Ubuntu 14.04 LTS 64-bit 於 VMWare
  - 下載**VMWare Workstation**並安裝
  - 下載**Ubuntu 14.04 LTS 64-bit**並安裝在VMWare上
    - 網址:  
<http://ftp.ubuntu-tw.org/mirror/ubuntu-releases/14.04.5/ubuntu-14.04.5-desktop-amd64.iso>

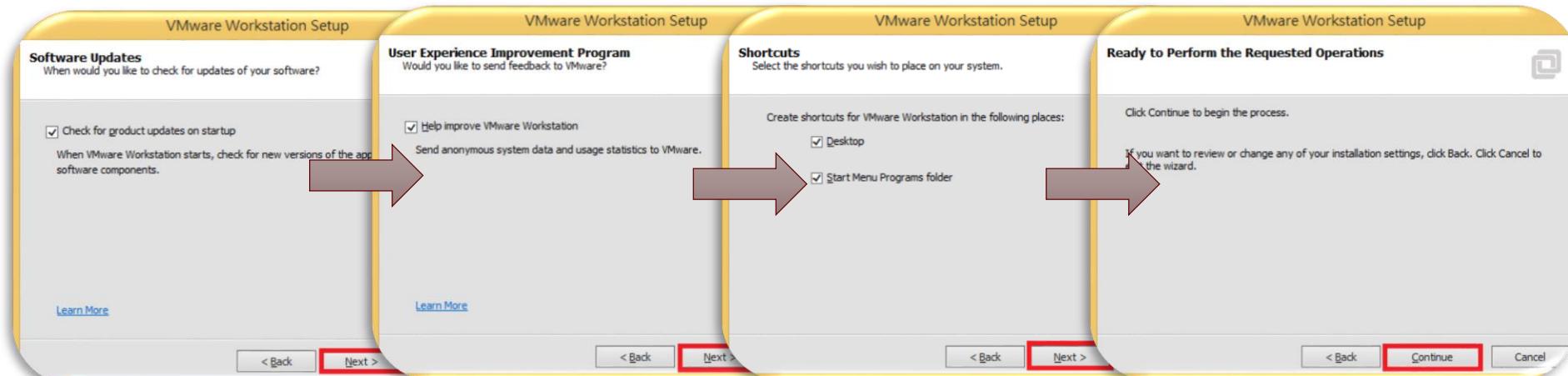
# 環境建立-安裝VMWare

- 安裝 VMWare
  - 基本上，建議都照預設值安裝



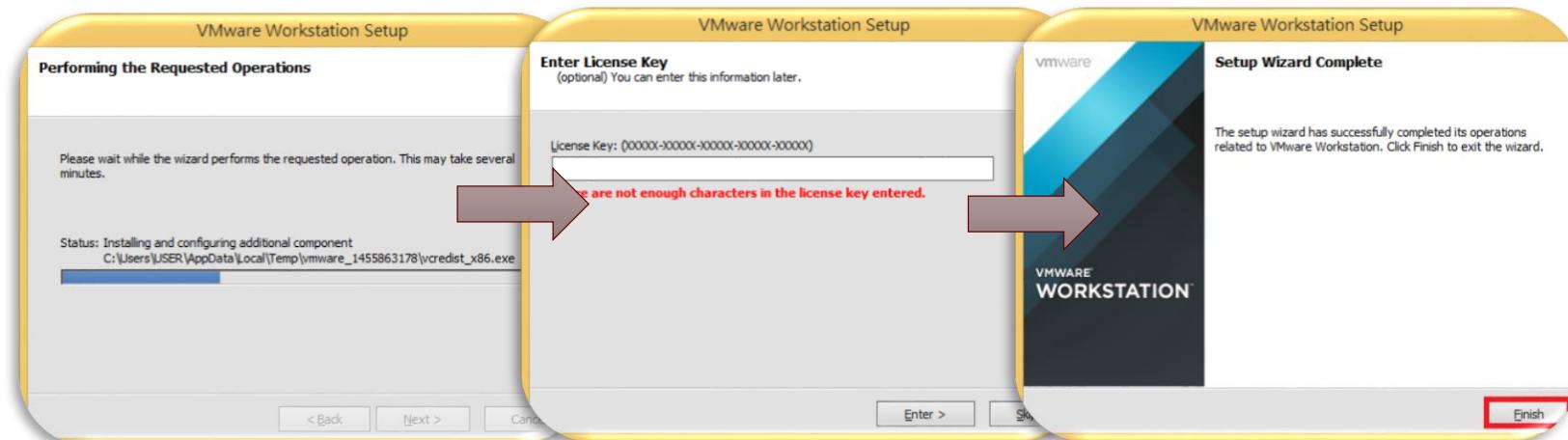
# 環境建立-安裝VMWare

- 安裝 VMWare
  - 基本上，建議都照預設值安裝



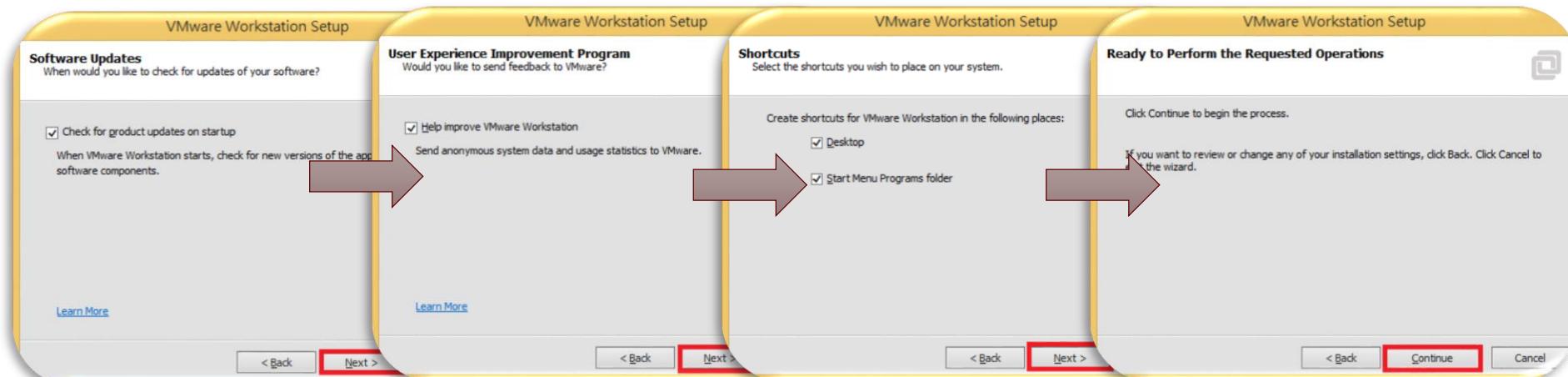
# 環境建立-安裝VMWare

- 安裝 VMWare
  - 基本上，建議都照預設值安裝



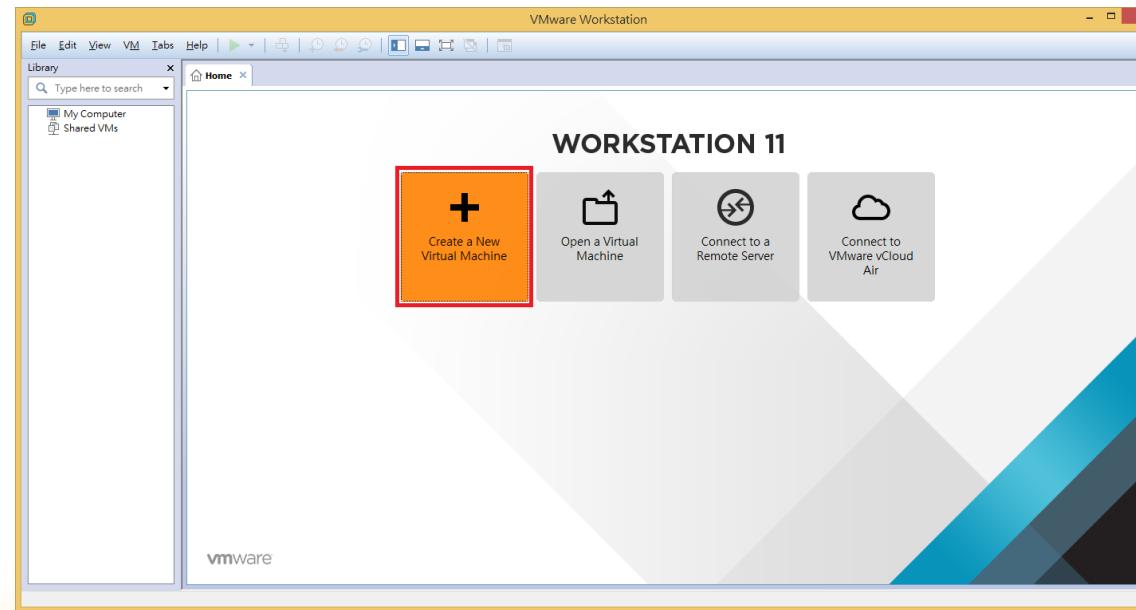
# 環境建立-安裝VMWare

- 安裝 VMWare
  - 基本上，建議都照預設值安裝



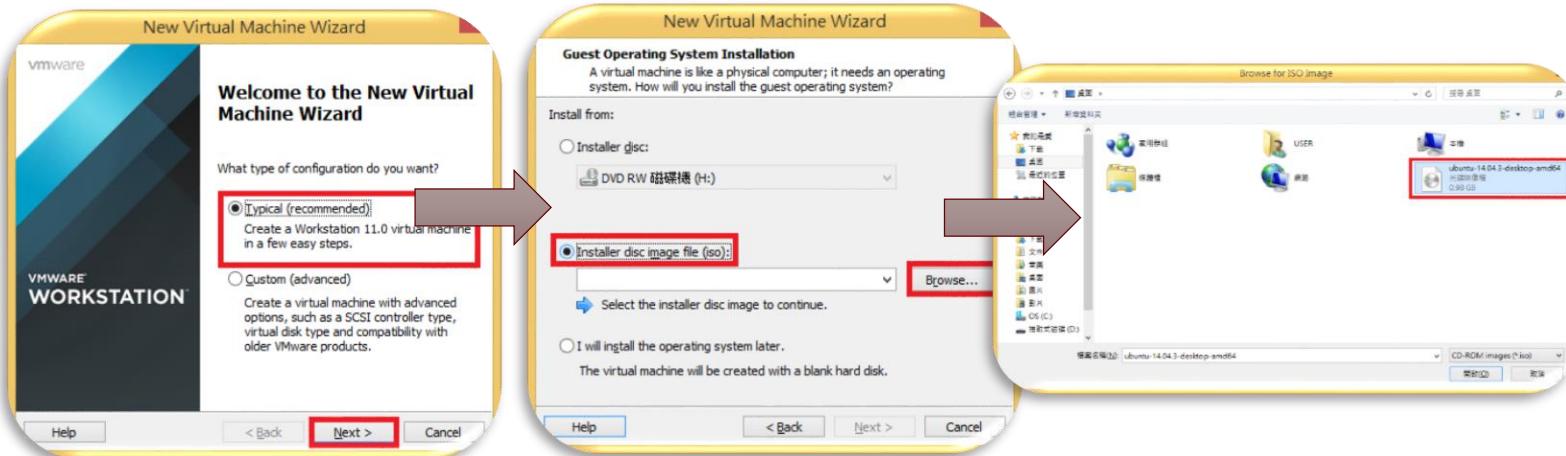
# 環境建立-安裝Ubuntu 14.04 LTS

- 新增模擬機器
  - 作業系統 : Linux
  - 版本 : Ubuntu-64bit



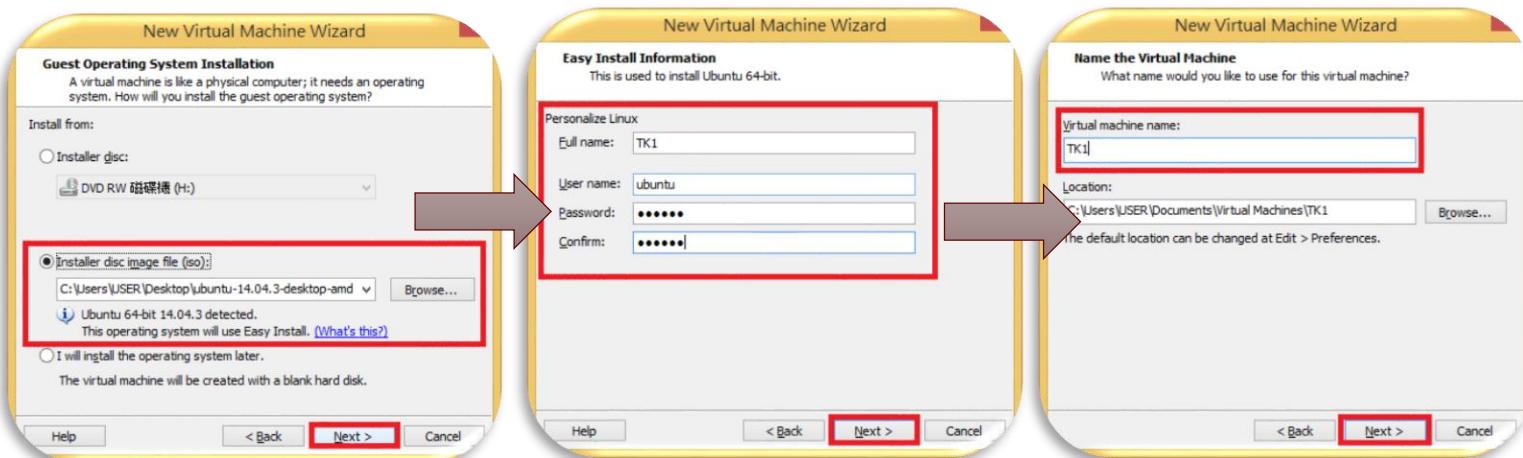
# 環境建立-安裝Ubuntu 14.04 LTS

- 載入剛下載的**Ubuntu 14.04 iso**檔



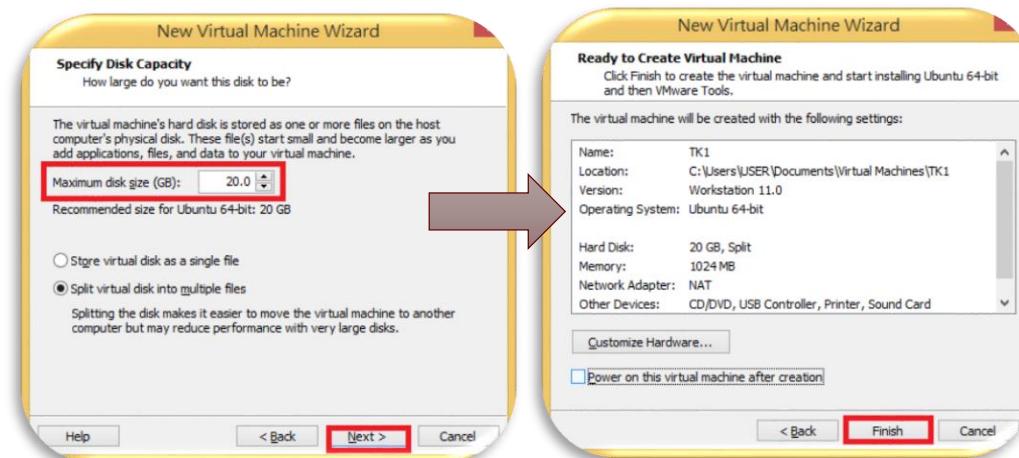
# 環境建立-安裝Ubuntu 14.04 LTS

- 輸入資訊
  - User Name: ubuntu
  - Password: ubuntu



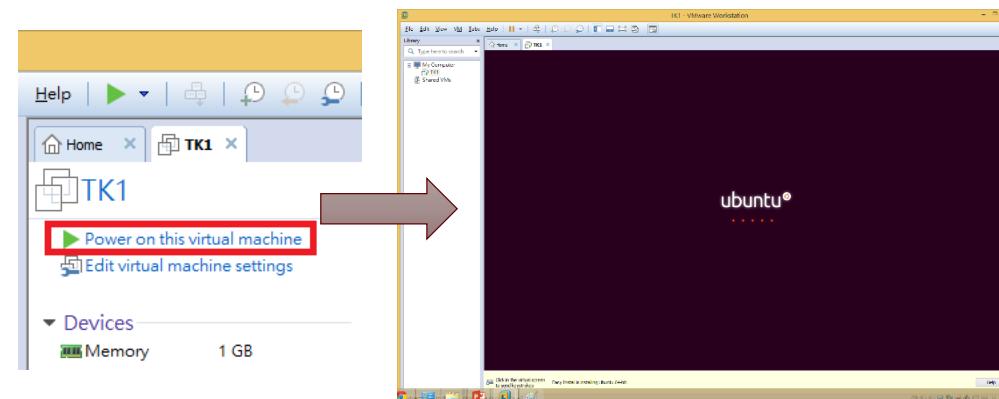
# 環境建立-安裝Ubuntu 14.04 LTS

- 選擇disk size
  - 推薦60GB，如硬碟容量夠的話，可以自行增加



# 環境建立-安裝Ubuntu 14.04 LTS

- VMware設定
  - 直接啟動剛剛所建立好的虛擬機
  - 等待虛擬機的初始設定完成



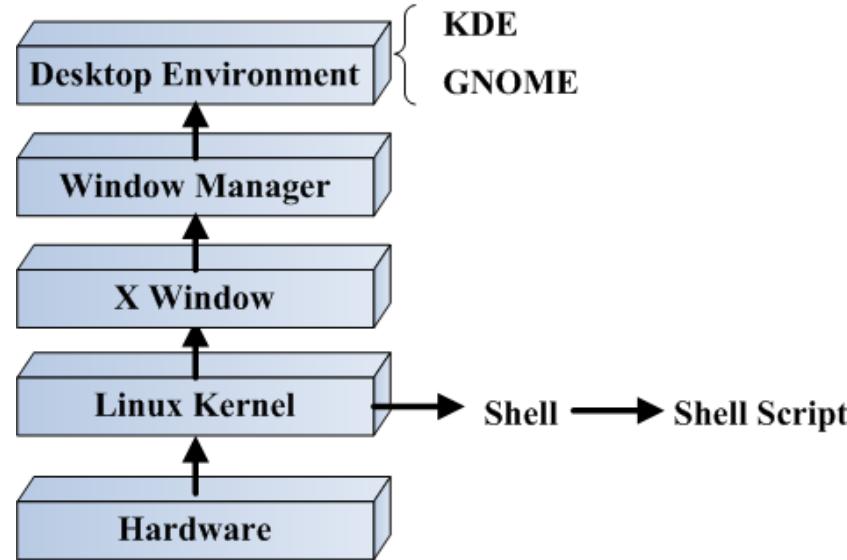
# 可以使用課程提供的虛擬機

- 為了方便學生在家可以自行操作課程內容，我們會提供已經配置好課程相關內容的VMWare虛擬機系統於課程網站可以下載使用，由於VMWare WorkStation為付費軟體所以課程不提供此虛擬機軟體的下載與序號，學生可以使用免費版本的VMWare Player來搭配課程網站提供的虛擬機環境使用

# Linux作業系統簡介

# Linux架構 (1)

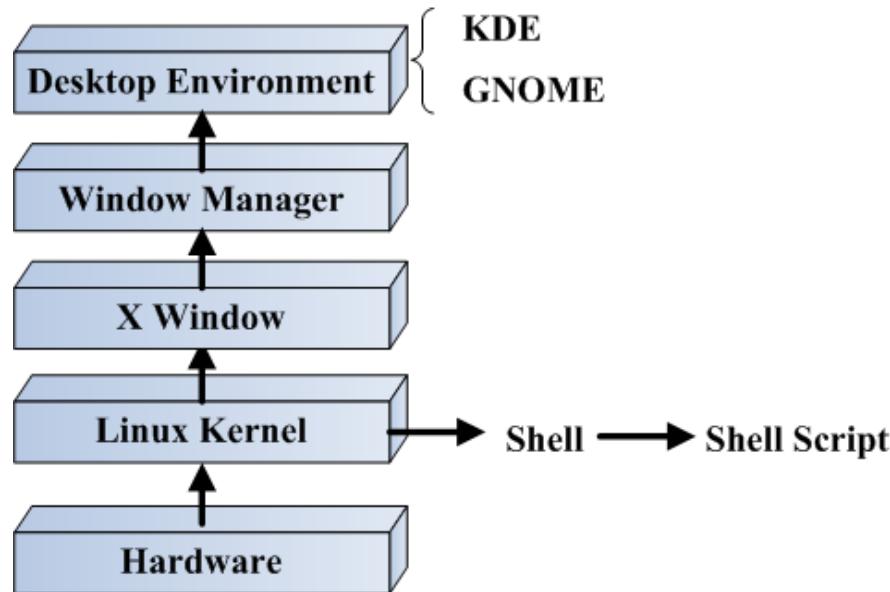
- Linux架構：



- **Hardware**：硬體設備
- **Linux Kernel**：用於控制硬體、管理程序、排程、檔案系統、I/O等等。版本編號：偶數為穩定版(正式釋出)，奇數為開發版(測試中)

## Linux架構 (2)

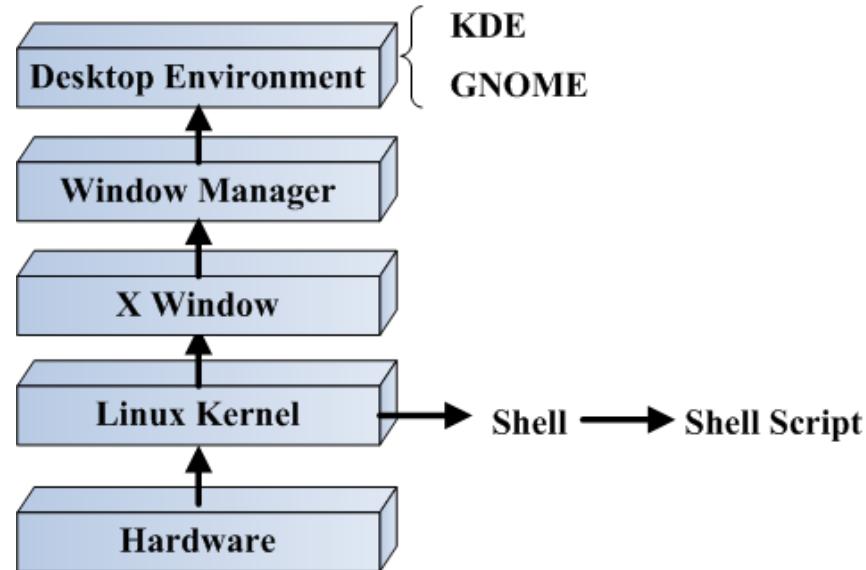
- Linux架構：



- **Shell**：讓使用者透過命令操作系統應用程式
- **Shell Script**：將各種命令操作寫成一個腳本，以利使用

# Linux架構 (3)

- Linux架構：



- X Window : OS上的應用程式，提供實現GUI基本功能
- Window Manager : 管理應用X Window的GUI程式
- Desktop Environment : 分為 : GNOME以及KDE

# 檔案系統 (1)

- 檔案系統：



A screenshot of a terminal window titled "mylinux@mylinux-desktop: /". The window has a menu bar with Chinese labels: 檔案(F) 編輯(E) 檢視(V) 終端機(T) 求助(H). The terminal prompt shows the user has run "cd /" followed by "ls". The output lists several directories: bin, dev, initrd.img, media, proc, selinux, tmp, vmlinuz, boot, etc, lib, mnt, root, srv, usr, cdrom, home, lost+found, opt, sbin, sys, and var. The "tmp" directory is highlighted with a green background.

```
mylinux@mylinux-desktop:~$ cd /
mylinux@mylinux-desktop:/$ ls
bin  dev  initrd.img  media  proc  selinux  tmp  vmlinuz
boot  etc  lib  mnt  root  srv  usr
cdrom  home  lost+found  opt  sbin  sys  var
```

- `cd /`: 代表移動到根目錄底下，`ls`：列出現在位址有哪些資料，在此將列出較重要之目錄進行說明。
  - `boot`：存放開機引導(boot loader)的檔案與開機選單(通常為grub)。
  - `bin`：重要系統程式執行檔存放地點。
  - `sbin`：存放系統管理員，會用到的指令。
  - `dev`：存放裝置與週邊設備。
  - `proc`：虛擬檔案系統，存放於記憶體中，不佔空間

## 檔案系統 (2)

- 檔案系統：



A screenshot of a terminal window titled "mylinux@mylinux-desktop: /". The window has a menu bar with Chinese labels: 檔案(E), 編輯(E), 檢視(V), 終端機(T), and 求助(H). The terminal prompt shows the user is at the root directory (~\$). The user then runs the command "ls" to list the contents of the root directory. The output shows the following files and directories:

```
mylinux@mylinux-desktop:~$ cd /
mylinux@mylinux-desktop:/$
bin dev initrd.img media proc selinux tmp vmlinuz
boot etc lib mnt root srv usr
cdrom home lost+found opt sbin sys var
```

- tmp : 放暫存資料用
- etc : 系統主要設定檔存放位址。
- lib : 系統使用到的函式庫存放位址
- root : 系統管理員(root)的目錄
- mnt : 掛載設備用，例如：CD/DVD或USB
- media : 掛載設備用，例如：CD/DVD或USB

# 檔案系統 (3)

- 檔案系統：



A screenshot of a terminal window titled "mylinux@mylinux-desktop: /". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with Chinese labels: 檔案(E), 編輯(E), 檢視(V), 終端機(T), and 求助(H). The main area of the terminal shows the command "ls" being run at the root prompt. The output lists several directories and files, with "tmp" highlighted in green. The list includes: bin, dev, initrd.img, media, proc, selinux, tmp, vmlinuz, boot, etc, lib, mnt, root, srv, usr, cdrom, home, lost+found, opt, sbin, sys, and var.

```
mylinux@mylinux-desktop:~$ cd /
mylinux@mylinux-desktop:/$ ls
bin    dev    initrd.img   media   proc   selinux  tmp   vmlinuz
boot   etc    lib        mnt     root   srv      usr
cdrom  home   lost+found  opt     sbin   sys      var
```

- `srv` : 服務啟動後，所需之資料存放位址。
- `usr` : 包含系統主要程式、圖形介面所需之檔案、函式庫等等。
- `home` : 存放使用者家目錄，此目錄下會有與使用者帳號同名的資料夾。
- `opt` : 額外安裝軟體所存放之位址。
- `var` : 這是系統在工作時，預設的工作目錄，例如這架主機使用者的登錄檔案資訊、尚未寄出的郵件存放地、接收的郵件放置處等等。

## 檔案系統 (4)

- 檔案系統：



A screenshot of a Linux terminal window titled "mylinux@mylinux-desktop: /". The window has a menu bar with Chinese labels: 檔案(E)、編輯(E)、檢視(V)、終端機(I)、求助(H). Below the menu is a command-line interface. The user has run the "cd /" command followed by "ls". The output shows the following directory structure:

```
mylinux@mylinux-desktop:~$ cd /
mylinux@mylinux-desktop:/$
bin    dev  initrd.img  media  proc  selinux  tmp  vmlinuz
boot   etc   lib       mnt    root  srv     usr
cdrom  home  lost+found  opt    sbin  sys     var
```

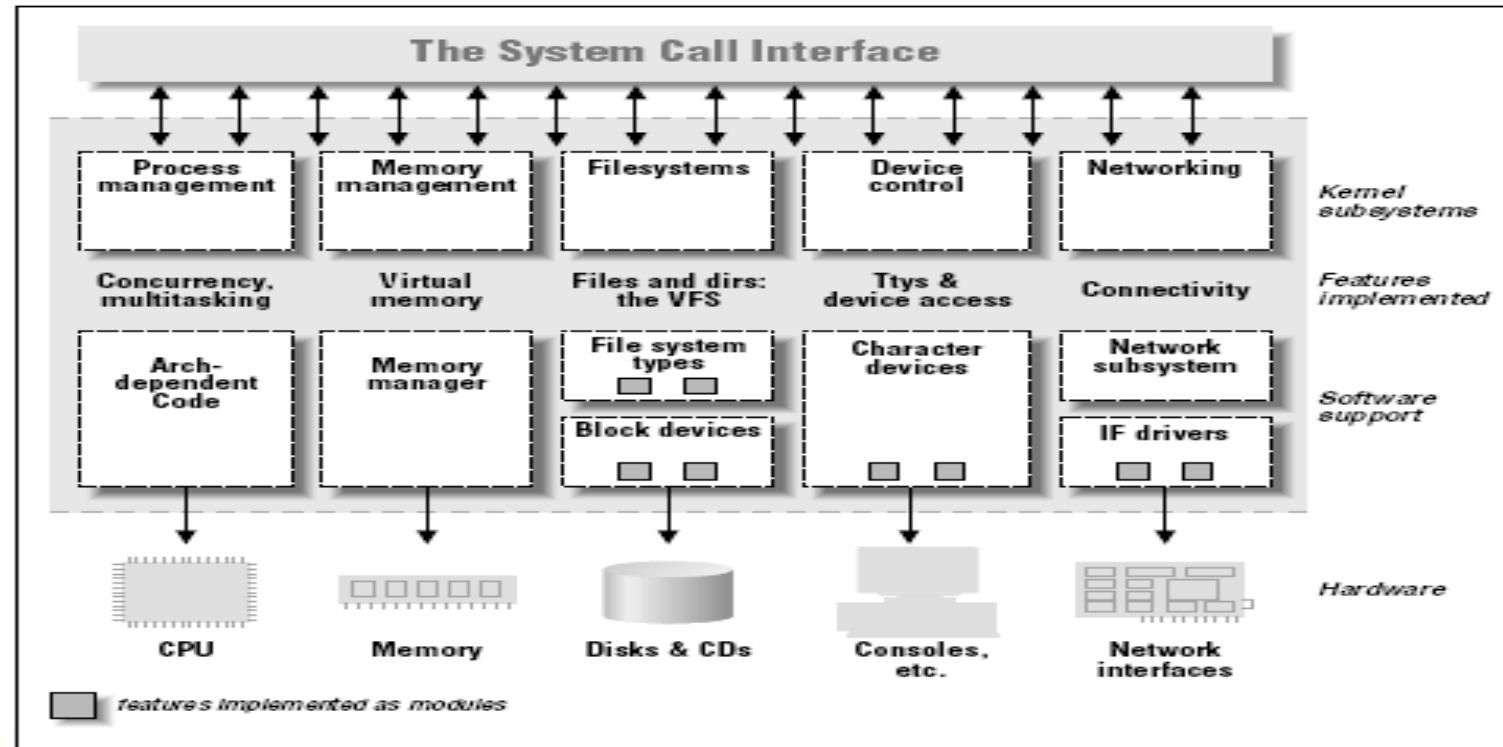
- sys : 功能與proc類似。
- lost+found : 使用ext2/ext3/ext4檔案格式所出現的目錄，用於檔案系統發生錯誤時，存放遺失的檔案。

# The Linux Kernel

- Task management
- System calls
- Memory management
- Scheduler
- IPC
- File system
- Device driver
- Network stack
- SMP support

# Linux Kernel architecture

- Linux Kernel architecture of a generic Linux system



# Execution mode

- User mode
- Kernel mode
- System interface



# From User to Kernel

- User level
  - User Program
  - Compiling
  - Linking
  - Execution
- Kernel
  - Loading program
  - System call
  - Device driver

# Linux-基本指令介紹

- 在這節，將簡單介紹比較常用的Linux Command，在開啟終端機，輸入指令即可得到相對應的功能。
- 介紹Linux檔案權限管理，以了解各種檔案與資料夾對於每個使用者皆有各個不一樣的使用權限。

# Linux-基本指令介紹

- man : 透過man可查詢指令用法與用途，按q離開
  - ls : 顯示檔案資訊以及資料夾
    - -l : 此參數顯示詳細資訊
    - -a : 此參數顯示資訊，包含隱藏檔案
    - -h : 此參數將檔案容量以人類較易讀的方式(例如 GB, MB, KB 等等)列出來
  - cd : 移動到資料夾
    - cd ~ : 移動到家目錄
    - cd .. : 移動到上一個目錄
    - cd <name> : 移動到<name>資料夾
  - cp : 複製檔案(cp 欲複製之檔案名稱 輸出地點)
    - -r : 此參數遞迴持續複製，用於目錄的複製行為
  - rm : 刪除檔案的指令 (rm <檔案名稱>)
    - -f : 此參數是 force 的意思，忽略不存在的檔案，不會出現警告訊息；
    - -r : 此參數為遞迴刪除啊，最常用在目錄的刪除，這是非常危險的選項！
  - mv : 移動檔案到指定地點、修改檔案名稱(mv <檔案名稱> <地點>)
  - mkdir : 建立新資料夾
  - rmdir : 刪除空的資料夾

# Linux-基本指令介紹

- df : 回報各個掛載之檔案系統使用情況
- ln : 建立連結給其他檔案(ln -s <原始檔案或位置> <捷徑名稱>)
- shutdown : 關機
  - -h : 此參數為關機
  - -r : 此參數為重新開機
- reboot : 重新開機
- pwd : 回報目前所在的目錄位址
- who : 查看有哪些使用者在此系統上
- mount : 掛載裝置至檔案系統上
- umount : 從檔案系統上卸載裝置
- uname : 顯示系統資訊(uname -a)
- passwd : 改變密碼
- login, logout : 登入登出系統
- startx : start X window

# Linux-基本指令介紹

- chon : 更換檔案或目錄的擁有使用者與擁有群組
  - chown <使用者>:<群組> <檔案或目錄>
- chmod : 更換檔案或目錄的存取權限
  - chmod ug+rx <目錄或檔案> (增加目錄或檔案的使用者與群組的讀取與執行權限)
  - chmod 755 <目錄或檔案> (將目錄或檔案的權限改為755)

\*權限內容於後面 Linux-權限管理 章節會再進一步說明

- mount : attach the device to the filesystem  
*<ex> mkdir /c (first make directory)  
          mount /dev/hdc /c*
- umount : detach the device from the filesystem  
*<ex> umount /dev/hdc /c*
- uname : 顯示系統資訊 *<ex> uname -a*
- cat : 查看ASCII文件內容 *<ex> cat a.txt*

# Linux-Network

- ping : requests packet echos from network hosts  
    <ex> ping 140.124.182.5
- telnet : to access remote computers  
    <ex> telnet 140.124.182.5
- ssh : to access remote computers  
    <ex> <username>@<addresss(ip/domain name)>
- ifconfig : display the state of network interfaces setup network
- netstat : display network state
- ftp : ftp tool <ex> ftp 140.124.182.5
  - bin : binary mode of ftp
  - get filename : download file
  - put filename : upload file
  - bye : finish ftp
  - ls : list file and directory

# Linux-Search

- `find` : `find [pathnames] [conditions]`
  - An extremely useful command for finding particular groups of files.
- `grep` : `grep [options] pattern [files]`
  - Search one or more files for lines that match a regular expression pattern.

# Linux-Resource manage

- ps : ps [options]
  - -a : list all processes on a terminal
  - -u : display processes for the specified users
  - -x : display processes without an associated terminal

```
miraculous@miraculous-desktop:~$ ps -aux
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
USER      PID %CPU %MEM   VSZ   RSS TTY      STAT START   TIME COMMAND
root       1  0.0  0.2  2528  1488 ?        Ss  Mar21   0:01 /sbin/init
root       2  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kthreadd]
root       3  0.0  0.0     0    0 ?        S<  Mar21   0:00 [migration/0]
root       4  0.0  0.0     0    0 ?        S<  Mar21   0:00 [ksoftirqd/0]
root       5  0.0  0.0     0    0 ?        S<  Mar21   0:00 [watchdog/0]
root       6  0.0  0.0     0    0 ?        S<  Mar21   0:00 [events/0]
root       7  0.0  0.0     0    0 ?        S<  Mar21   0:00 [cpuset]
root       8  0.0  0.0     0    0 ?        S<  Mar21   0:00 [khelper]
root       9  0.0  0.0     0    0 ?        S<  Mar21   0:00 [netns]
root      10  0.0  0.0     0    0 ?        S<  Mar21   0:00 [async/mgr]
root      11  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kintegrityd/0]
root      12  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kblockd/0]
root      13  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kacpid]
root      14  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kacpi_notify]
root      15  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kacpi_hotplug]
root      16  0.1  0.0     0    0 ?        S<  Mar21   1:57 [ata/0]
root      17  0.0  0.0     0    0 ?        S<  Mar21   0:00 [ata_aux]
root      18  0.0  0.0     0    0 ?        S<  Mar21   0:00 [ksuspend_usbd]
root      19  0.0  0.0     0    0 ?        S<  Mar21   0:00 [khubd]
root      20  0.0  0.0     0    0 ?        S<  Mar21   0:00 [kseriod]
```

- Kill : kill [options] [pids | commands]
  - Send a signal to terminate one or more process IDs.

# Linux-權限管理

- 檔案權限：
  - Read : 讀取
  - Write : 寫入
  - Execute : 執行
- 使用者分類：
  - 系統管理員(root) : 為最上層管理者，可取得一切資訊
  - 使用者(user) : 單一使用者所建立之帳號
  - 群組(group) : 若很多使用者，可歸類為一個群組，則可讓user同時加入一個group，並針對該group開啟權限。
  - 其他(other) : 其他拜訪者。

# 檔案權限 (1)

- 在終端機下輸入 `ls -al` 列出該路徑下所有檔案之屬性內容，其格式輸出將會呈現：

目錄 檔案 連結	r : 代表讀取權限 w : 代表寫入權限 x : 代表執行權限 - : 無此權限	[ ]	rwX	rwX	rwX	n	root	root	4096	yyyy-mm-dd hh:mm	檔名
			擁有者 的權限	同群組 的權限	其他人 的權限	連結數	檔案 擁有者	檔案 所屬 群組	檔案 容量	檔案最後修改時間	

- [d] : 目錄
- [-] : 檔案
- [l] : 連結檔
- [b] : 可供儲存的週邊設備
- [c] : 序列埠設備，如鍵盤、滑鼠等

## 檔案權限 (2)

- 在執行上述指令後，以下是在根目錄下所得到的結果，在bin中，為`drwxr-xr-x`，代表他為一個資料夾，而擁有人的權限為可讀、可寫、可執行，同群組的權限可讀、可執行，其他人的權限為可讀、可執行。

```
mylinux@mylinux-desktop:/$ ls -al

drwxr-xr-x  21 root root 4096 2009-11-07 18:03 .
drwxr-xr-x  21 root root 4096 2009-11-07 18:03 ..
drwxr-xr-x   2 root root 4096 2009-11-01 17:56 bin
drwxr-xr-x   3 root root 4096 2009-11-01 17:58 boot
lrwxrwxrwx   1 root root    11 2009-11-01 17:43 cdrom -> media/cdrom
drwxr-xr-x  16 root root 3640 2009-11-08 18:53 dev
drwxr-xr-x 131 root root 12288 2009-11-08 18:55 etc
drwxr-xr-x   3 root root 4096 2009-11-01 17:49 home
lrwxrwxrwx   1 root root    33 2009-11-01 17:54 initrd.img -> boot/initrd.img-2.
6.31-14-generic
drwxr-xr-x  18 root root 12288 2009-11-01 17:56 lib
drwx-----  2 root root 16384 2009-11-01 17:43 lost+found
drwxr-xr-x   4 root root 4096 2009-10-29 04:55 media
drwxr-xr-x   2 root root 4096 2009-10-20 08:04 mnt
drwxr-xr-x   2 root root 4096 2009-10-29 04:55 opt
dr-xr-xr-x 153 root root     0 2009-11-09 02:52 proc
drwx-----  10 root root 4096 2009-11-02 19:52 root
drwxr-xr-x   2 root root 4096 2009-11-03 11:32 sbin
```

## 檔案權限 (3)

- 執行 `cd /home/mylinux`，進入自己帳號路徑，輸入 `ls -al`。
- 可以發現檔案的擁有人以及群組，為自己帳號。

```
-rw----- 1 mylinux mylinux 2478 2009-11-08 18:55 .ICEauthority
drwx----- 3 mylinux mylinux 4096 2009-11-03 20:28 .local
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 .nautilus
-rw-r--r-- 1 mylinux mylinux 675 2009-11-01 17:49 .profile
drwx----- 2 mylinux mylinux 4096 2009-11-08 18:55 .pulse
-rw----- 1 mylinux mylinux 256 2009-11-01 18:19 .pulse-cookie
-rw----- 1 mylinux mylinux 218 2009-11-08 09:15 .recently-used.xbel
drwx----- 2 mylinux mylinux 4096 2009-11-01 18:19 .ssh
-rw-r--r-- 1 mylinux mylinux 0 2009-11-01 18:21 .sudo_as_admin_successful
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:20 .update-manager-core
drwx----- 2 mylinux mylinux 4096 2009-11-01 18:37 .update-notifier
-rw-r---- 1 mylinux mylinux 5 2009-11-08 18:55 .vboxclient-autoresize.pid
-rw-r---- 1 mylinux mylinux 5 2009-11-08 18:55 .vboxclient-clipboard.pid
-rw-r---- 1 mylinux mylinux 5 2009-11-08 18:55 .vboxclient-seamless.pid
-rw----- 1 mylinux mylinux 2764 2009-11-08 19:08 .xsessions-errors
-rw----- 1 mylinux mylinux 5766 2009-11-08 09:15 .xsessions-errors.old
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 共
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 圖片
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 影片
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 文件
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-07 18:03 桌面
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 模板
drwxr-xr-x 2 mylinux mylinux 4096 2009-11-01 18:19 音樂
mylinux@mylinux-desktop:~$ 
```

## 檔案權限 (4)

- 當在屬於自己權限的路徑底下時，進行讀檔、寫檔、執行等動作，皆不會進行使用者確認，因為該資料夾的擁有人屬於執行這些權限的人。



A screenshot of a Linux desktop environment showing a terminal window titled "mylinux@mylinux-desktop: ~". The window contains the following terminal session:

```
mylinux@mylinux-desktop:~$ cd /home/mylinux/
mylinux@mylinux-desktop:~$ mkdir mDoc
mylinux@mylinux-desktop:~$ ls
Downloads examples.desktop mDoc 公共 圖片 影片 文件 桌面 模板 音樂
mylinux@mylinux-desktop:~$
```

- 進入不屬於自己權限的路徑底下時，進行讀檔、寫檔、執行等動作，皆因為權限不足而遭受系統拒絕，此時可利用root的權限進行這些動作。
- sudo 指令可讓使用者暫時取得root的權限進行操作。



A screenshot of a Linux desktop environment showing a terminal window titled "mylinux@mylinux-desktop: /home". The window contains the following terminal session:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ mkdir mAccount
mkdir: 無法建立「mAccount」目錄：拒絕不具權限的操作
mylinux@mylinux-desktop:/home$ sudo mkdir mAccount
[sudo] password for mylinux:
mylinux@mylinux-desktop:/home$ ls
mAccount mylinux
mylinux@mylinux-desktop:/home$
```

## 檔案權限 (5)

- 什麼是UID與GID
  - UID與GID為系統主要運作基礎，幫助識別用戶權限。
  - UID : userid
  - GID : groupid
- 使用者密碼資料，存放於/etc/passwd
- 映射資料，存放於/etc/shadow
  - 由於每個程序都需要透過UID，GID判斷權限，故在passwd的權限則為-rw-r--r--，如此將會造成任何人都可看到每個人的密碼，故將密碼移入到shadow並加密，透過映射方式進行存取，以增加帳戶安全性。
- 使用者群組資料，存放於/etc/group

## 檔案權限 (6)

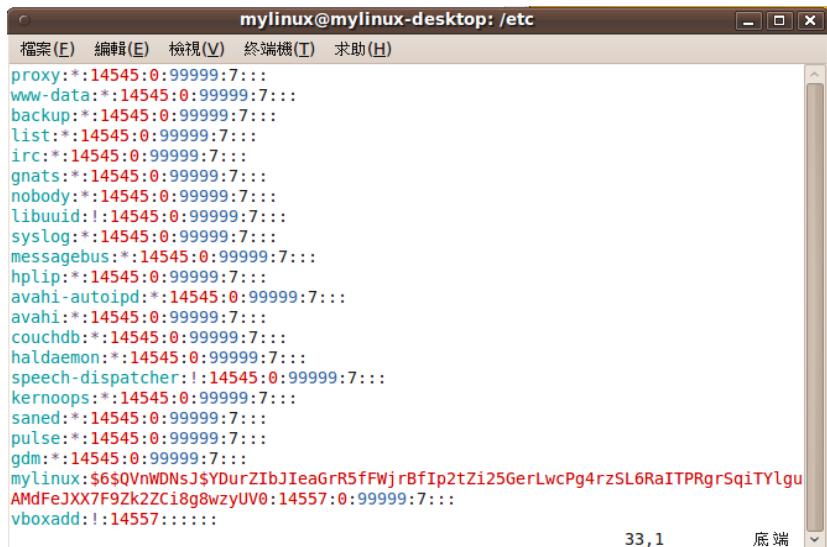
- 如何增加與移除使用者以及群組：
  - adduser : 增加使用者
  - deluser : 刪除使用者
  - groupadd : 增加群組
  - groupdel : 刪除群組
  - gpasswd : 設定群組與密碼
- 如何改變檔案權限：
  - chgrp : 改變檔案所屬群組
  - chown : 改變檔案所屬人
  - chmod : 改變檔案的屬性

## 檔案權限 (7) - /etc/passwd

- 3. UID
- 4. GID
- 5. 一般資料
- 6. 家目錄
- 7. 和系統溝通所使用的Shell

# 檔案權限 (8) - /etc/shadow

- /etc/shadow



A screenshot of a terminal window titled "mylinux@mylinux-desktop: /etc". The window displays the contents of the /etc/shadow file. The file contains numerous entries, each consisting of nine fields separated by colons. The first few entries are:

```
proxy:*:14545:0:99999:7:::
www-data:*:14545:0:99999:7:::
backup:*:14545:0:99999:7:::
list:*:14545:0:99999:7:::
irc:*:14545:0:99999:7:::
gnats:*:14545:0:99999:7:::
nobody:*:14545:0:99999:7:::
libuuid!:14545:0:99999:7:::
syslog!:14545:0:99999:7:::
messagebus!:14545:0:99999:7:::
hplip!:14545:0:99999:7:::
avahi-autoipd!:14545:0:99999:7:::
avahi!:14545:0:99999:7:::
couchdb!:14545:0:99999:7:::
haldaemon!:14545:0:99999:7:::
speech-dispatcher!:14545:0:99999:7:::
kernoops!:14545:0:99999:7:::
saned!:14545:0:99999:7:::
pulse!:14545:0:99999:7:::
gdm!:14545:0:99999:7:::
mylinux:$6$QnW$NSJ$YDurZIbJIeaGrR5fFWj$BfIp2tZi25GerLwcPg4rzSL6RaITPRgrSqiTYlg$AMdFeJXX7F92k22ZCi8g8wzyUV0:14557:0:99999:7:::
vboxadd!:14557:::::::
```

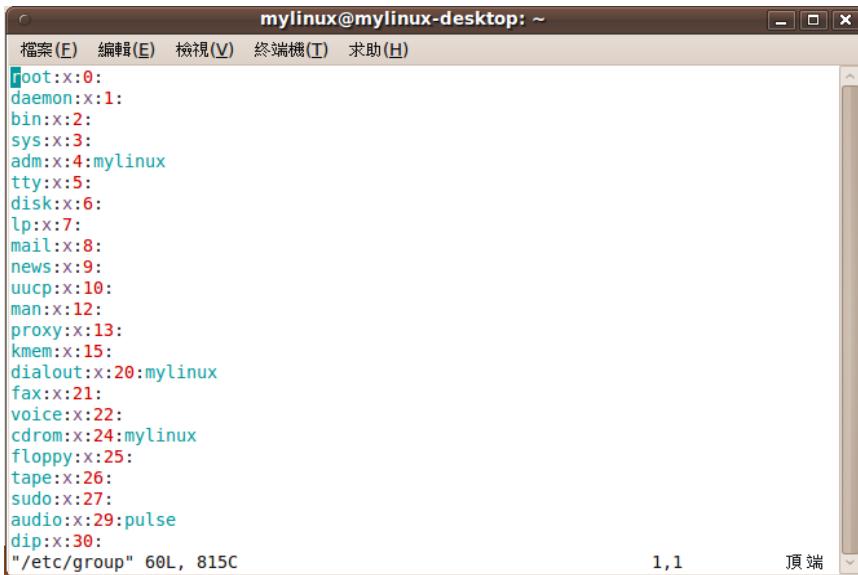
- 資料格式，皆用 : 當區隔，共九個欄位
  - 1.使用者名稱
  - 2. 加密後的密碼

## 檔案權限 (9) - /etc/shadow

- 3. 上次改密碼的時間(Linux日期以1970/01/01為起始)
- 4. 密碼經過幾天可以變更
- 5. 密碼經過幾天必須變更
- 6. 密碼過期之前幾天需警告擁有人
- 7. 密碼過期幾天後帳號會被取消
- 8. 帳號何時失效
- 9. 保留，目前沒用

# 檔案權限 (10) - /etc/group

- /etc/group



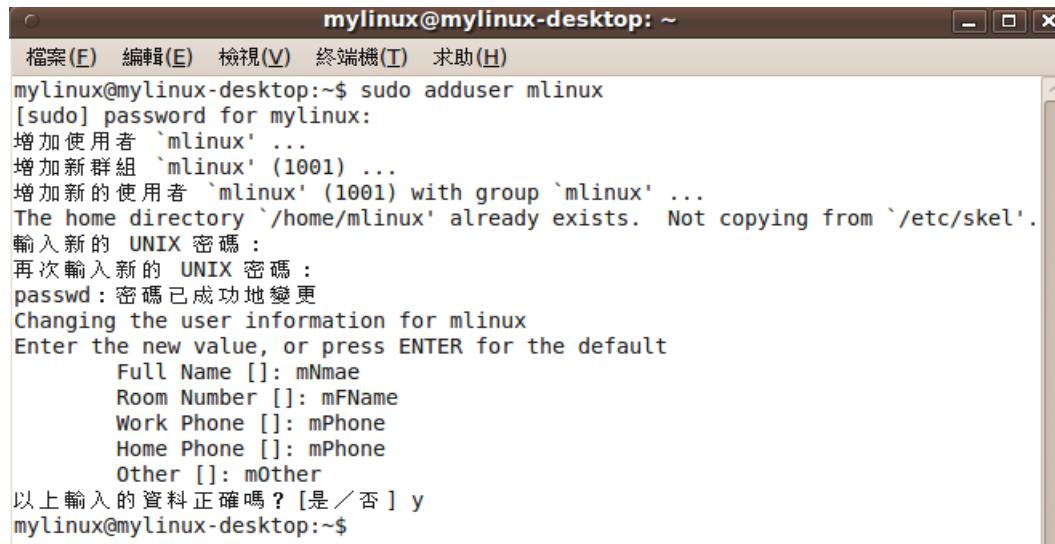
A screenshot of a terminal window titled "mylinux@mylinux-desktop: ~". The window displays the contents of the "/etc/group" file. The file contains a list of group entries, each consisting of a group name followed by a colon, then a list of users separated by colons, and finally a GID at the end. Some entries also include a password field. The entries shown are:

```
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:mylinux  
tty:x:5:  
disk:x:6:  
lp:x:7:  
mail:x:8:  
news:x:9:  
uucp:x:10:  
man:x:12:  
proxy:x:13:  
kmem:x:15:  
dialout:x:20:mylinux  
fax:x:21:  
voice:x:22:  
cdrom:x:24:mylinux  
floppy:x:25:  
tape:x:26:  
sudo:x:27:  
audio:x:29:pulse  
dip:x:30:  
"/etc/group" 60L, 815C
```

- 資料格式：
  - 1. 群組名稱
  - 2. 群組密碼
  - 3. GID
  - 4. 群組成員，使用“,”分隔

# 檔案權限 (11) - adduser

- 當要增加使用者時，輸入
  - sudo adduser “帳戶名稱” 之後進入設定帳號詳細資料
  - 帳戶僅接受小寫



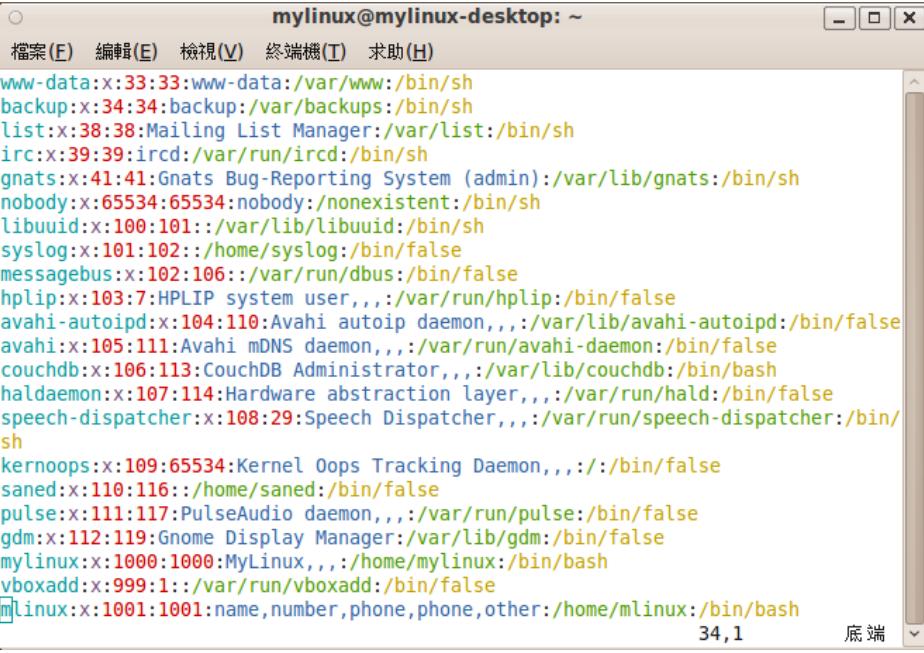
mylinux@mylinux-desktop: ~

檔案(F) 編輯(E) 檢視(V) 終端機(I) 求助(H)

```
mylinux@mylinux-desktop:~$ sudo adduser mlinux
[sudo] password for mylinux:
增加使用者 `mlinux' ...
增加新群組 `mlinux' (1001) ...
增加新的使用者 `mlinux' (1001) with group `mlinux' ...
The home directory `/home/mlinux' already exists. Not copying from `/etc/skel'.
輸入新的 UNIX 密碼 :
再次輸入新的 UNIX 密碼 :
passwd: 密碼已成功地變更
Changing the user information for mlinux
Enter the new value, or press ENTER for the default
    Full Name []: mNmae
    Room Number []: mFName
    Work Phone []: mPhone
    Home Phone []: mPhone
    Other []: mOther
以上輸入的資料正確嗎？ [是／否] y
mylinux@mylinux-desktop:~$
```

# 檔案權限 (12) - adduser

- 在新增完使用者帳戶後，可經由/etc/passwd進行確認是否有新增成功。



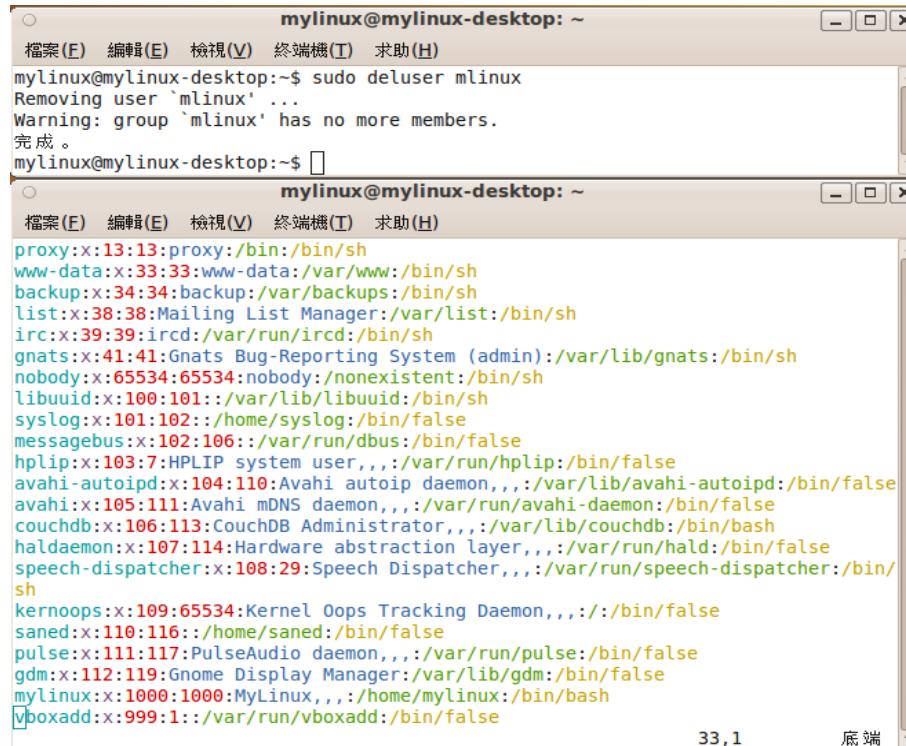
The screenshot shows a terminal window titled "mylinux@mylinux-desktop: ~". The window displays the contents of the /etc/passwd file. The file lists various system users and their details, such as their user ID (UID), group ID (GID), home directory, and shell. The "mylinux" user is listed at the bottom of the list.

```
mylinux@mylinux-desktop: ~
檔案(E) 編輯(E) 檢視(V) 終端機(T) 求助(H)
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
hplip:x:103:7:HPLIP system user,,,,:/var/run/hplip:/bin/false
avahi-autoipd:x:104:110:Avahi autoip daemon,,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:105:111:Avahi mDNS daemon,,,,:/var/run/avahi-daemon:/bin/false
couchdb:x:106:113:CouchDB Administrator,,,,:/var/lib/couchdb:/bin/bash
haldaemon:x:107:114:Hardware abstraction layer,,,,:/var/run/hald:/bin/false
speech-dispatcher:x:108:29:Speech Dispatcher,,,,:/var/run/speech-dispatcher:/bin/
sh
kernoops:x:109:65534:Kernel Oops Tracking Daemon,,,,:/bin/false
saned:x:110:116::/home/saned:/bin/false
pulse:x:111:117:PulseAudio daemon,,,,:/var/run/pulse:/bin/false
gdm:x:112:119:Gnome Display Manager:/var/lib/gdm:/bin/false
mylinux:x:1000:1000:MyLinux,,,,:/home/mylinux:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
mlinux:x:1001:1001:name,number,phone,phone,other:/home/mlinux:/bin/bash
34,1 底端
```

- 其資料對應前章所提格式。

# 檔案權限 (13) - deluser

- 當要移除使用者時，則輸入
  - sudo deluser “帳戶名稱”



```
mylinux@mylinux-desktop: ~
檔案(E) 編輯(E) 檢視(V) 終端機(T) 求助(H)
mylinux@mylinux-desktop:~$ sudo deluser mlinux
Removing user `mlinux' ...
Warning: group `mlinux' has no more members.
完成。
mylinux@mylinux-desktop:~$ 
```

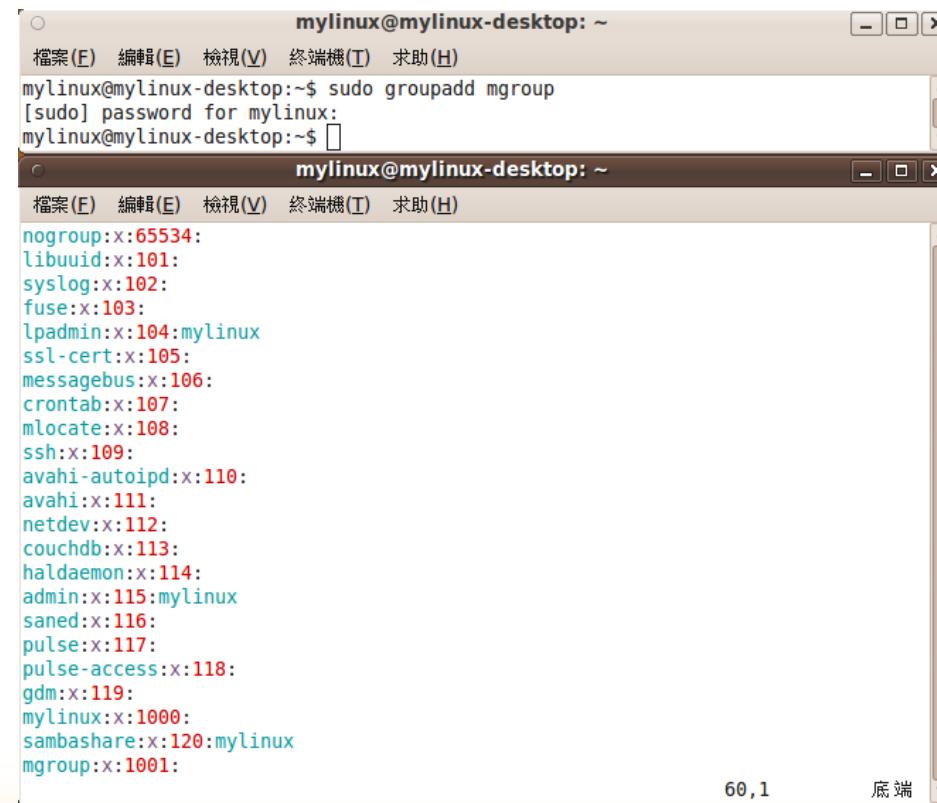
```
mylinux@mylinux-desktop: ~
檔案(E) 編輯(E) 檢視(V) 終端機(T) 求助(H)
proxy:x:13:13:proxy:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuuid:x:100:101::/var/lib/libuuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
hplip:x:103:7:HPLIP system user,,,:/var/run/hplip:/bin/false
avahi-autoipd:x:104:110:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:105:111:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
couchdb:x:106:113:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash
haldaemon:x:107:114:Hardware abstraction layer,,,:/var/run/hald:/bin/false
speech-dispatcher:x:108:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/
sh
kernoops:x:109:65534:Kernel Ooops Tracking Daemon,,,:/bin/false
saned:x:110:116::/home/saned:/bin/false
pulse:x:111:117:PulseAudio daemon,,,:/var/run/pulse:/bin/false
gdm:x:112:119:Gnome Display Manager:/var/lib/gdm:/bin/false
mylinux:x:1000:1000:MyLinux,,,:/home/mylinux:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
```

33,1

底端

# 檔案權限 (14) - groupadd

- 新增一個群組：
  - sudo groupadd "群組名稱"



The screenshot shows a dual-terminal window on a Linux desktop. The top terminal window is titled "mylinux@mylinux-desktop: ~" and contains the command:

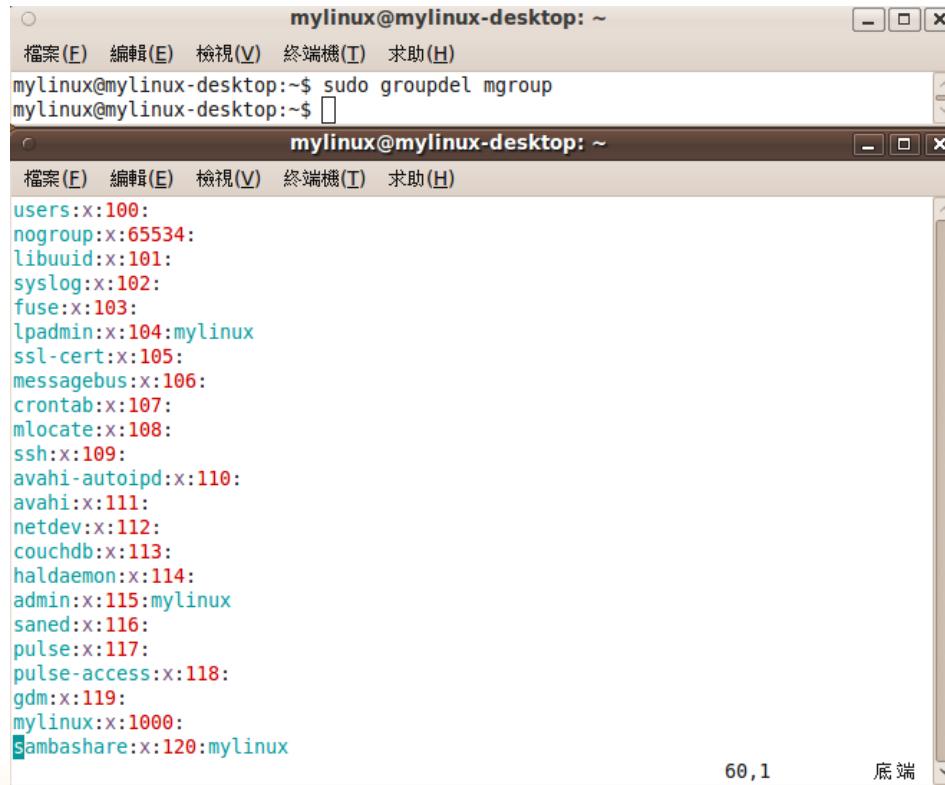
```
mylinux@mylinux-desktop:~$ sudo groupadd mgroup  
[sudo] password for mylinux:  
mylinux@mylinux-desktop:~$
```

The bottom terminal window is also titled "mylinux@mylinux-desktop: ~" and displays the contents of the /etc/group file, which includes the newly created group "mgroup".

```
nogroup:x:65534:  
libuuid:x:101:  
syslog:x:102:  
fuse:x:103:  
lpadmin:x:104:mylinux  
ssl-cert:x:105:  
messagebus:x:106:  
crontab:x:107:  
mlocate:x:108:  
ssh:x:109:  
avahi-autoipd:x:110:  
avahi:x:111:  
netdev:x:112:  
couchdb:x:113:  
haldaemon:x:114:  
admin:x:115:mylinux  
saned:x:116:  
pulse:x:117:  
pulse-access:x:118:  
gdm:x:119:  
mylinux:x:1000:  
sambashare:x:120:mylinux  
mgroup:x:1001:
```

# 檔案權限 (15) - groupdel

- 刪除一個群組：
  - sudo groupdel “群組名稱”



The screenshot shows a dual-terminal window on a Linux desktop. The top terminal window has the title "mylinux@mylinux-desktop: ~". It contains the command "sudo groupdel mgroup" followed by a blank line. The bottom terminal window also has the title "mylinux@mylinux-desktop: ~". It displays the contents of the /etc/group file, which lists various system groups with their GID and other details. The "mgroup" entry is present in the list.

```
mylinux@mylinux-desktop:~$ sudo groupdel mgroup
mylinux@mylinux-desktop:~$ 
mylinux@mylinux-desktop:~$ cat /etc/group
users:x:100:
nogroup:x:65534:
libuuid:x:101:
syslog:x:102:
fuse:x:103:
lpadmin:x:104:mylinux
ssl-cert:x:105:
messagebus:x:106:
crontab:x:107:
mlocate:x:108:
ssh:x:109:
avahi-autoipd:x:110:
avahi:x:111:
netdev:x:112:
couchdb:x:113:
haldaemon:x:114:
admin:x:115:mylinux
saned:x:116:
pulse:x:117:
pulse-access:x:118:
gdm:x:119:
mylinux:x:1000:
Sambashare:x:120:mylinux
```

# 檔案權限 (16) - gpasswd

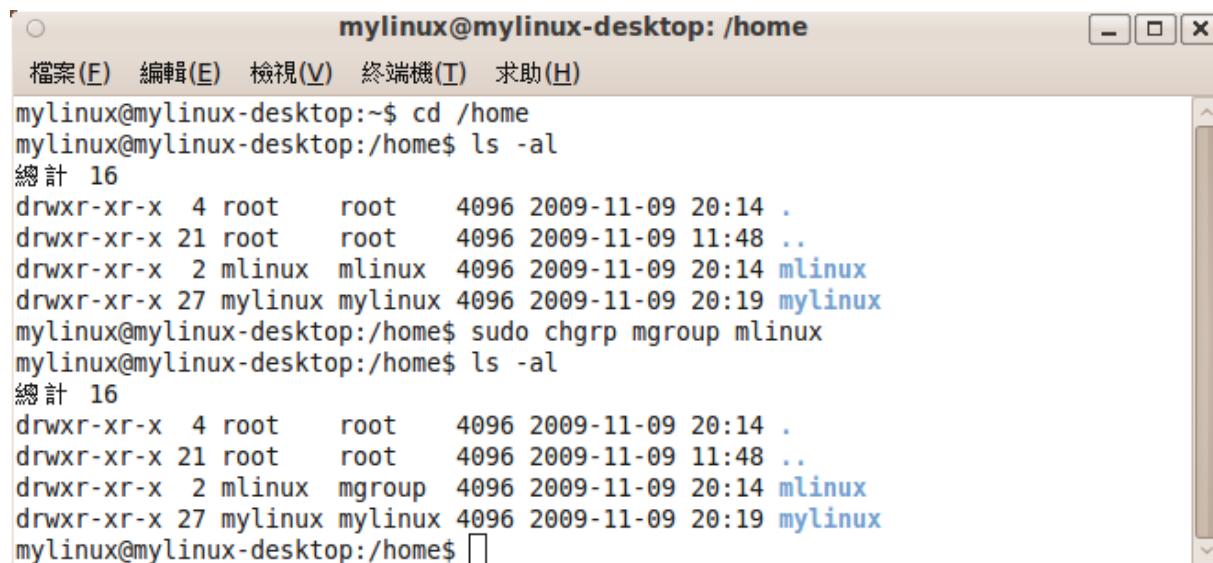
- 當我們想要把使用者設定為某一群組成員時，可以透過gpasswd的指令進行設定，除了設定群組成員外，也可以進行群組密碼的設定。
  - gpasswd “群組名稱”：指定該群組密碼
  - gpasswd [-A user, ...] “群組名稱”：指定user為群組管理員
  - gpasswd [-M user, ...] “群組名稱”：將user加入該群組
  - gpasswd [-r] “群組名稱”：將群組密碼移除
- 現在將之前建立的帳戶，加入到新增群組之中
  - sudo gpasswd -M mlinux mgroup



The screenshot shows two terminal windows side-by-side. The top terminal window has a title bar "mylinux@mylinux-desktop: ~" and a menu bar "檔案(F) 編輯(E) 檢視(V) 終端機(T) 求助(H)". It contains the command "mylinux@mylinux-desktop:~\$ sudo gpasswd -M mlinux mgroup" and its output "mylinux@mylinux-desktop:~\$". The bottom terminal window also has a title bar "mylinux@mylinux-desktop: ~" and a menu bar. It displays the output of the command: "mgroup:x:1002:mlinux". The status bar at the bottom of the bottom window shows "62,1" and "底端".

## 檔案權限 (17) - chgrp

- chgrp : 改變所屬群組，而所改變的群組，必須要在/etc/group裡所存在的名稱。
  - sudo chgrp [ -R ] “群組名稱” “改變的檔案或資料夾”
  - [ -R ] 參數：代表是否持續改變該資料夾底下所有檔案

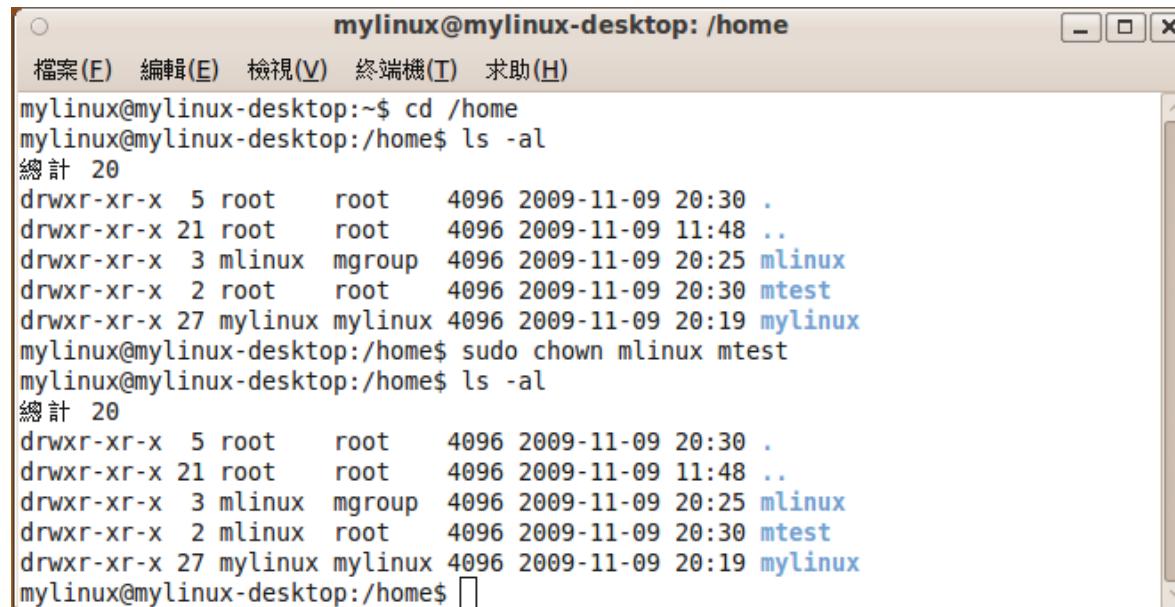


The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The terminal history is as follows:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 16
drwxr-xr-x  4 root      root    4096 2009-11-09 20:14 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxr-xr-x  2 mlinux    mlinux   4096 2009-11-09 20:14 mlinux
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chgrp mgroup mlinux
mylinux@mylinux-desktop:/home$ ls -al
總計 16
drwxr-xr-x  4 root      root    4096 2009-11-09 20:14 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxr-xr-x  2 mlinux    mgroup   4096 2009-11-09 20:14 mlinux
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ 
```

## 檔案權限 (18) - chown

- chown : 改變檔案所有人，而所改變的擁有者，必須是在 /etc/passwd 中所紀錄的使用者名稱。
  - sudo chown [ -R ] “帳號名稱” “檔案或資料夾”
  - [ -R ] 參數：代表是否持續改變該資料夾底下所有檔案



The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The terminal displays the following command sequence:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root    4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr-xr-x  2 root      root    4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chown mlinux mtest
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root    4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr-xr-x  2 mlinux   root    4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ 
```

## 檔案權限 (19) - chmod

- chmod : 改變檔案屬性，修改擁有者、群組、其他人使用檔案或資料夾的權限。其屬性對照如下：
  - r : 4
  - w : 2
  - x : 1
- 檔案的權限，如果是777，則代表擁有者、群組、其他人，皆可對該檔案進行讀、且、執行的權限。
- 而修改方式為：
  - chmod [ -R ] {i}{j}{k} "檔案或目錄"
  - [ -R ] 參數：代表是否持續改變該資料夾底下所有檔案
  - {i} {j} {k} : 分別代表各個使用者權限計算後之結果。

## 檔案權限 (20) - chmod

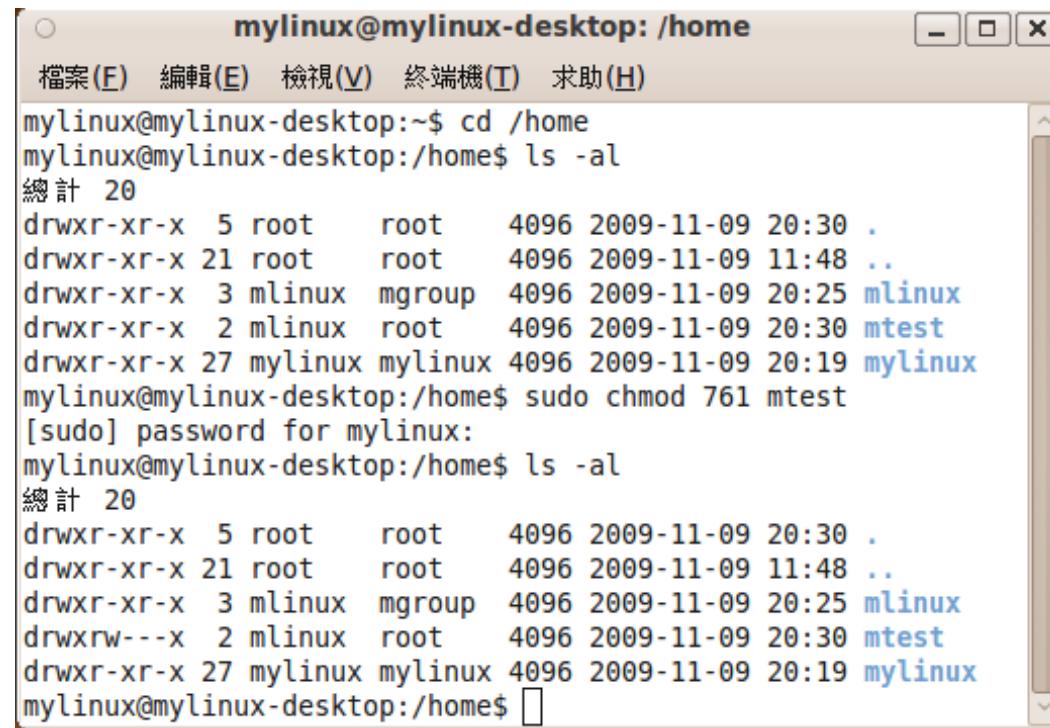
- 另外一個修改方式，則是將該群組分類，如下表：

chmod	u g o a	+ (加入) - (除去) = (設定)	r w x	檔案或目錄
-------	------------------	----------------------------	-------------	-------

- u : user
- g : group
- o : other
- a : all

## 檔案權限 (21) - chmod

- sudo chmod 761 mtest，擁有者擁有全部權限，群組擁有讀取與寫入權限，其他擁有執行權限。

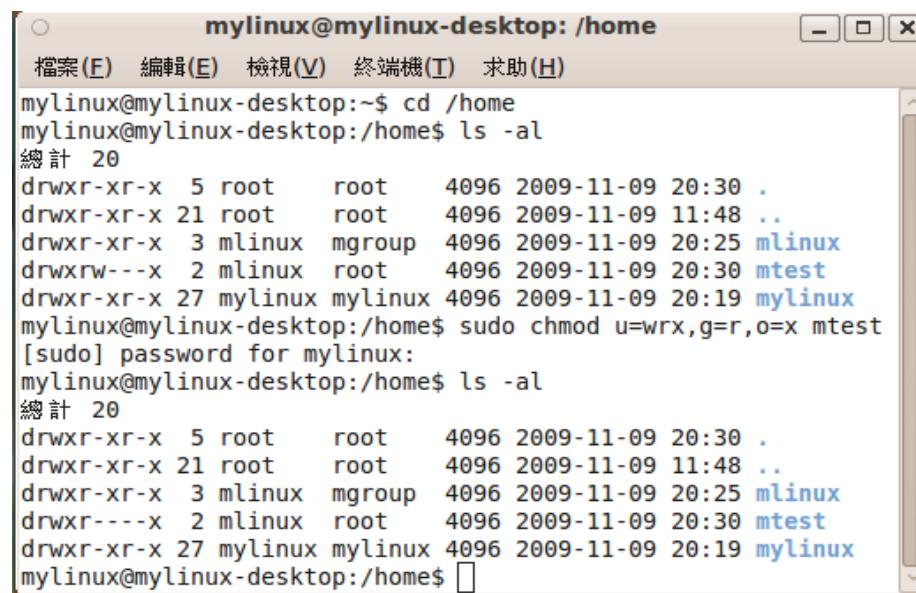


The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The window contains the following command history:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr-xr-x  2 mlinux    root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chmod 761 mtest
[sudo] password for mylinux:
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxrwx---x  2 mlinux    root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ 
```

## 檔案權限 (22) - chmod

- `sudo chmod u=rwx,g=r,o=x mtest`，「`u=rwx,g=r,o=x`」中間並無任何空白，使`mtest`檔案，擁有者有讀取、寫入及執行權限，群組擁有讀取權限，其他擁有執行權限。

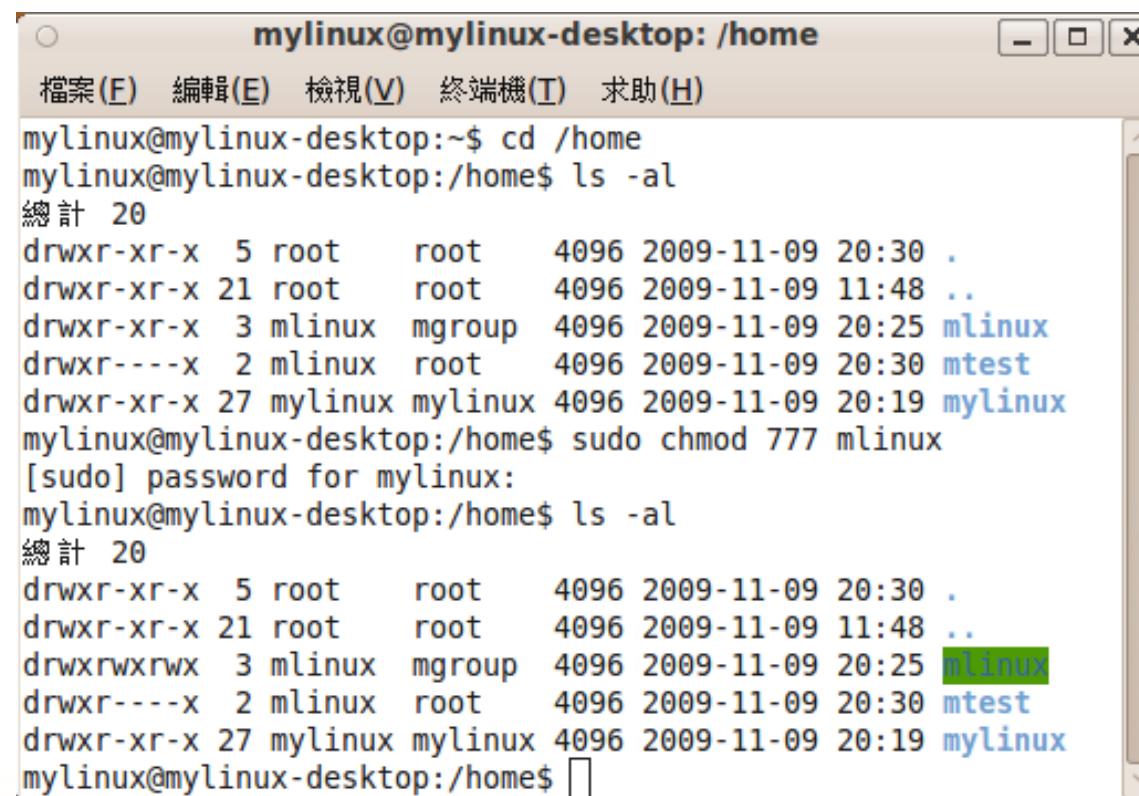


The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The window contains the following command history and output:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxrwx---x  2 mlinux   root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chmod u=rwx,g=r,o=x mtest
[sudo] password for mylinux:
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxrwx---x  2 mlinux   root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
```

## 檔案權限 (23) - chmod

- sudo chmod 777 mlinux，使mlinux所有使用者擁有全部權限。



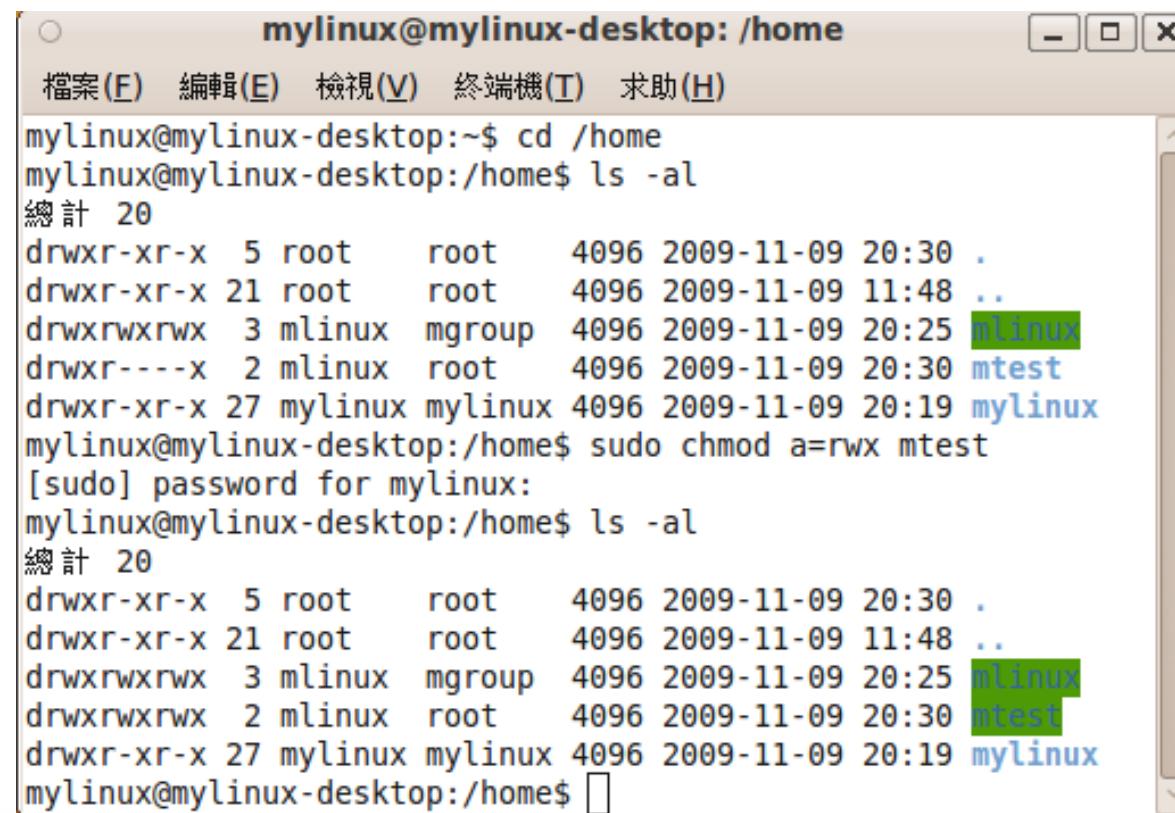
The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The window has a menu bar with Chinese labels: 檔案(E)、編輯(E)、檢視(V)、終端機(T)、求助(H). The terminal content is as follows:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxr-xr-x  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr----x  2 mlinux    root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chmod 777 mlinux
[sudo] password for mylinux:
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root     4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root     4096 2009-11-09 11:48 ..
drwxrwxrwx  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr----x  2 mlinux    root     4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ 
```

In the second "ls -al" command output, the file "mlinux" has its permissions changed to "drwxrwxrwx" (777), indicated by the green background color.

## 檔案權限 (24) - chmod

- sudo chmod a=rwx mtest，使mtest所有使用者擁有全部權限。

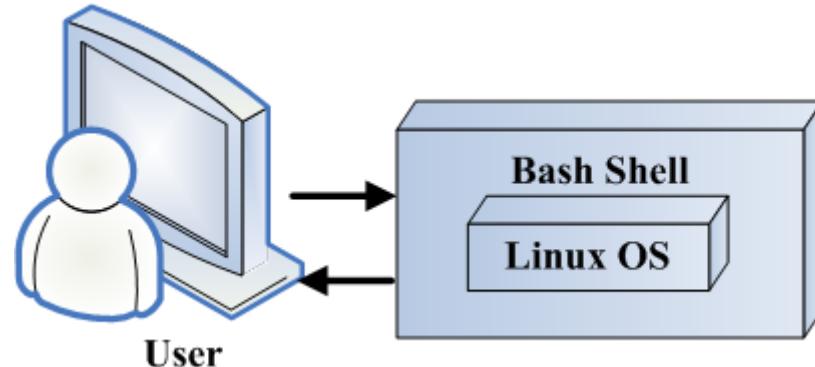


The screenshot shows a terminal window titled "mylinux@mylinux-desktop: /home". The window contains the following terminal session:

```
mylinux@mylinux-desktop:~$ cd /home
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root    4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxrwxrwx  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxr----x  2 mlinux    root    4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ sudo chmod a=rwx mtest
[sudo] password for mylinux:
mylinux@mylinux-desktop:/home$ ls -al
總計 20
drwxr-xr-x  5 root      root    4096 2009-11-09 20:30 .
drwxr-xr-x 21 root      root    4096 2009-11-09 11:48 ..
drwxrwxrwx  3 mlinux    mgroup   4096 2009-11-09 20:25 mlinux
drwxrwxrwx  2 mlinux    root    4096 2009-11-09 20:30 mtest
drwxr-xr-x 27 mylinux  mylinux  4096 2009-11-09 20:19 mylinux
mylinux@mylinux-desktop:/home$ 
```

# Shell

- 使用者與作業系統之間溝通的橋樑，當使用者輸入指令時，將轉換成為作業系統所知道的指令進行溝通。



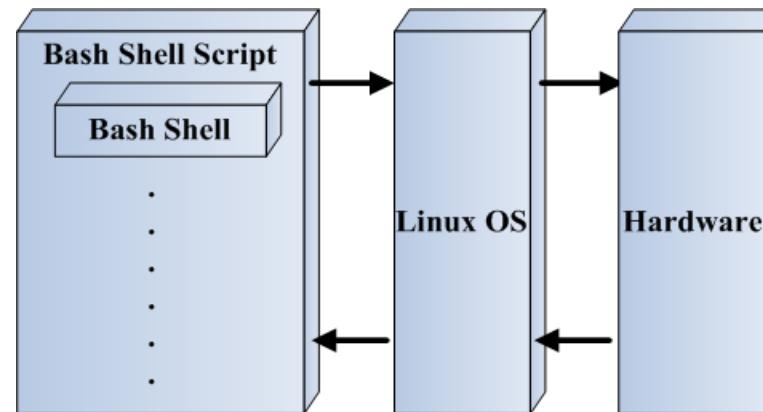
- 例如：`cd`，代表切換路徑，`ls`，代表列出該路徑下所有的檔案。

A screenshot of a terminal window titled 'mylinux@mylinux-desktop: /'. The window has a menu bar with '檔案(F) 編輯(E) 檢視(V) 終端機(I) 求助(H)'. The terminal output shows:

```
mylinux@mylinux-desktop:~$ cd /
mylinux@mylinux-desktop:/$
mylinux@mylinux-desktop:~$ ls
bin  dev  initrd.img  media  proc  selinux  tmp  vmlinuz
boot  etc  lib       mnt   root  srv    usr
cdrom  home  lost+found  opt  sbin  sys    var
mylinux@mylinux-desktop:/$
```

# Shell Script (1)

- 可透過Shell Script建立指令腳本，方便使用者進行大量Shell指令與作業系統進行溝通



- 在每個Script中皆有他特別的意義存在，例如：作業系統啟動流程，也靠著Script進行一連串的系統啟動服務

# Shell Script (2)

- 撰寫一個簡單的Shell Script：

```
#!/bin/bash

cd /          ←切換到根目錄
ls           ←顯示該目錄所有檔案
```

- 將該檔案加入執行權限：
  - sudo chmod +x mShellScript
- 執行輸出



The screenshot shows a terminal window titled "mylinux@mylinux-desktop: ~/桌面". The window menu bar includes "檔案(F)", "編輯(E)", "檢視(V)", "終端機(T)", and "求助(H)". The terminal command history shows:

```
mylinux@mylinux-desktop:~$ cd 桌面
mylinux@mylinux-desktop:~/桌面 $ sudo ./mShellScript
bin  dev  initrd.img  media  proc  selinux  tmp  vmlinuz
boot  etc  lib  mnt  root  srv  usr
cdrom  home  lost+found  opt  sbin  sys  var
mylinux@mylinux-desktop:~/桌面 $
```

The command "sudo ./mShellScript" and its output are highlighted with a red rectangle.

# 嵌入式系統軟體開發

# 嵌入式系統軟體開發

## Embedded Software vs. Non-Embedded Software

- Development technologies
  - Hard timing constraints
  - Limited memory resources and power usage
  - Predefined hardware platform technology
  - Related cost drivers of the hardware
  - Reliability, cost and time-to-market demands

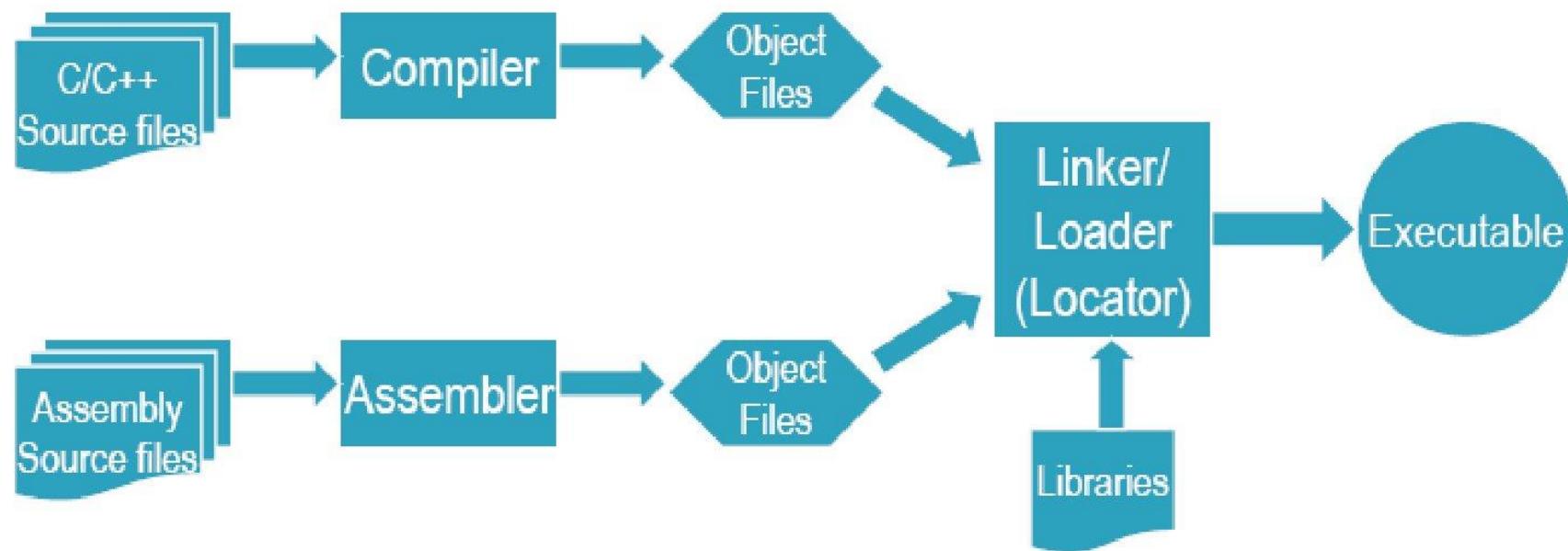
# Embedded Software vs. Non-Embedded Software

- Embedded software usually packaged with hardware product for selling
- Embedded software engineering and other processes such as mechanical engineering and electrical engineering are in fact sub-processes of system engineering.
- Coordinating these sub-processes to develop quality products is one of the embedded software development's most challenging aspects.

# Software Development for Desktop Computer Systems

- Typically software development for desktop computer systems are usually done on the same machine (Native development):
  - Create/edit the source code
  - Compile the source code to create executable program
  - Execute the code
    - OS creates the process and loads into memory
    - OS transfers control to the process and the process begins execution

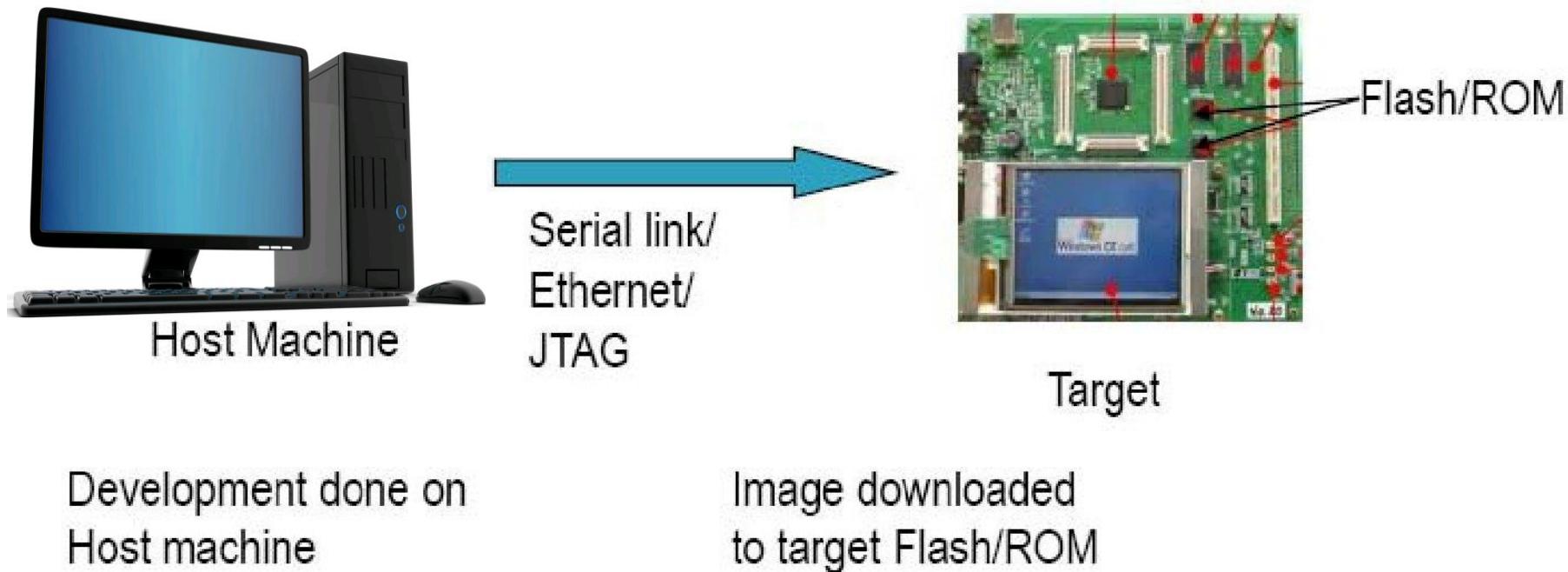
# Creating Executable Code



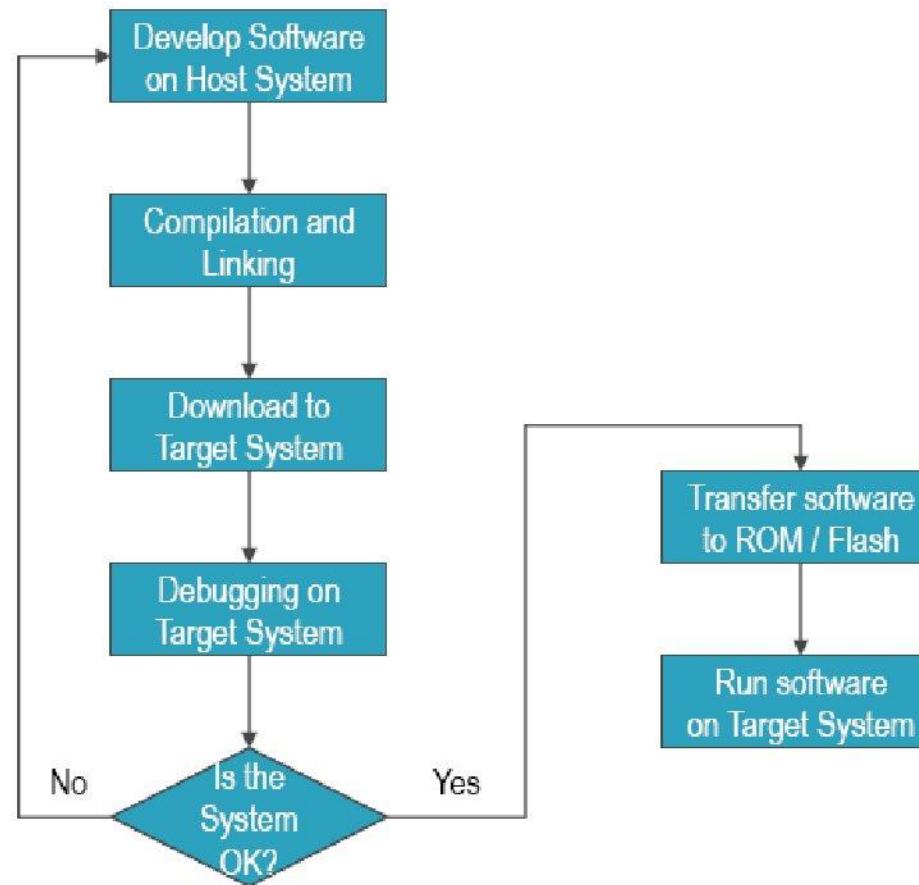
# Embedded Software Development

- The above process will not work exactly the same way for embedded systems:
  - Application programs are typically developed, compiled, and run on the host system
  - Embedded software is targeted for execution on a target processor(different from the development/host processor and operating environment)
  - We cannot directly develop the applications on the target hardware
  - In a general purpose computer environment, the OS and application software are distinct, but in embedded systems everything is often a single piece of code
  - Memory has to be explicitly managed (often no memory management/virtual memory)

# Embedded Software Development



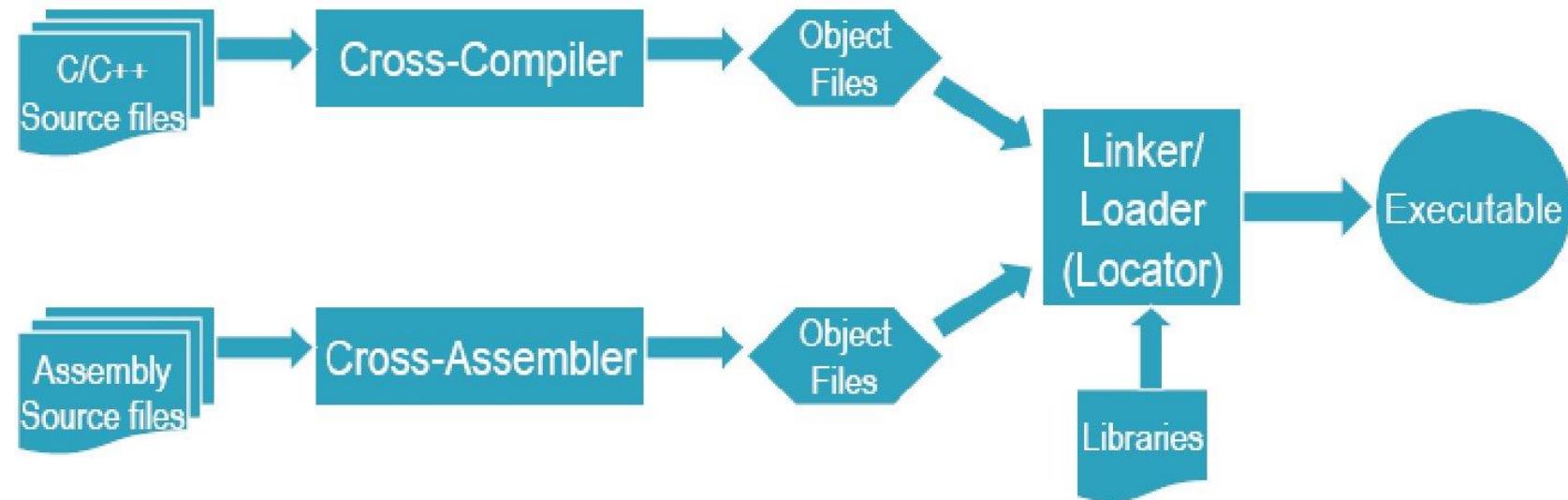
# Embedded Software Development



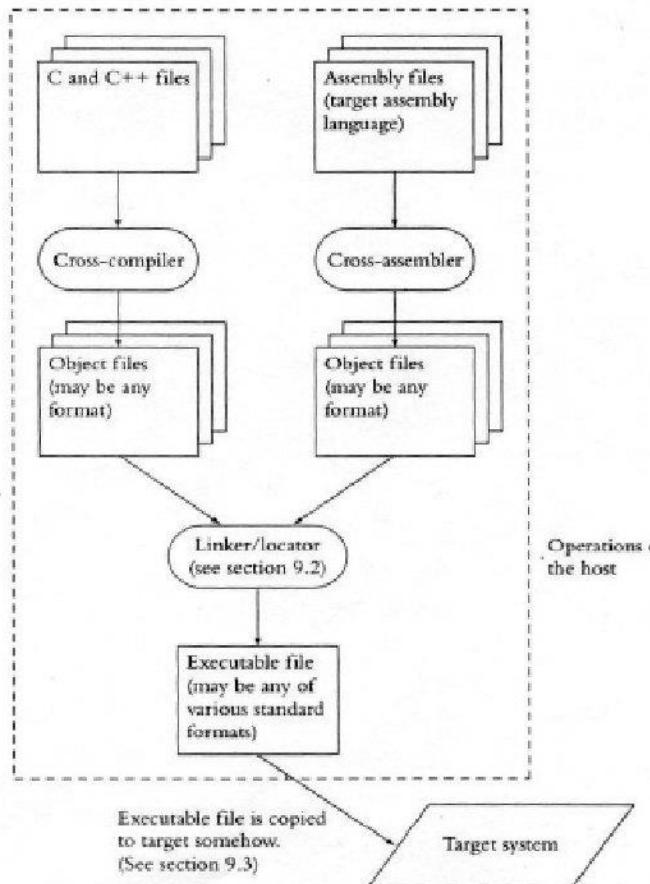
# Embedded Software Development

- What tools are needed to develop, test, and locate embedded software into the target processor and its operating environment?
- Host: Where the embedded software is developed, compiled, tested, debugged, and optimized prior to its translation into target device.
  - Because the host has keyboards, editors, monitors, printers, more memory, etc. for development, while the target may not have these capabilities for developing the software.
- Target: After development, the code is **cross-compiled**, translated –**cross-assembled, linked** (into target processor instruction set) and located into the target

# Creating Executable Code For Embedded System



# Tools Chain for Building Embedded Software



- Cross-compiler
- Cross-assembler
- Cross-linker
- Cross-debugger
- Cross-compiled libraries
- OS dependent libraries and headers for target processor

# GCC與Makefile簡介

- 介紹GNU Compiler Collection，及其相關參數及使用方式，以了解其使用情況。
- 接著將介紹Makefile，可將一系列繁雜的編譯過程寫成一個Makefile檔，並透過指令自動執行編譯。

# GCC簡介

- gcc : GNU project C and C++ compiler
- Features of GNU cc :
  - The compilation process includes up to four stages :
    - - Preprocessing
    - - Compilation Proper
    - - Assembly
    - - Linking

# Interprets Filename Extensions

Extension	Type
.c	C language source code
.C, .cc	C++ language source code
.i	Preprocessed C source code
.ii	Preprocessed C++ source code
.S, .s	Assembly language source code
.o	Compiled object code
.a, .so	Compiled library code

# 範例

- Hello word:

```
#include <stdio.h>
int main(void)
{
    printf("Hello Word.\n");
    return 0;
}
```

```
root@miraculous-desktop:~# vim hello.c
root@miraculous-desktop:~# gcc hello.c -o hello
root@miraculous-desktop:~# ./hello
Hello Word.
root@miraculous-desktop:~#
```

# GCC Command-Line Options

- `-o output_file` :
  - Specify the output filename; not necessary when compiling to object code. If FILE is not specified, the default name is `a.out`

```
root@miraculous-desktop:~/hello# ls
hello.c Makefile
root@miraculous-desktop:~/hello# gcc hello.c -o hello.out
root@miraculous-desktop:~/hello# ls
hello.c hello.out Makefile
root@miraculous-desktop:~/hello#
```

- `-C` :
  - Compile or assemble the source files, but do not link
- `-g` :
  - Include standard **debugging information in the binary**
- `-E` :
  - Stop after the preprocessing stage; do not run the compiler proper

```
root@miraculous-desktop:~/hello# gcc -E hello.c -o hello.cpp
root@miraculous-desktop:~/hello# ls
hello.c hello.cpp hello.out Makefile
root@miraculous-desktop:~/hello#
```

- `-O` :
  - The bare `-O` option tells gcc to **reduce both code size and execution time**. It is equivalent to `-O1`

# GCC Command-Line Options

- **Code optimization problem**
  - Code optimization is an attempt to improve performance
  - The trade-off is lengthened compile times and increased memory usage during compilation
- **-I :**
  - Add the directory to the list of directories to be searched for header files.
  - Example : `gcc program.c -I/usr/include`
- **-L :**
  - Add directory to the list of directories to be searched for library files.
  - Example : `gcc program.c -L/usr/lib`

# Makefile

- Makefile
  - The GNU make utility automatically determines which pieces of a large program need to be recompiled, and issues the commands to recompile them.

# Makefile

- A rule consists of the following
  - Target : A target is usually the name of a file that is generated by a program; examples of targets are **executable** or **object files**. A target can also be the name of an **action** to carry out, such as 'clean'.
  - Dependency : A dependency is a file that is used as input to create the target. A target often depends on several files.
  - Command : A command is an action that make carries out.

**target** : dependency dependency [...]

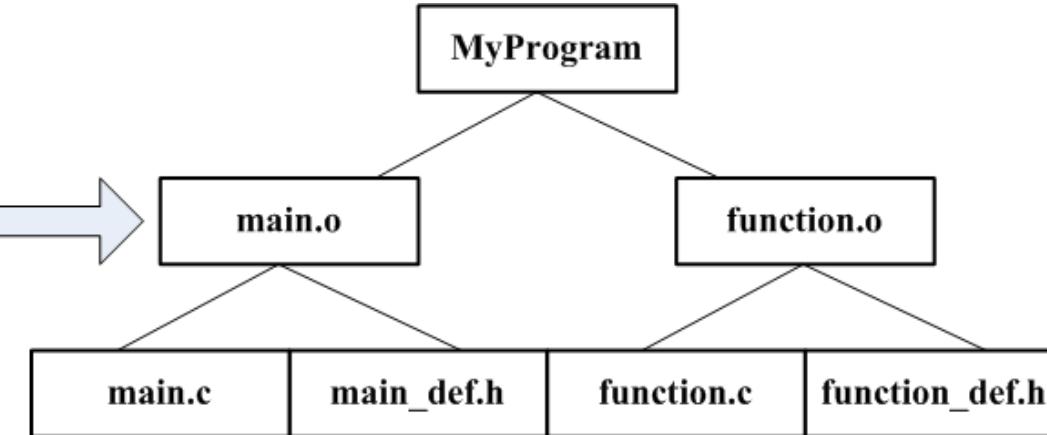
command

command

[...]

# A Simple Makefile

```
MyProgram:main.o function.o
    gcc -o MyProgram main.o function.o
main.o:main.c main_def.h
    gcc -c main.c
function.o:function.c function_def.h
    gcc -c function.c
```



- This makefile has three rules
- The first **target**, MyProgram, is called the *default* target
- MyProgram has two **dependencies**, main.o and function.o; these two files must exist in order to build MyProgram.

# Makefile

- How make processes a Makefile :
  1. By default, make starts with the first target.
  2. Thus, when you give the command: **make**; make reads the “**Makefile**” in the current directory and begins by processing the first rule.
  3. Make compares the **timestamp** on the target to the timestamp of the dependencies.
  4. If one or more of the dependencies is newer than the target, make rebuilds the target, assuming that the newer dependency implies some code change that must be incorporated into the target.

# Using Variables in Makefile

- Variables :
  - To simplify editing and maintaining makefiles, make allows you to create and use variables.
  - Define variables using the general form :
    - VARNAME = text [...]
    - Example :
      - OBJS = main.o function.o
      - HDRS = main.c main\_def.h
  - To obtain VARNAME' s value, enclose it in parentheses and prefix it with a \$:
    - \$(VARNAME)
    - Example:
      - MyProgram:\$(OBJS)

# Using Variables in Makefile

- `$@` : the filename of a rule' s target
- `$<` : the name of the first dependency in a rule
- `$^` : space-delimited list of all the dependencies in a rule
- `$?` : space-delimited list of all the dependencies in a rule that are newer than the target
- `$(@D)` : the directory part of a target filename, if the target is in a subdirectory
- `$(@F)` : the filename part of a target filename, if the target is in a subdirectory

# Using Variables in Makefile

<b>Variable</b>	<b>Description</b>
CC	Program for compiling C programs; default value = gcc
CPP	C Preprocessor program; default value = cpp
RM	Program to remove files; default value = "rm -f"
ARFLAGS	Flags for the archive-maintenance program; default = rv
ASFLAGS	Flags for the assembler program; no default
CFLAGS	Flags for the C compiler; no default
CPPFLAGS	Flags for the C preprocessor; no default
LDFLAGS	Flags for the linker (ld); no default
AR	Archive-maintenance programs; default value = ar
AS	Program to do assembly; default value = as

## ❖ Example:

- CROSS\_COMPILE = arm-unknown-linux-gnu-
- CC = \$(CROSS\_COMPILE)gcc /\* arm-unknown-linux-gnu-gcc \*/
- LD = \$(CROSS\_COMPILE)ld /\* arm-unknown-linux-gnu-ld \*/

# Pattern Rules

- %.o : %.c
  - tells make to build any object file **filename.o** from a source file **filename.c**
- Example :
  - %.o: \$(SDIR)%.c
    - \$(CC) -c \$(CFLAGS) \$(CPPFLAGS) \$< -o \$@

# A simple makefile example

- programA.c

```
#include <stdio.h>
void programA()
{
    printf("ProgramA\n");
```

- programB.c

```
#include <stdio.h>
void programB()
{
    printf("ProgramB\n");
```

- programC.c

```
#include <stdio.h>
void programC()
{
    printf("ProgramC\n");
```

- def.h

```
void programA();
void programB();
void programC();
```

# A simple makefile example

- Program.c

```
#include <stdio.h>
#include "def.h"
int main(void)
{
    programA();
    programB();
    programC();
    return 0;
}
```

- Makefile

```
FILE=Program
CFLAGS= -g -Wall
CC=gcc
LIBS=-lm
INCLUDES=
OBJS=programA.o programB.o programC.o
SRCS=programA.c programB.c programC.c Program.c

all:${FILE}
${FILE}: ${FILE}.o ${OBJS}
        $(CC) $(CFLAGS) $(INCLUDES) -o ${FILE}.o ${OBJS} $(LIBS)

clean:
        rm *.o ${FILE}
```

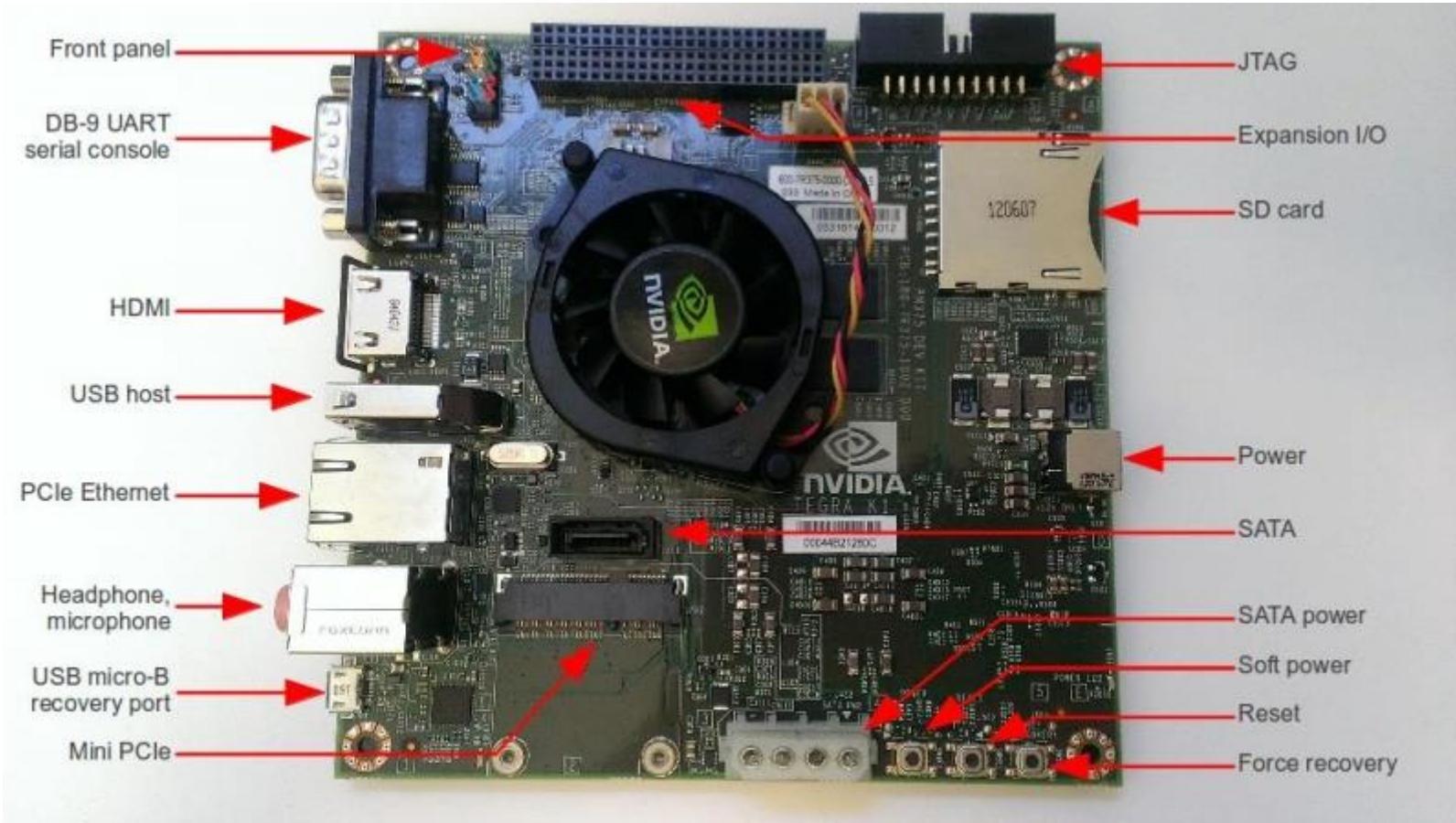
# A simple makefile example

- Type “make” to make this example

```
root@miraculous-desktop:~/Example# ls
def.h  Makefile  programA.c  programB.c  Program.c  programC.c
root@miraculous-desktop:~/Example# make
gcc    -c -o Program.o Program.c
gcc    -c -o programA.o programA.c
gcc    -c -o programB.o programB.c
gcc    -c -o programC.o programC.c
gcc    -o Program Program.o programA.o programB.o programC.o -lm
root@miraculous-desktop:~/Example# ./Program
ProgramA
ProgramB
ProgramC
```

# Jetson TK1 開發環境建立

# Jetson TK1 硬體介紹



# Jetson TK1開發環境

- Nsight

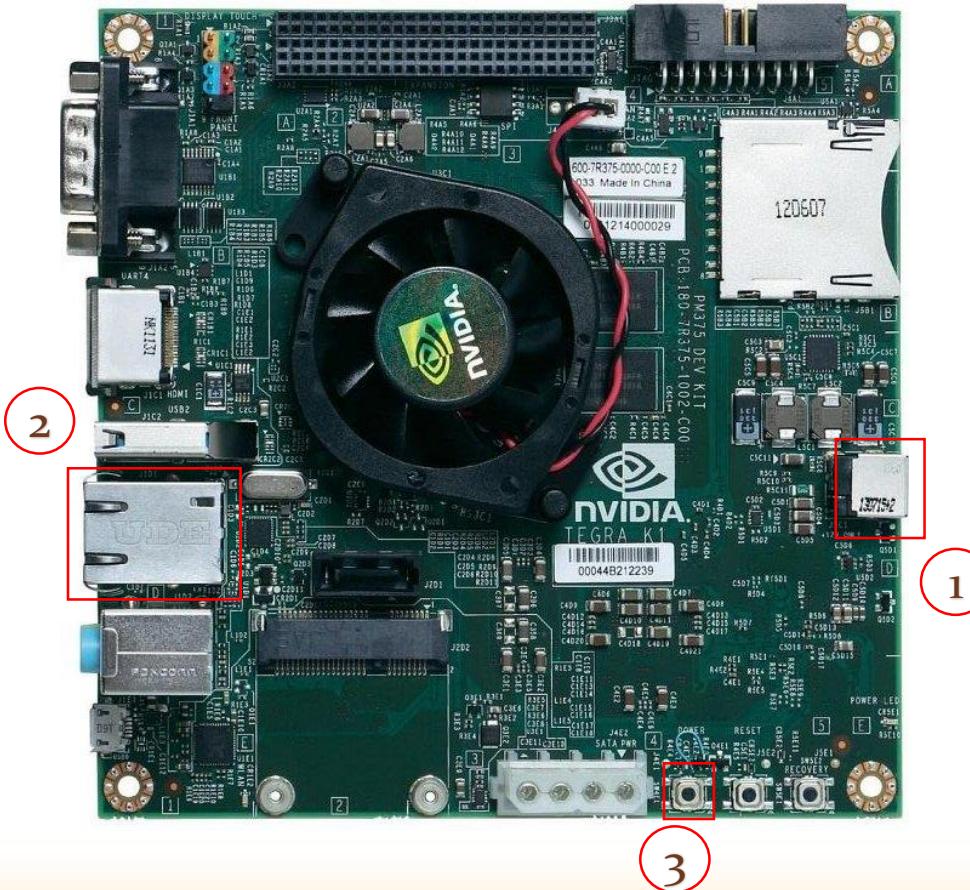
- 為了從虛擬機中同步專案到TK1上，所以使用CUDA提供的Nsight程式開發工具，來撰寫和同步程式專案。
- CUDA是 NVIDIA 平行運算架構。利用 GPU 的強大威力，此架構能大幅提昇運算效能。

- OpenCV

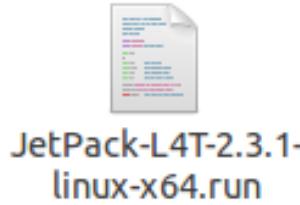
- OpenCV是由Intel公司所開發的開放原始碼電腦影像視覺函式庫(Open Source Computer Vision Library)，它由一系列 C 函數和少量 C++ 構成，實現了圖像處理和電腦視覺方面的很多通用演算法。

# Jetson TK1建置步驟

1. 使用Jetpack燒TK1
2. 將桌上的網路線插至TK1
3. 按下Power按鈕開機取得TK1的IP位置
4. 每次重新啟動TK1時都會改變
5. Xming與Putty連入使用



# 使用JetPack燒錄TK1的檔案系統



JetPack可於Nvidia網路下載或於課程網站中的連結下載，於課程網站中提供的虛擬機環境已經有放入此檔案

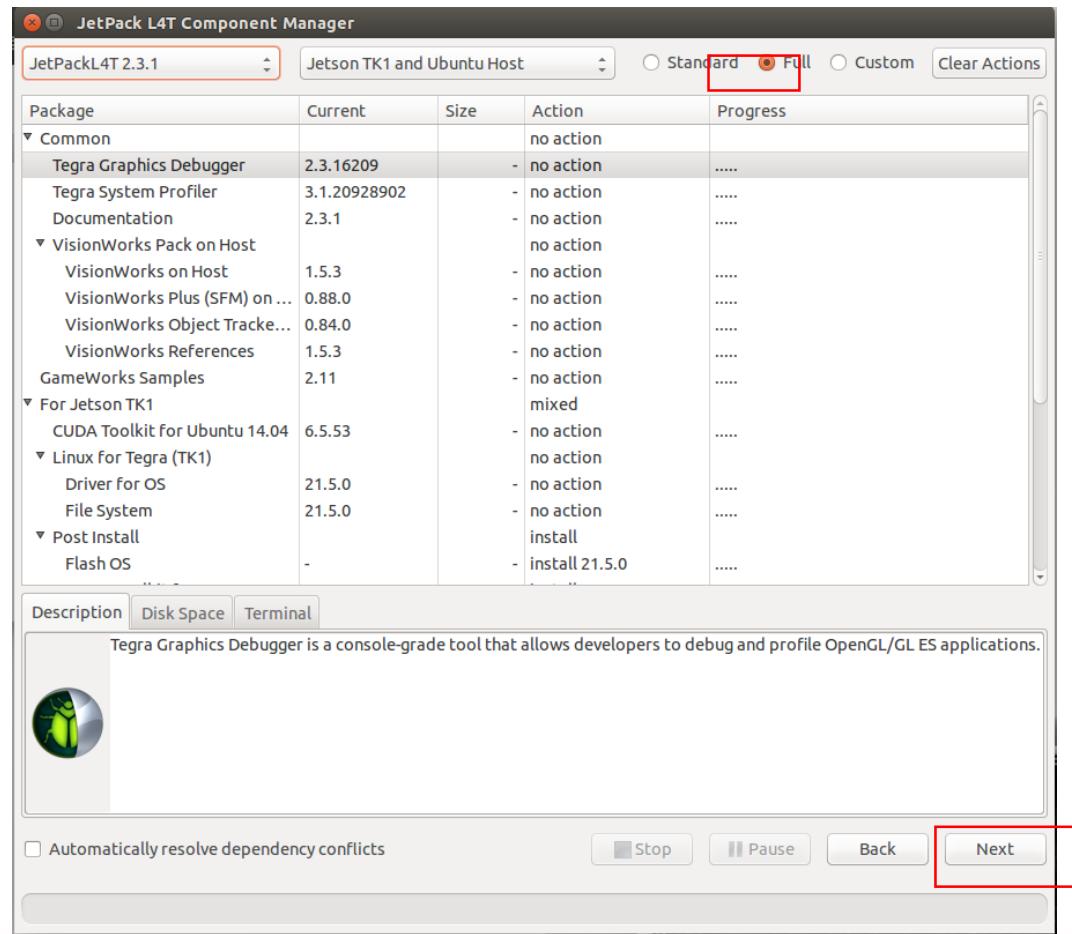
```
$ sudo ./JetPack-L4T-2.3.1-linux-x64.run
```

\*此章節在有需要將TK1系統損毀時需要系統重建才使用，一般上課時TK1已經完成安裝檔案系統，所以不用另外進行

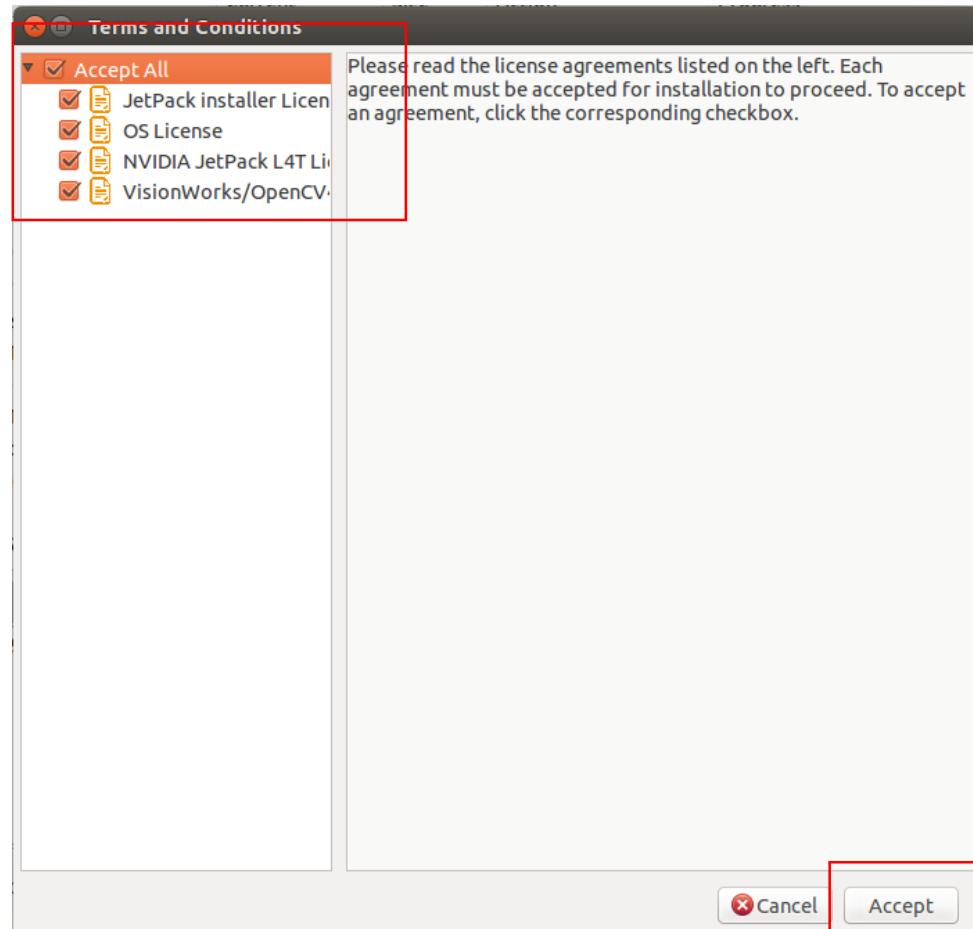
# 使用JetPack燒TK1



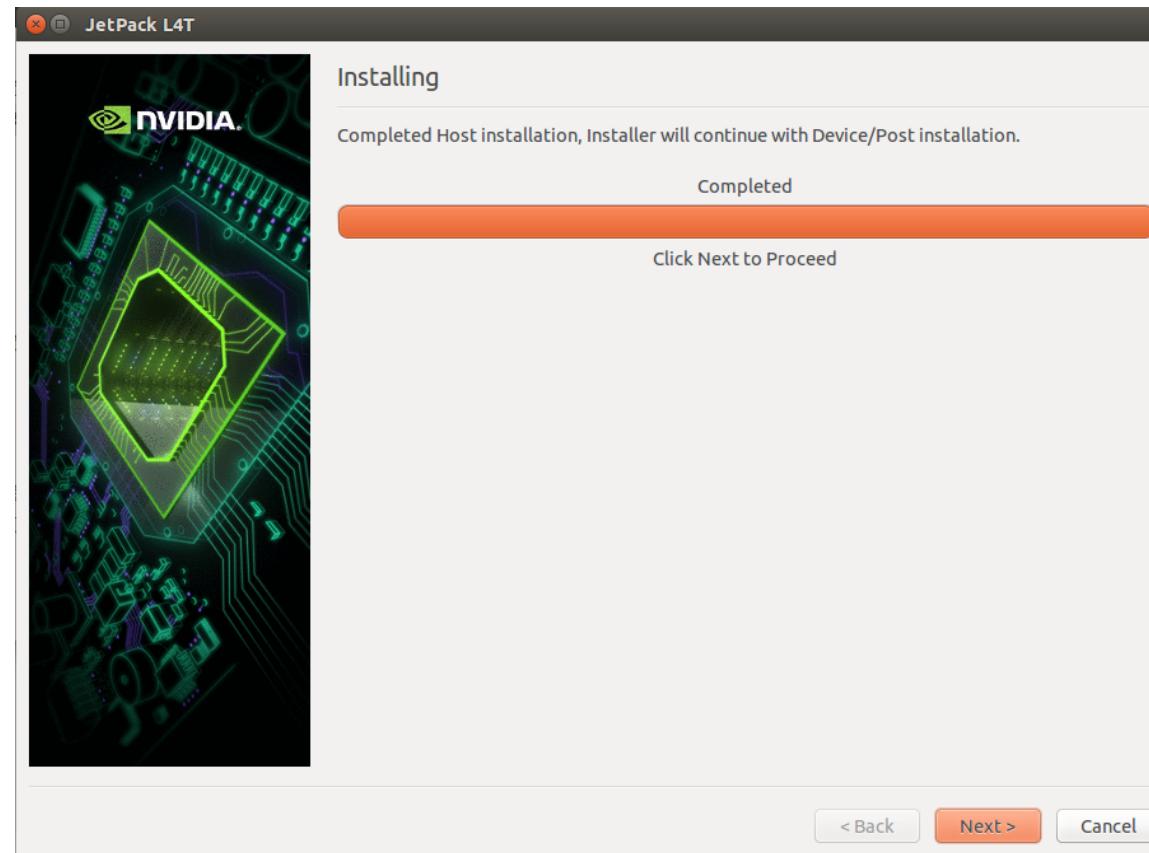
# 使用JetPack燒TK1



# 使用JetPack燒TK1



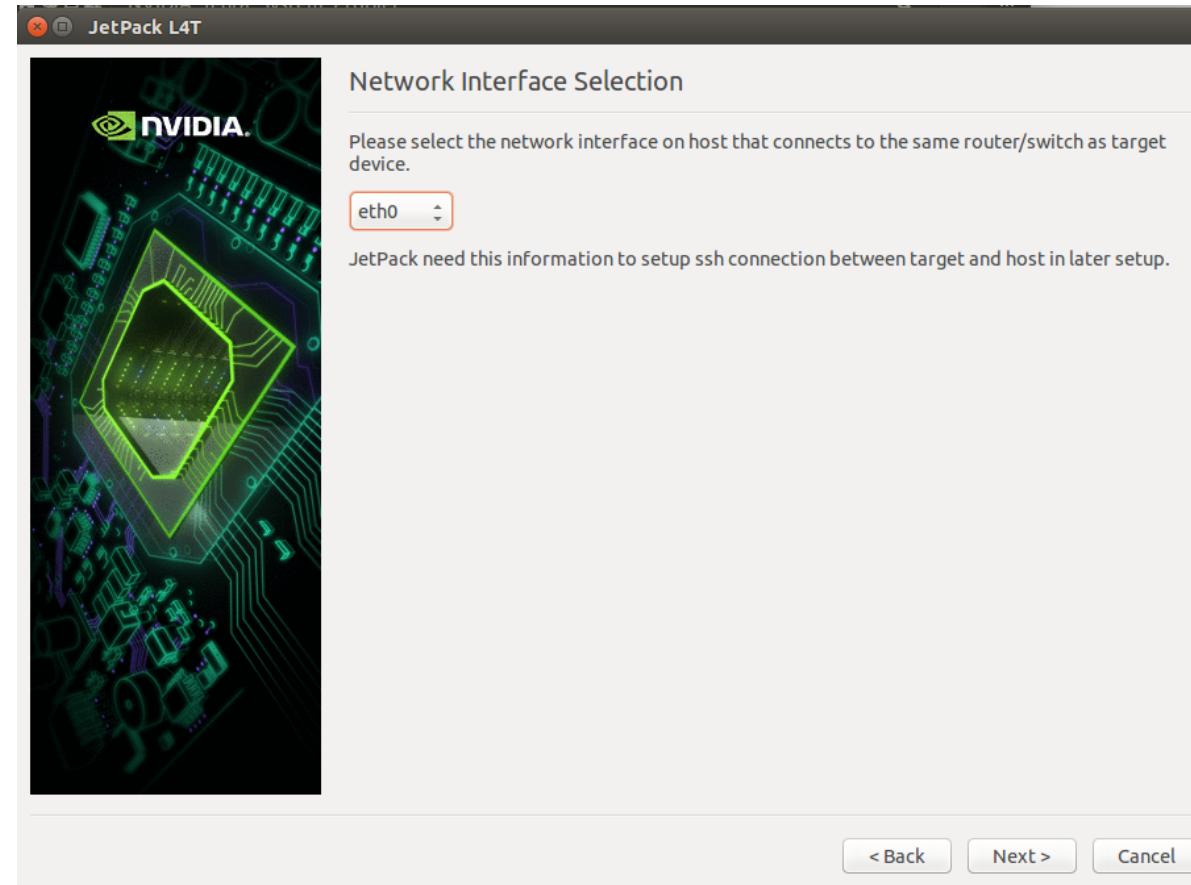
# 使用JetPack燒TK1



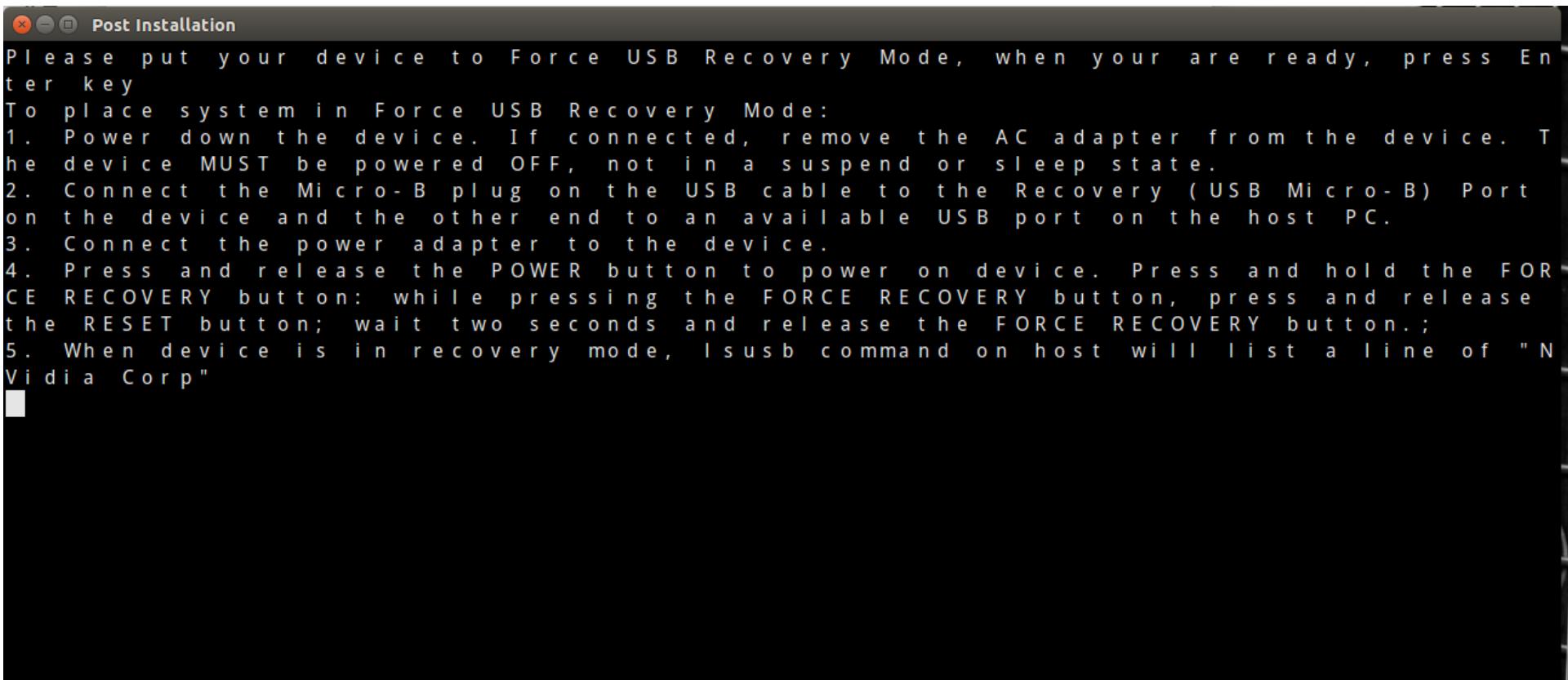
# 使用JetPack燒TK1



# 使用JetPack燒TK1



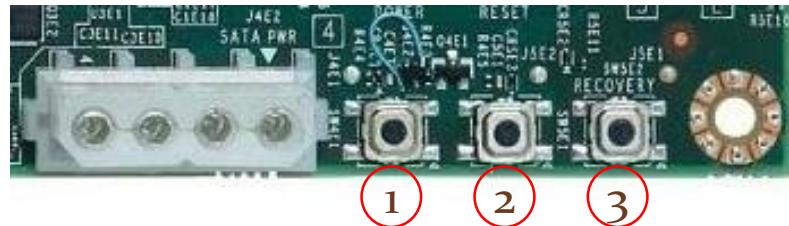
# 使用JetPack燒TK1



# 使用JetPack燒TK1

- 關閉**TK1**並拔除電源
- 接上**USB**線與網路線
- 接上電源線
- 按下電源鍵開機 (1)
- 按住還原按鍵不放(3) , 再按重開機鍵(2)此時還原鍵(3)持續按住
- 在終端機打入**lsusb**查看是否有此裝置(編號可能不同)  

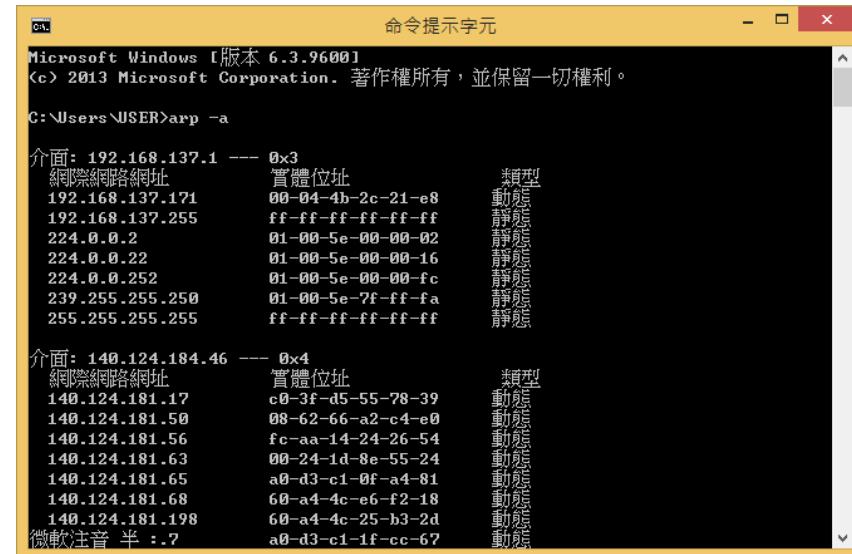
```
Bus 003 Device 005: ID 0955:7140 NVidia Corp.
```
- 對著剛剛步驟教學視窗按下迴車鍵開始燒錄



# 取得TK1的IP位址(Windows端)

- Step. 1
  - 開啟cmd
- Step. 2
  - 輸入arp -a
- Step. 3
  - 查看板子上的MAC address
  - 找到介面: 192.168.8.4
    - (IP可能與此範例不同)
  - 以MAC address找到TK1的IP

\* 此取得TK1的IP方始僅適用於授課教室的環境，若於家中是連到IP分享器的話，請查看IP分享器配發給TK1的IP為何



網際網路網址	實體位址	類型
192.168.137.1	00-04-4b-2c-21-e8	動態
192.168.137.255	ff-ff-ff-ff-ff-ff	靜態
224.0.0.2	01-00-5e-00-00-02	靜態
224.0.0.22	01-00-5e-00-00-16	靜態
224.0.0.252	01-00-5e-00-00-fc	靜態
239.255.255.250	01-00-5e-7f-ff-fa	靜態
255.255.255.255	ff-ff-ff-ff-ff-ff	靜態

網際網路網址	實體位址	類型
140.124.184.46	c0-3f-d5-55-78-39	動態
140.124.181.17	08-62-66-a2-c4-e0	動態
140.124.181.50	fc-aa-14-24-26-54	動態
140.124.181.56	00-24-1d-8e-55-24	動態
140.124.181.63	a0-d3-c1-0f-a4-81	動態
140.124.181.65	60-a4-4c-e6-f2-18	動態
140.124.181.68	60-a4-4c-25-b3-2d	動態
140.124.181.198	a0-d3-c1-1f-cc-67	動態

# 使用JetPack燒TK1



# 使用JetPack燒TK1



出現此畫面代表Jetpack成功連線到TK1，  
按下一步開始安裝上面這些套件(透過網路)，  
此需要一些時間與對外的網路連線

# 使用JetPack燒TK1

燒錄與安裝套件完成!!

不外接螢幕時  
連結到Jetson TK1的方式

# SSH是甚麼？與telnet差別為何？

Secure Shell（縮寫為SSH）為一項建立在應用層和傳輸層基礎上的安全協定，為電腦上的Shell（殼層）提供安全的傳輸和使用環境。

傳統的網路服務程式，如rsh、FTP、POP和Telnet其本質上都是不安全的；因為它們在網路上用明文傳送資料、用戶帳號和用戶口令，很容易受到中間人（man-in-the-middle）攻擊方式的攻擊。就是存在另一個人或者一台機器冒充真正的伺服器接收用戶傳給伺服器的資料，然後再冒充用戶把資料傳給真正的伺服器。

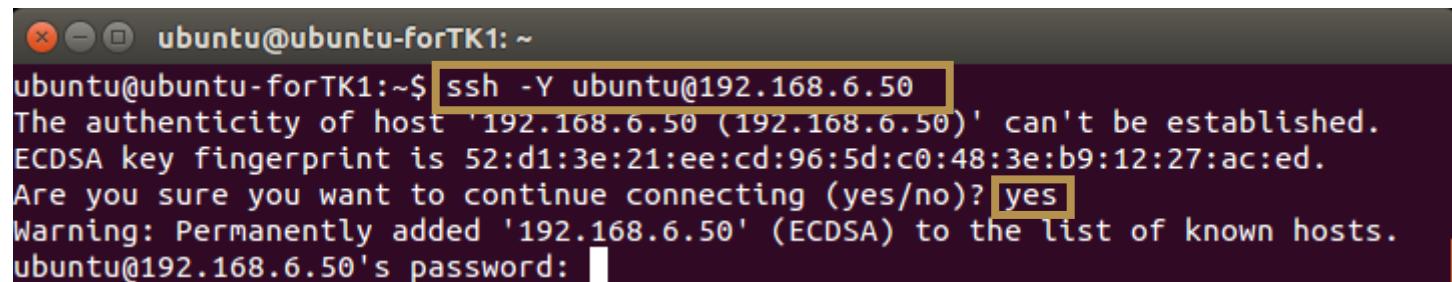
而SSH是目前較可靠，專為遠端登入對談和其他網路服務提供安全性的協定。利用SSH協定可以有效防止遠端管理過程中的資訊泄露問題。透過SSH可以對所有傳輸的資料進行加密，也能夠防止DNS欺騙和IP欺騙。

SSH之另一項優點為其傳輸的資料可以是經過壓縮的，所以可以加快傳輸的速度。SSH有很多功能，它既可以代替Telnet，又可以為FTP、POP、甚至為PPP提供一個安全的「通道」。

# 在Ubuntu虛擬機中使用ssh連到TK1

- Step. 1
  - 開啟Ubuntu虛擬機中的終端機(快捷鍵：Ctrl + Alt + T)
- Step. 2
  - 輸入 `ssh -Y <使用者>@<TK1的IP>`
    - 假設TK1的IP為192.168.6.50，使用者為ubuntu
    - 輸入: `ssh -Y ubuntu@192.168.6.50`
    - 其中 -Y 參數為支援XWindow顯示
- Step. 3
  - 第一此連線會詢問是否同意連結到此裝置，輸入yes
  - 接著輸入使用者的密碼就可以完成登入了

# 在Ubuntu虛擬機中使用ssh連到TK1

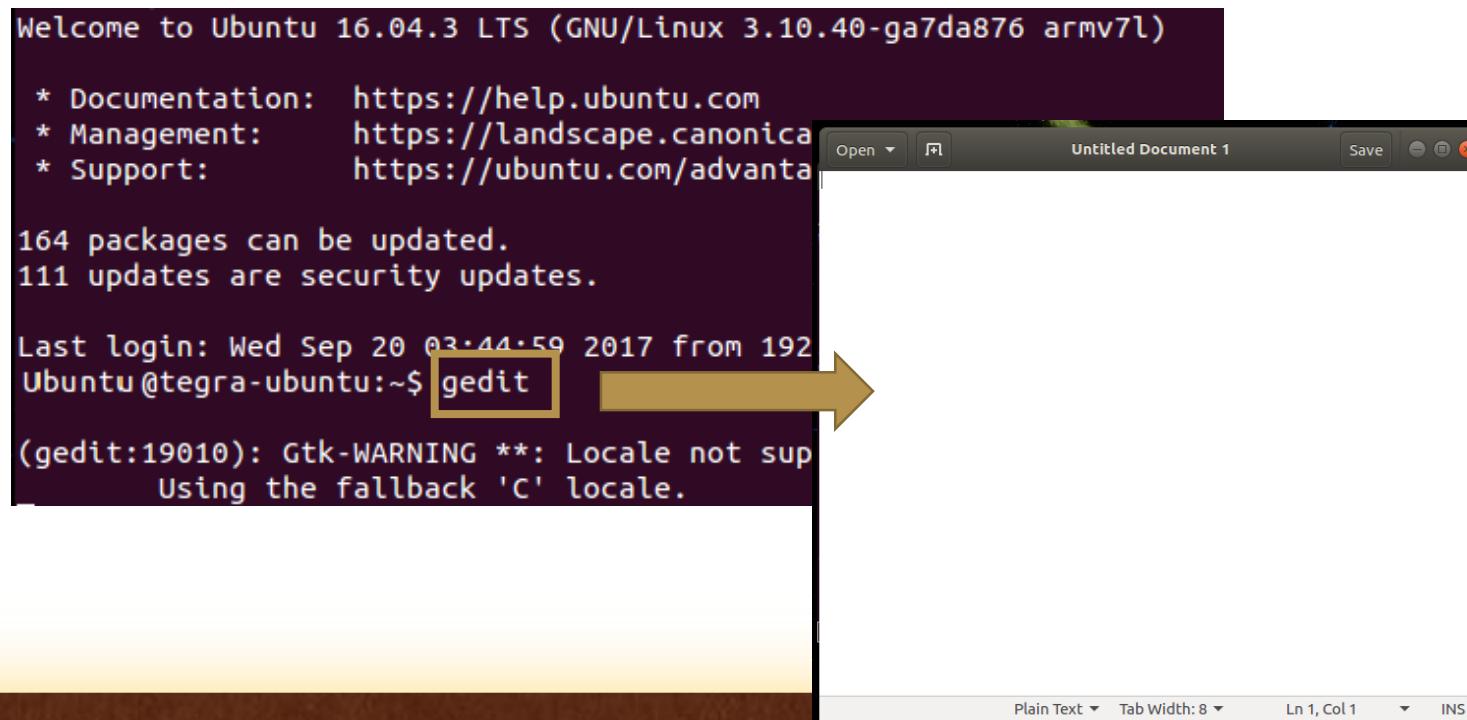


```
ubuntu@ubuntu-forTK1:~$ ssh -Y ubuntu@192.168.6.50
The authenticity of host '192.168.6.50 (192.168.6.50)' can't be established.
ECDSA key fingerprint is 52:d1:3e:21:ee:cd:96:5d:c0:48:3e:b9:12:27:ac:ed.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.6.50' (ECDSA) to the list of known hosts.
ubuntu@192.168.6.50's password:
```

輸入密碼，輸入的密碼為了安全不會顯示於上面

# 在Ubuntu虛擬機中使用ssh連到TK1

- 使用帶有參數-Y的ssh連入TK1可以映射XWindow到電腦上，我們可以輸入gedit來開啟圖型化視窗的文字編輯器看看是否成功

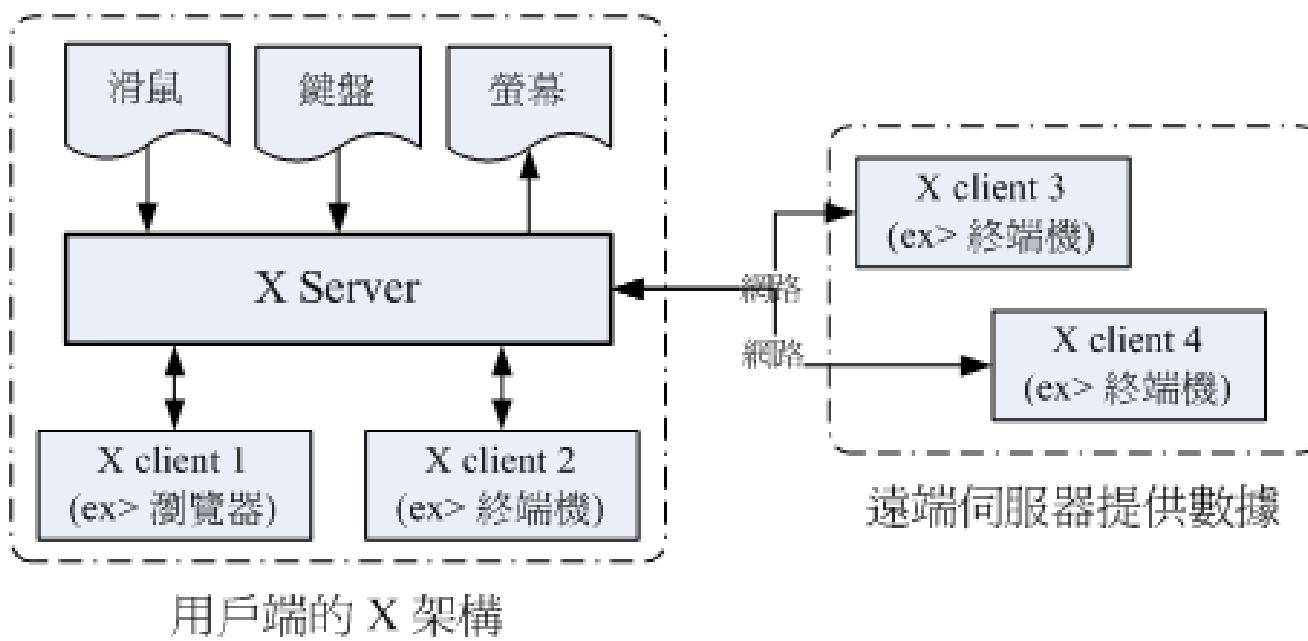


# XWindow

- X Window system為利用網路架構的圖形使用者介面軟體
  - X Server
    - 硬體管理、螢幕繪製與提供字型功能
  - X Client
    - 負責 X Server 要求的『事件』之處理
  - EX: X Client應用程式會將所想要呈現的畫面告知 X Server，由X server將管理的硬體繪製出來



# XWindow



# 傳送檔案到TK1嵌入式平台

# 傳送檔案到TK1

- 當我們需要將檔案或編譯過後的執行檔傳送到嵌入式平台TK1的時候，由於TK1內的Ubuntu預設有安裝ssh Server所以可以使用scp或sftp這兩種方式傳送檔案
  - scp (Secure copy):是指在本地主機與遠端主機或者兩台遠端主機之間基於Secure Shell ( SSH ) 協定安全地傳輸電腦檔案。適合單一的小檔案傳送
    - 複製本地端檔案到遠端主機
      - `scp <本地端檔案路徑> <使用者>@<遠端IP>:<目的端的路徑>`
      - 從遠端主機複製檔案到本地端主機
        - `scp <使用者>@<遠端IP>:<目的端的檔案路徑> <本地端存的路徑位置>`

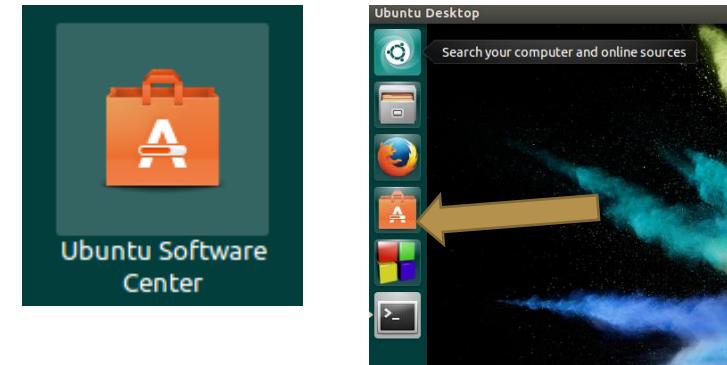
# 傳送檔案到TK1

- 當我們需要將檔案或編譯過後的執行檔傳送到嵌入式平台TK1的時候，由於TK1內的Ubuntu預設有安裝ssh Server所以可以使用scp或sftp這兩種方式傳送檔案
  - sftp (Secret File Transfer Protocol): 是一資料流連線，提供檔案存取、傳輸和管理功能的網路傳輸協定。

\* 在上課期間我們傳送檔案到TK1嵌入式平台，會比較推薦使用sftp的方式傳輸

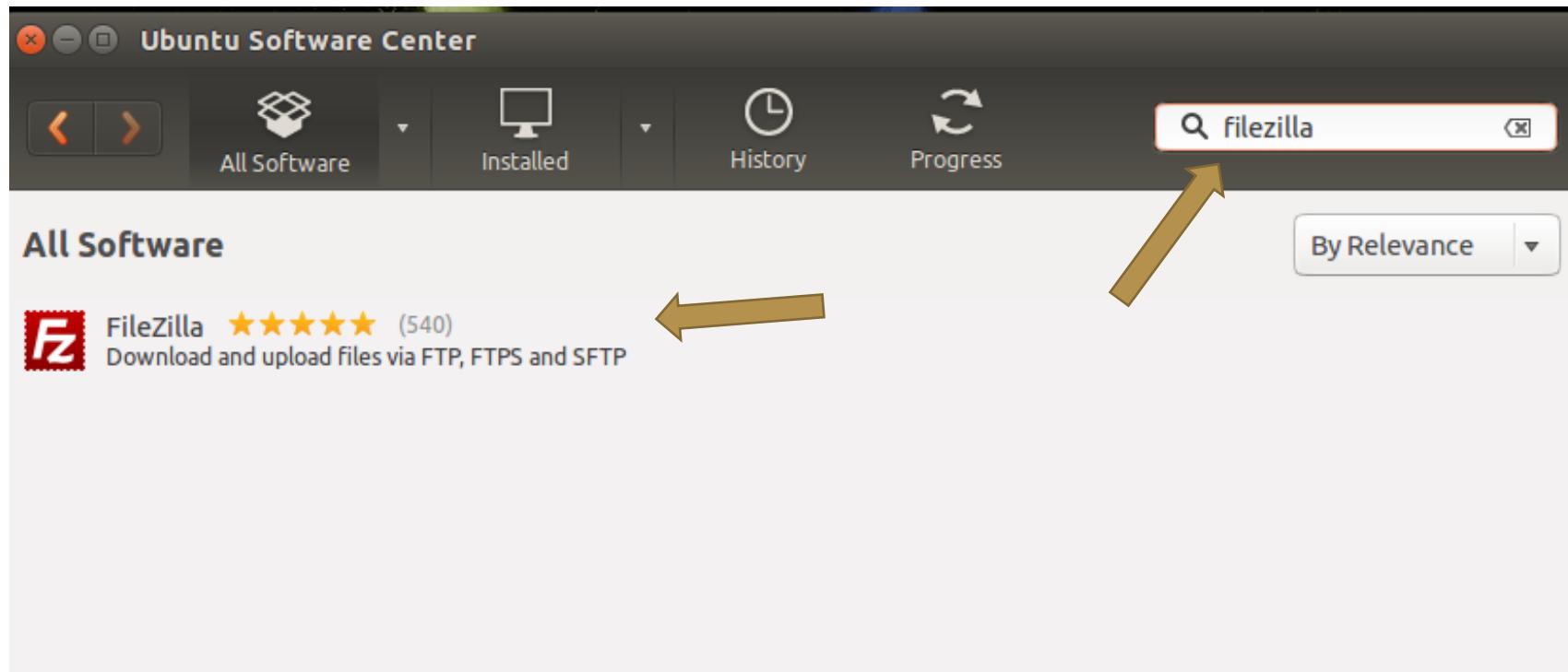
# 如何使用sftp傳送檔案到TK1

- 在我們開始使用sftp傳送檔案到TK1嵌入式平台之前，我們必須安裝FTP軟體，我們可以在Ubuntu上透過Ubuntu軟體中心來安裝filezilla這套FTP軟體
- 步驟 1：打開Ubuntu軟體中心，位於Ubuntu側邊欄或是應用程式目錄可以找到如下圖的icon



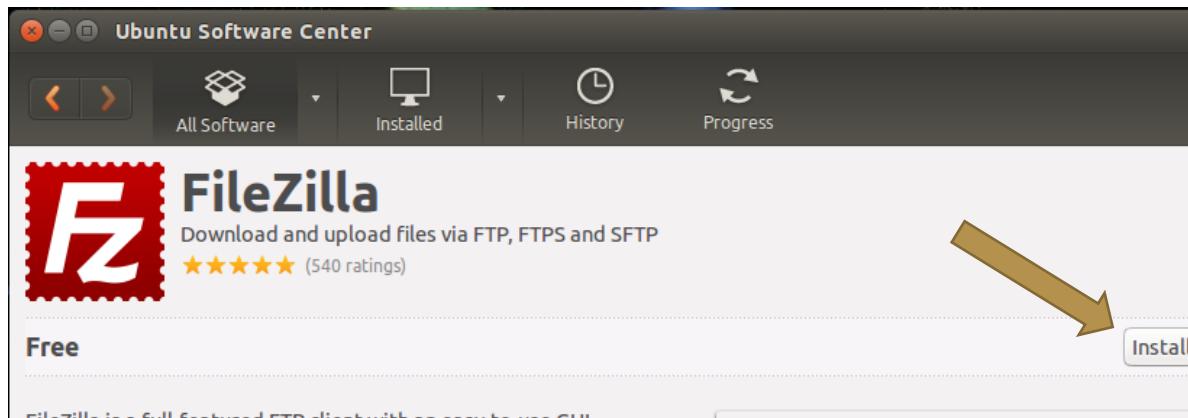
# 如何使用sftp傳送檔案到TK1

- 步驟2：打開後我們在右上角的搜尋框內打入filezille找尋此ftp軟體，並點擊兩下方找到的FileZilla軟體，就可以進入此軟體的說明頁面

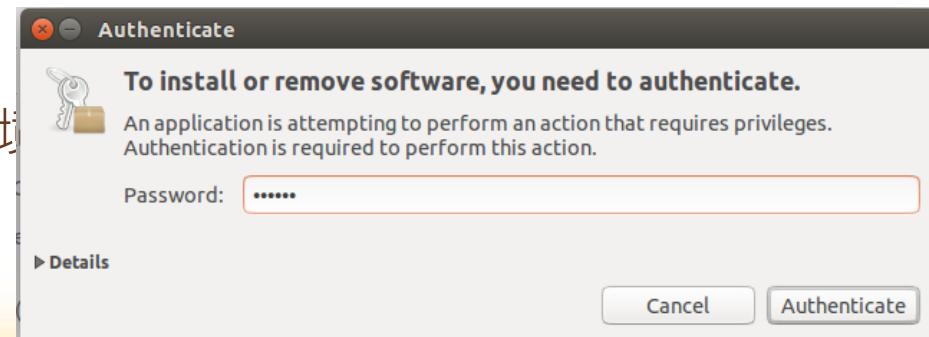


# 如何使用sftp傳送檔案到TK1

- 步驟3：點擊install開始安裝此軟體



- 步驟4：輸入使用者密碼(課程環境已開啟root權限)



# 如何使用sftp傳送檔案到TK1

- 步驟4：安裝好後會在側邊欄或應用程式放置區中看到如下圖的FileZilla的icon，請點選並執行此軟體

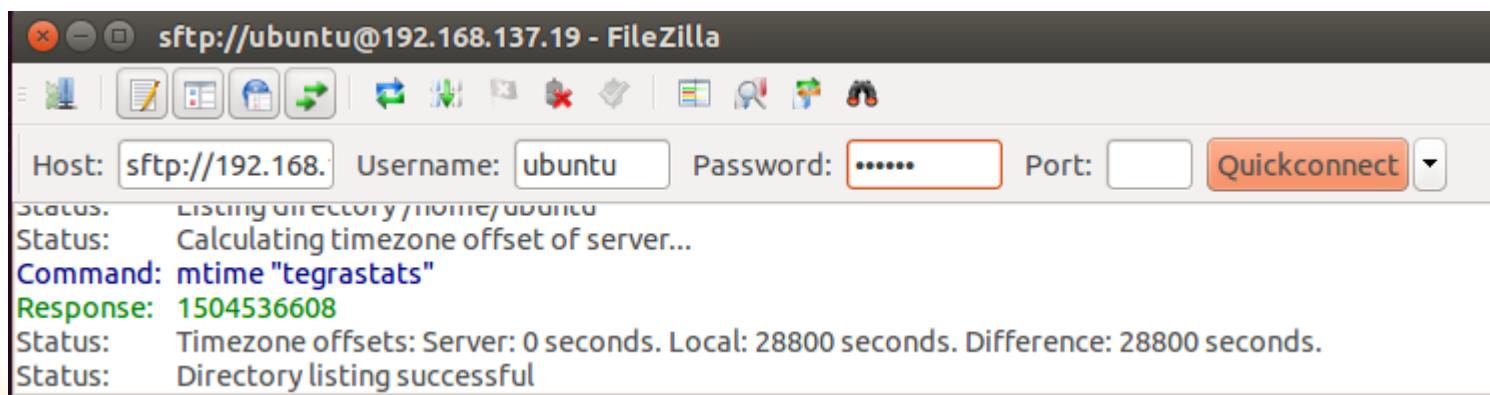


# 如何使用sftp傳送檔案到TK1

- 步驟4：開啟 FileZilla 後在上方有數個欄位
  - Host: 輸入 **sftp://<TK1的IP>**
    - 例如IP為192.168.137.19 則輸入 **sftp://192.168.137.19**
  - Username: 輸入使用者名稱(預設為ubuntu)
  - Password: 輸入使用者密碼(預設為ubuntu)
  - Port: 我們沒有更改port號，所以**此欄位空白不用輸入**

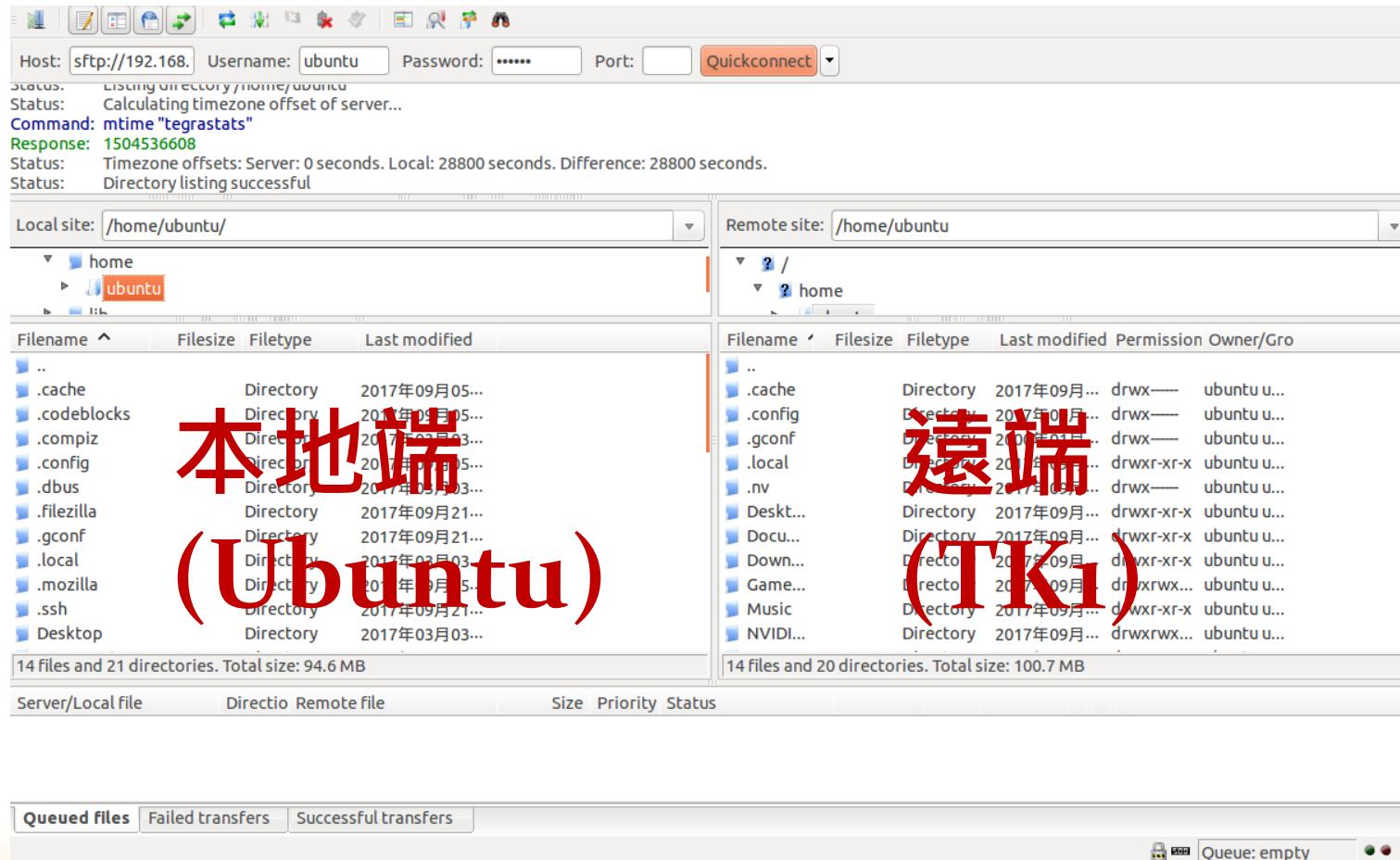
輸入完成後按下右方的Quickconnect後就可以快速連到TK1嵌入式平台了

\*若連結時有出現Unknown host key的訊息請直接按下OK



# 如何使用sftp傳送檔案到TK1

- 連線完成後可以使用簡單的拖拉放方式傳送檔案到遠端有寫入權限的目錄下



# GPIO

# GPIO簡介

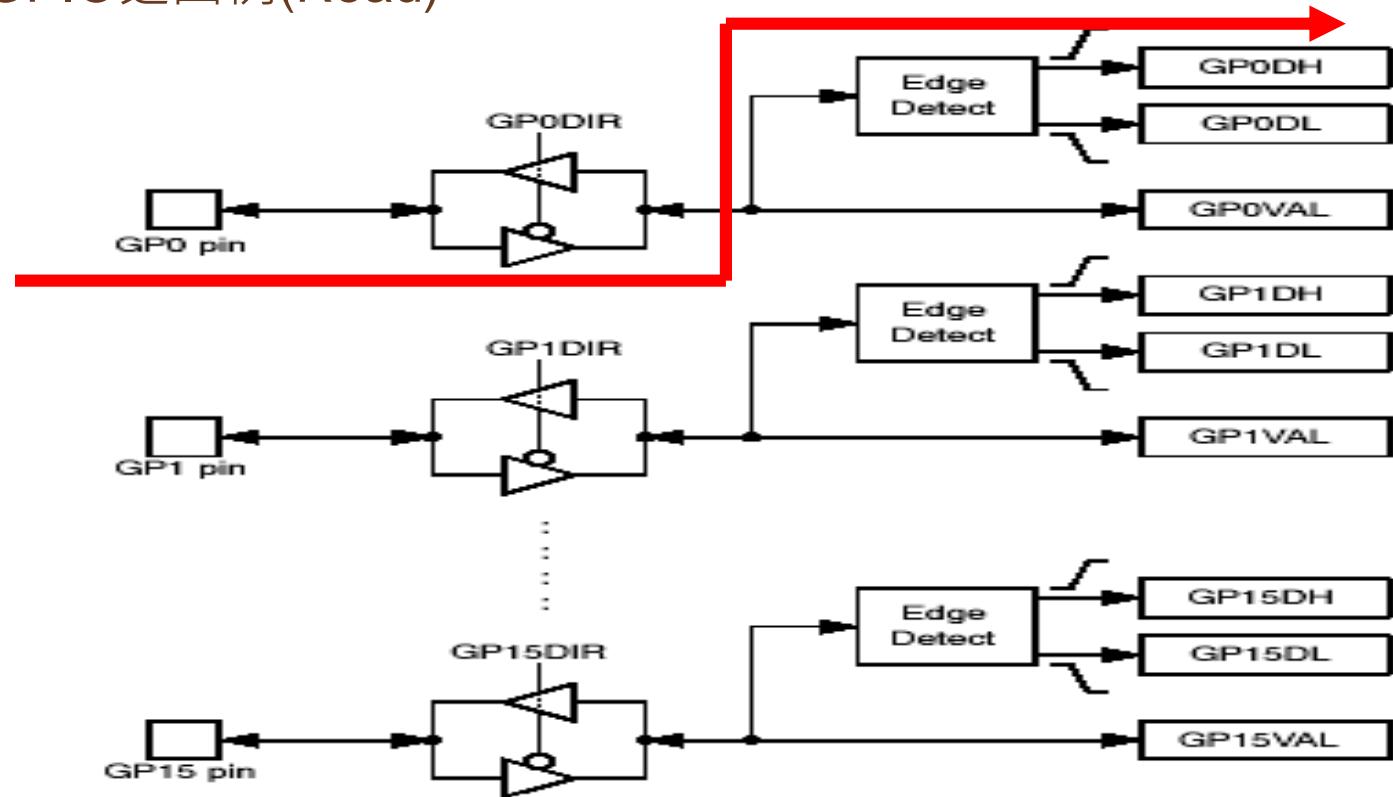
**General Purpose Input Output** (通用型之輸入輸出接口,GPIO)，在嵌入式系統中，常常有很多結構比較簡單的外部設備或電路，對於這些設備或電路來說，有的需要嵌入式平台的CPU為它提供控制，有的則需要當作輸入信號。且許多這樣的設備/電路只要求一個位元，也就是說，只要有開與關兩種狀態就夠了，比如LED的亮與滅。在實現這些控制時，使用傳統的串列埠或並列埠都不合適。所以在嵌入式平台微算機上一般都會提供GPIO。有無GPIO接口也是嵌入式平台微處理器區別於微型處理器的重要特徵之一。

# GPIO簡介

- GPIO可作為輸入與輸出
- 使用輸出功能(out)時
  - 可分別輸入High(1)或Low(0)
- 使用輸入功能(in)時
  - 可方便地讀出外部輸入的高低電壓
- 所有的配置均可通過暫存器的配置快速實現

# GPIO簡介

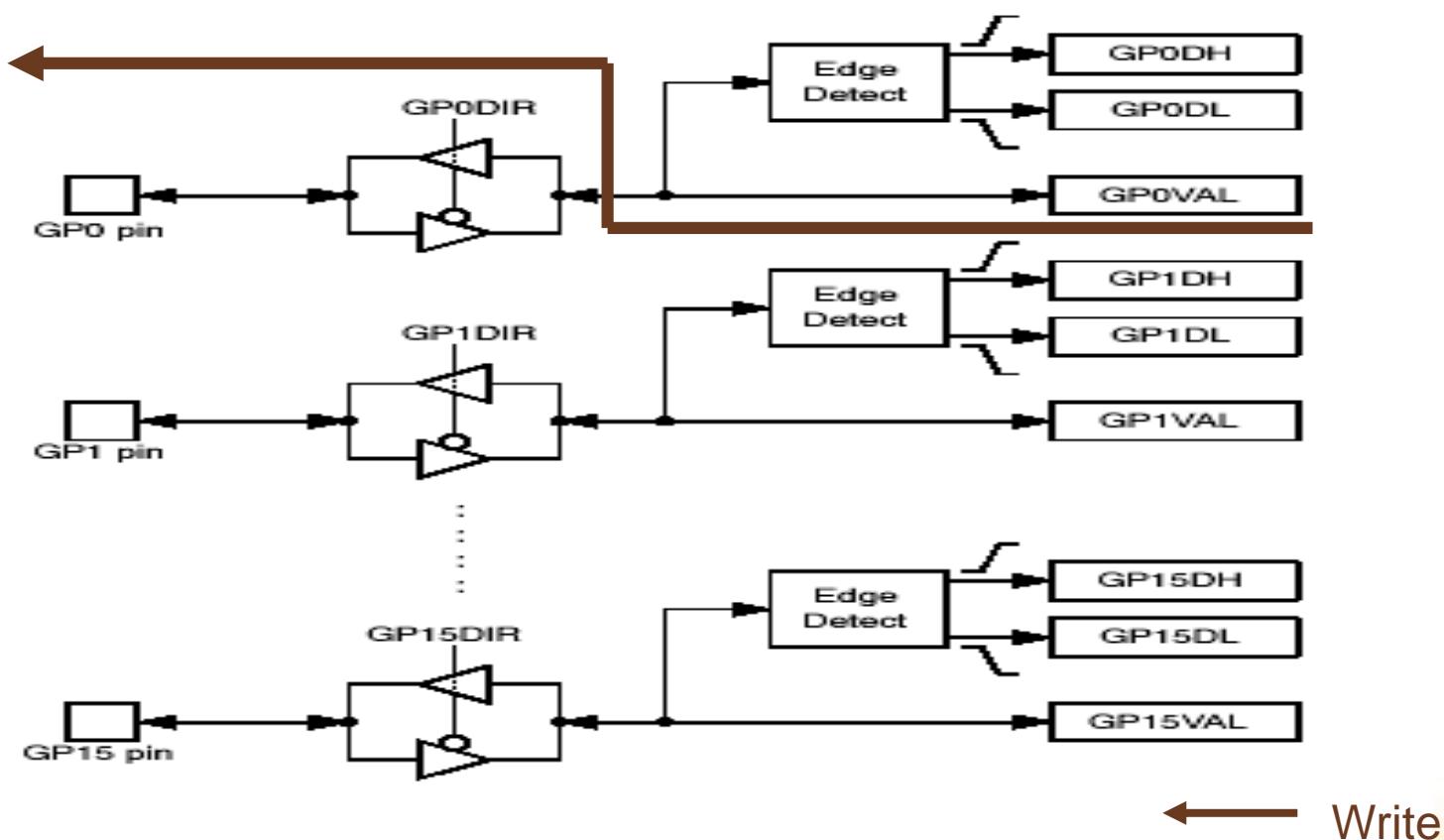
- 功能性GPIO之圖例(Read)



→ Read

# GPIO簡介

- 功能性GPIO之圖例(Write)



# GPIO簡介

- GPIO訊號至少有兩種暫存器
  - 通用IO控制暫存器
  - 通用IO數據暫存器
- 控制暫存器用來控制GPIO通訊埠的信號方向，用來決定當前通訊埠處於輸出狀態(out)還是輸入狀態(in)
- 數據暫存器中的每一位元都對應著一個GPIO接腳的狀態。
  - 當接口為輸出狀態時，可以通過數據暫存器的寫入來控制接口輸出電壓高低狀態
  - 當接口為輸入狀態時，可以通過數據暫存器讀入當前接口的電壓高低狀態

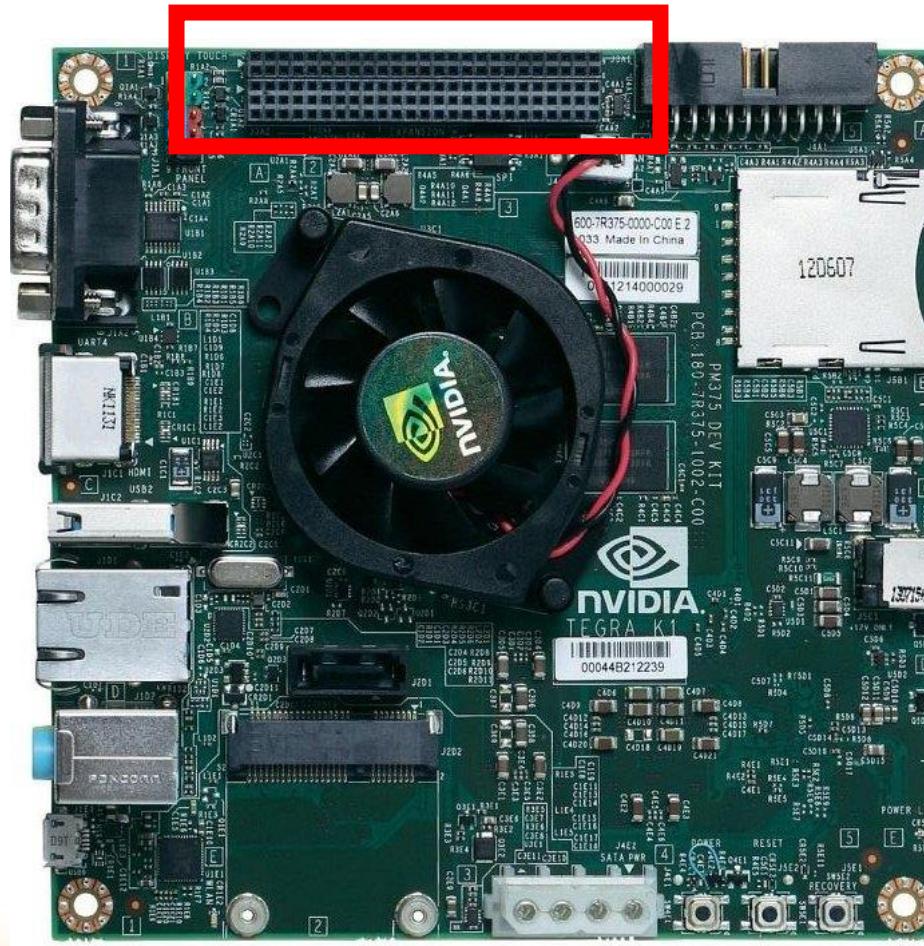
# GPIO簡介

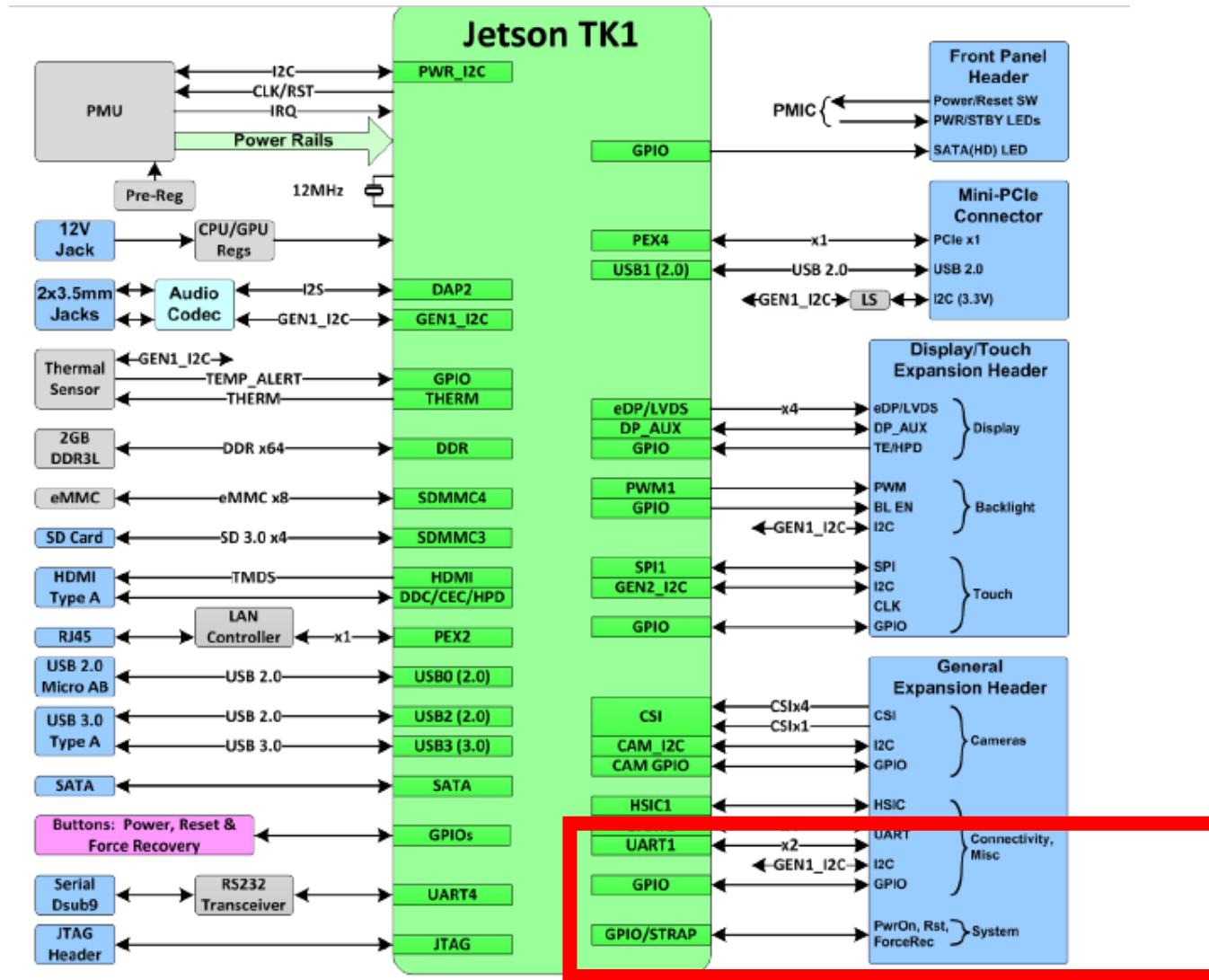
- GPIO的使用非常廣泛。用戶可以通過GPIO通訊埠與硬體進行數據交換、控制硬體(如LED、蜂鳴器、鏡頭等)工作、讀取硬體的工作狀態信號(如中斷信號)等。
- 一些傳輸協議相對簡單的低速匯流排(如I<sup>2</sup>C、SPI匯流排等)，也可以通過軟件控制一組GPIO通訊埠，從而模擬匯流排傳輸協議的通信。
- GPIO通訊埠是嵌入式平台中應用最廣泛和最靈活的接口之一。

# GPIO簡介

- 部份GPIO亦能夠在使用其他用途
  - 通常較為價廉的解決方案，即是利用特定的控制器與特定的pin腳來滿足不同的需求與用途
- 例子
  - 記憶體介面
  - I<sup>2</sup>C, SPI 介面
  - UART 信號
  - Timer 輸出
  - LCD 信號
  - 外部中斷

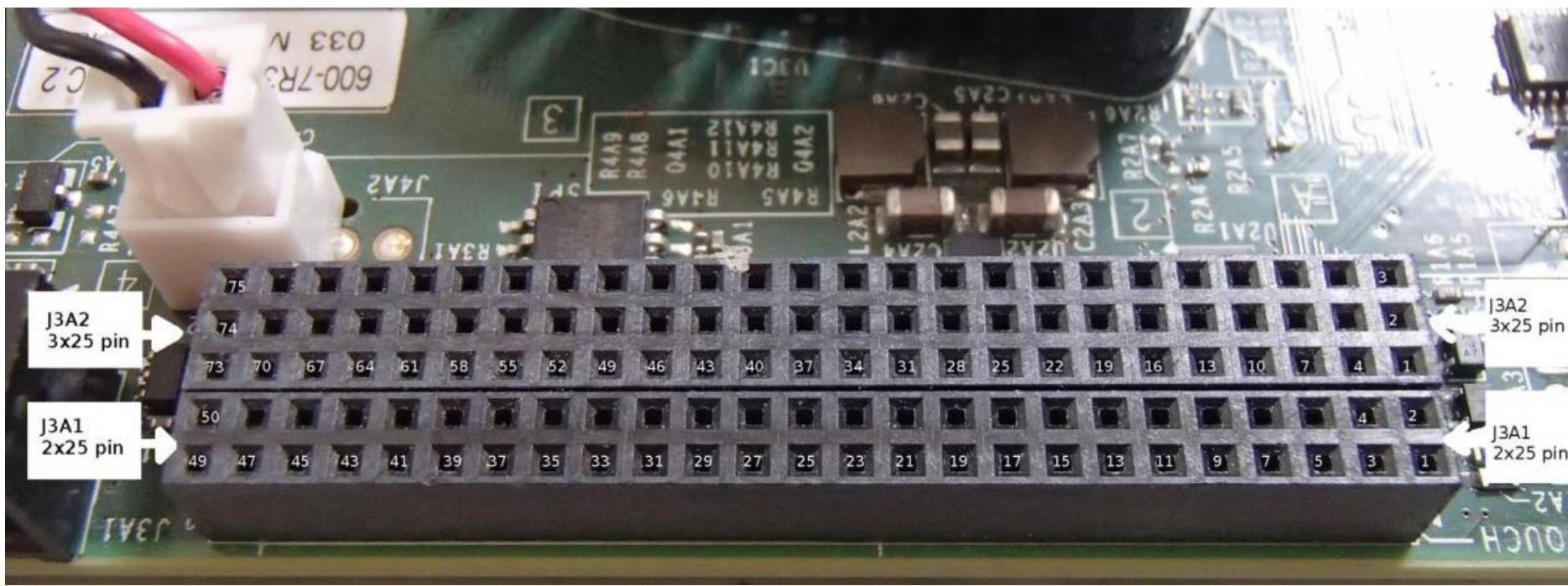
# TK1 的 GPIO 腳位介紹





# TK1 的 GPIO 腳位介紹

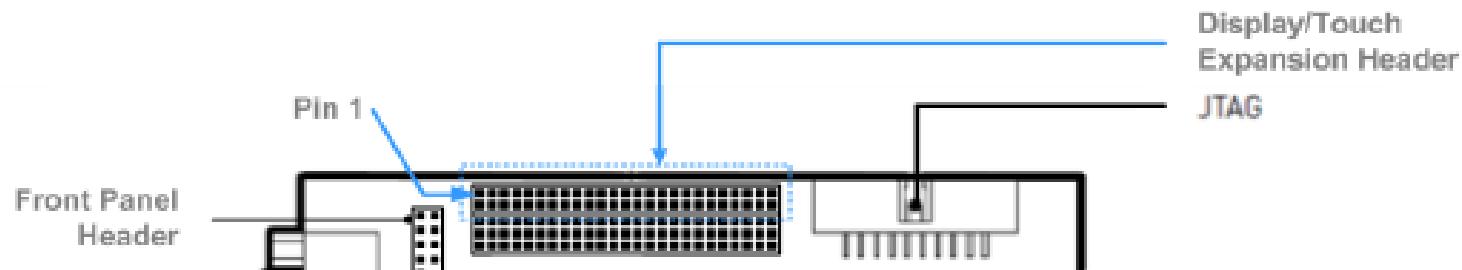
- TK1 上的 GPIO 主要分為兩大區塊
  - J3A1(靠近外側兩排一組)
  - J3A2(靠近內側三排一組)



# TK1 的 GPIO 腳位介紹

- TK1 上的 GPIO 並非所有的 GPIO 腳位都讓使用者自定 in/out 與控制使用
- TK1 只提供八個腳位可以讓使用者自行定義，在 J3A1 一個、 J3A2 七個
- J3A1 兩排 GPIO 其實主要是設計給觸控螢幕使用

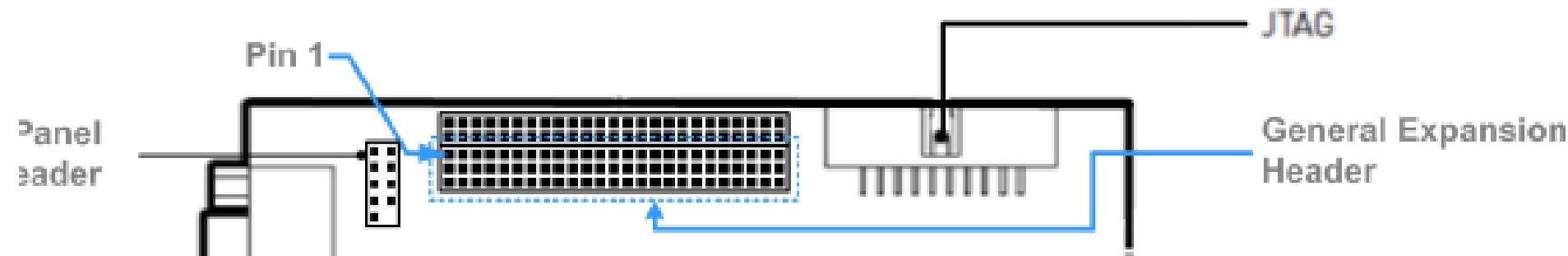
Display/Touch Header Location on Jetson TK1 (J3A1)



# TK1 的 GPIO 腳位介紹

- J3A2三排GPIO設計給一般通用型擴增接口，但實際只有7個接腳可以由使用者自行定義控制

General Expansion Header Location on Jetson TK1 (J3A2)



# TK1 的 GPIO 腳位介紹

- 可自訂的接角列表
- sysfs filename很重要，Ubuntu對應的gpio編號是依照這個

Port	sysfs filename	Physical pin	Notes
GPIO_PU0	gpio160	Pin 40 on J3A2	
GPIO_PU1	gpio161	Pin 43 on J3A2	
GPIO_PU2	gpio162	Pin 46 on J3A2	(Disabled by default)
GPIO_PU3	gpio163	Pin 49 on J3A2	
GPIO_PU4	gpio164	Pin 52 on J3A2	
GPIO_PU5	gpio165	Pin 55 on J3A2	
GPIO_PU6	gpio166	Pin 58 on J3A2	
GPIO_PH1	gpio57	Pin 50 on J3A1	

# TK1 的 GPIO 腳位介紹

- 接腳位置編號(上面兩排為J3A1、下面兩排為J3A2)
- 綠色為可自定義腳位(High: 1.8V)
- 黃色為接地(GND)
- 紅色為固定VCC腳位(5V)
- J3A1與J3A2請勿跨接

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50
1	4	7	10	13	16	19	22	25	28	31	34	37	40	43	46	49	52	55	58	61	64	67	70	73
2	5	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	53	56	59	62	65	68	71	74
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60	63	66	69	72	75

# Ubuntu 操作 TK1 上的 GPIO

- 在Ubuntu上已於Kernel層將GPIO的操作包裝過了，使用者無法使用ioctl這類的方式直接存取GPIO
- 需要透過/sys/class/gpio/下的export先啟用指定的GPIO Pin (Ubuntu只認識sysfs filename的編號)
- 啟用的sysfs filename會產生一個資料夾，編輯裡面的文件可以對對應的GPIO Pin產生相對應的控制

# Ubuntu 操作 TK1 上的 GPIO

1. 操作前必須為Root才能取得控制權

```
$ sudo su
```

2. 進入/sys/class/gpio

- # cd /sys/class/gpio/

3. 使用export啟用一個GPIO Pin，這裡示範pin 58 (sysfs filename: **gpio166**)

- 指令: echo (sysfs filename編號) > export

```
root@tegra-ubuntu:/sys/class/gpio# echo 166 > export
```

# Ubuntu 操作 TK1 上的 GPIO

3. 查看 /sys/class/gpio/ 的目錄下是否有剛剛啟用的 sysfs filename 對應資料夾(範例為 gpio166，對應 pin 腳為 58)

```
root@tegra-ubuntu:/sys/class/gpio# ls  
export  gpio143  gpio166  gpiochip0  gpiochip1016  unexport
```

4. 進入剛剛產生出來的 sysfs filename 對應資料夾，並且查看其資料夾下的內容

```
root@tegra-ubuntu:/sys/class/gpio# cd gpio166  
root@tegra-ubuntu:/sys/class/gpio/gpio166# ls  
active_low  device  direction  edge  power  subsystem  uevent  value
```

# Ubuntu 操作 TK1 上的 GPIO

- 主要會使用到direction與value
  - direction:用來設定in/out
  - value: 用來設定1/0
- 設定請用 **echo** (內容 ex: out) > (檔案 ex: direction)

```
root@tegra-ubuntu:/sys/class/gpio# cd gpio166
root@tegra-ubuntu:/sys/class/gpio/gpio166# ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

# Ubuntu 操作 TK1 上的 GPIO

5. 步驟五:將剛剛啟用的sysfs filename(範例為gpio166)資料夾下的direction設定為out，並且用cat查看direction內容是否已經為out

```
root@tegra-ubuntu:/sys/class/gpio/gpio166# echo out > direction
root@tegra-ubuntu:/sys/class/gpio/gpio166# cat direction
out
```

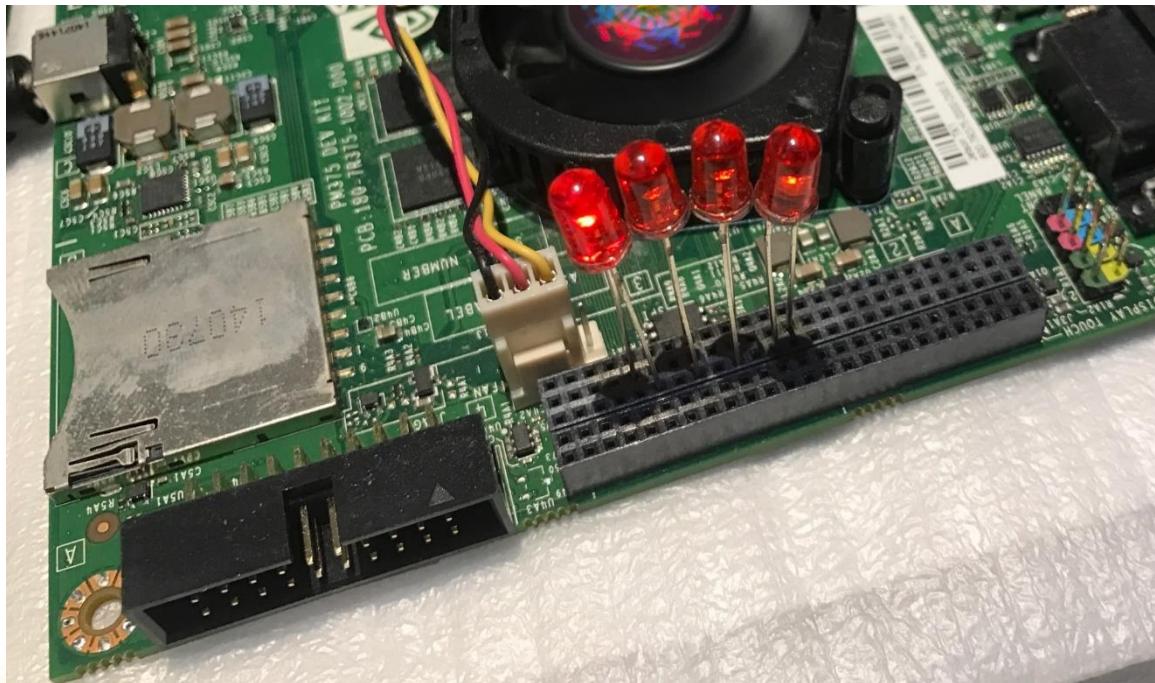
# Ubuntu 操作 TK1 上的 GPIO

6. 將剛剛啟用的sysfs filename (範例為gpio166)資料夾下的value設定為1，並且用cat查看value內容是否已經為1

```
root@tegra-ubuntu:/sys/class/gpio/gpio166# echo 1 > value
root@tegra-ubuntu:/sys/class/gpio/gpio166# cat value
1
```

# Ubuntu 操作 TK1 上的 GPIO

- 若已完成上面的操作步驟，則 sysfs filename(範例為 gpio166) 對應的 Pin 腳(範例為 Pin 58) 狀態為 1，輸出為 1.8v，若有接 LED 可以看到該腳位的 LED 是亮的



# Ubuntu 操作 TK1 上的 GPIO

7. 若要停用剛剛啟用的GPIO需要使用/sys/class/gpio/目錄下的unexport
  - 指令: echo (sysfs filename編號) > unexport

```
[root@tegra-ubuntu:/sys/class/gpio# ls
export gpio143 gpio166 gpiochip0 gpiochip1016 unexport
[root@tegra-ubuntu:/sys/class/gpio# echo 166 > unexport
[root@tegra-ubuntu:/sys/class/gpio# ls
export gpio143 gpiochip0 gpiochip1016 unexport
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 因為Ubuntu已於Kernel層將GPIO的操作封裝過了，使用者只能從上面範例提到的/sys/class/gpio/底下的介面來使用GPIO，所以在C/C++上我們不能使用ioctl的方式直接控制硬體
- 要以C/C++控制GPIO必須透過開檔/關檔/寫檔/讀檔的方式存取/sys/class/gpio/底下的介面來操作GPIO
- 以下為使用 `export unexport direction value` 的 C++ function 範例 code (僅供參考實際情形請按照使用環境需求改寫)

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- **export**

```
*****  
* gpio_export 啟用GPIO  
*****  
int gpio_export(unsigned int gpio) //傳入gpio的編號  
{  
    int fd, len;  
    char buf[64];  
  
    fd = open("/sys/class/gpio/export", O_WRONLY); //對/sys/class/gpio/export此程式做開檔讀寫  
    if (fd < 0) {  
        perror("gpio/export"); //開檔失敗  
        return fd;  
    }  
  
    len = snprintf(buf, sizeof(buf), "%d", gpio); //將應啟用的gpio編號放入buf變數中  
    write(fd, buf, len); //寫入export(啟用此gpio)  
    close(fd); //關檔  
  
    return 0;  
}
```

\* snprintf函式為格式化字串內容使用，在後方會再深入說明！

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- unexport

```
*****  
* gpio_unexport 關閉GPIO  
*****  
  
int gpio_unexport(unsigned int gpio) //傳入gpio的編號  
{  
    int fd, len;  
    char buf[64];  
  
    fd = open("/sys/class/gpio/unexport", O_WRONLY); //對/sys/class/gpio/unexport此程式做開檔讀寫  
    if (fd < 0) {  
        perror("gpio/export"); //開檔失敗  
        return fd;  
    }  
  
    len = snprintf(buf, sizeof(buf), "%d", gpio); //將應要關閉的gpio編號放入buf變數中  
    write(fd, buf, len); //寫入unexport(關閉此gpio)  
    close(fd); //關檔  
    return 0;  
}
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 設定direction

```
*****
 * gpio_set_dir 設定GPIO輸入或輸出
 *****
int gpio_set_dir(unsigned int gpio, string dirStatus) //傳入參數分別為gpio編號與欲改為out或in
{
    int fd;
    char buf[64];

    //使用snprintf將字串組合
    snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/direction", gpio);

    fd = open(buf, O_WRONLY); //對direction此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/direction"); //開檔失敗
        return fd;
    }

    if (dirStatus == "out")
        write(fd, "out", 4); //如果傳入的為out
    else
        write(fd, "in", 3); //如果傳入的不是為out(代表要為in)

    close(fd); //關檔
    return 0;
}
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 設定value

```
*****
 * gpio_set_value 設定gpio的值為1或0
 *****
int gpio_set_value(unsigned int gpio, int value) //傳入參數分別為gpio編號與欲改為狀態為1或0
{
    int fd;
    char buf[64];

    //使用snprintf將字串組合
    snprintf(buf, sizeof(buf), "/sys/class/gpio/gpio%d/value", gpio);

    fd = open(buf, O_WRONLY); //對value此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/set-value"); //開檔失敗
        return fd;
    }

    if (value == 0)
        write(fd, "0", 2); //如果傳入的狀態為0
    else
        write(fd, "1", 2); //如果傳入的狀態不是為0(代表為1)

    close(fd); //關檔
    return 0;
}
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 可能需要include的library(僅供參考)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <iostream>

using namespace std;
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 範例主程式

```
//使用範例
int main()
{
    int input;
    cout << "輸入1為啟用gpio166並設定狀態為1\n輸入2為將gpio166設定狀態為0並停用gpio166\n>>>";
    cin >> input;

    if (input == 1)
    {
        //啟用GPIO166
        gpio_export(166);
        //將此gpio166設定為輸出用途out
        gpio_set_dir(166, "out");
        //將此gpio166設定狀態為1
        gpio_set_value(166,1);
    }
    else if (input == 2)
    {
        //將此gpio166設定狀態為0
        gpio_set_value(166,0);
        //停用gpio166
        gpio_unexport(166);
    }
    return 0;
}
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 跨平台編譯並傳到TK1上執行
  - 我們需要使用`arm-linux-gnueabihf-g++`對此程式編譯成可以在TK1上執行的程式
    1. 使用終端機將目錄移動到欲編譯的cpp檔位置
    2. 使用`arm-linux-gnueabihf-g++ -o <輸出的執行檔名稱> <cpp檔案名稱>`
    3. 可以用`file`查看編譯出來的執行檔是否為arm平台使用的
      - `file <執行檔的名稱>`

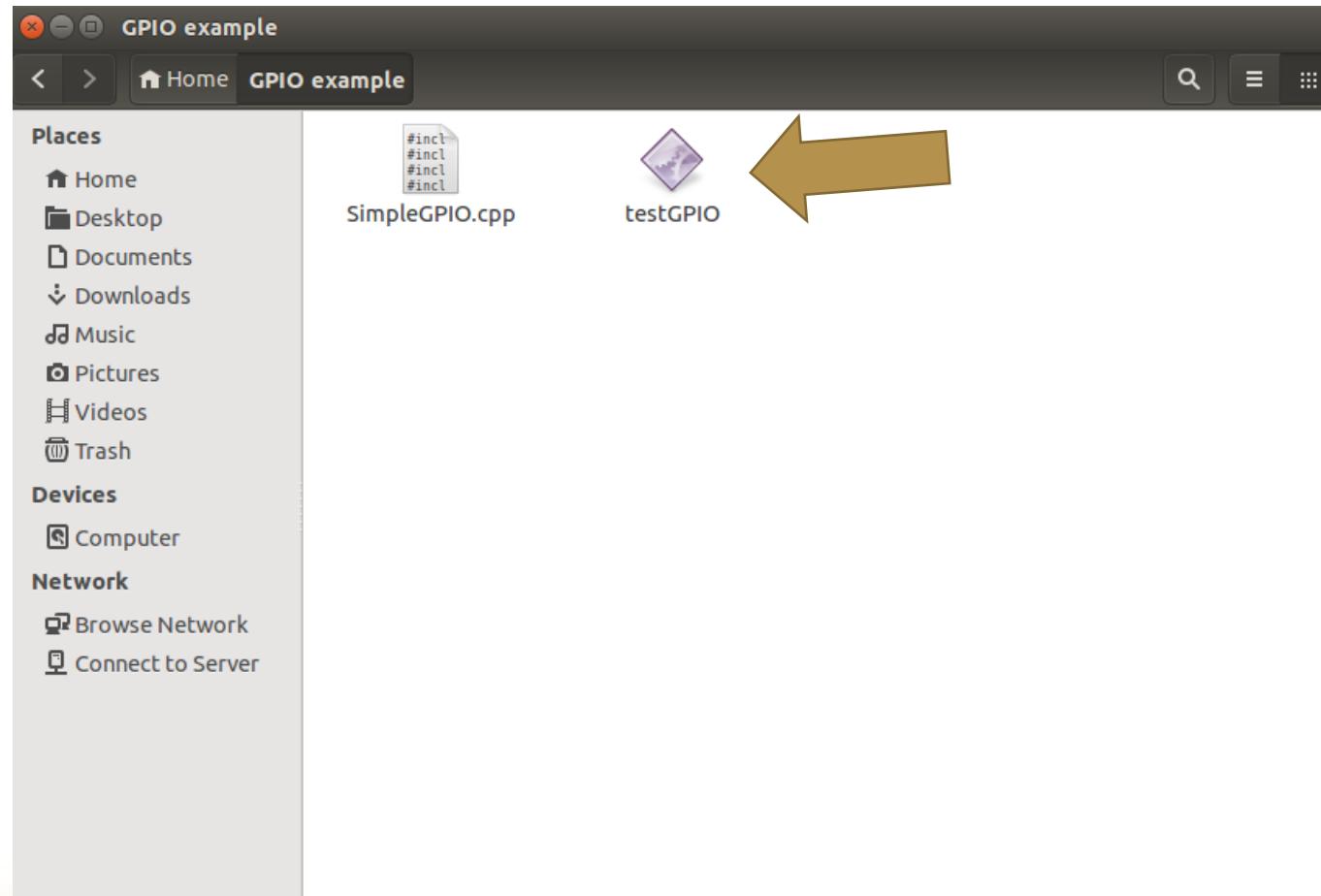
The screenshot shows a terminal window with the following session:

```
ubuntu@ubuntu-forTK1:~/GPIO example
ubuntu@ubuntu-forTK1:~$ cd GPIO\ example/
ubuntu@ubuntu-forTK1:~/GPIO example$ ls 1
SimpleGPIO.cpp
ubuntu@ubuntu-forTK1:~/GPIO example$ arm-linux-gnueabihf-g++ -o testGPIO SimpleGPI
O.cpp 2
ubuntu@ubuntu-forTK1:~/GPIO example$ ls
SimpleGPIO.cpp  testGPIO
ubuntu@ubuntu-forTK1:~/GPIO example$ file testGPIO 3
testGPIO: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically l
inked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=e05ba8796fa95c72f0
a4ea32b11f261da94cb4aa, not stripped
```

Annotations with numbers 1, 2, and 3 highlight specific parts of the terminal output:

- Annotation 1 highlights the file `SimpleGPIO.cpp`.
- Annotation 2 highlights the command `arm-linux-gnueabihf-g++ -o testGPIO SimpleGPI
O.cpp`.
- Annotation 3 highlights the output of the `file` command showing the executable details.

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO



# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 跨平台編譯完成後可以透過前面所教的sftp將此執行檔(範例為:testGPIO)傳送到TK1嵌入式平台上執行
- 透過ssh進入TK1後移動到指定目錄找到剛剛船上去的程式，要執行該程式要先確認他有執行權限(x)，可以用 ls -l查看此執行檔是否有執行權限，若沒有可以使用chmod +x <執行檔名稱>來增加此權限

A terminal window titled "ubuntu@tegra-ubuntu: ~/GPIOExample". The user runs "cd GPIOExample/" followed by "ls -l". The output shows a file named "testGPIO" with permissions "-rw-rw-r--". A yellow arrow points to the first three characters of the permission string with the text "沒權限". The user then runs "chmod +x testGPIO", and another yellow arrow points to the command with the text "增加權限". Finally, the user runs "ls -l" again, and the output shows the same file "testGPIO" but with new permissions "-rwxrwxr-x". A yellow arrow points to the first three characters of the new permission string with the text "有權限".

```
ubuntu@tegra-ubuntu:~/GPIOExample
ubuntu@tegra-ubuntu:~$ cd GPIOExample/
ubuntu@tegra-ubuntu:~/GPIOExample$ ls -l
total 16
-rw-rw-r-- 1 u n 14100 Sep 21 15:52 testGPIO
ubuntu@tegra-ubuntu:~/GPIOExample$ chmod +x testGPIO
ubuntu@tegra-ubuntu:~/GPIOExample$ ls -l
total 16
-rwxrwxr-x 1 u n 14100 Sep 21 15:52 testGPIO
```

- 有執行權限後只需要輸入./<執行檔名稱> (範例為:./testGPIO)就可以執行此程式了

A terminal window titled "ubuntu@tegra-ubuntu: ~/GPIOExample". The user runs "./testGPIO". The program outputs two lines of text: "輸入1為啟用gpio166並設定狀態為1" and "輸入2為將gpio166設定狀態為0並停用gpio166".

```
ubuntu@tegra-ubuntu:~/GPIOExample$ ./testGPIO
輸入1為啟用gpio166並設定狀態為1
輸入2為將gpio166設定狀態為0並停用gpio166
>>>
```

# 以 C++ 透過 Ubuntu 操作 TK1 上的 GPIO

- 依照上面的方式執行此程式後，當我們輸入1會發現系統告訴我們**Permission denied** 原因是要控制GPIO必須要有root權限
- 解決方式
  - 方法1(推薦):使用**sudo**執行 ex: **sudo ./testGPIO**
  - 方法2(懶人版，但要小心):使用**sudo su**將自己改為**root**就可以直接執行

```
ubuntu@tegra-ubuntu:~/GPIOExample$ sudo ./testGPIO
[sudo] password for ubuntu:
輸入1為啟用gpio166並設定狀態為1
輸入2為將gpio166設定狀態為0並停用gpio166
>>>1
ubuntu@tegra-ubuntu:~/GPIOExample$ ls /sys/class/gpio/
export gpio143 gpio166 gpiochip0 gpiochip1016 unexport
ubuntu@tegra-ubuntu:~/GPIOExample$ cat /sys/class/gpio/gpio166/value
1
```

```
ubuntu@tegra-ubuntu:~/GPIOExample$ sudo ./testGPIO
輸入1為啟用gpio166並設定狀態為1
輸入2為將gpio166設定狀態為0並停用gpio166
>>>2
ubuntu@tegra-ubuntu:~/GPIOExample$ ls /sys/class/gpio/
export gpio143 gpiochip0 gpiochip1016 unexport
```

# TK1 GPIO 相關參考資料

- **snprintf**
  - **snprintf()** 的功能是，就好像 **printf()/fprintf()/sprintf()** 一樣，給定一個 **format specifier**，以及額外的一些不定個數的參數，將會依序依指定的格式，填入 **format specifier** 裡面，以 % 開頭的欄位。**printf()** 和 **fprintf()** 會把結果，輸出到 **STDOUT** 或指定的 **FILE stream**，而 **sprintf()** 則是會把結果，填入第一個參數：一個 **C-style** 字串 **buffer**。然而，由於 **sprintf()** 在 **protocol** 設計上，沒有辦法讓實作 **sprintf()** 的程式庫，在被呼叫後，得知這個 **buffer** 的大小，因此，若結果比實際上的 **buffer** 還要長，就會造成 **buffer overflow** 的問題（所以我們範例中的 **buf** 變數宣告大小為64個字元大小）。因此，**snprintf()** 多出了第二個參數：**buffer** 的大小，以避免這個問題。
- 參考網站
  - <http://blog.xuite.net/tzeng015/twblog/113272245-snprintf>
  - <http://www.cplusplus.com/reference/cstdio/snprintf/>
  - [https://wirelessr.gitbooks.io/working-life/content/snprintf\\_miao\\_wu\\_qiong.html](https://wirelessr.gitbooks.io/working-life/content/snprintf_miao_wu_qiong.html)

# TK1 GPIO 相關參考資料

- 以下是TK1 GPIO相關參考資料，提供更進階的學習內容：
  - <http://elinux.org/Jetson/GPIO>
  - <http://elinux.org/Jetson/Tutorials/GPIO>
  - [http://elinux.org/Jetson/Tutorials/Vision-controlled\\_GPIO](http://elinux.org/Jetson/Tutorials/Vision-controlled_GPIO)

# Qt簡介

# Qt是什麼？

- Qt目前是Digia公司的產品，是一個跨平台的C++應用程式開發框架。廣泛用於開發GUI程式，這種情況下又被稱為部件工具箱。也可用於開發非GUI程式，比如控制台工具和伺服器。
- Qt使用標準的C++和特殊的代碼生成擴充功能（稱為元物件編譯器（Meta Object Compiler））以及一些巨集。
- 通過語言綁定，其他的程式語言也可以使用Qt。
  - 例如結合Python使用(PyQT)。
- 在Linux平台上C/C++ GUI函式庫常見有GTK+與Qt，GTK+以C語言開發為主，Qt則為C++為主。

# Qt是什麼？

- Qt是自由且開放原始碼的軟體，在GNU較寬鬆公共許可證（LGPL）條款下發布。所有版本都支援廣泛的編譯器，包括GCC的C++編譯器和Visual Studio。
- Qt提供也有提供商業授權版本，可以避免LGPL更改函式庫需要公開原始碼的問題
- 課堂上屬於學術用途，所以我們使用的版本為LGPL版本

# Qt跨平台

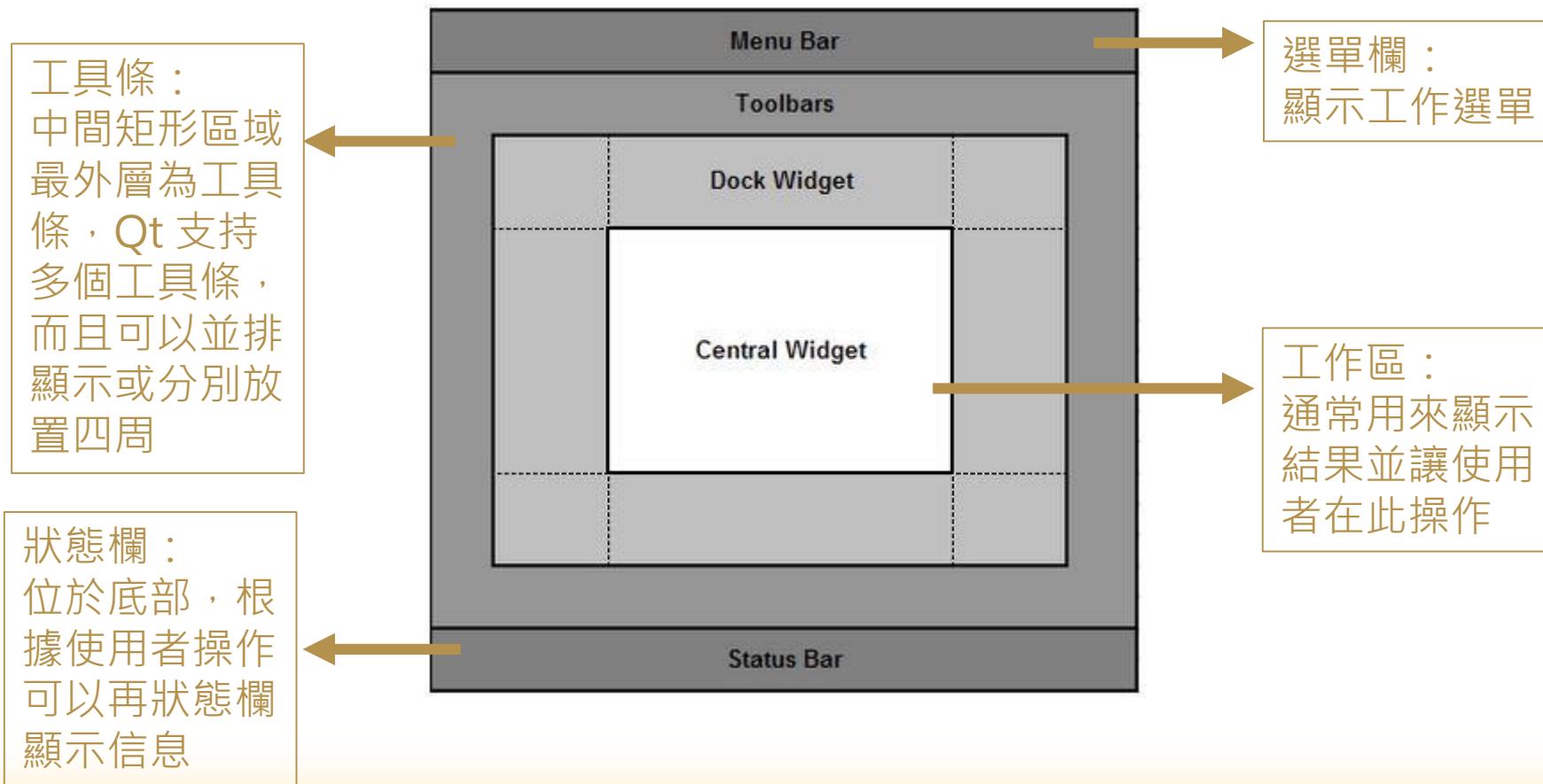
- Qt可以跨多種平台
  - Linux
  - MacOS
  - Windows
  - 嵌入式Linux系統 (Embedded Linux)
  - Windows CE
  - Android
  - iOS
  - IBM OS/2
  - 以及 Symbian/ Maemo / MeeGo 等Nokia早期系統（因為Qt曾經為Nokia的產品，後來賣出）

# QT GUI

- Qt的圖形使用者介面的基礎是QWidget。
- Qt中所有類型的GUI組件如按鈕、標籤、工具列等都衍生自QWidget，而QWidget本身則為QObject的子類別。
- Widget負責接收滑鼠，鍵盤和來自視窗系統的其他事件，並描繪了自身顯示在螢幕上。
- 每一個GUI組件都是一個widget，widget還可以作為容器，在其內包含其他Widget。

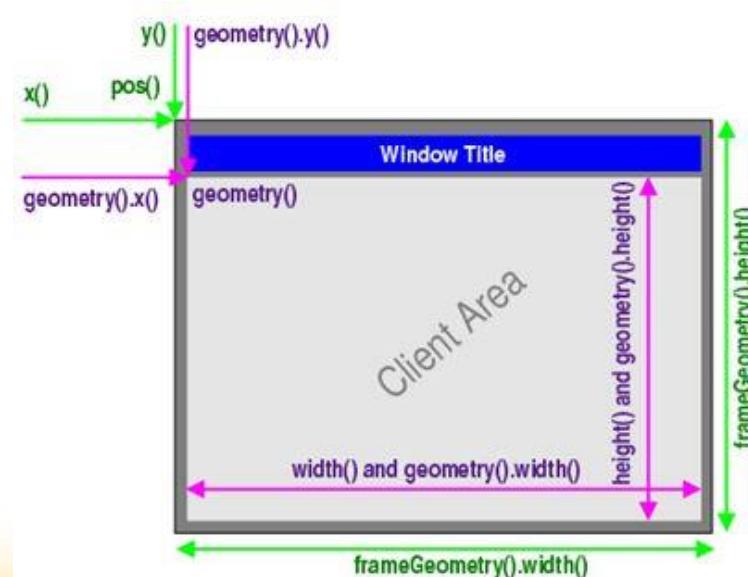
# Qt GUI 設計介紹與說明

- Qt QMainWindow 主視窗類



# Qt GUI 設計介紹與說明

- 關於Qt主視窗的視窗大小和位置操作函式，可以分為包含框架和不含框架
  - 含框架：`x()`、`y()`、`pos()`、`frameGeometry()`、`move()`。
  - 不含框架：`geometry()`、`width()`、`height()`、`rect()`、`size()`。

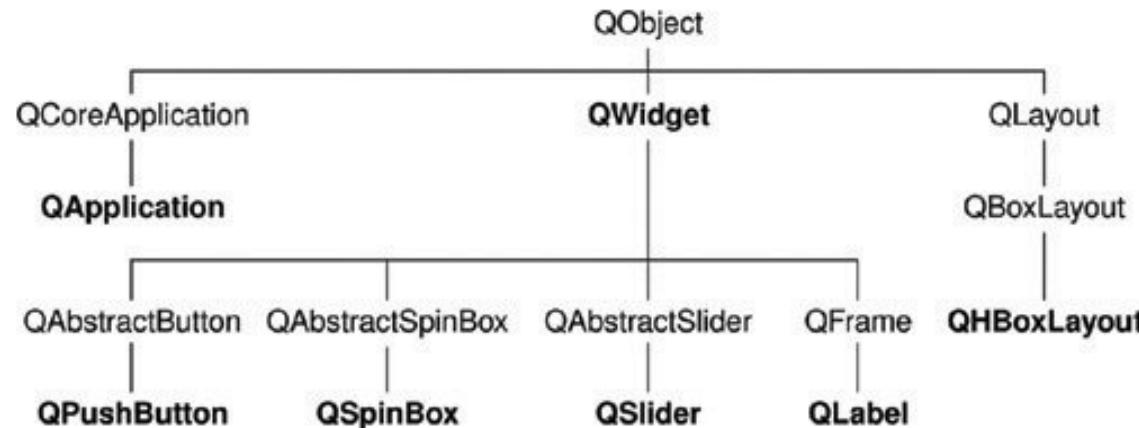


# Qt常用排版方法

- Qt內建的布局管理類型有：QHBoxLayout、QVBoxLayout、QGridLayout和QFormLayout。
  - QHBoxLayout：配置widget成橫向一列
  - QVBoxLayout：配置widget成垂直一行
  - QGridLayout：配置widget在平面網格
  - QFormLayout：配置widget用於2欄標籤- field

# Qt GUI 物件繼承

- QWidget繼承QObject，是建立使用者界面的主要類別。
- 建置時沒有指定父元件的元件稱為視窗，需指定父元件的元件稱為子元件（如：標籤、按鈕等等）
- QMainWindow就是一種視窗，也稱為頂級部件（top-level widget）



# Qt GUI 設計介紹與說明

- 通常在main()函數內只生成頂層視窗
- 在父元件的建構式內，new出我們需要的子元件
- 只要最頂層的視窗（top-level widget）能夠正確釋放記憶體，且父子元件都連接成鏈，每一個組件記憶體就能被正確釋放

## Qt物件訊號處理 (訊號與槽)

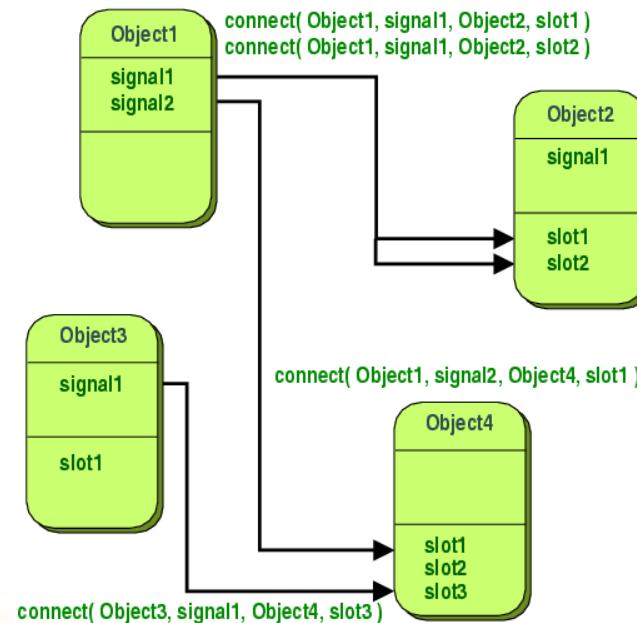
- Qt利用訊號與槽 ( **signals/slots** ) 機制取代傳統的callback來進行物件之間的溝通。
- 當操作事件發生的時候，物件會發送出一個訊號 ( **signal** ) ；而槽 ( **slot** ) 則是一個函式接受特定信號並且執行槽本身設定的動作。
- 訊號與槽之間，則透過QObject的靜態函數**connect**來連結。
- 訊號在任何執行點上皆可發射，甚至可以在槽裡再發射另一個訊號，訊號與槽的連結不限定為一對一的連結，一個訊號可以連結到多個槽或多個訊號連結到同一個槽，甚至訊號也可連接到訊號。

# Qt物件訊號處理 (Signal and Slot)

- 以往的callback缺乏類型安全，在呼叫處理函式時，無法確定是傳遞正確型態的參數。但訊號和其接受的槽之間傳遞的資料型態必須要相符合，否則編譯器會提出警告。訊號和槽可接受任何數量、任何型態的參數，所以訊號與槽機制是完全類型安全。
- 訊號與槽機制也確保了低耦合性，發送訊號的類別並不知道是哪個槽會接受，也就是說一個訊號可以呼叫所有可用的槽。此機制會確保當在"連接"訊號和槽時，槽會接受訊號的參數並且正確執行。

# Qt物件訊號處理 (Signal and Slot)

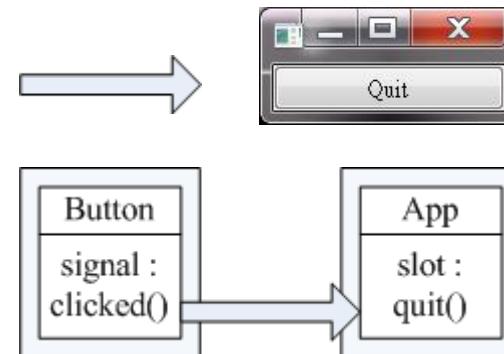
- Qt 在不同Object之間，傳遞訊息是透過Signal與Slot的機制，當一個Object發出Signal後，將會啟動相對應的Slot進行運作。



# Qt物件訊號處理 (Signal and Slot)

- 下列範例程式將說明其運作過程

```
#include <QApplication>
#include <QPushButton>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QPushButton *button = new QPushButton("Quit");
    QObject::connect(button, SIGNAL(clicked(bool)), &a, SLOT(quit()));
    button->show();
    return a.exec();
}
```



- 首先建立兩個物件之間的溝通連結

Button中signal : clicked() // 當按鈕被按下時觸發事件

App中slot : quit() // 當被呼叫時離開程式

- 執行後，當Quit button被按下時，將會傳送訊號signal : clicked()，並啟動slot : quit()，當完成後便會關閉視窗

# Qt更多參考資料

- Qt介紹
  - <https://zh.wikipedia.org/wiki/Qt>
- Qt物件種類與使用
  - <http://doc.qt.io/qt-5/qtwidgets-index.html>
- LGPL版本開發者網頁
  - <https://www1.qt.io/developers/>

•————•

# 使用Qt Creator 開發Qt GUI程式

•————•

# Qt Creator介紹

- Qt Creator 是一款跨平台的整合開發環境，特別針對Qt開發者，是Qt SDK組成的一部分，可執行於Windows, Linux/X11及Mac OS X等桌面作業系統，允許開發者為多桌面環境及行動裝置平台建立應用程式。
- Qt Creator 包括一個可視化偵錯工具和整合的 GUI 版面和外形設計師，這個編輯器的功能包括語法高亮度顯示和自動完成。
- Qt Creator 在 Linux 上，使用 GCC 的 C++ 編譯器。在 Windows，預設安裝它可以用 MinGW 或 MSVC。
- Qt Creator 整合了跨平台自動化建構系 qmake 與 Cmake，可以匯入不使用 qmake 或 CMake 的專案，並指定 Qt Creator 忽略你的建構系統。

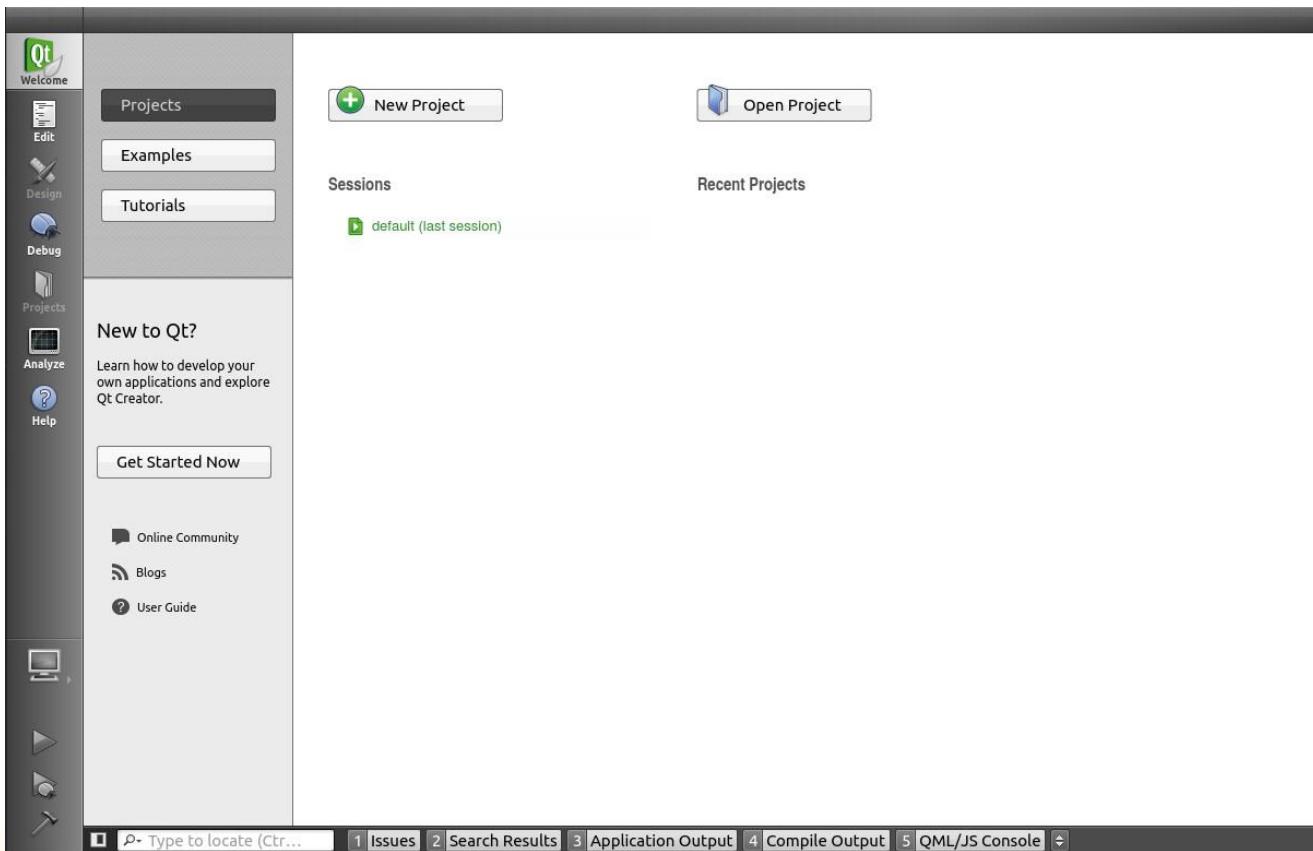
# Qt Creator

- 於課程提供的Ubuntu虛擬機上已經有安裝好Qt Creator了可以直接開啟使用，若於Ubuntu系統上沒有安裝Qt Creator的話可以執行下列指令進行安裝
  - `sudo apt-get install qtcreator`
  - 也可於軟體中心搜尋並安裝
- 安裝好後可於應用程式放置區找到Qt Creator



# Qt Creator

- 打開Qt Creator

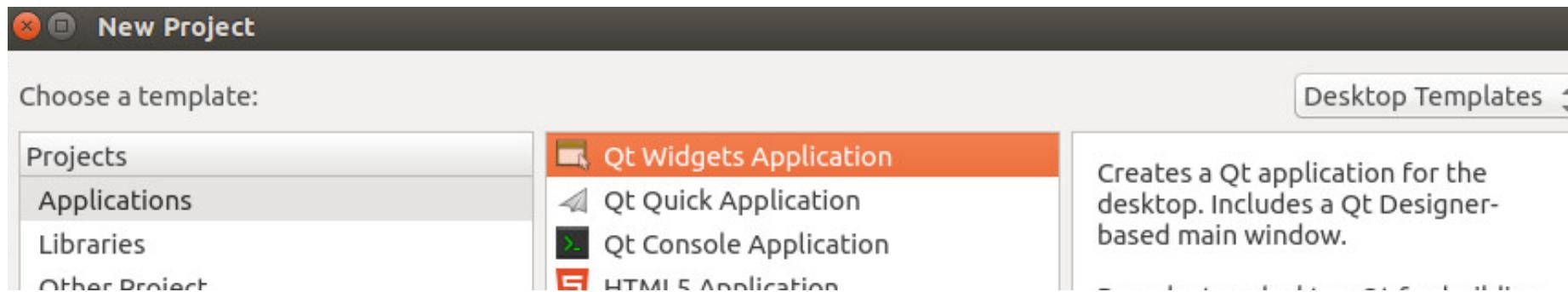


# Qt Creator

- 按下 New Project 建立新專案

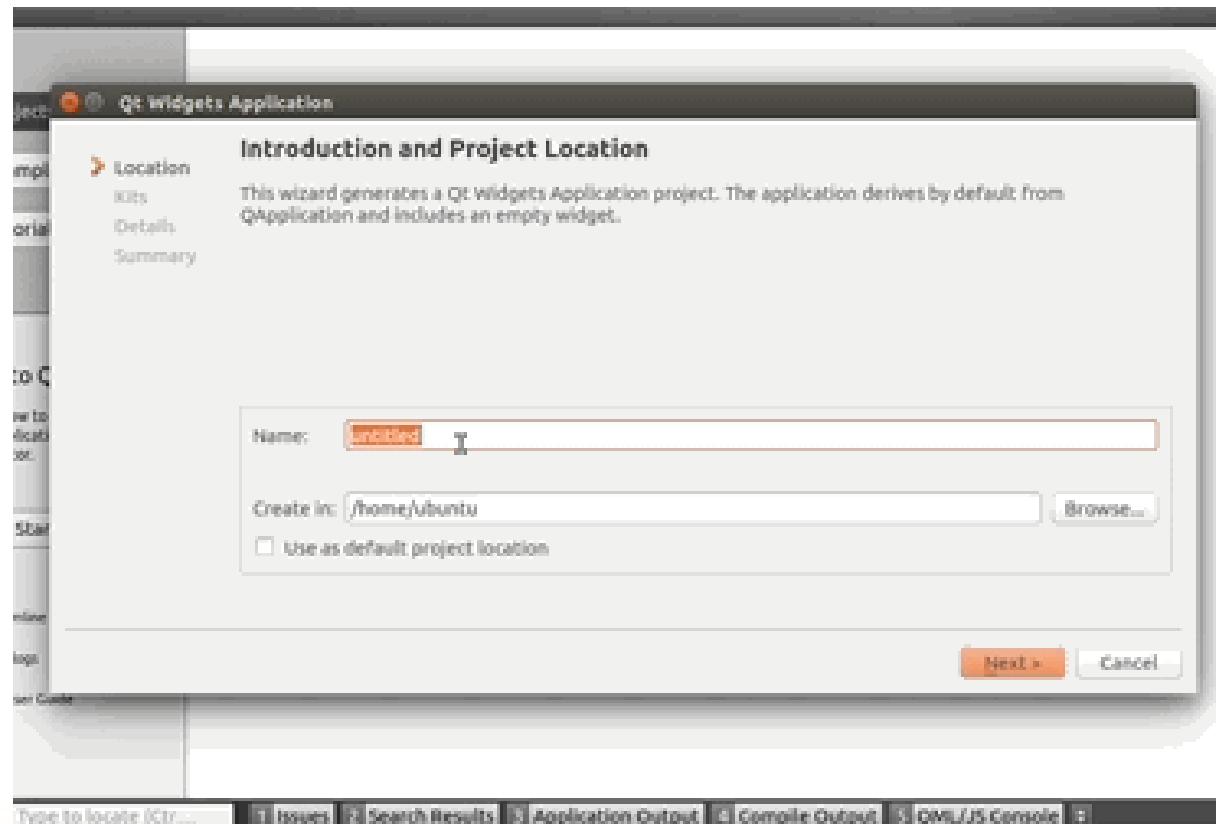


- 選擇 Qt Widgets Application



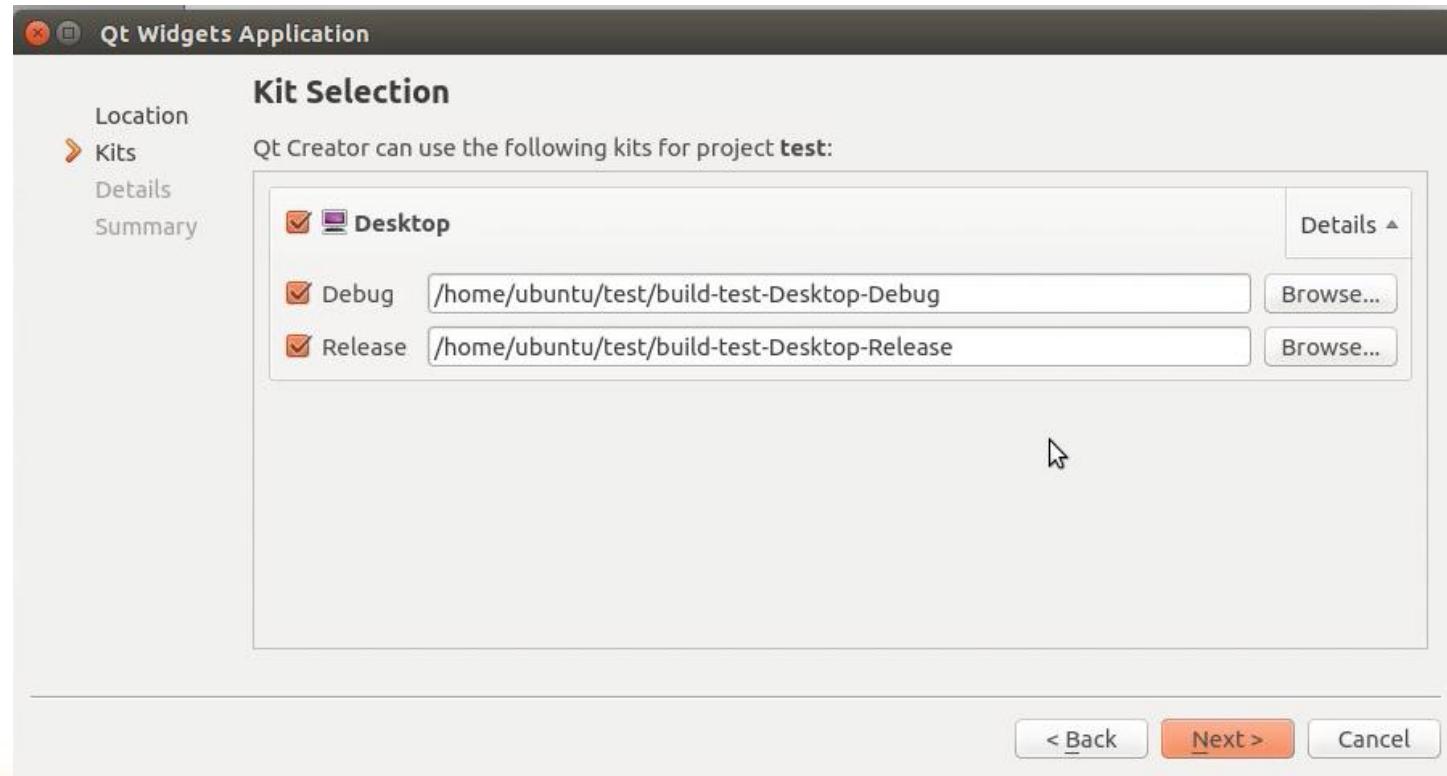
# Qt Creator

- 選擇專案名稱與要存放的資料夾路徑



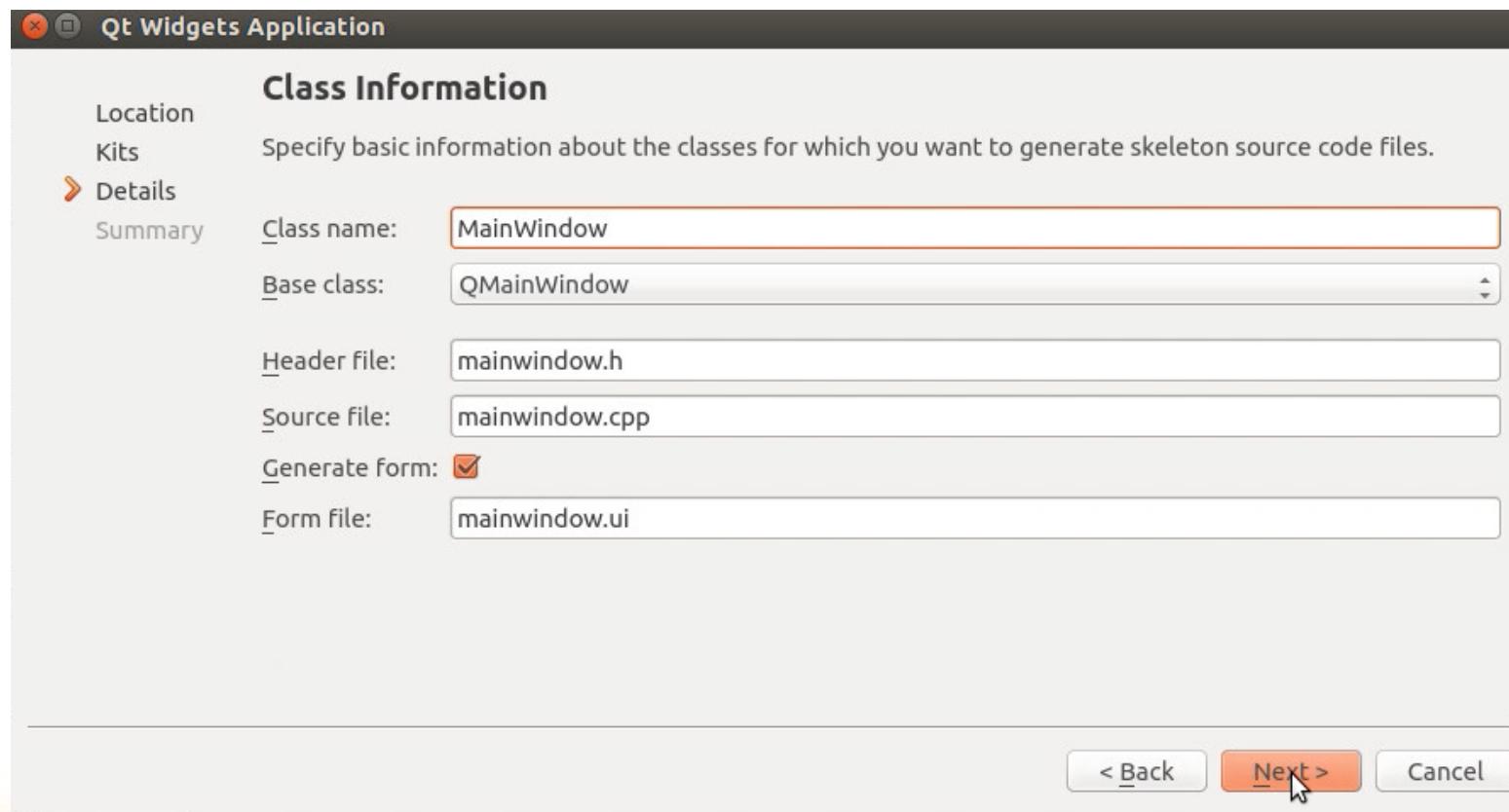
# Qt Creator

- 對於編譯的細節做些設定 (我們這次範例是要編譯出Ubuntu虛擬機上執行的程式所以不用更改預設設定)



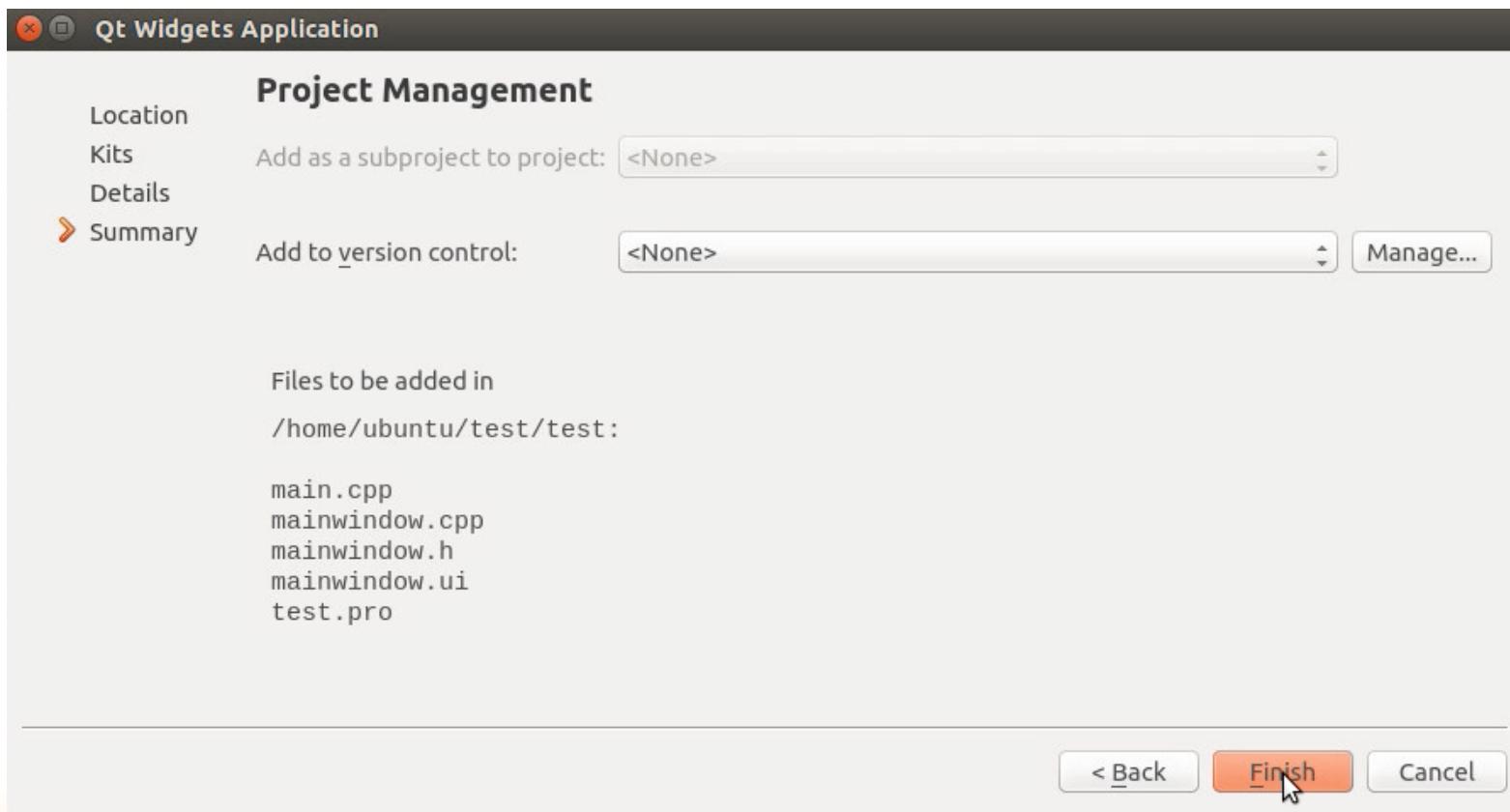
# Qt Creator

- 設定主視窗class (不用更改預設設定)

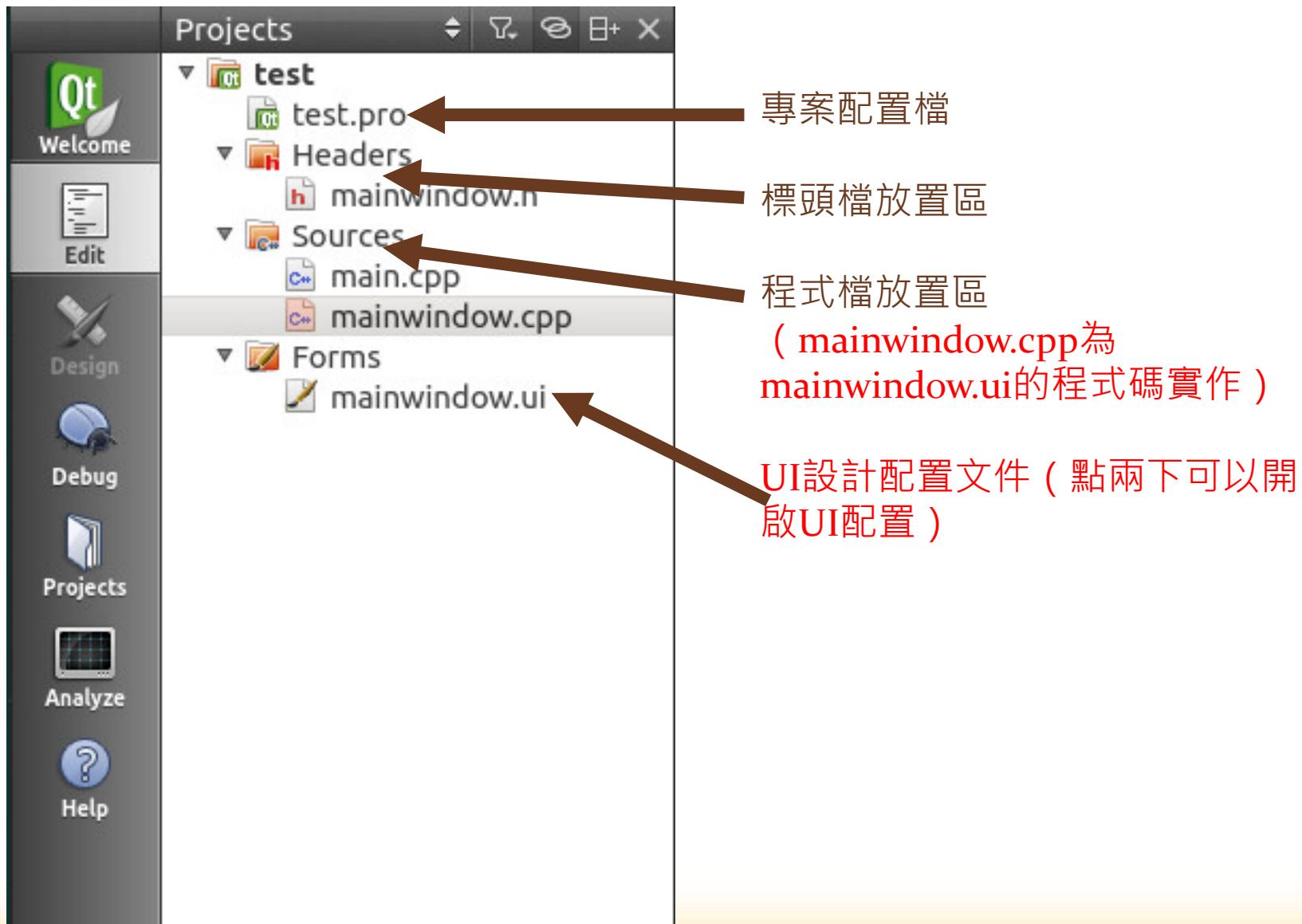


# Qt Creator

- 完成設定 Finish

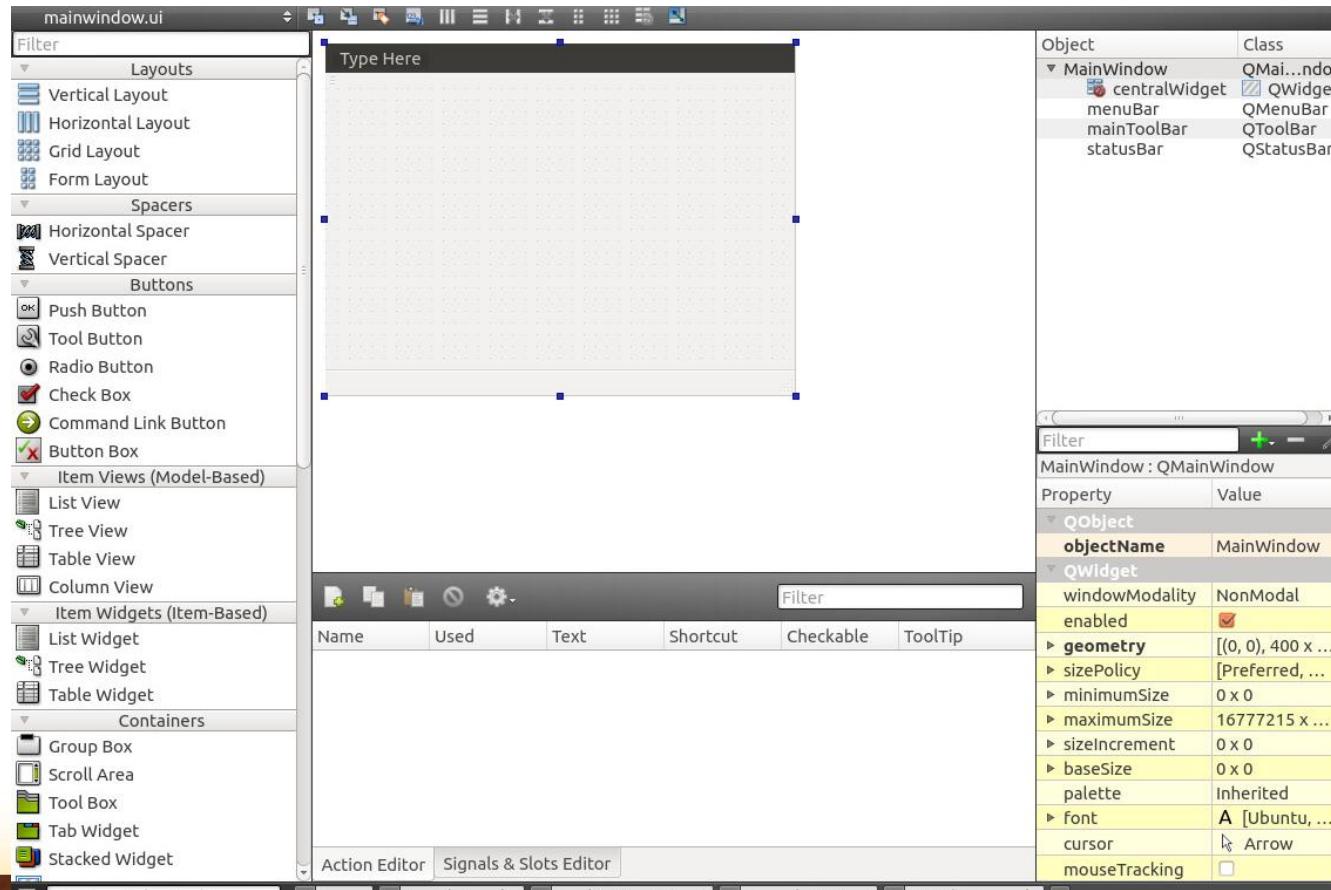


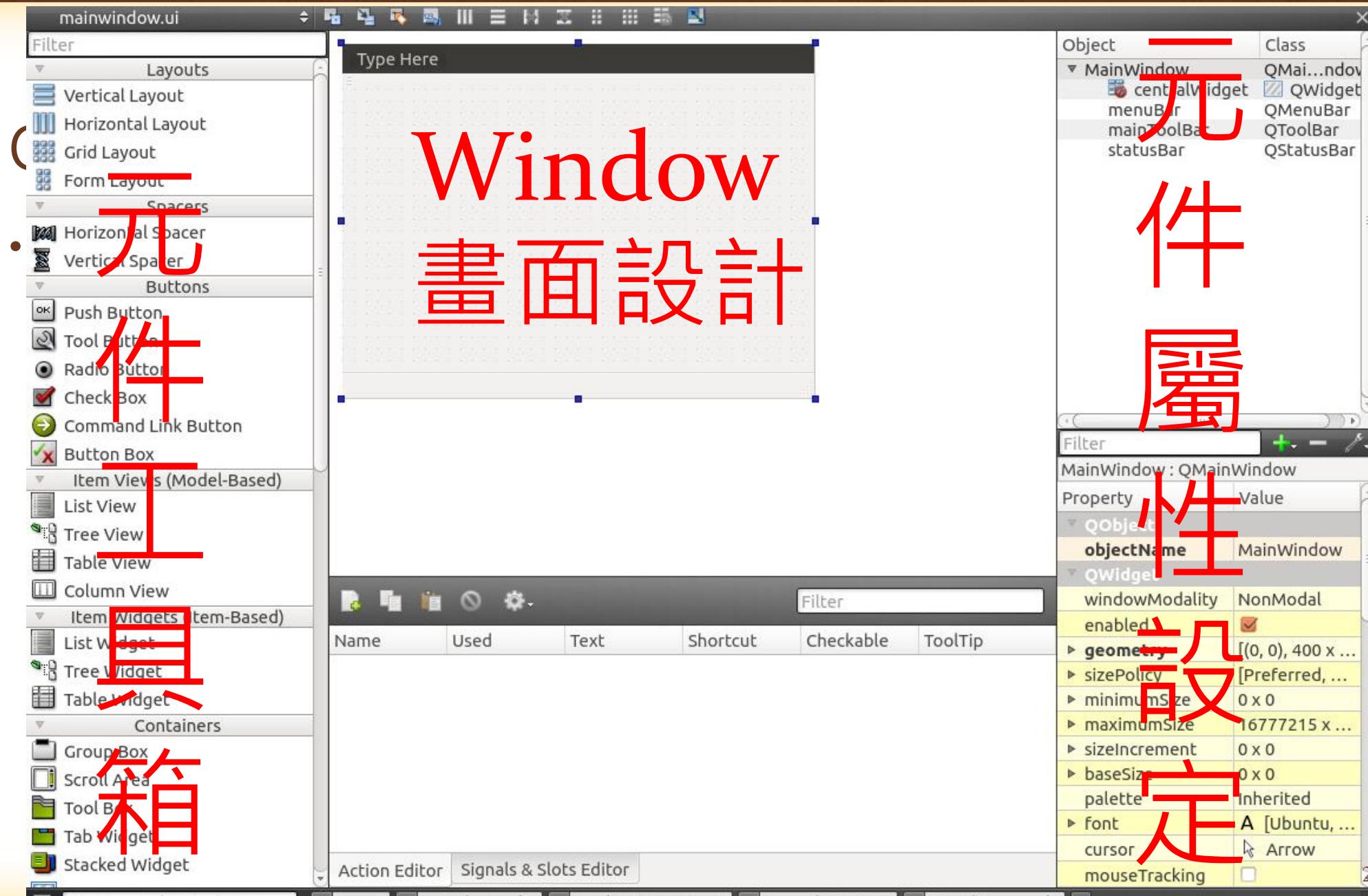
# Qt Creator



# Qt Creator

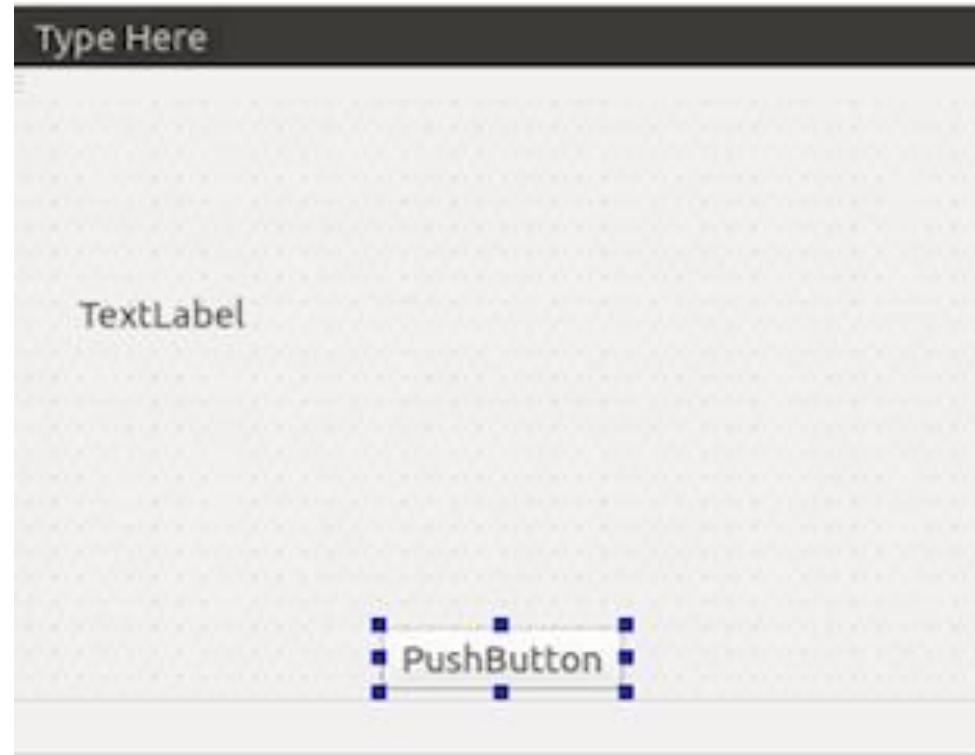
- 點兩下mainwindow.ui





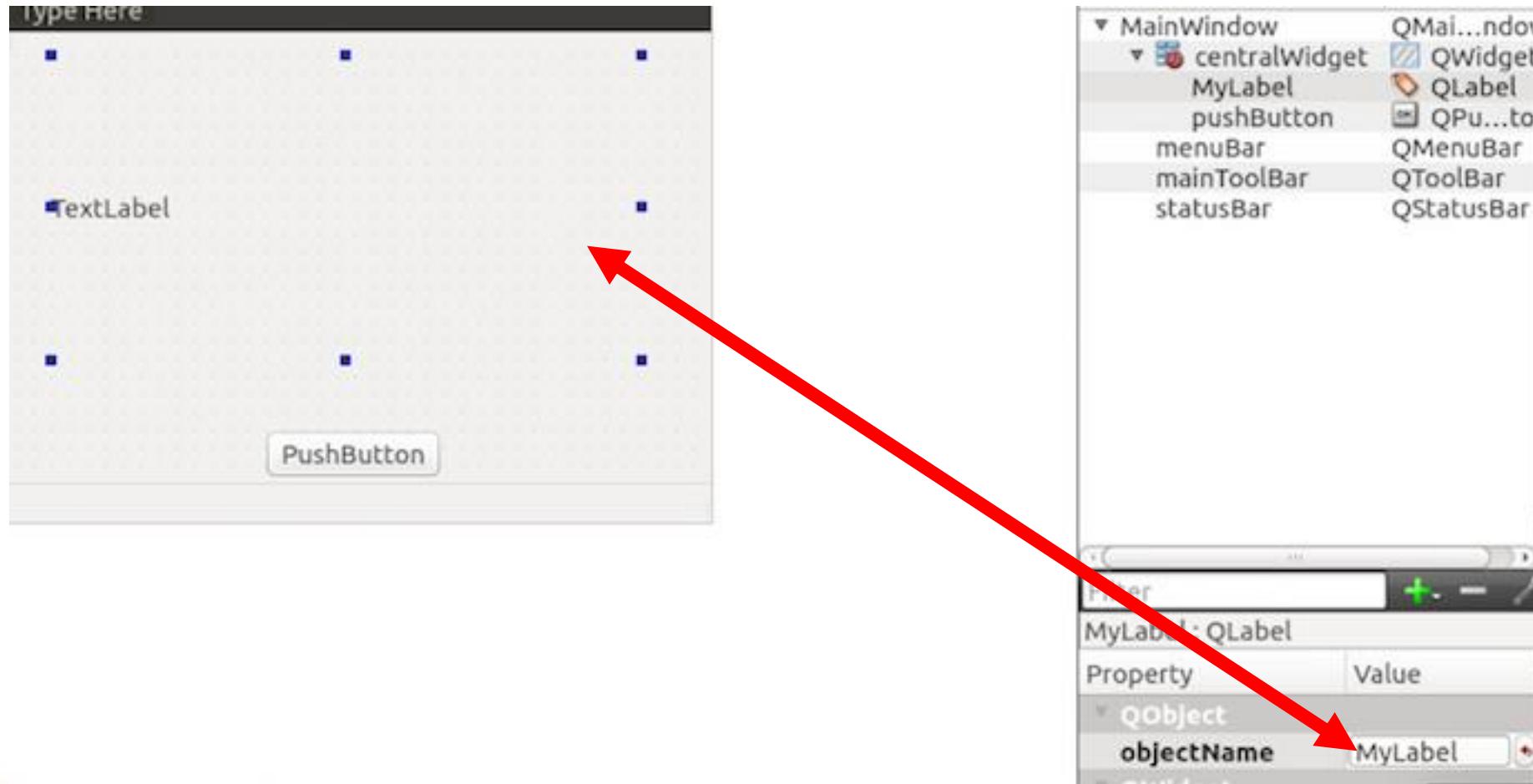
# Qt Creator

- 拉兩個物件分別是Push Button與Label於Window畫面上



# Qt Creator

- 將Label命名成MyLabel



# Qt Creator

- 將 Push Button 命名成 MyButton1



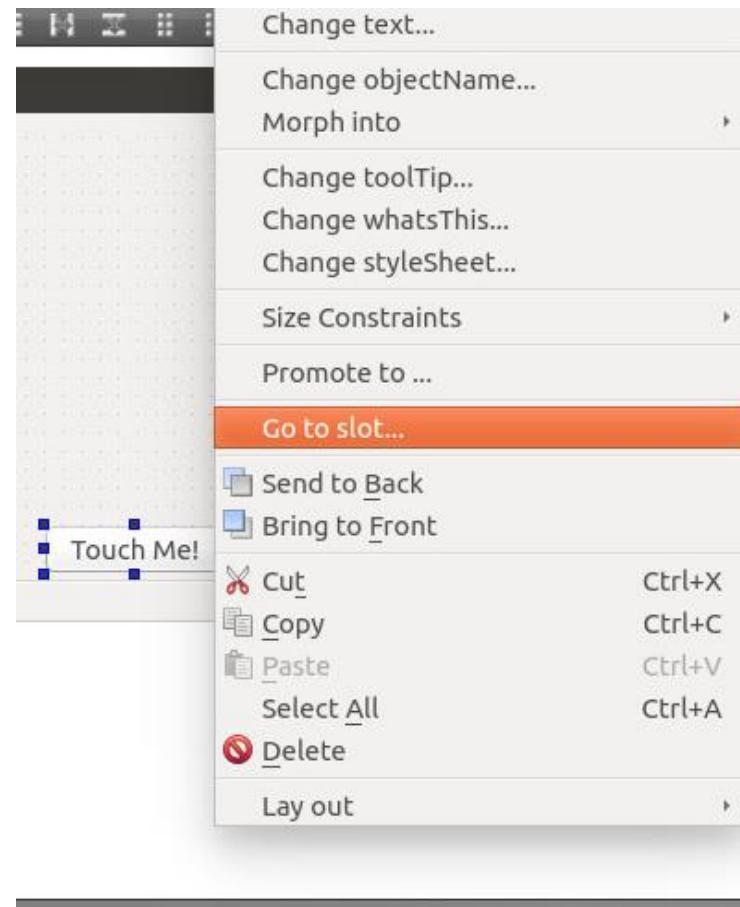
# Qt Creator

- 在Push Button上點兩下標題改成“Touch Me!”



# Qt Creator

- 點擊Push Button右鍵選擇 Go to slot... 來建立對應的訊號槽



# Qt Creator

- 選擇**clicked()**建立點擊此按鈕後會觸發的訊號槽



# Qt Creator

- IDE會幫我們再mainwindow.h和mainwindow.cpp中建立對應的程式碼如下圖

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QMainWindow>
5
6 namespace Ui {
7     class MainWindow;
8 }
9
10 class MainWindow : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit MainWindow(QWidget *parent = nullptr) : QMainWindow(parent)
16     ~MainWindow();
17
18 private slots:
19     void on_MyButton1_clicked();
20
21 private:
22     Ui::MainWindow *ui;
23 };
24
25 #endif // MAINWINDOW_H
```

**mainwindow.h**

## mainwindow.cpp

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 MainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9 }
10
11 ~MainWindow()
12 {
13     delete ui;
14 }
15
16 void MainWindow::on_MyButton1_clicked()
17 {
18 }
19
```

# Qt Creator

- 我們可以於mainwindow.cpp中剛剛建立的訊號觸發的function中撰寫按下按鈕後要執行的程式動作

## mainwindow.cpp

```
15  
16 void MainWindow::on_MyButton1_clicked()  
17 {  
18     在這裡寫程式  
19 }
```

# Qt Creator

- 範例是撰寫一個按下按鈕時會更改Label上的文字，按一下顯示“Hello Qt Creator!!”按第二下顯示“This is an apple”，第三下回到第一下時候的狀態（循環） \* 程式碼於下頁說明

```
void MainWindow::on_MyButton1_clicked()
{
    QString getLebelString = ui->MyLabel->text();

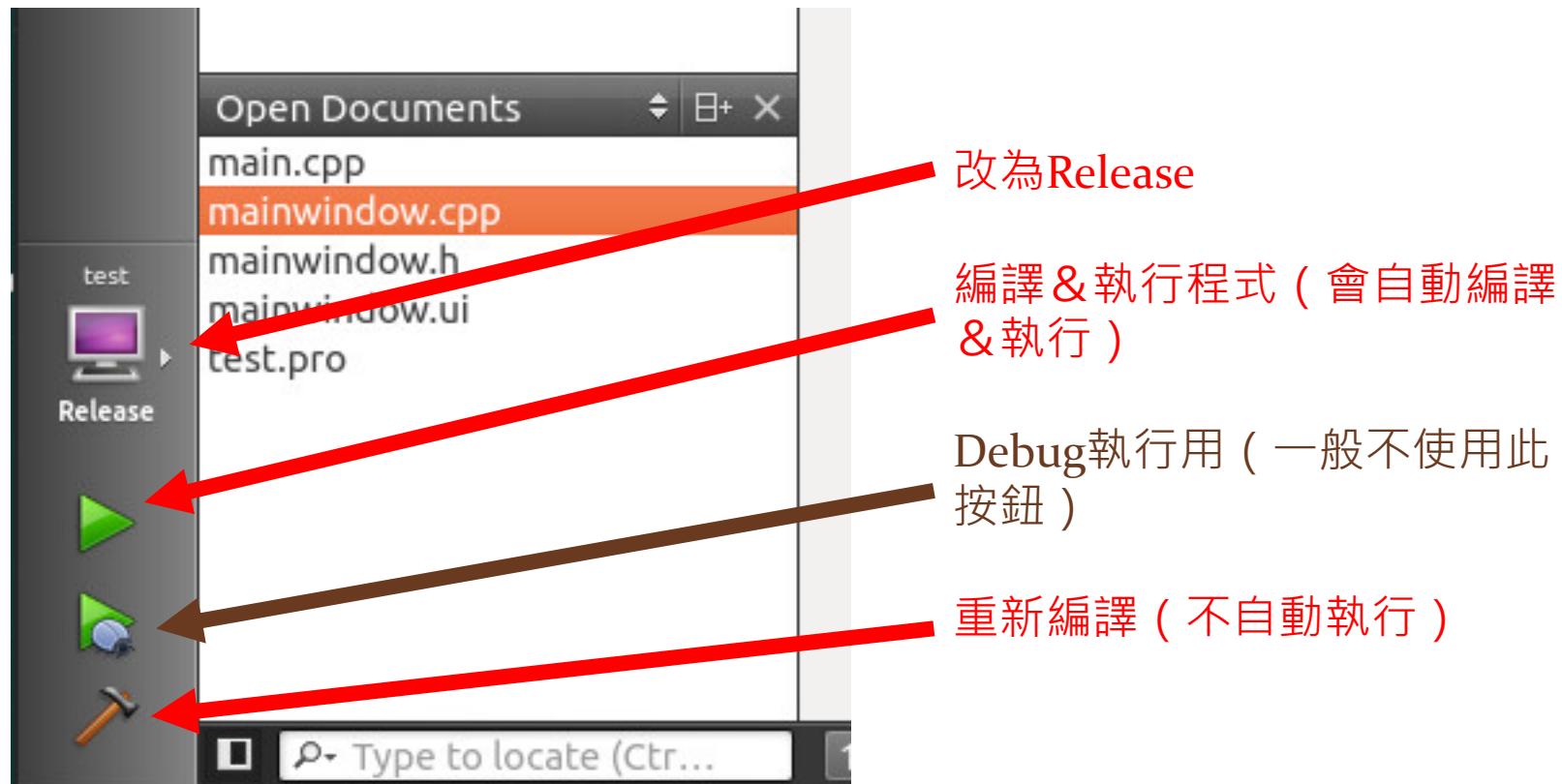
    if (getLebelString == "Hello Qt Creator!!")
    {
        ui->MyLabel->setText("This is an apple");
    }
    else
    {
        ui->MyLabel->setText("Hello Qt Creator!!");
    }
}
```

# Qt Creator

```
1 void MainWindow::on_MyButton1_clicked()
2 {
3     //QString為Qt的字串型態，可以轉換為C++ String
4     //此段程式碼式向ui上我們定義Label(名稱為MyLabel)取出上面的文字並存在變數中
5     QString getLebelString = ui->MyLabel->text();
6
7     //如果文字是Hello Qt Creator!!
8     if (getLebelString == "Hello Qt Creator!!")
9     {
10         //將Label的文字改成This is an apple
11         ui->MyLabel->setText("This is an apple");
12     }
13     else //若不是
14     {
15         //將Label的文字改成Hello Qt Creator!!
16         ui->MyLabel->setText("Hello Qt Creator!!");
17     }
18 }
```

# Qt Creator

- 編譯（於左下角），點選**編譯&執行程式按鈕**



# Qt Creator

- 執行結果

