EMBEDDED SYSTEM

LINUX ON EMBEDDED SYSTEM

HCMUTE – Faculty of Mechanical Engineering

Lecturer: PhD. Bui Ha Duc

MEMORABLE LINUX MILESTONES

CELEBRATING 20 YEARS OF LINUX

1991 — LINUS TORVALDS POSTS FAMOUS MESSAGE - *"HELLO EVERYBODY OUT THERE..."* - AND RELEASES FIRST LINUX CODE

1992 — LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS

1993 — SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION

1996 — LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT

1998 — TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX

1999 — RED HAT GOES PUBLIC

2003 — IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL

2005 — LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

2007 — THE LINUX FOUNDATION IS FORMED TO PROMOTE PROTECT AND STANDARDIZE LINUX LINUS IS A FELLOW

2010 — THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE

2011 — LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON

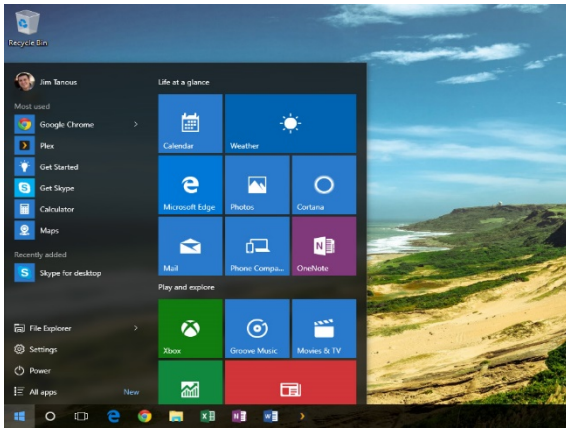
20 YEARS OF LINUX 1991-2011

THE LINUX FOUNDATION
http://www.linuxfoundation.org/

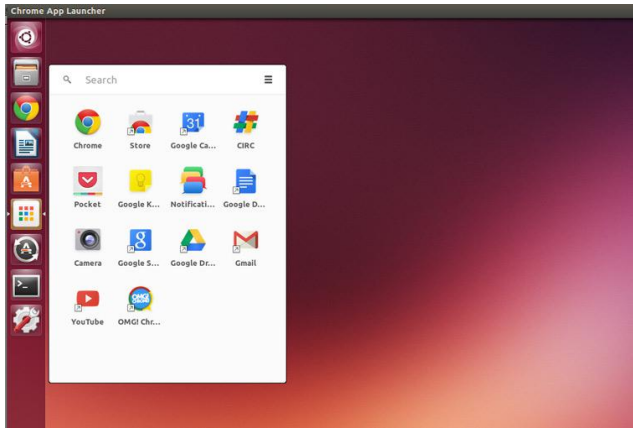
Linux vs Windows

- First look - Graphic user interface:

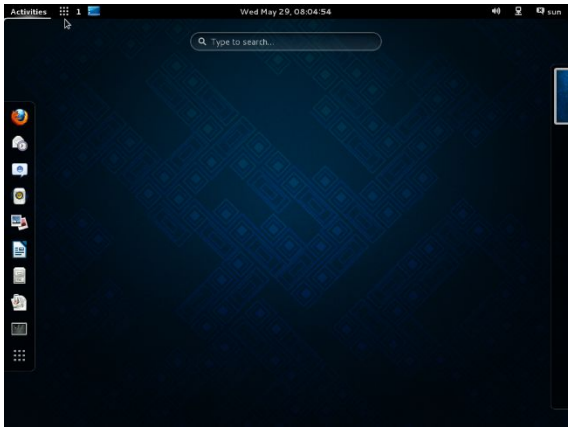
Windows



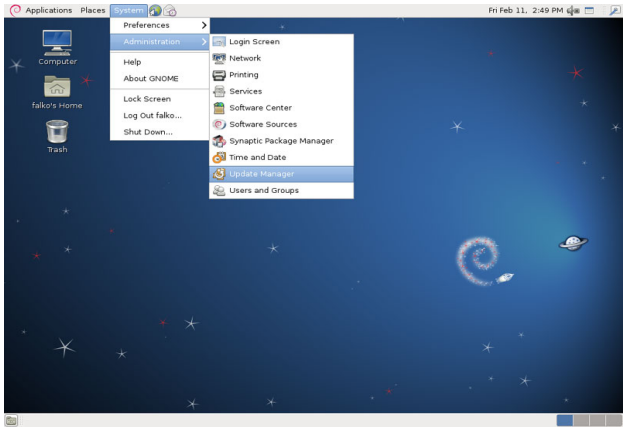
Linux Ubuntu



Linux Fedora

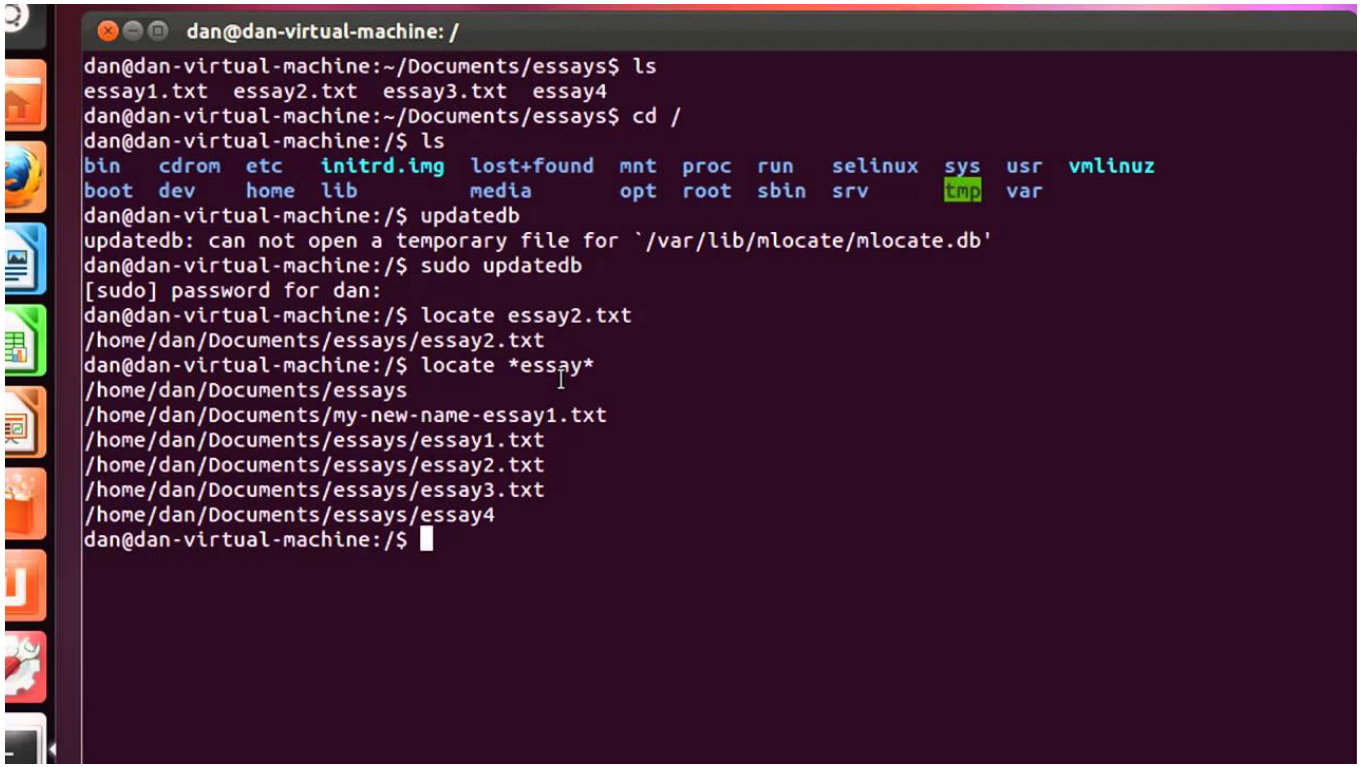


Linux Debian



Linux vs Windows

- First look - Terminal:



Linux
Terminal

Linux vs Windows

Linux	Windows
Open Source; **“Free”** to distribute, download, modify	$$
Can be installed on a **wide variety of computer hardware** (mobile phones, tablet computers, video game consoles, supercomputers)	On PC's desktops, laptops, servers and some phones
File system support: Ext2, Ext3, Ext4	File system support: FAT, FAT32, NTFS
Text command: **BASH**	Text command: DOS – like commands
User interface: GNOME or KDE	Windows Aero

Linux Kernel

- Kernel is the core of Linux operating systems
- Official website:

  https://www.kernel.org/
- OS based on Linux kernel – Linux Distributions
  - E.g. Ubuntu, Fedora, Red hat, Debian…
- Almost 600 Linux distributions exist.
- Kernel versions:
  - Linux 2.6.x: Friendly Arm
  - Linux 3.4.x: Orange Pi
  - Linux 4.x: Raspberry, BeagleBone

Linux Kernel

- Check your linux kernel:
  - Open terminal
  - Type “uname –r”

2.6.32-431.11.2.el6.x86_64

Kernel Version Major Revision Minor Revision

- Newer version has: new features, enhancements, bug fixes, apps

Can we modify Linux Kernel ?

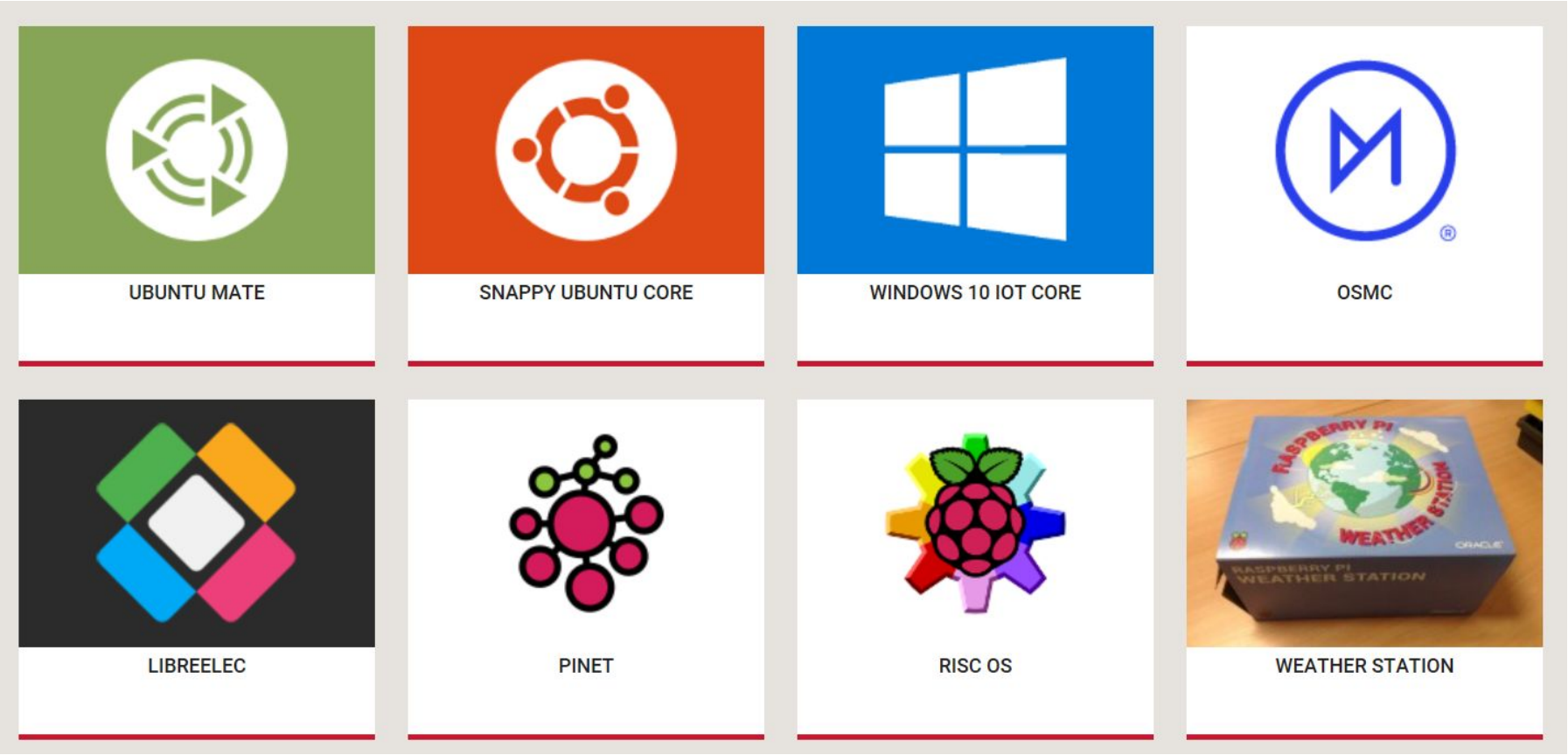
When ? How ?

Debug, add drivers, modify modules

How to install Linux OS to embedded system ?

Depend on board. Not easy as install linux on computer !

- OS for embedded system



Linux Kernel Organization

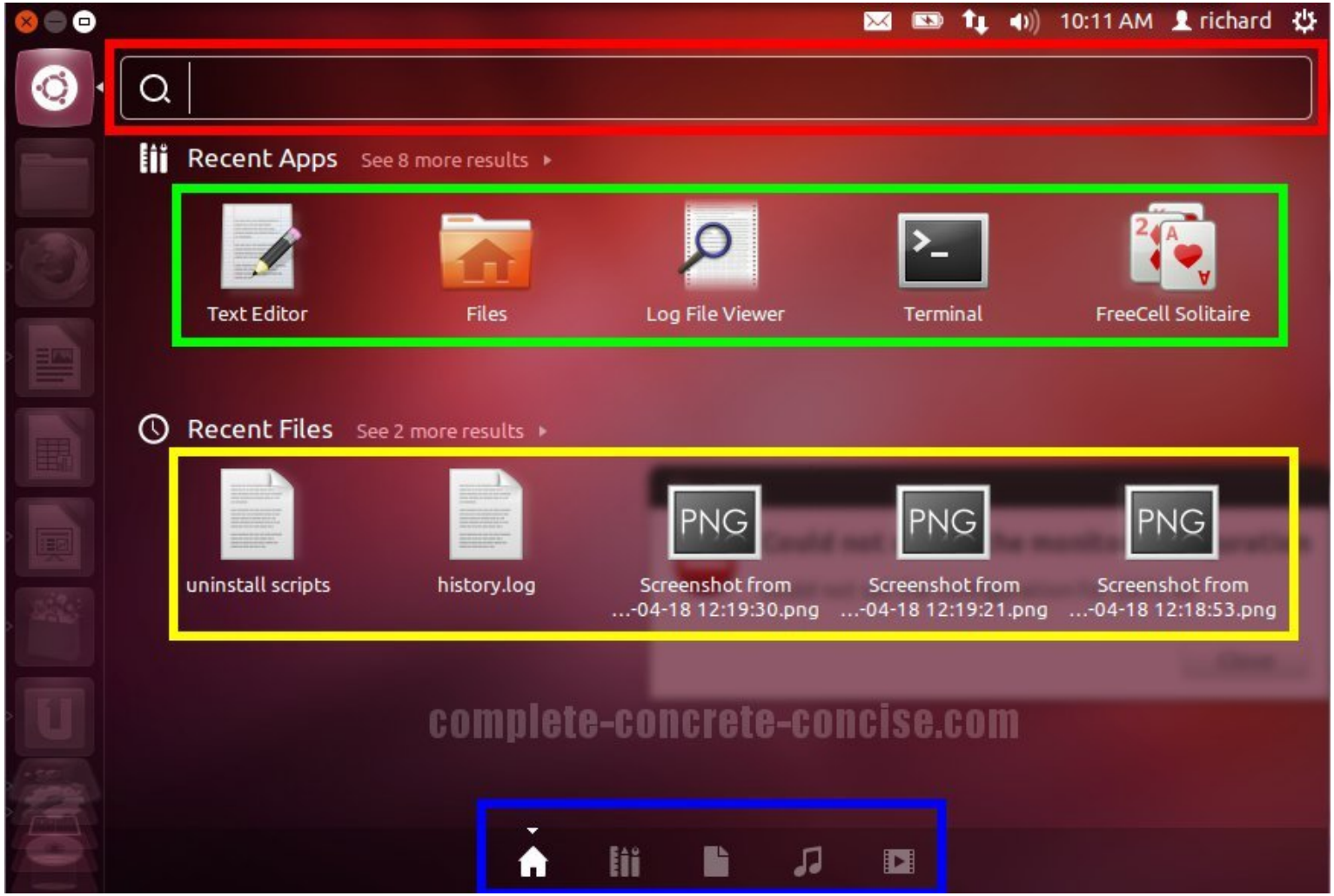
```
arch/            firmware/   kernel/    scripts/
block/           fs/         lib/       security/
crypto/          include/    mm/        sound/
Documentation/   init/       net/       usr/
drivers/         ipc/        samples/   virt/
```

- **/drivers**: largest folder, contains drivers for numerous hardware

- **/arch**: next largest, contains support for processors architectures

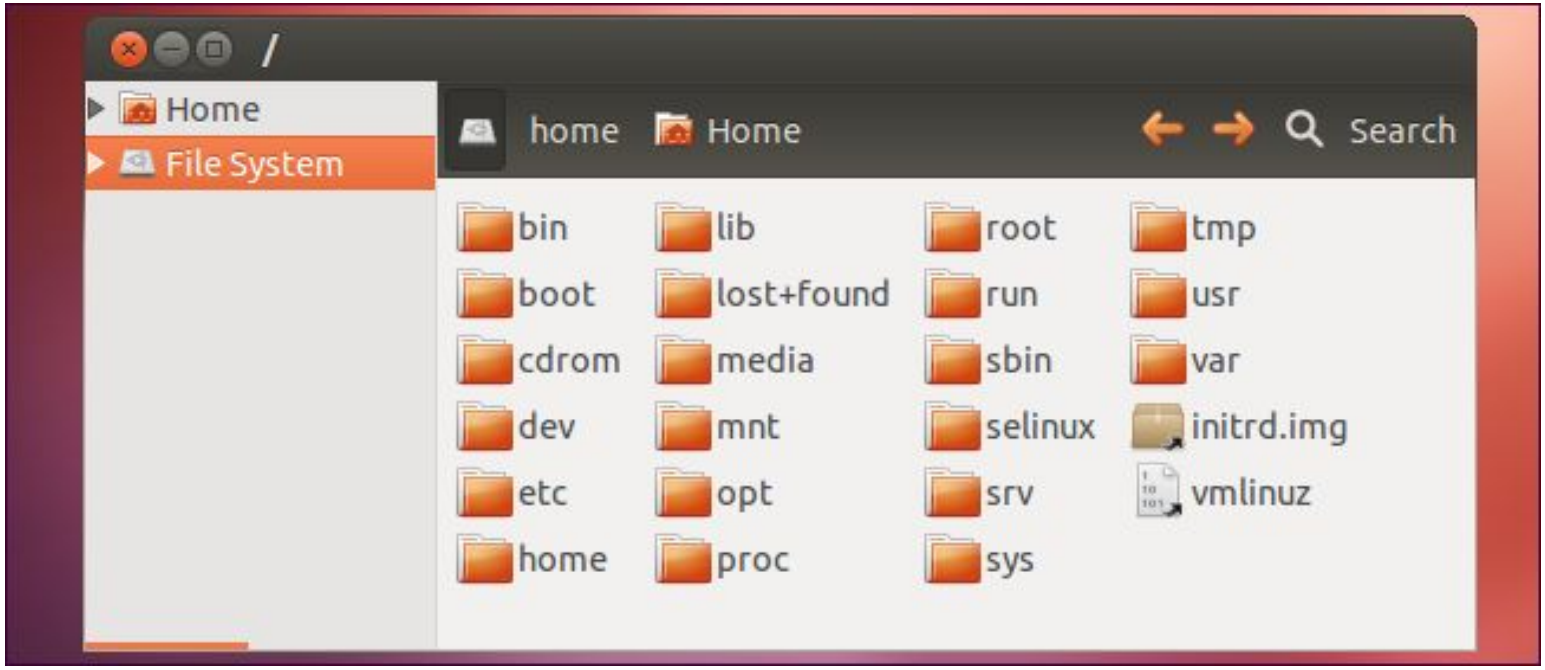
Ubuntu desktop

Ubuntu desktop



Linux File System

- Top-level Directory:



- **/bin**: Contains executable binary, commands used by all the users of the system are located here

Linux File System

- **/sbin**: contains executable binary files used typically by system administrator, for system maintenance purpose.
- **/etc**: contains configuration files required by all programs, startup and shutdown shell scripts
- **/dev**: contains device files
  - E.g. /dev/tty1, /dev/usbmon0
- **/proc**: contains information about running system process
- **/home**: store user personal files

Linux File System

- **/boot**: contains boot loader related files
- **/usr**: contains binaries, libraries, documentation, and source code for second level programs.
- **/opt**: contains add-on applications from user, individual vendors
- Refer to this link for more information

http://www.thegeekstuff.com/2010/09/linux-file-system-structure/?utm_source=tuicool

Warm up

- Open **Ubuntu** -> **Files** -> **Home**
- Create a new folder name “**bai_tap**”
- Open **Text Editor**
- Type “Hello world” and save as a new file name “**bt_1**” in **Home** folder
- Right click on “**bt_1**”. Select **Properties** -> **Permission**
- Change Owner Acess to **Read Only**, then Close
- Try to edit the file and save it
- Try to create a new file

Linux Terminal

- The command line is an interesting beast
- Most of time you communicate with the embedded board via Terminal
- Terminal use **BASH** language
- Linux ignores the extension and looks inside the file to determine what type of file it is.
- Linux is Case Sensitive

  e.g. file1.TXT ≠ file1.txt
- Careful with space in file names
- The Linux command line does not have an undo feature. Perform destructive actions carefuly.

Linux Terminal

Useful commands:

- **pwd** – Print Working Directory

  user@bash: pwd

  /home/haduc

- **ls** – list all folders and files in the working directory

  user@bash: ls

  bin Documents public_html

  user@bash: ls -l /etc

- **cd [location]** – move to the [location folder]

  e.g. cd /usr/local/bin

  cd ../games

Absolute and Relative Paths

Absolute paths specify a location (file or directory) in relation to the root directory, always begin with a forward slash (/)

e.g. /home/ryan

Relative paths specify a location (file or directory) in relation to where we currently are in the system, not begin with a slash.

e.g. Documents

Linux Terminal

- **man <command>** - Look up the manual page for a particular command.
- **mkdir [options] <Directory>** - Create a new folder

  e.g. mkdir chuyende
- **rmdir [options] <Directory>** - Remove a folder
- **touch [options] <filename>** - Create a blank file
- **cp [options] <source> <destination>** - copy a file or folder
- **mv [options] <source> <destination>** - move a file or folder
- **rm [options] <file>** - remove a file
- **gedit <file>** - open Text Editor to edit file

Linux Terminal

Permission: specify what a particular person may or may not do with respect to a file or directory

```
user@bash: ls -l frog.png
-rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
```

- : file **d** : folder	permissions for the **owner**	permissions for **groups**	permissions for **others**

- **r** read - you may view the contents of the file.
- **w** write - you may change the contents of the file.
- **x** execute - you may execute or run the file if it is a program or script.

- chmod [permissions] [path] - change permissions on a file or directory

```
user@bash: chmod g+x frog.png
user@bash: ls -l frog.png
-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
```

Linux Terminal

- **chown [option] [path]** – Change the ownership of a file or directory

  E.g. to change the owner of /foo and subfiles to “root”, run:

  chown -R root /foo

  E.g. get permission to modify /var/www/html :

  sudo chown -R pi:pi /var/www/html
- **passwd** – change user password
- **su** – switch from user to root account
- **exit** – return to user account
- **history** – display all previous command
- **clear** – clear the terminal screen

Linux Terminal

- **df** – shows the size, used space, and available space on the mounted filesystems

  e.g. df -h -> show list of file system

- **echo** – print a string or string variable on terminal screen

  e.g. echo Hello

  echo $PATH

  echo $HOME

- **grep** – searches for lines which contain a search pattern

  e.g grep train *.txt

  searching for the word “train” in all text files in the current directory

Linux Terminal

- **Unzip Files in Linux**
  - **Installing unzip**

    sudo apt-get install unzip
  - **Unzip a file**

    unzip filename.zip

- **Download file from web**
  - **Installing wget**

    sudo apt-get install wget
  - **Download file with wget**

    wget http://website.com/files/file.zip

  e.g. Download opencv from web

  wget https://github.com/opencv/opencv/archive/3.2.0.zip -O opencv_source.zip

Create your 1st program on Linux

- Move file “bt_1” to “bai_tap” folder.
- Open “bt_1” with Text editor
- Type the source code for the program as follows:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf(“Hello World\n”);
    return 0;
}
```

- Save as “hello.c”.
- On Terminal, move to “bai_tap” folder and type

  gcc -o hello hello.c
- To run the program, type

  ./hello

“Hello World” Program Explain

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf(“Hello World\n”);
    return 0;
}
```

hello.c

Need to include **stdio.h**

The code is compiled with GCC compiler

```
$ gcc  -o hello  hello.c
```

And create a binary file named “hello”

When we type a command on the command line, the system runs through a preset series of directories, looking for the program we specified.

```
user@bash: echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib/mit/bin:/usr/lib/mit/sbin
```

To override this behavior, we can provide path for the program with “**./**”

Compile “hello” program

- **Method 1**: type command

```
gcc -o hello hello.c
```

Useful when compile simple program

- **Method 2**: Create and run a Makefile

Very useful when build a program that may need various modules/source files.

- **Method 3**: Use Cmake

Makefile

- Makefile is a program building tool which runs on Unix, Linux
- Makefile simplify the procedure of building a program that may need various modules.
- Makefile determine how the modules need to be compiled or recompiled together
- For example, let's assume we have the following source files:
    - main.cpp
    - hello.cpp
    - factorial.cpp
    - functions.h

Makefile

- We can compile using command:

```
gcc  main.cpp hello.cpp factorial.cpp -o hello
```

But need to careful about the sequence of the function calls.

What happen if we have few hundred source files?

- Makefile:

```
CC = gcc
CFLAGS = -g
hello: main.cpp hello.cpp factorial.cpp
    $(CC) $(CFLAGS) $? -o $@
clean:
    -rm *.o hello
```

Makefile

target
action

```
CC = gcc
CFLAGS = -g
hello: main.cpp hello.cpp factorial.cpp
    $(CC) $(CFLAGS) $? -o $@
clean:
    rm *.o hello
```

- Structure of target:

  target: *dependencies*
- Action lines should **start after a tab**
- CC, CFLAGS: Macro, need to predefine
  - E.g. `CC = gcc` Program for compiling C programs

    `CFLAGS = -g` Extra flags to give to the C compiler

    `LDFLAGS = -lGL` Extra flags to give to compiler when they are supposed to invoke the linker

Refer to this link for more detail on macro
https://www.tutorialspoint.com/makefile/makefile_macros.htm

Makefile

target
action

```
CC = gcc
CFLAGS = -g
hello: main.cpp hello.cpp factorial.cpp
    $(CC) $(CFLAGS) $? -o $@
clean:
    rm *.o hello
```

- $@ : is the name of the file to be made.
- $? : is the names of the changed source files

- Run the makefile by command: **make** or **make hello**
- Clear target: determine what will be done when command **make clean** is called

Example

- What will happen if this makefile is called ?

```
CC=gcc
OUTPUT=Hello
all:Hello.o display.o
   $(CC) -o $(OUTPUT) Hello.o display.o
Hello.o:Hello.c
   $(CC) -c Hello.c
display.o:display.c
   $(CC) -c display.c
```

CMake

- CMake is a meta build system.

(refer to https://cgold.readthedocs.io/en/latest/overview/cmake-can.html for more info)

- Installing Cmake

```
sudo apt-get -y install cmake
```

- Compile a project with Cmake
  - Step 1: create an empty folder and put your code files into it
  - Step 2: create a CMakeLists.txt in the same folder

CMake

- Step 3: Open CMakeLists.txt, insert following lines

```
cmake_minimum_required(VERSION 2.8)

# set the project name
project(Example)
# Find the require packet (e.g. opencv)
find_package( OpenCV REQUIRED )

# Set the include folder (e.g. opencv include folder)
include_directories( ${OpenCV_INCLUDE_DIRS} )

# Add the executable
add_executable(example example.cpp)

# Add library (e.g. opencv library)
target_link_libraries(example ${OpenCV_LIBS} )
```

CMake

- Step 4: Build the project

  cmake .

  make
- Step 5: Recompile if the source code is changed

  make

Source code management

- **Source code management** (SCM) is used to
  - track modifications to a source code repository
  - revert selected files back to a previous state
  - collaborate with developers on other systems

- Tracking code modifications
  - Manual: use Linux commands such as diff, patch
  - Auto: git, github

Source code management

- Compare files in Linux

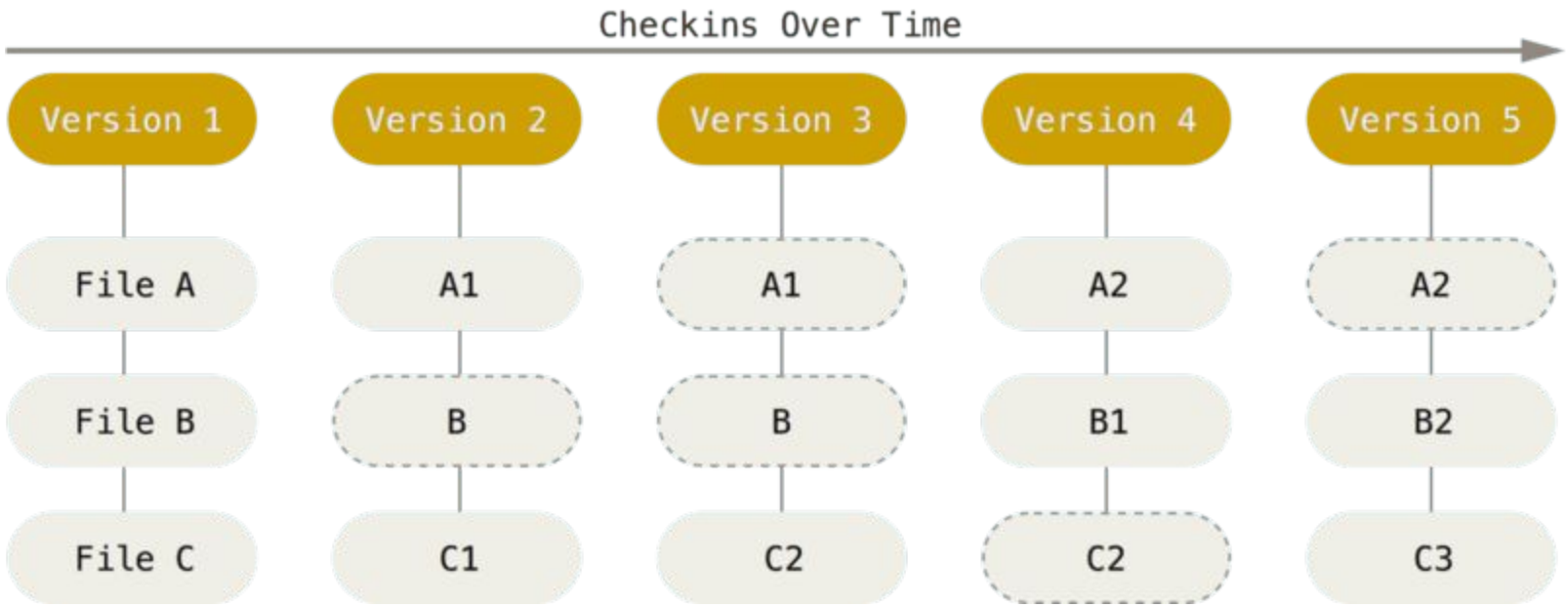
```
diff [options] old_file new_file

diff -u old_file new_file  > change.diff
```

- Apply changes:

```
patch old_file < change.diff
```

Source code management

- Git: Created by Linus Torvald in 2005



Source code management

- Git: