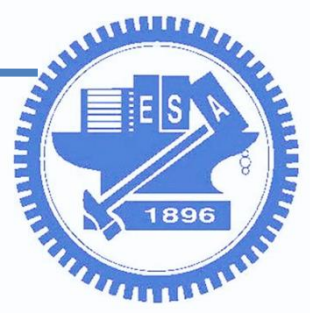
Outline

- 嵌入式系統: cross compile
  - 1. prepare a Linux system (Virtualbox + Ubuntu 16)
  - 2. download toolchain for PI
  - 3. compile code on Virtualbox
  - 4. execute code on Raspberry PI

https://www.tensorflow.org/install/source_rpi



Ex: Build Tensorflow

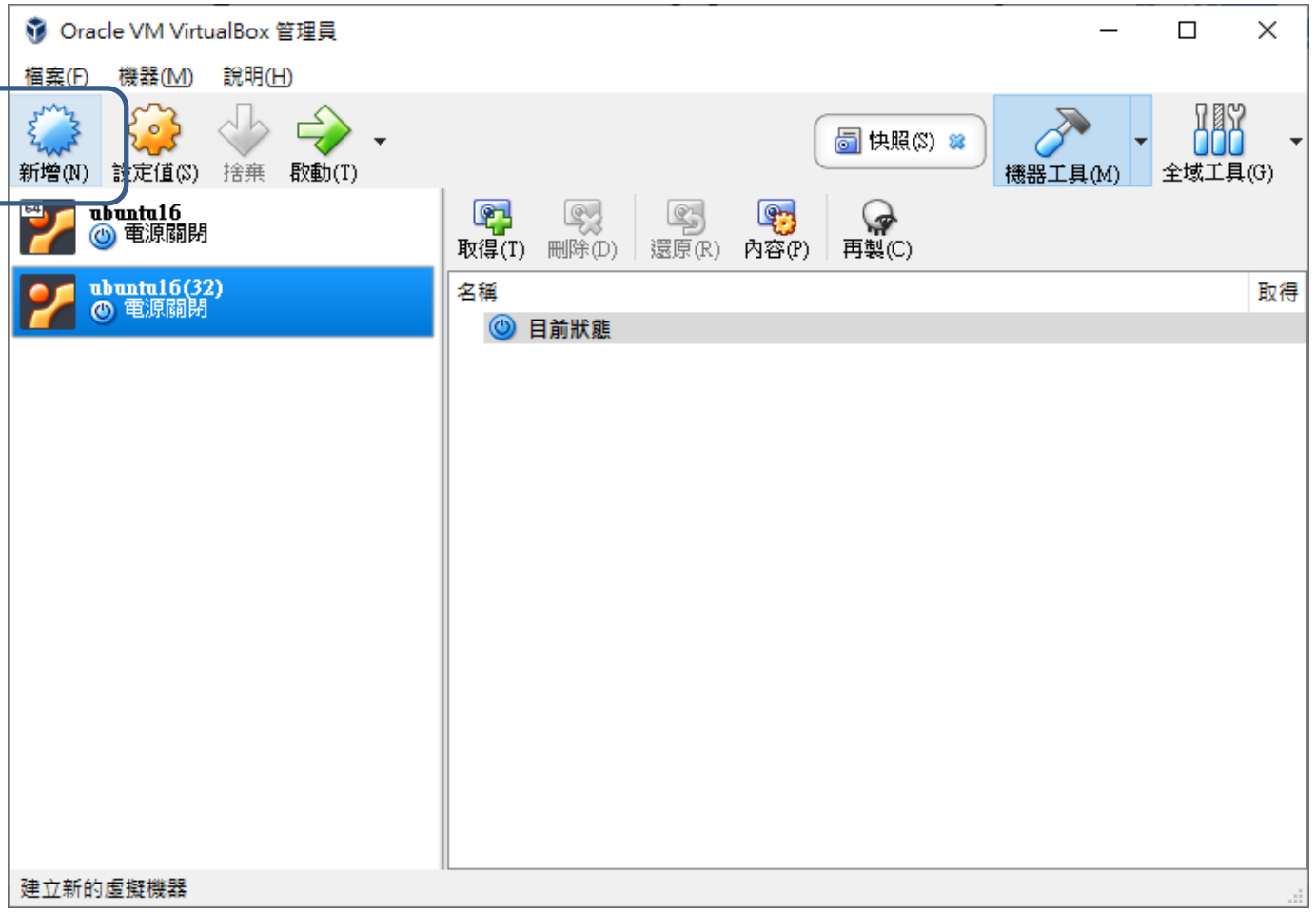
Build from source for the Raspberry Pi

This guide builds a TensorFlow package for a Raspberry Pi device running Raspbian 9.0. While the instructions might work for other Raspberry Pi variants, it is only tested and supported for this configuration.

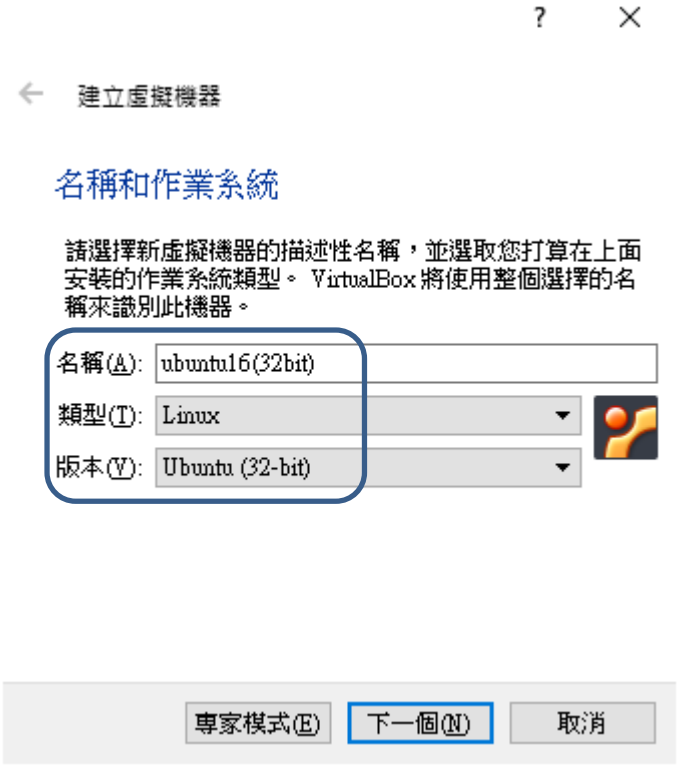
We recommend *cross-compiling* the TensorFlow Raspbian package. Cross-compilation is using a different platform to build the package than deploy to. Instead of using the Raspberry Pi's limited RAM and comparatively slow processor, it's easier to build TensorFlow on a more powerful host machine running Linux, macOS, or Windows.

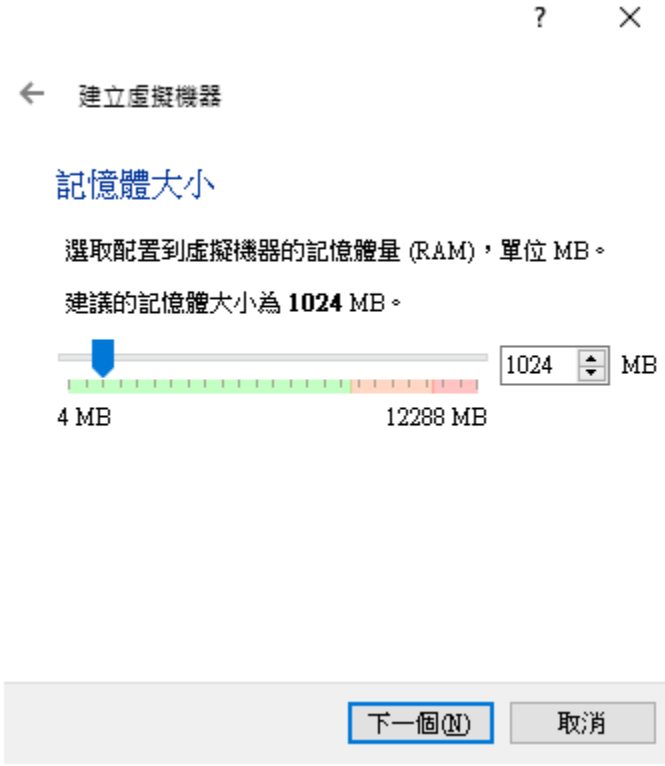
★ **Note:** We already provide well-tested, pre-built TensorFlow packages for Raspbian systems.

1. Prepare a Linux system

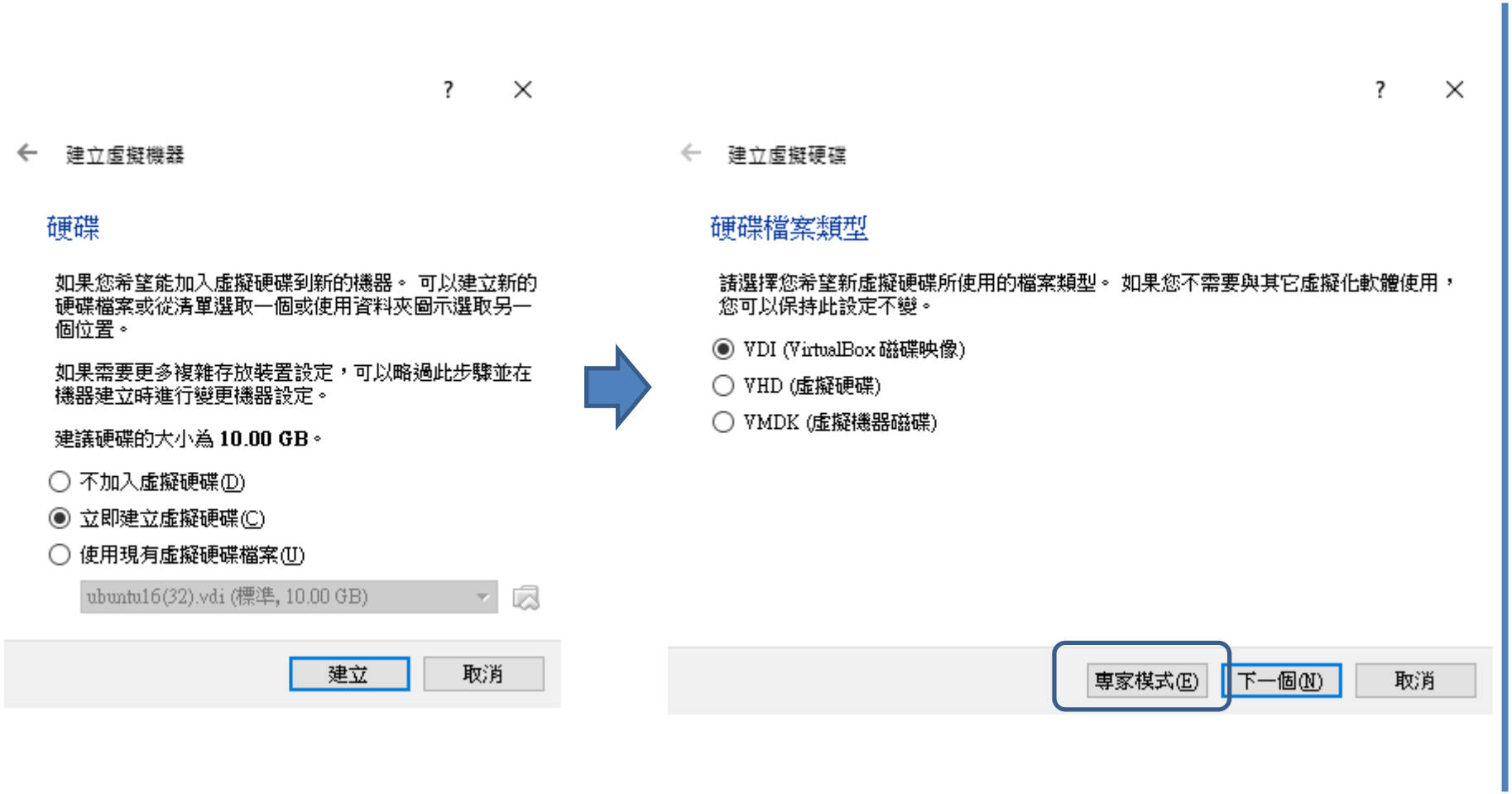


1. Prepare a Linux system



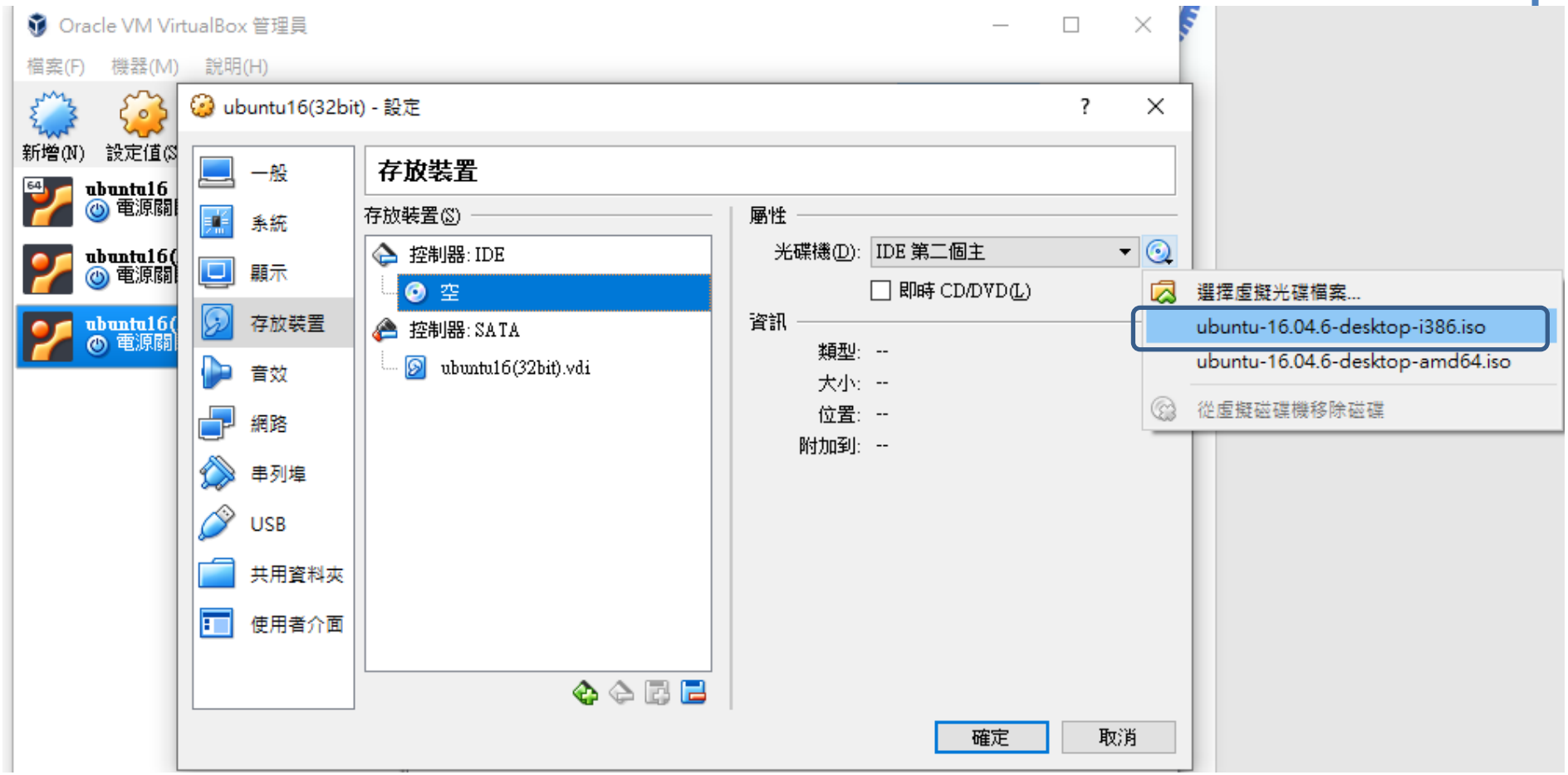


1. Prepare a Linux system



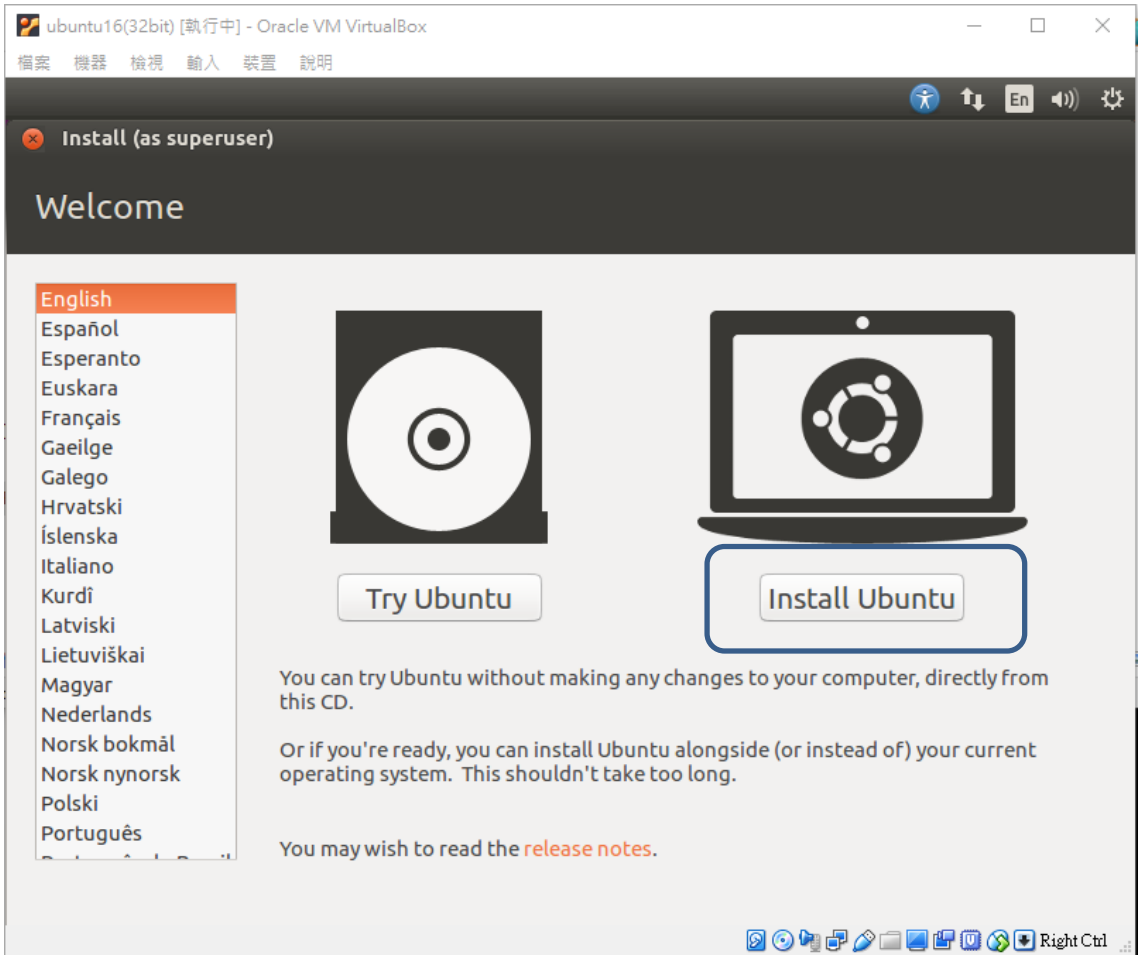
1. Prepare a Linux system

1. Prepare a Linux system

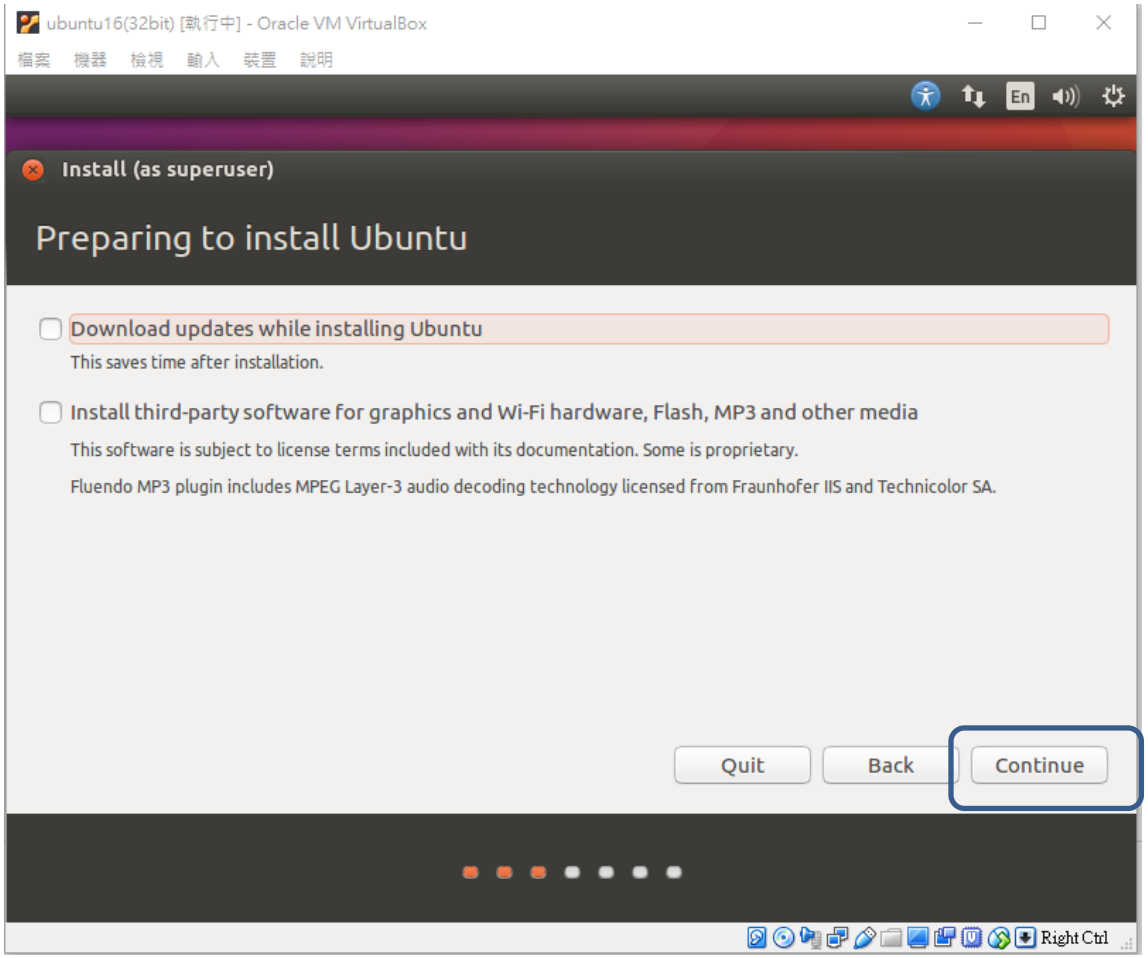


http://ubuntu.cs.nctu.edu.tw/ubuntu-release/16.04.6/ubuntu-16.04.6-desktop-i386.iso

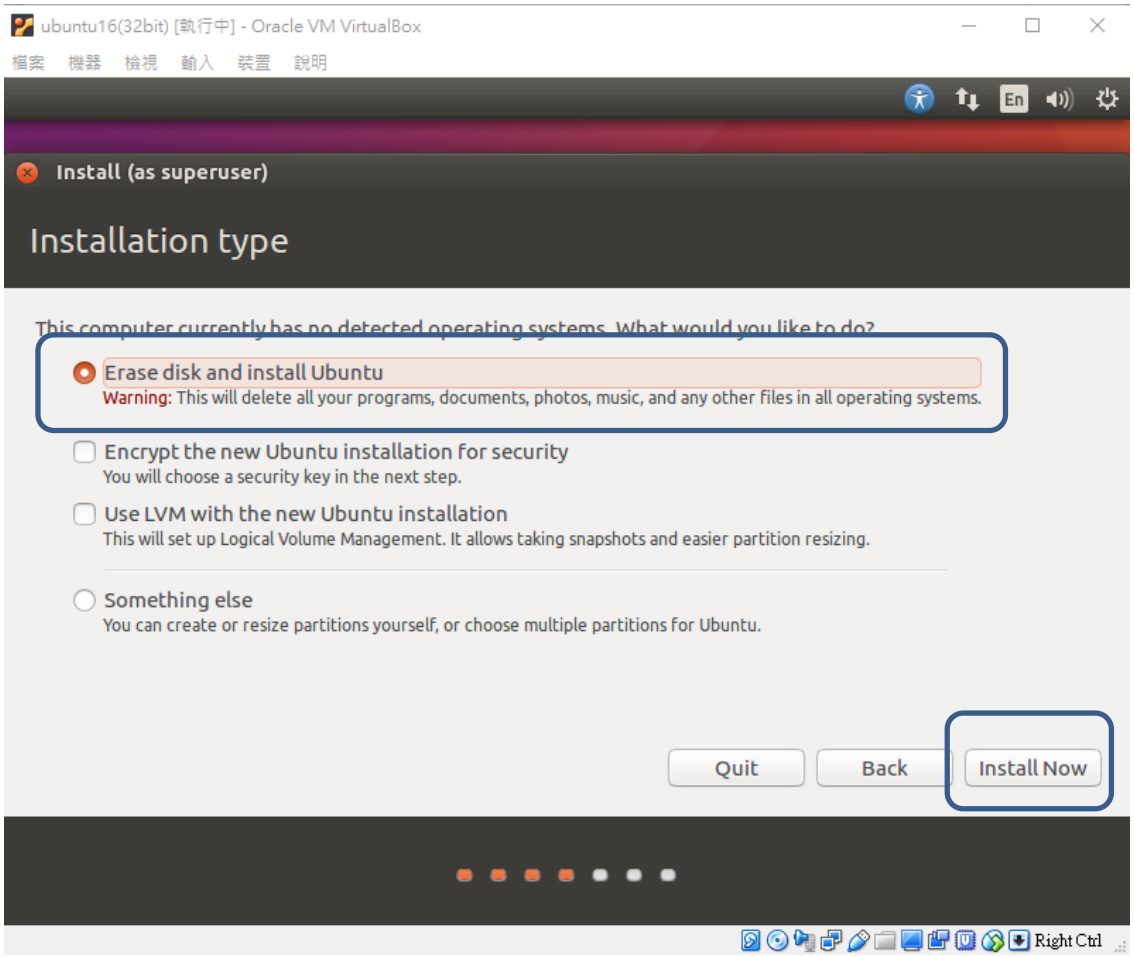
1. Prepare a Linux system



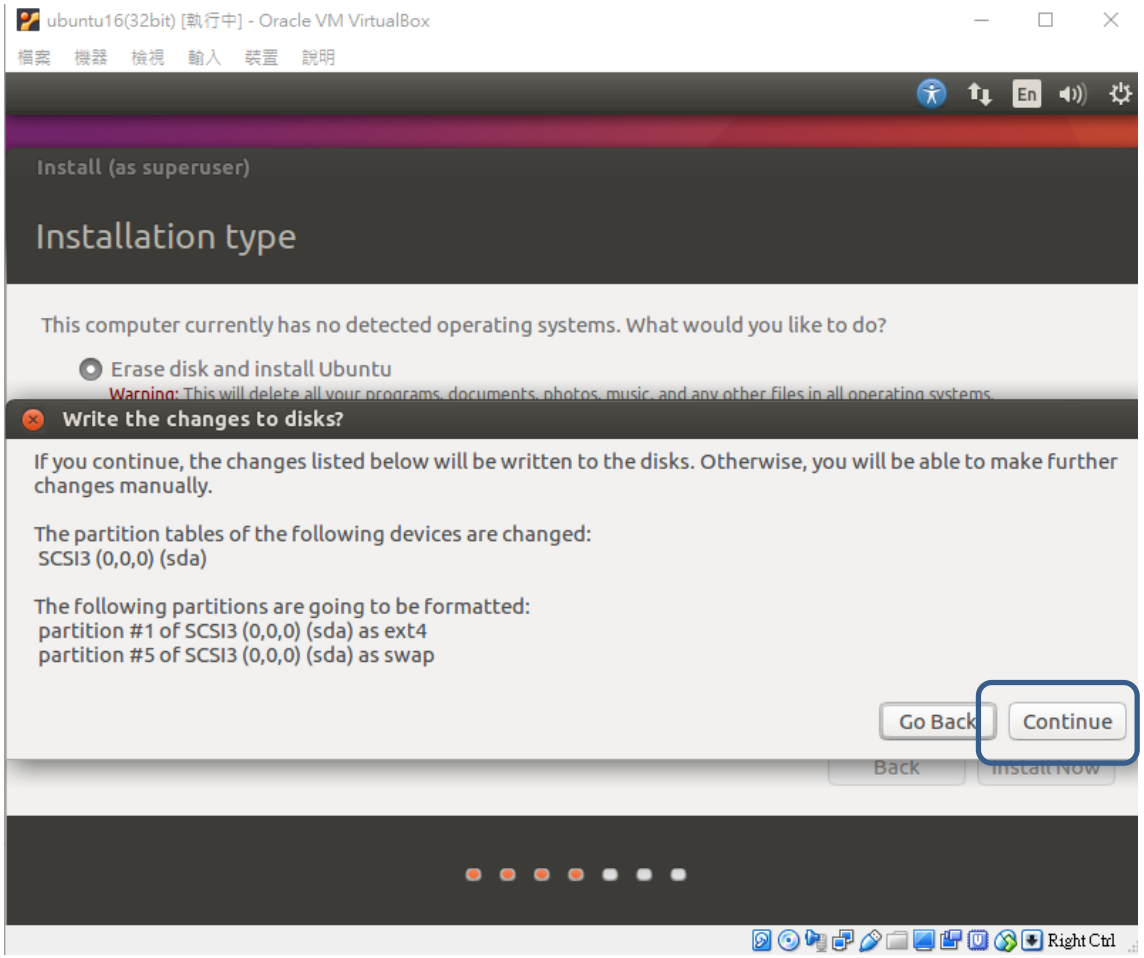
1. Prepare a Linux system



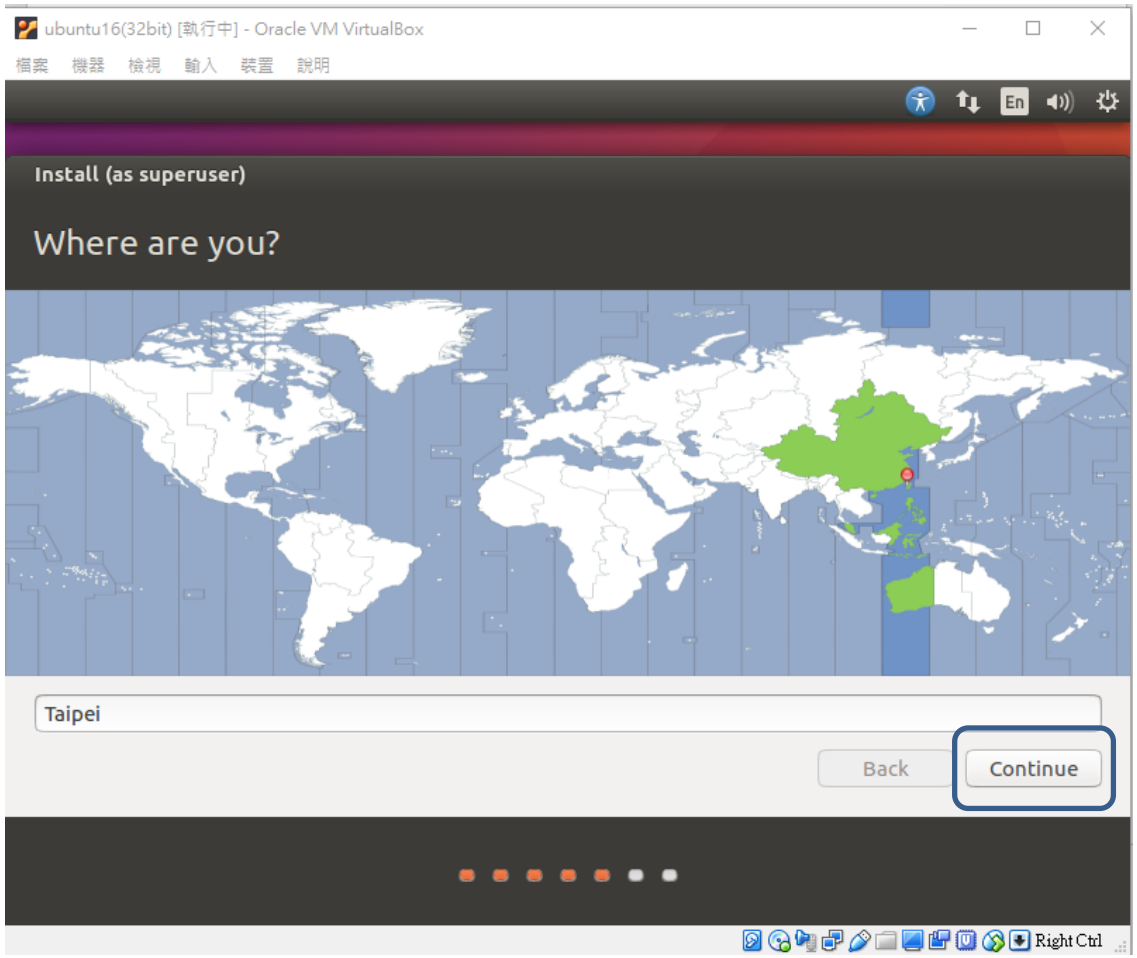
1. Prepare a Linux system



1. Prepare a Linux system

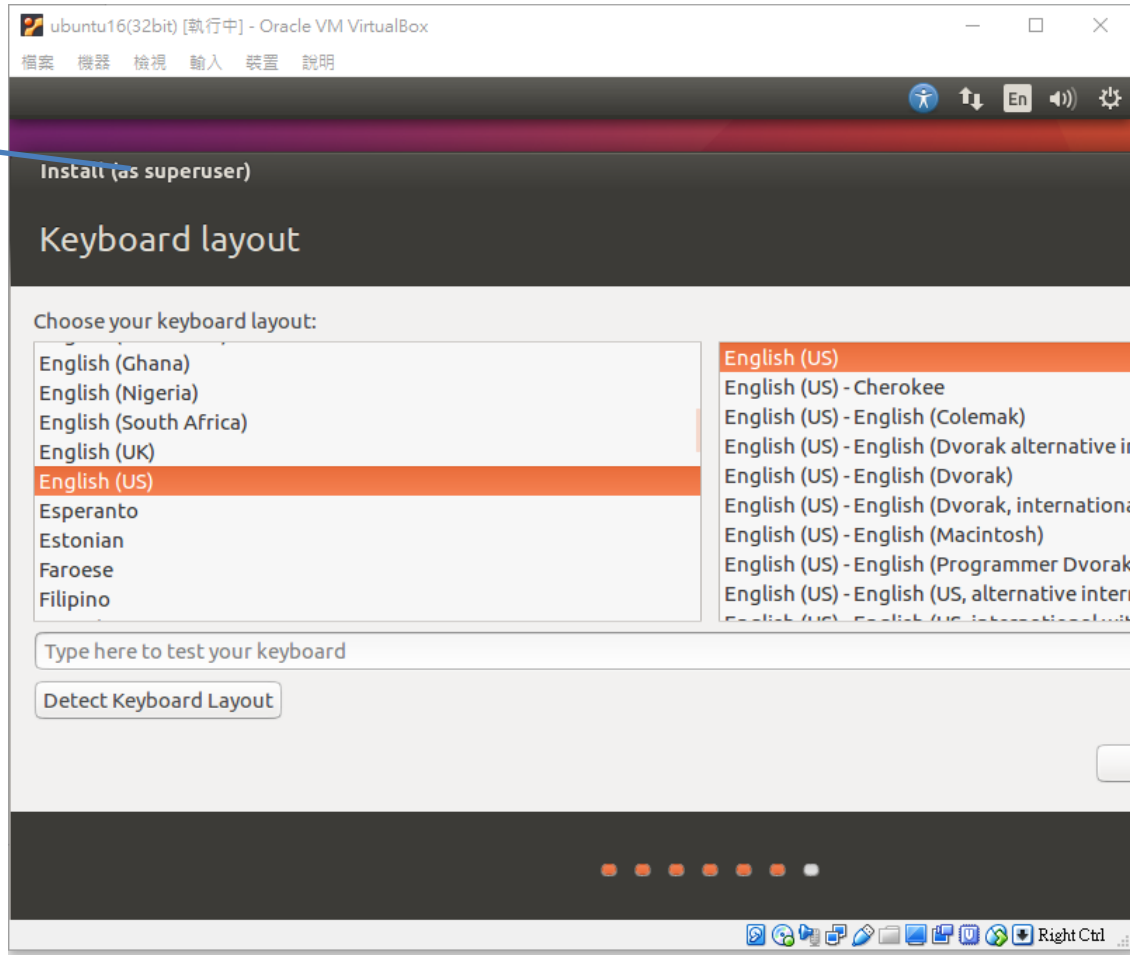


1. Prepare a Linux system



1. Prepare a Linux system

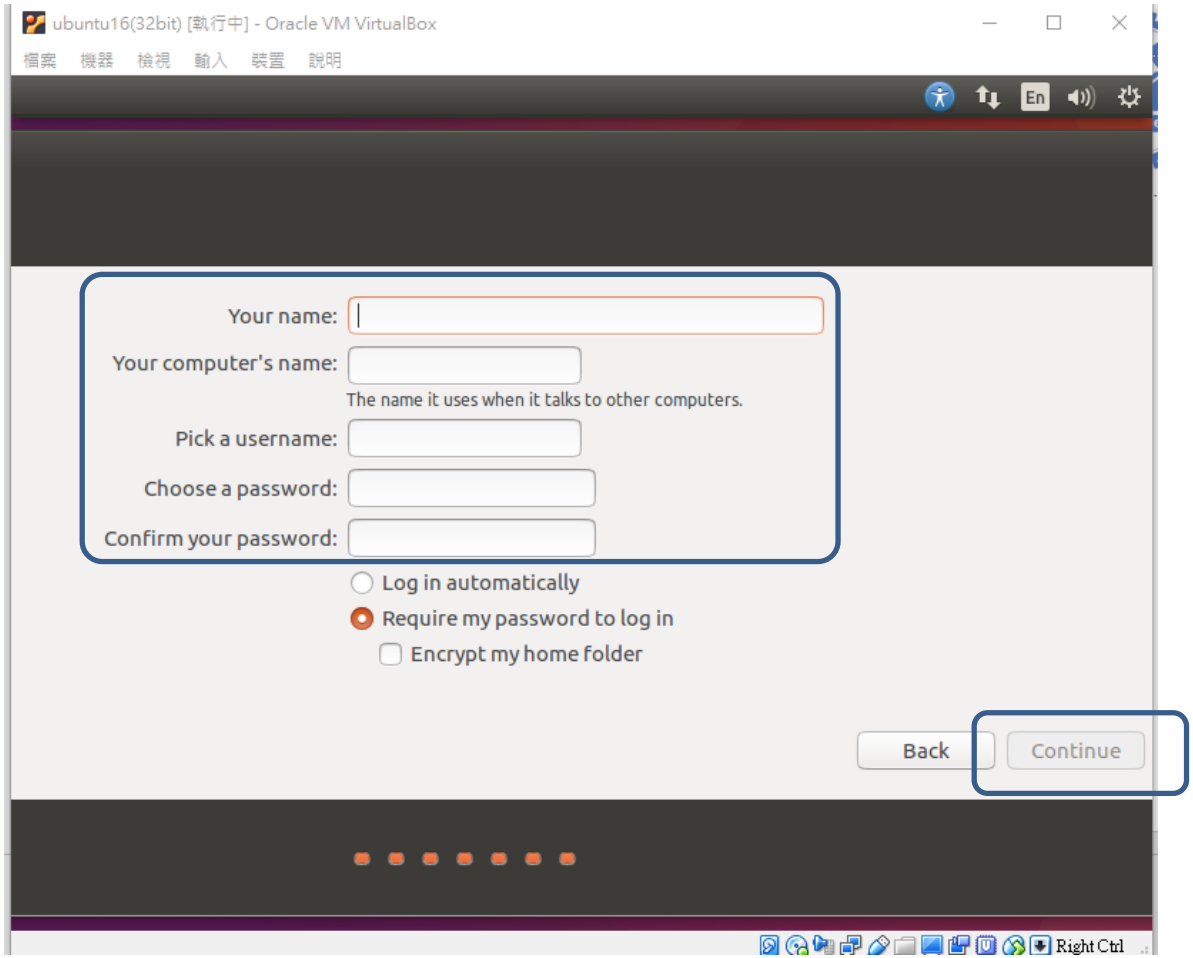
可用滑鼠拖曳



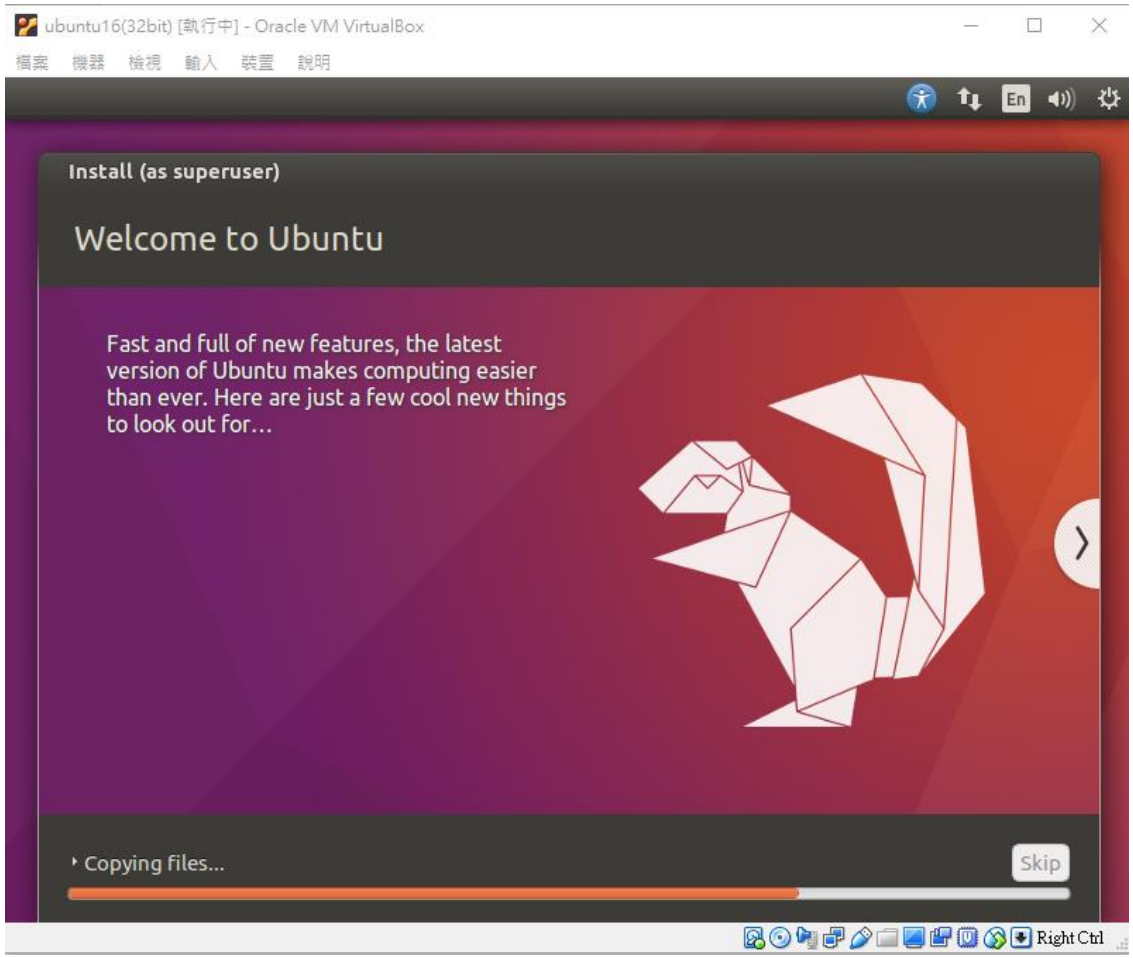
continue

按鈕在畫面外

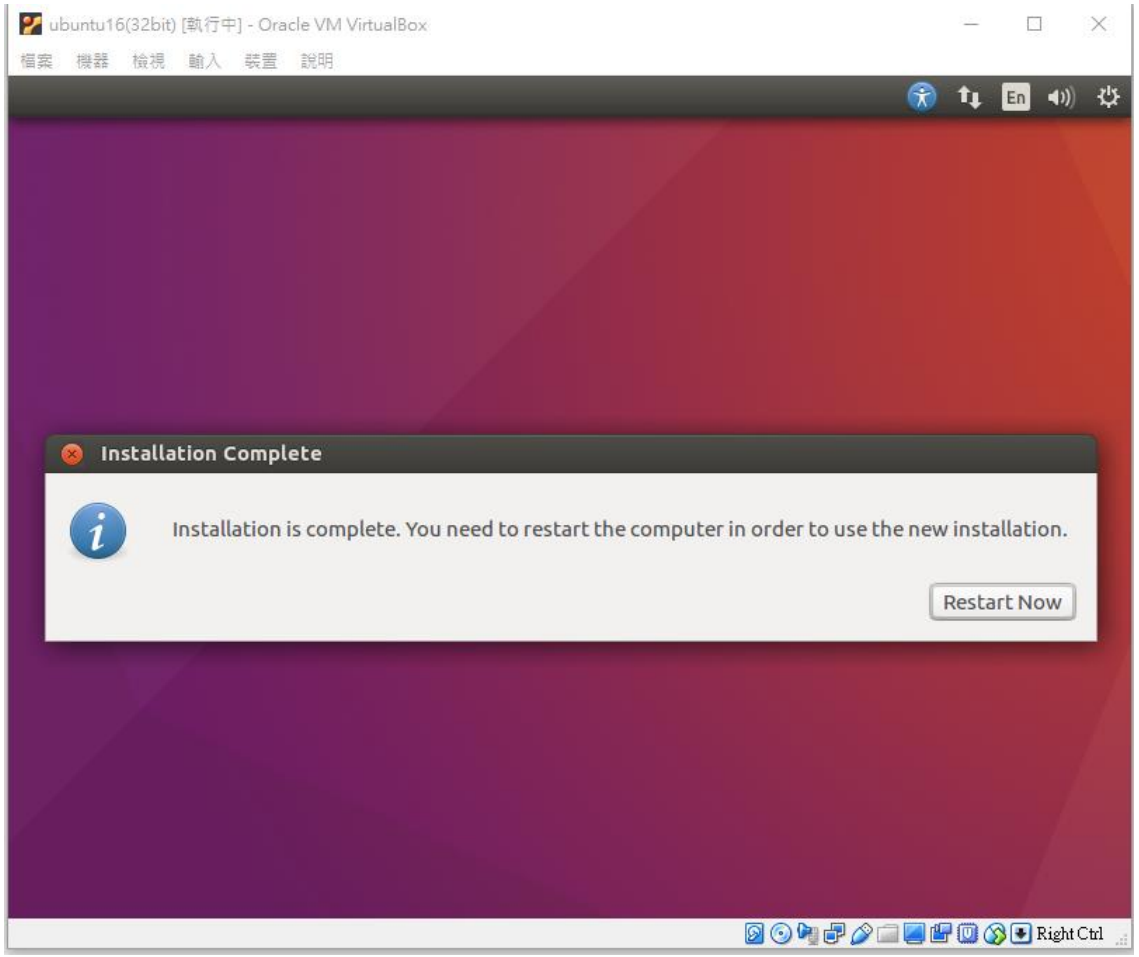
1. Prepare a Linux system



1. Prepare a Linux system

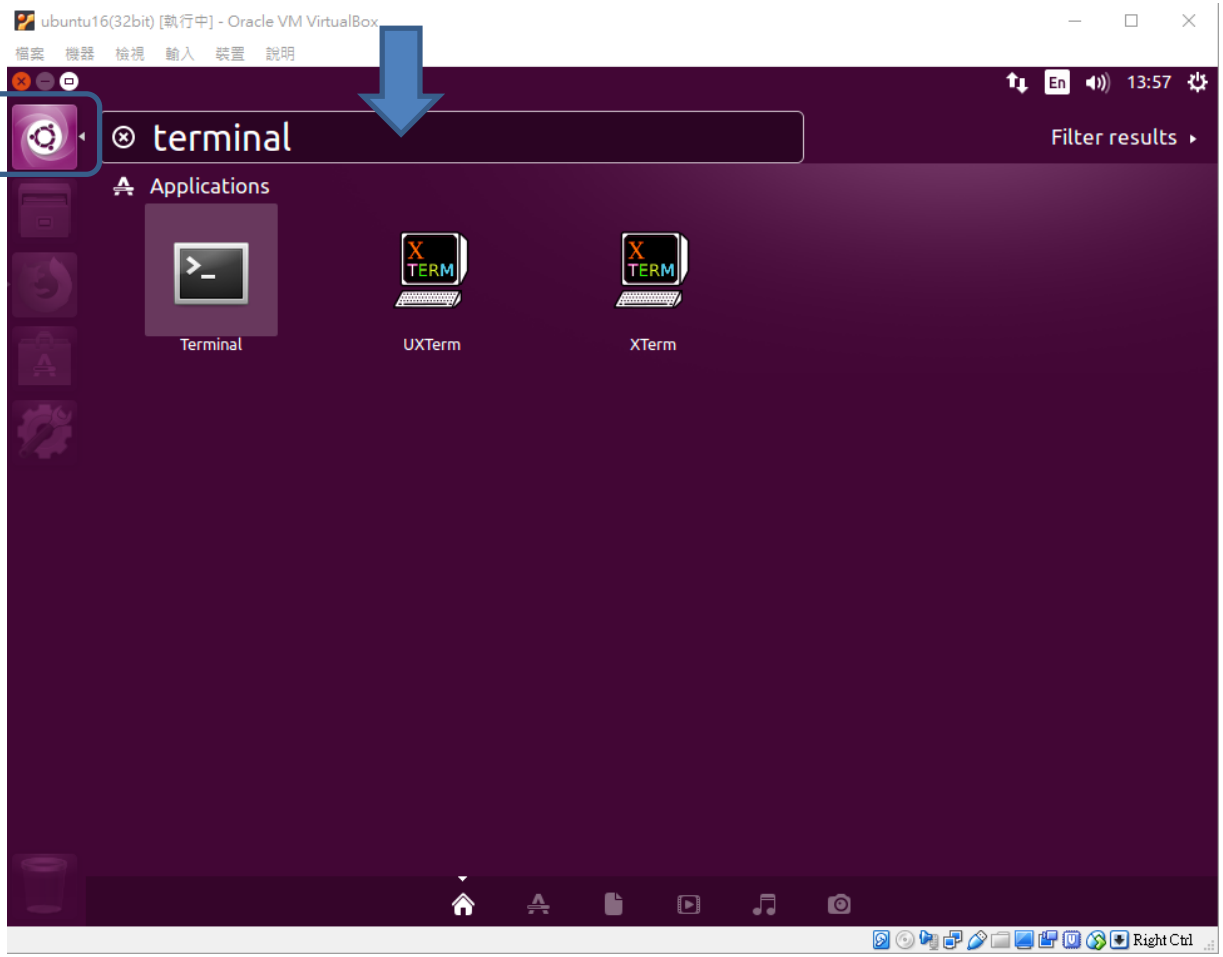


1. Prepare a Linux system



2. Virtualbox

- Open terminal: (type **terminal** here)



https://www.raspberrypi.org/documentation/linux/kernel/building.md
https://www.raspberrypi.com.tw/tag/cross-compiler/

2. Virtualbox

- Install toolchain:
  - sudo apt-get update
  - sudo apt-get install make git-core ncurses-dev
  - git clone https://github.com/raspberrypi/tools ~/tools
  - echo PATH=\$PATH:~/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin >> ~/.bashrc
  - source ~/.bashrc

2. Check environment

- In terminal:
  - Type **arm**, then press *tab* twice

```
class@class-VirtualBox: ~
class@class-VirtualBox:~$ arm
arm2hpdl                          arm-linux-gnueabihf-gfortran
arm-linux-gnueabihf-addr2line     arm-linux-gnueabihf-gprof
arm-linux-gnueabihf-ar            arm-linux-gnueabihf-ld
arm-linux-gnueabihf-as            arm-linux-gnueabihf-ld.bfd
arm-linux-gnueabihf-c++           arm-linux-gnueabihf-ldd
arm-linux-gnueabihf-c++filt       arm-linux-gnueabihf-ld.gold
arm-linux-gnueabihf-cpp           arm-linux-gnueabihf-nm
arm-linux-gnueabihf-dwp           arm-linux-gnueabihf-objcopy
arm-linux-gnueabihf-elfedit       arm-linux-gnueabihf-objdump
arm-linux-gnueabihf-g++           arm-linux-gnueabihf-pkg-config
arm-linux-gnueabihf-gcc           arm-linux-gnueabihf-pkg-config-real
arm-linux-gnueabihf-gcc-4.8.3     arm-linux-gnueabihf-ranlib
arm-linux-gnueabihf-gcc-ar        arm-linux-gnueabihf-readelf
arm-linux-gnueabihf-gcc-nm        arm-linux-gnueabihf-size
arm-linux-gnueabihf-gcc-ranlib    arm-linux-gnueabihf-strings
arm-linux-gnueabihf-gcov          arm-linux-gnueabihf-strip
arm-linux-gnueabihf-gdb
```

2. Check environment

- Test: arm-linux-gnueabihf-gcc -v

```
class@class-VirtualBox: ~
class@class-VirtualBox:~$ arm-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
COLLECT_LTO_WRAPPER=/home/class/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf
-raspbian/bin/../libexec/gcc/arm-linux-gnueabihf/4.8.3/lto-wrapper
Target: arm-linux-gnueabihf
Configured with: /cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabihf-ra
spbian-linux/.build/src/gcc-linaro-4.8-2014.01/configure --build=i686-build_pc-l
inux-gnu --host=i686-build_pc-linux-gnu --target=arm-linux-gnueabihf --prefix=/c
build/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabihf-raspbian-linux/instal
```

```
ith-libelf=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabihf-raspbian
-linux/.build/arm-linux-gnueabihf/build/static --enable-threads=posix --disable-
libstdcxx-pch --enable-linker-build-id --enable-plugin --enable-gold --with-loca
l-prefix=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabihf-raspbian-l
inux/install/arm-linux-gnueabihf/libc --enable-c99 --enable-long-long --with-flo
at=hard
Thread model: posix
gcc version 4.8.3 20140106 (prerelease) (crosstool-NG linaro-1.13.1-4.8-2014.01
- Linaro GCC 2013.11)
```

Write code

- Write C code:
  - nano hello.c // write your code
  - gcc hello.c -o hello.o // compile it, the output file is hello.o
  - ./hello.o // execute hello.o

```
#include <stdio.h>
int main()
{
        printf("hello, world\n");
    return 0;
}
```

Compile

- Compile it and execute on PC:

```
class@class-VirtualBox: ~
class@class-VirtualBox:~$ cat hello.c
#include <stdio.h>
int main()
{
        printf("hello XD\n");
        return 0;
}
class@class-VirtualBox:~$ gcc hello.c -o hello.o
class@class-VirtualBox:~$ ./hello.o
hello XD
```

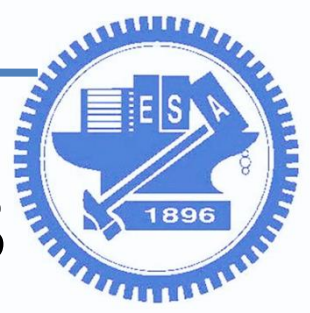
- Cross compile, then copy it to PI and execute:
  - arm-linux-gnueabihf-gcc hello.c -o hello-arm
  - ./hello-arm // run this on PI

```
pi@raspberrypi:~$ ./hello.arm
hello XD
pi@raspberrypi:~$
```

For reference

- Raspberry Pi boot modes
- Network boot: boot via Ethernet
- Flash your AP
- How to flash or recover your device
- Kernel building
  - Raspberry PI
  - Openwrt
- Openwrt requirements

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/

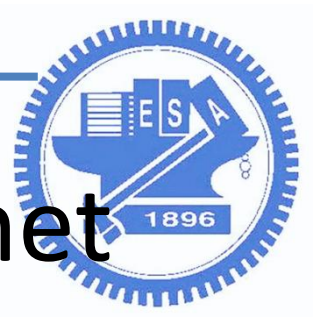


Raspberry Pi boot modes

- Boot Sequence

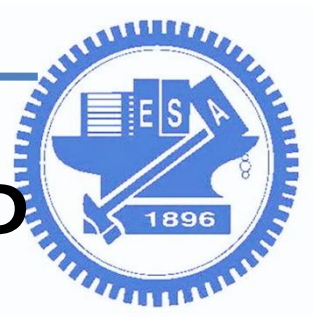
- SD card boot
- USB boot comprises the following two modes:
  - Device boot: booting as a mass storage device
  - Host boot: booting as a USB host using one of the following:
    - Mass storage boot: boot from mass storage device
    - Network boot: boot via Ethernet

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/net.md



Network boot: boot via Ethernet

- To network boot, the boot ROM does the following:
  - Initialise on-board Ethernet device (Microchip LAN9500 or LAN7500)
  - Send DHCP request
  - Receive DHCP reply
  - (optional) Receive DHCP proxy reply
  - ARP to tftpboot server
  - ARP reply includes tftpboot server ethernet address
  - TFTP RRQ 'bootcode.bin'
    - File not found: Server replies with TFTP error response with textual error message
    - File exists: Server will reply with the first block (512 bytes) of data for the file with a block number in the header
    - Pi replies with TFTP ACK packet containing the block number, and repeats until the last block which is not 512 bytes
  - TFTP RRQ 'bootsig.bin'
    - This will normally result in an error file not found. This is to be expected, and TFTP boot servers should be able to handle it.

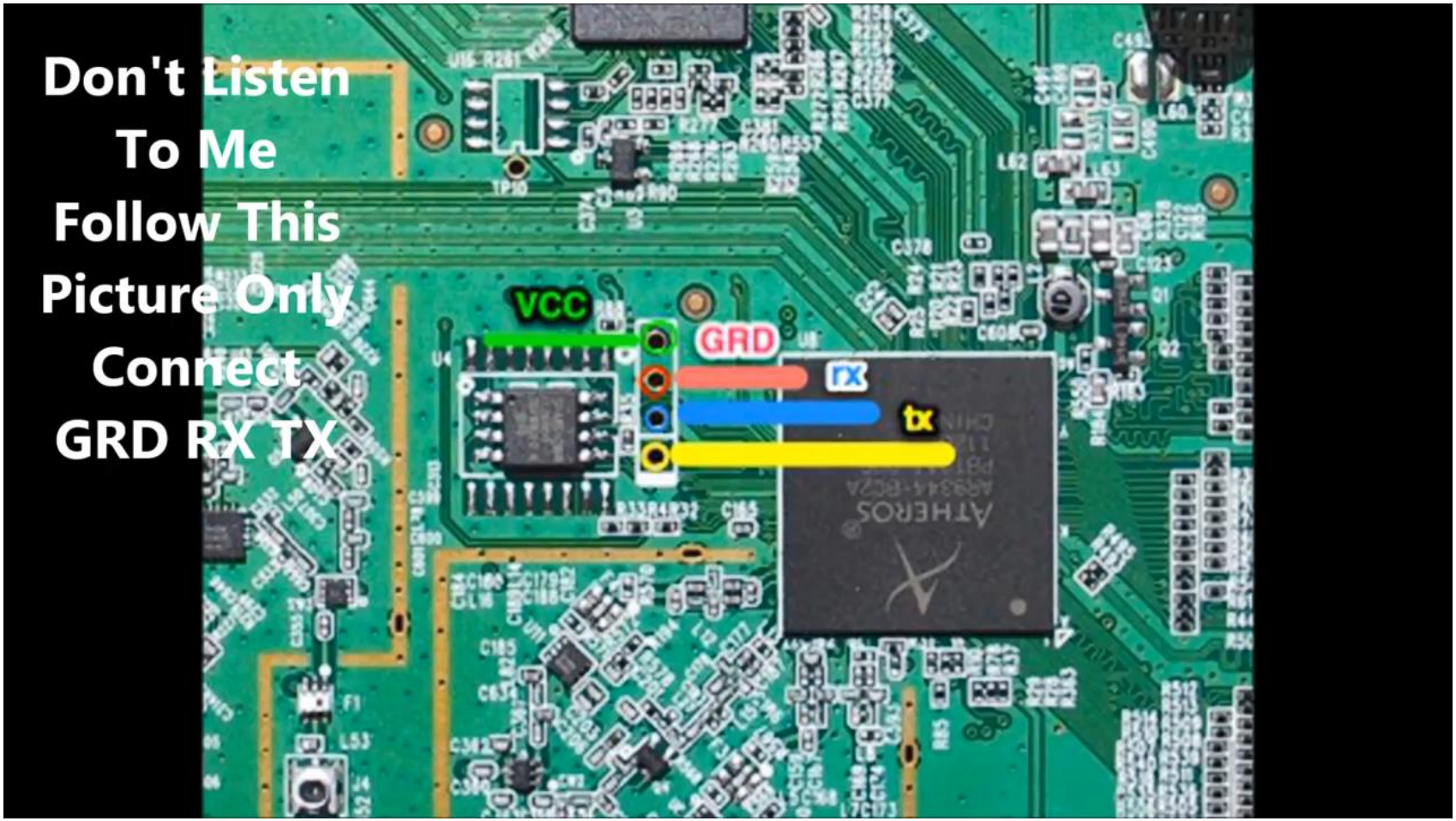


Application: Flash your AP

- Contents
  1. Installing DD-WRT
     1. Flashing from Buffalo Firmware
     2. DD-WRT Upgrade Flashes
  2. Specific configuration
     1. DDNS
  3. Important Notes
  4. Going back to Buffalo Firmware
  5. Recovery from Bricking, Semi-bricking

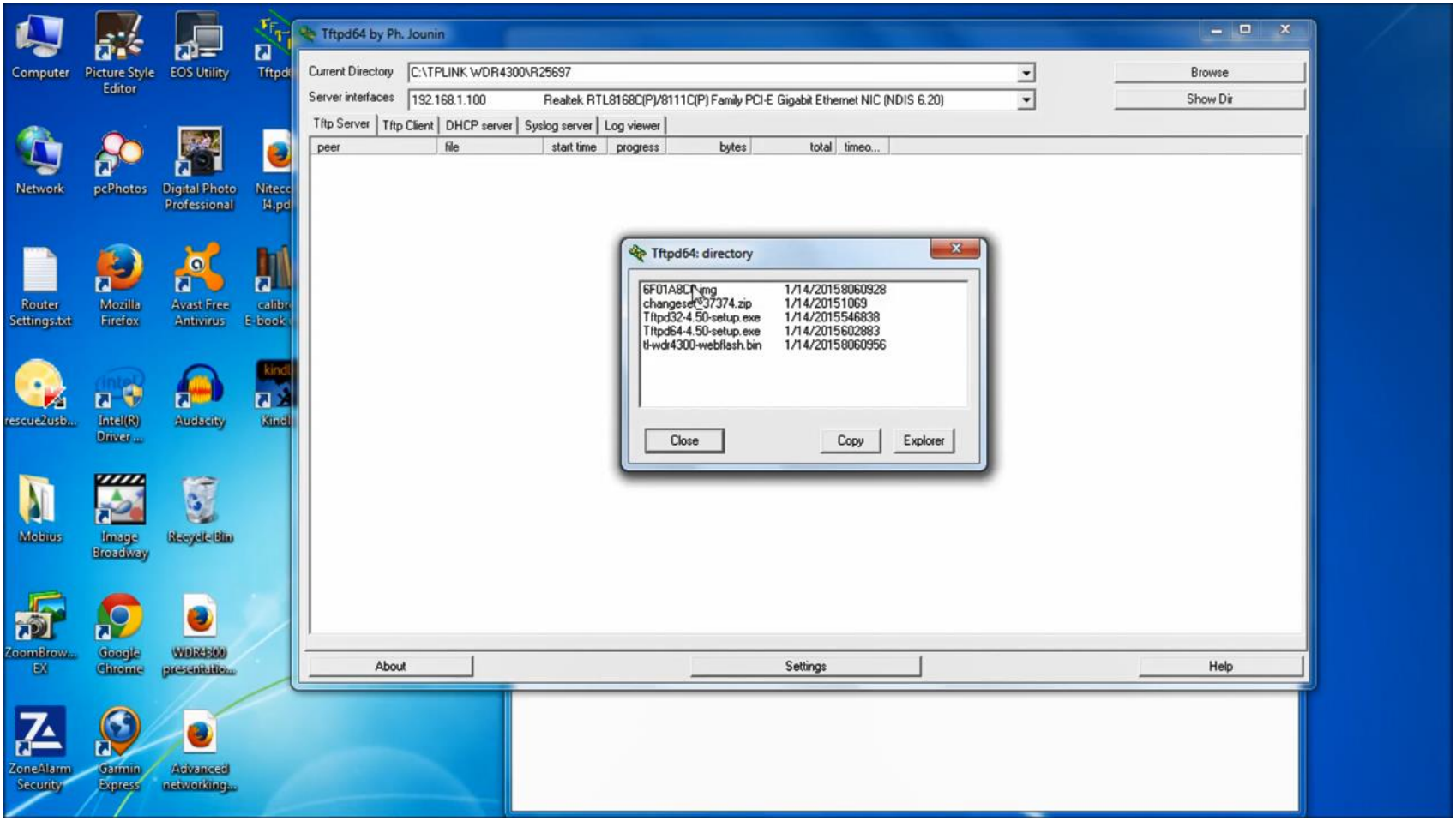
https://www.youtube.com/watch?v=t35Pei_eb6o

Bricked TP-Link WDR4300 Router Recovery Using UART Serial Converter - 1



https://www.youtube.com/watch?v=wCwBvoueBG4

Bricked TP-Link WDR4300 Router Recovery Using UART Serial Converter - 2



https://www.raspberrypi.org/documentation/linux/kernel/building.md

Kernel building (PI)

Raspberry Pi 2, Pi 3, Pi 3+, and Compute Module 3 default build configuration

```
cd linux
KERNEL=kernel7
make bcm2709_defconfig
```

Cross-compiling

First, you will need a suitable Linux cross-compilation host. We tend to use Ubuntu; since Raspbian is also a Debian distribution, it means many aspects are similar, such as the command lines.

You can either do this using VirtualBox (or VMWare) on Windows, or install it directly onto your computer. For reference, you can follow instructions online at Wikihow.

Kernel building (PI)

Finally, copy the kernel and Device Tree blobs onto the SD card, making sure to back up your old kernel:

```
sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
sudo umount mnt/fat32
sudo umount mnt/ext4
```

```
wufish@wufish-aBeing-One-AZ10:~/linux$ ls arch/arm/boot
bootp         deflate_xip_data.sh  Image        Makefile
compressed    dts                  install.sh   zImage
```

https://openwrt.org/zh-tw/doc/howto/build
https://openwrt.org/docs/guide-developer/quickstart-build-images

Build Openwrt

Get the OpenWrt source code:

```
git clone https://git.openwrt.org/openwrt/openwrt.git/
cd openwrt

./scripts/feeds update -a
./scripts/feeds install -a

make menuconfig
```

The last command will open a menu.

If you want to build images for the “TL-WR841N v11” Wifi-Router, select:

- “Target System” ⇒ “Atheros AR7xxx/AR9xxx”
- “Subtarget” ⇒ “Devices with small flash”
- “Target Profile” ⇒ “TP-LINK TL-WR841N/ND v11”

Build Openwrt

編譯OpenWRT：

```
make
```

最後產生出來的image(.gz)會放在bin/x86下面。

解壓縮指令

```
gzip -d openwrt-x86-generic-combined-ext4.img.gz
```

使用dd把image複製到CF卡上

把剛剛解壓縮得到的openwrt-x86-generic-combined-ext4.img複製到cf卡

```
dd if=openwrt-x86-generic-combined-ext4.img of=/dev/sdX    sdX為CF卡的代號
```

https://openwrt.org/supported_devices

Openwrt requirements

1. General requirements for OpenWrt support
2. SoC / target supported by OpenWrt
3. Sufficient Flash to accommodate OpenWrt firmware image
   - 4MB min (won't be able to install GUI (LuCI))
   - 8MB better (will fit GUI and some other applications)
4. Sufficient RAM for stable operation
   - 32MB min, 64MB better

- Is your device supported?
  - Go to https://wikidevi.com
  - Ex: ASUS_RT-AC86U
  - https://wikidevi.com/wiki/ASUS_RT-AC86U