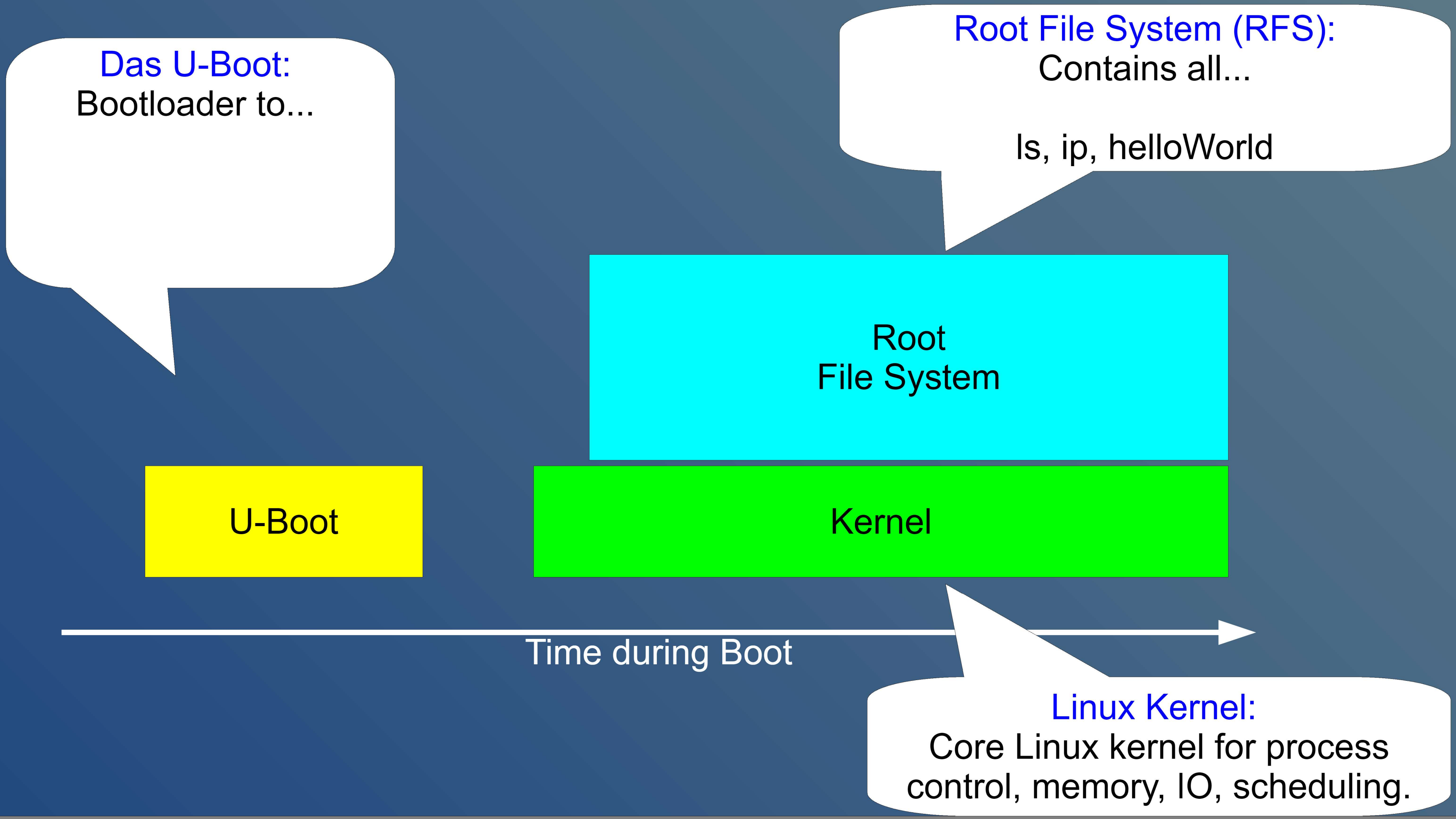
Launching & Building Embedded Software

U-Boot,
Cross Compiling,
Make, CMake
& Editors

Topics

1) What software components run on the board?
2) How can we build our software?
3) How can we edit files via just text console?

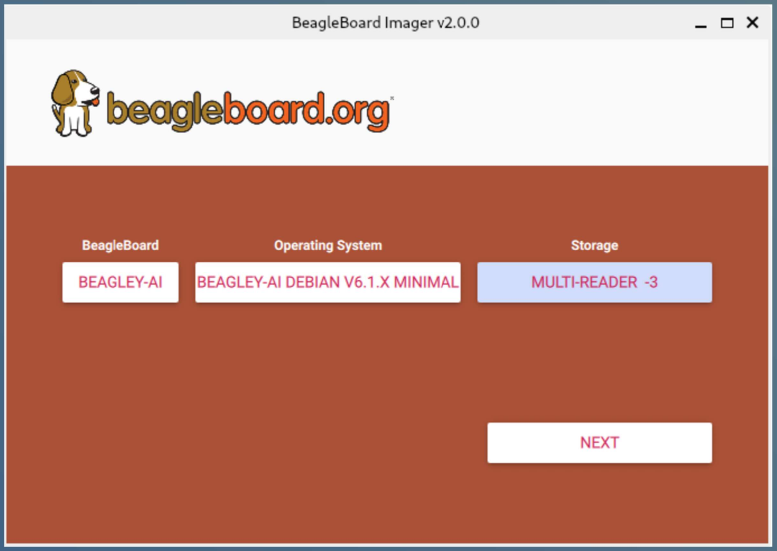
Software Components



Making a micro SD Card

- Micro SD
  - The micro SD (uSD) card contains all software to boot the board.
- bb-imager
  - ..
  - Easy to configure your settings (user name, password, wifi, ...)

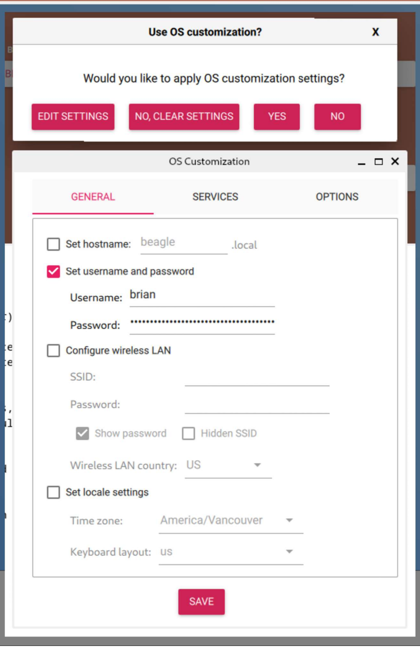
    I had poor experiences under Windows; so run on Linux (or VM in Windows)


BeagleBoard Imager v2.0.0

BeagleBoard: BEAGLEY-AI

Operating System: BEAGLEY-AI DEBIAN V6.1.X MINIMAL

Storage: MULTI-READER -3


Use OS customization?

Would you like to apply OS customization settings?

OS Customization

☐ Set hostname: beagle .local

☑ Set username and password

Username: brian

Password:

☐ Configure wireless LAN

SSID:

Password:

☑ Show password ☐ Hidden SSID

Wireless LAN country: US

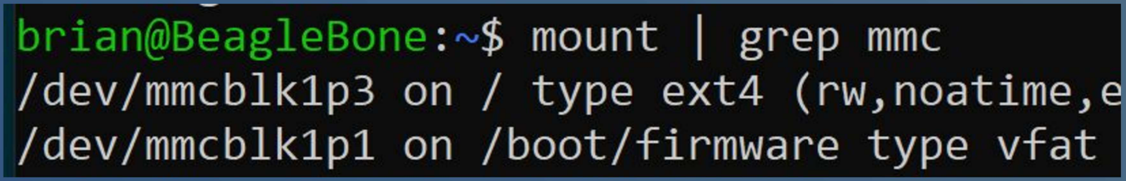
☐ Set locale settings

Time zone: America/Vancouver

Keyboard layout: us

uSD Contents

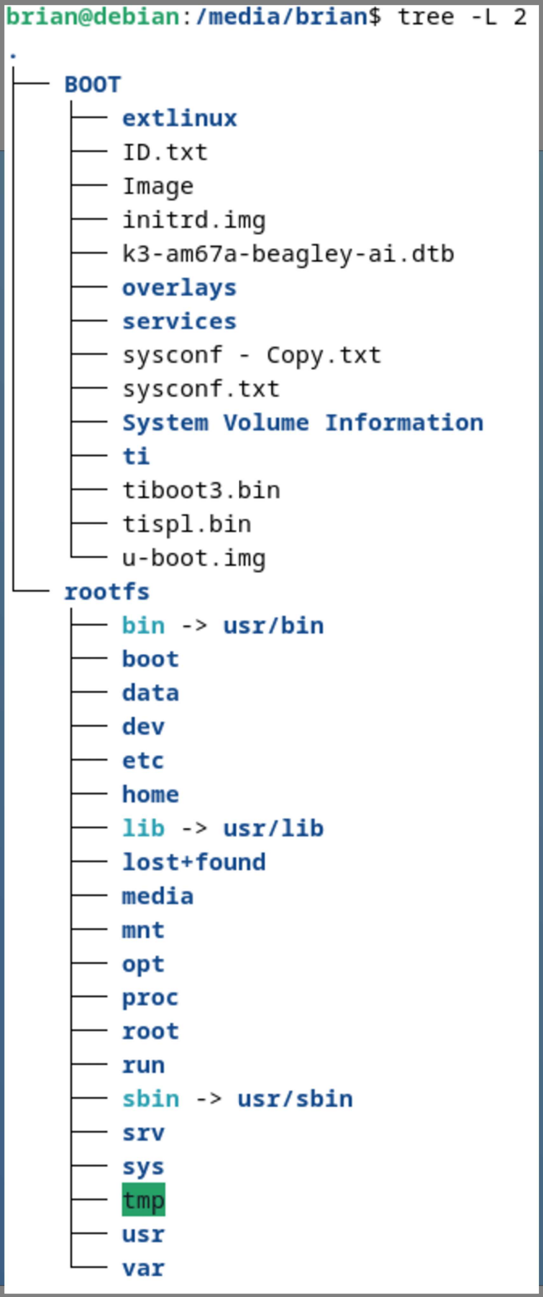
- ..



```
brian@BeagleBone:~$ mount | grep mmc
/dev/mmcblk1p3 on / type ext4 (rw,noatime,e
/dev/mmcblk1p1 on /boot/firmware type vfat
```

Physical Partition | Mount Location | File system type (ext4 or vfat)

  - BOOT
    Contains config info for boot.
    Readable by Windows/MacOS.
    Mounts on BYAI to /boot/firmware/
  - rootfs
    All files for root fs

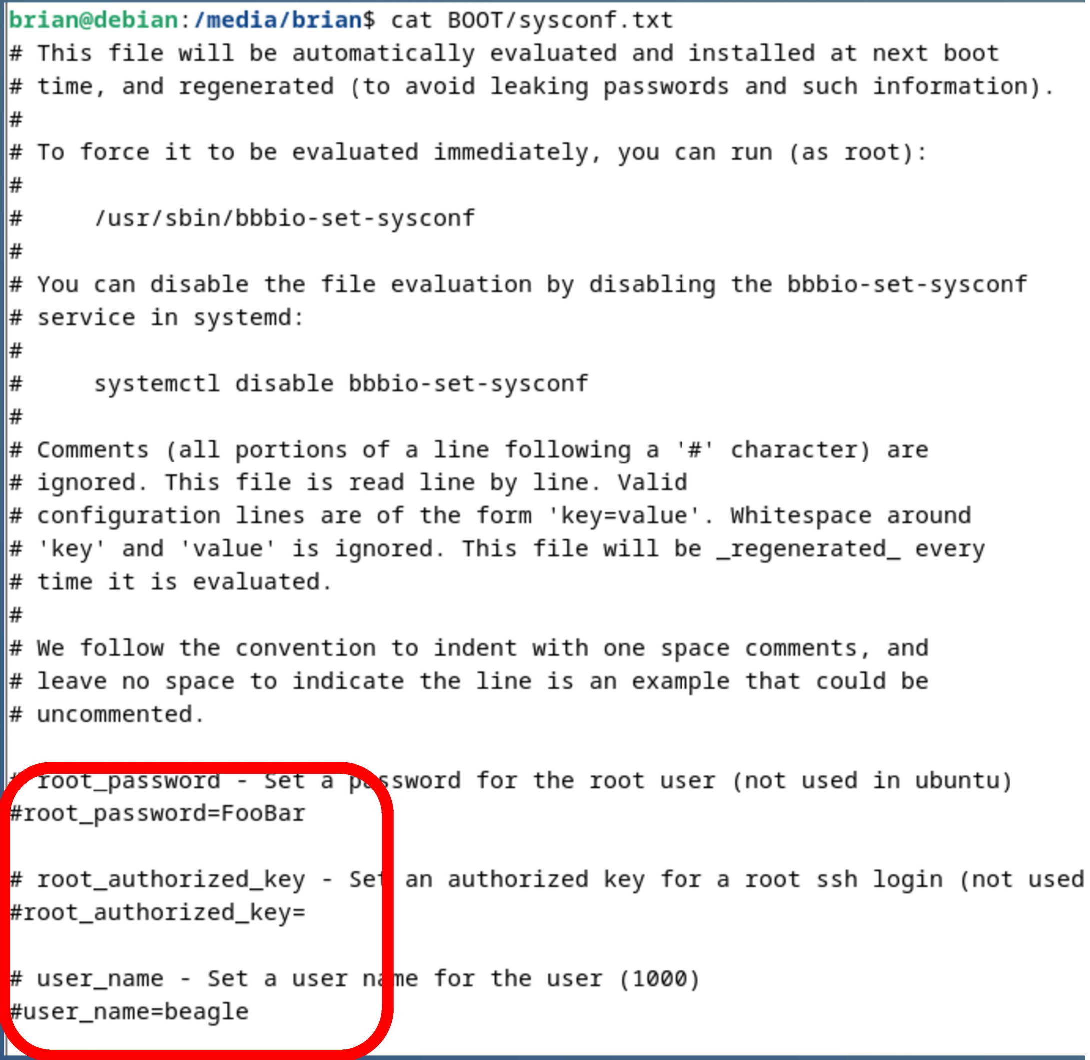


```
brian@debian:/media/brian$ tree -L 2
.
├── BOOT
│   ├── extlinux
│   ├── ID.txt
│   ├── Image
│   ├── initrd.img
│   ├── k3-am67a-beagley-ai.dtb
│   ├── overlays
│   ├── services
│   ├── sysconf - Copy.txt
│   ├── sysconf.txt
│   ├── System Volume Information
│   ├── ti
│   ├── tiboot3.bin
│   ├── tispl.bin
│   └── u-boot.img
└── rootfs
    ├── bin -> usr/bin
    ├── boot
    ├── data
    ├── dev
    ├── etc
    ├── home
    ├── lib -> usr/lib
    ├── lost+found
    ├── media
    ├── mnt
    ├── opt
    ├── proc
    ├── root
    ├── run
    ├── sbin -> usr/sbin
    ├── srv
    ├── sys
    ├── tmp
    ├── usr
    └── var
```

Files in each partition

Configuring BYAI

- sysconf.txt
  - Read by Linux on BYAI at boot.
  - ..
  - Then program wipes the file and reboots target. (no password leak)
- bb_imager sets up this file with all your custom options about user name, password, and wifi.



```
brian@debian:/media/brian$ cat BOOT/sysconf.txt
# This file will be automatically evaluated and installed at next boot
# time, and regenerated (to avoid leaking passwords and such information).
#
# To force it to be evaluated immediately, you can run (as root):
#
#     /usr/sbin/bbbio-set-sysconf
#
# You can disable the file evaluation by disabling the bbbio-set-sysconf
# service in systemd:
#
#     systemctl disable bbbio-set-sysconf
#
# Comments (all portions of a line following a '#' character) are
# ignored. This file is read line by line. Valid
# configuration lines are of the form 'key=value'. Whitespace around
# 'key' and 'value' is ignored. This file will be _regenerated_ every
# time it is evaluated.
#
# We follow the convention to indent with one space comments, and
# leave no space to indicate the line is an example that could be
# uncommented.

# root_password - Set a password for the root user (not used in ubuntu)
#root_password=FooBar

# root_authorized_key - Set an authorized key for a root ssh login (not used
#root_authorized_key=

# user_name - Set a user name for the user (1000)
#user_name=beagle
```

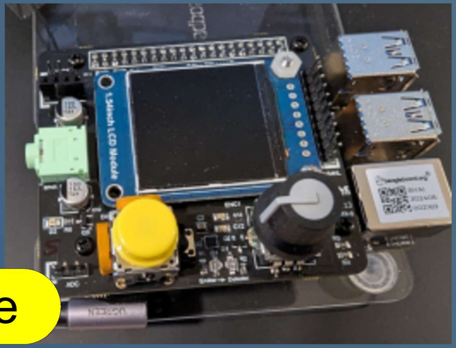
Servers & Directories

- Work (private) Directory
  - ..
    E.g., .c, .h, filelists.txt, makefile
    Suggestion: *Put this into GitHub!*
- Public Directory
  - Holds files to...
  - Unprotected by passwords!
    Only for compiled code.

$HOME/ensc351/work

$HOME/ensc351/public

/mnt/remote



Host

Target

Cross-compile demo

- Compile on host for target

```
(host)$ aarch64-linux-gnu-gcc hello.c -o hello
```

- Check compiled file

```
(host)$ readelf -h hello
```

- Run on board via NFS (one line each)

```
(byai)$ sudo mount -t nfs \
          192.168.7.1:/home/matt/ensc351/public \
         /mnt/remote

(byai)$ cd /mnt/remote/
(byai)$ ./hello
```

Boot, sysconf.txt

- What sequence of software runs during the target's boot?

a) RFS > Kernel > UBoot
b) RFS > UBoot > Kernel
c) Kernel > UBoot > RFS
d) UBoot > Kernel > RFS

- What is the purpose of the sysconf.txt file?

a) Change Linux settings on the target.
b) Store Linux settings on the target.
c) Select a cross compiler targeting the BYAI.
d) Mount folders off micro SD card or target.

ABCD: Running from Target

- When *SSH'd into the target*, and having performed the standard setup described above, which of the following will run a cross-compiled **`helloworld`** app?

```
a) ~/ensc351/public/myApps/helloworld
b) /media/rfs/myApps/helloworld
c) /mnt/remote/myApps/helloworld
d) /nfs/myApps/helloworld
```

Building Software With

Make & CMake

Makefile Basics

- Makefiles are
  ..
  - Name your script Makefile
  - Build a specific make-target with:..

    ```
    (host)$
    ```
  - Build default make-target with:

    ```
    (host)$ make
    ```
- Examples

  ```
  (host)$ make clean
  (host)$ make all
  ```

Simple Makefile

```
# Simple Makefile for building Hello world!

CC_C = aarch64-linux-gnu-gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror

app:
	$(CC_C) $(CFLAGS) helloWorld.c -o hello
	cp hello ~/ensc351/public/myapps/
    ...

clean:
	rm hello
```

Define custom variables for later use.

Targets of form `targetName:`

Command(s) for this target.

clean is a common target to remove all build files.

More Makefile

```
OUTFILE = helloWorld
OUTDIR = $(HOME)/ensc351/public/myApps

CROSS_COMPILE = aarch64-linux-gnu-
CC_C = $(CROSS_COMPILE)gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror

help:
	@echo "Build Hello World program for BeagleY-AI"
	@echo "Targets include all, app, and clean."

all: app nestedDir done

app:
	$(CC_C) $(CFLAGS) helloWorld.c -o $(OUTDIR)/$(OUTFILE)
	ls -l $(OUTDIR)/$(OUTFILE)

nestedDir:
	make --directory=myNestedFolder

done:
	@echo "Finished building application."

clean:
	rm $(OUTDIR)/$(OUTFILE)
```

Setup output info once, used twice.

Compiler Flags

```
OUTFILE = factorial
OUTDIR = $(HOME)/ensc351/public/myApps

CROSS_COMPILE = aarch64-linux-gnu-
CC_C = $(CROSS_COMPILE)gcc
CFLAGS = -Wall -g -std=c11 -D _POSIX_C_SOURCE=200809L -Werror
  ..
```

Debug symbols

Explicit POSIX support (for nanosleep() function).

Warnings as errors.

..... rest of makefile omitted...

CMake

- CMake =..
  - Manage software build process
    ..
  - Supports intelligently recompiling only the files that changed
  - CMake Scripts:
    Describe the build process: CMakeLists.txt

    Can have multiple scripts:
    one to build each part, one to combine, etc.
- CMake is a Meta Build System
  1) CMake processes CMakeLists.txt files to..
  2) Use GNU Make to build the software using those Makefiles

Anatomy of CMakeLists.txt

CMakeLists.txt

```
# Minimum version. Run on the host.
cmake_minimum_required(VERSION 3.18)

# Project info
project(
    SimpleCMakePrj
    VERSION 1.0
    DESCRIPTION "Simple demo of CMake"
    LANGUAGES C
)

# Compiler options
set(CMAKE_C_STANDARD 11)
add_compile_options(-Wall -Werror -Wpedantic -Wextra)

add_executable( simple_cmake
    src/main.c
    src/funstuff.c
)
```

Required Elements

Lowest CMake version that will build our system (on host).

Many commands take key-value pair like: VERSION 3.18

Info about project: name, version, necessary compilers, etc.

Generate this executable (1st arg) using these source files

Running CMake - Terminal (for host)

- Regenerate build/ folder and makefiles:

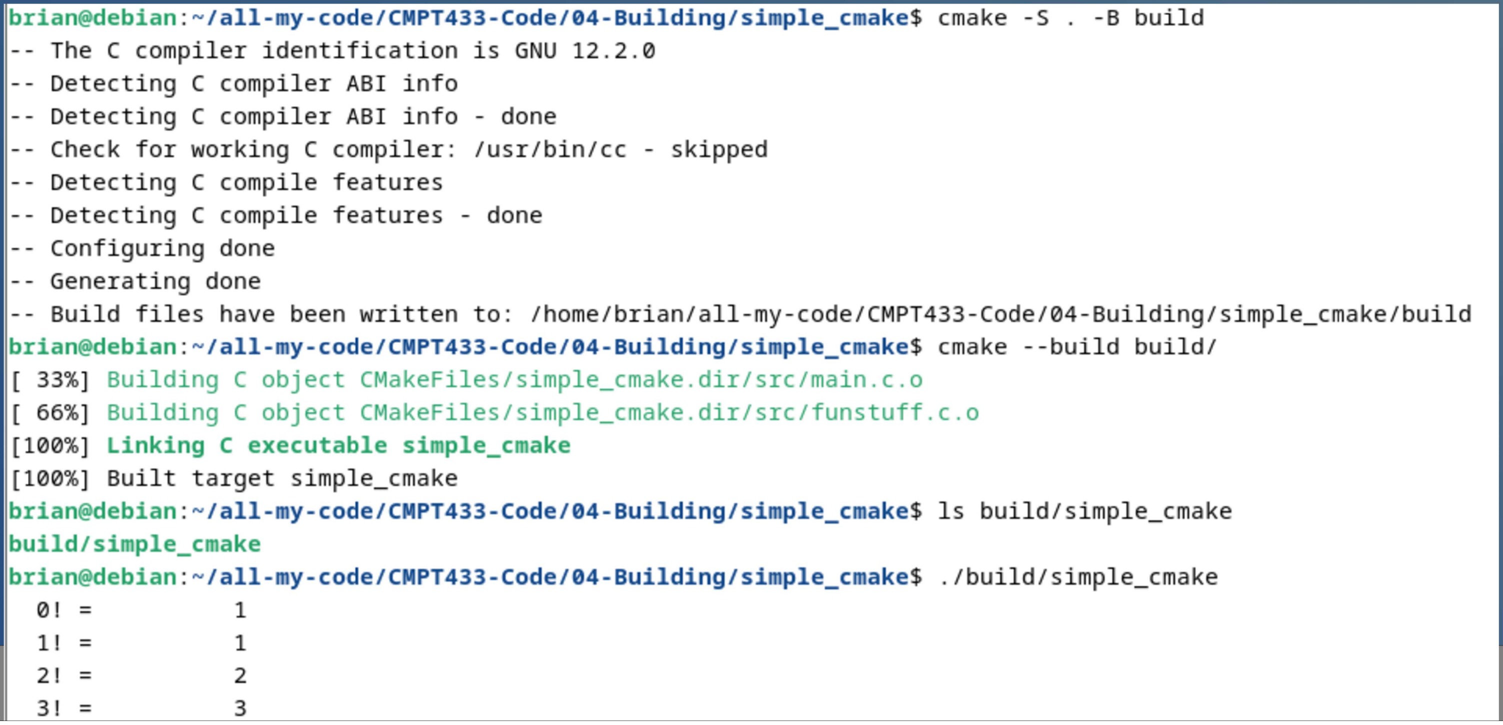
```
(host)$ cmake -S . -B build
```

- Build (compile & link) the project

```
(host)$ cmake --build build/
```

- Clean up temporary build folder (when needed)

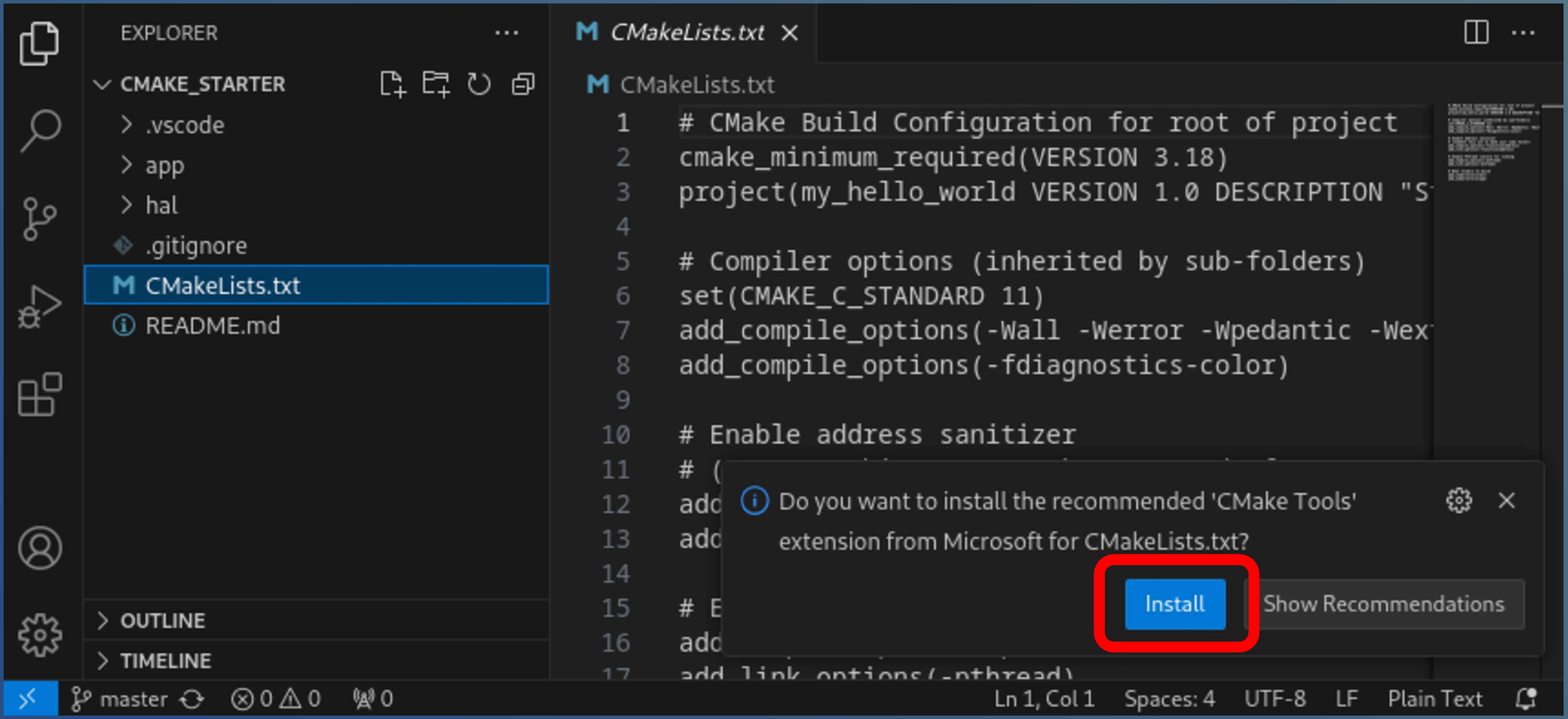
```
(host)$ rm -r build/
```



```
brian@debian:~/all-my-code/CMPT433-Code/04-Building/simple_cmake$ cmake -S . -B build
-- The C compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/brian/all-my-code/CMPT433-Code/04-Building/simple_cmake/build
brian@debian:~/all-my-code/CMPT433-Code/04-Building/simple_cmake$ cmake --build build/
[ 33%] Building C object CMakeFiles/simple_cmake.dir/src/main.c.o
[ 66%] Building C object CMakeFiles/simple_cmake.dir/src/funstuff.c.o
[100%] Linking C executable simple_cmake
[100%] Built target simple_cmake
brian@debian:~/all-my-code/CMPT433-Code/04-Building/simple_cmake$ ls build/simple_cmake
build/simple_cmake
brian@debian:~/all-my-code/CMPT433-Code/04-Building/simple_cmake$ ./build/simple_cmake
  0! =          1
  1! =          1
  2! =          2
  3! =          3
```

Running CMake - VS Code's Addon

- CMake Tool addon loaded with project with a CMakeLists.txt



```
# CMake Build Configuration for root of project
cmake_minimum_required(VERSION 3.18)
project(my_hello_world VERSION 1.0 DESCRIPTION "S

# Compiler options (inherited by sub-folders)
set(CMAKE_C_STANDARD 11)
add_compile_options(-Wall -Werror -Wpedantic -Wex
add_compile_options(-fdiagnostics-color)

# Enable address sanitizer
# (
add
add

# E
add
add_link_options(-pthread)
```

Do you want to install the recommended 'CMake Tools' extension from Microsoft for CMakeLists.txt?

Install   Show Recommendations

Running CMake - VS Code's Addon

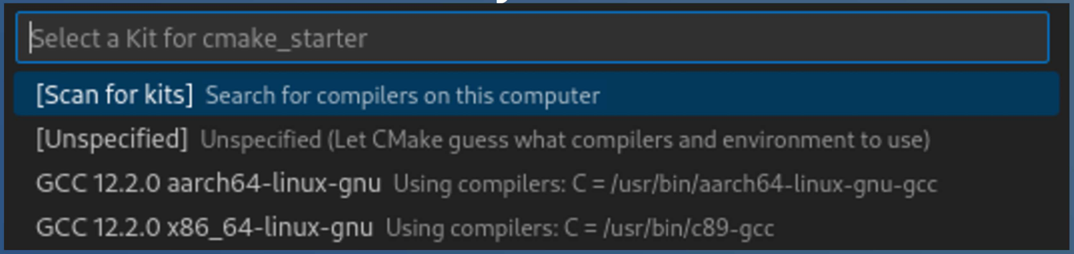
- ..

  "A kit encompasses project-agnostic and configuration-agnostic information about how to build code." [1]

  – Specifies compiler toolchain and version

  – We'll have one for native, one for cross-compile (Use "unspecified" to build natively)

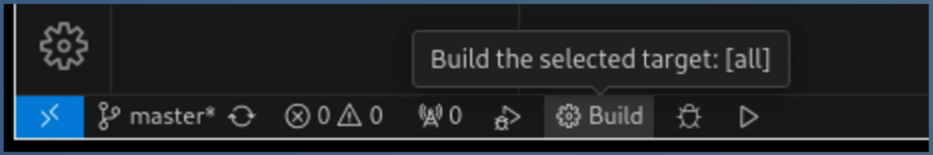
  – Addon scans host system for available toolchains



```
Select a Kit for cmake_starter
[Scan for kits]  Search for compilers on this computer
[Unspecified]  Unspecified (Let CMake guess what compilers and environment to use)
GCC 12.2.0 aarch64-linux-gnu  Using compilers: C = /usr/bin/aarch64-linux-gnu-gcc
GCC 12.2.0 x86_64-linux-gnu  Using compilers: C = /usr/bin/c89-gcc
```

- Building

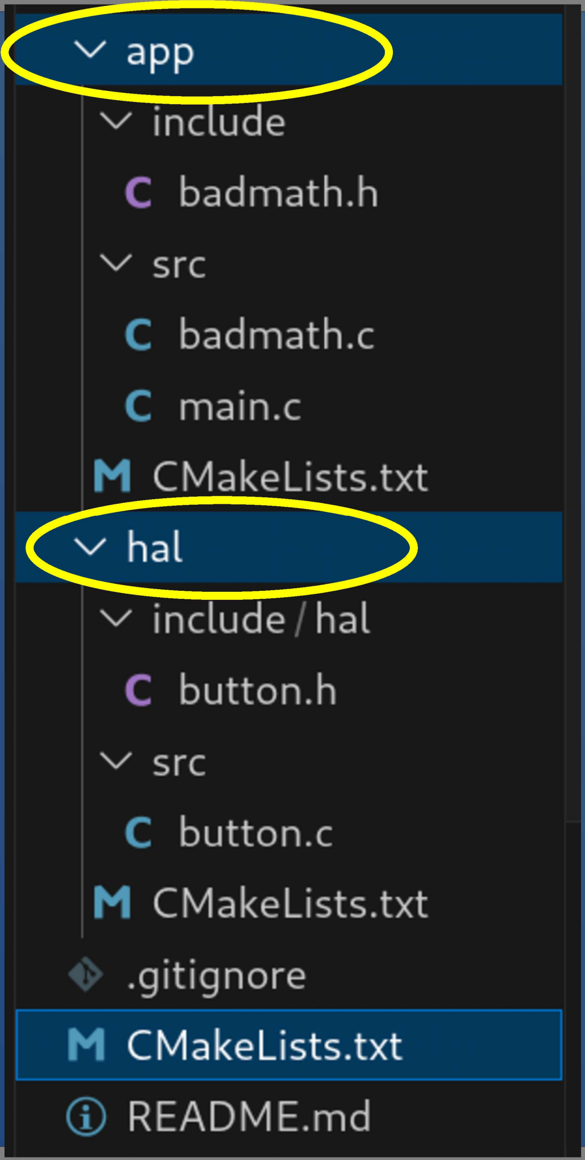
  – Generate then run makefiles:



  – Run makefiles: Ctrl + Shift + B
  Terminal > Configure Default Build Task... > CMake:Build

1. https://vector-of-bool.github.io/docs/vscode-cmake-tools/kits.html

CMake Starter Project



```
v app
  v include
    C badmath.h
  v src
    C badmath.c
    C main.c
  M CMakeLists.txt
v hal
  v include / hal
    C button.h
  v src
    C button.c
  M CMakeLists.txt
.gitignore
M CMakeLists.txt
README.md
```

- hal/ ..
  - Low-level *modules* with hardware specific details.
- app/ ..
  - Organized into *modules* for better organization and encapsulation
- build/
  - Created by CMake; *temporary*
- 3 CMakeLists.txt
  - One in root to control full build
  - One in each of hal/ and app/

ABCD: CMake

- What is a *primary benefit of CMake*?

a) It puts all build commands in one file.

b) Compiler independent make file.

c) Configures project options.

d) Removes need to install Make

- How does CMake *support cross-compiling*?

a) Uses toolchain file to select compiler.

b) Generates `CMakeLists.txt` from `Makefile`.

c) Writes all output into `build/` folder.

d) Allows for a HAL layer.

Nano

- Nano is a somewhat easier to use text editor.

  ```
  $ nano myfileToEdit.txt
  ```

  - Just type and edit text as you might expect.
- Commands
  - : Displays help. Ctrl+x to quit help.
  - : Quit, asks you if you want to save.

Simple create/view a file

- Redirect text to a file

```
$  echo      “Overwrite file with text”          test.txt
$            “Adding this to end of file”        test.txt
```

- View a file

```
$  cat       daFile
```

concatenate the file, outputs to stdout (terminal)

```
$  less      daLongFile
```

shows page-by-page view of long file

```
$  tail      -20 daLongFile
```

Shows last 20 lines of the file.

- Pipe output from one tool to another

```
$  dmesg
```

displays kernel messages

  -
    ```
    $ dmesg | less
    $ dmesg | tail -20
    ```

vi

To me vi is zen.
To use vi is to practice zen.
Every command is a koan.
Profound to the user, unintelligible to the uninitiated.
You discover truth every time you use it.

-- Satish Reddy

vi – **THE** editor

- vi is a text based editor build into most *unix's
- Launch by:
  vi <filename>
- 2 Modes of operation:

  - Used to move cursor, delete lines, save/quit.
  - Press to get to this mode.

  - Used to enter text.
  - Press to get from command mode to here.

Command: in Command mode!

Save / Quit

:w	-
:q	-
:wq	- Save and quit
:q!	- Quit without saving

Delete, undo, copy/paste

dd	-
u	- Undo *1* change (not on target!).
yy	- Copy current line (yank)
p	- Past copied line

Cursor Movement

Arrow keys: may work.. may not (do on board, not under Ubuntu).

h	- left
j	- down
k	- right
l	- right (a lower-case L)

Page Up/Down

Ctrl+f	- Forward a page
Ctrl+b	- Back a page

Note: Case sensitive commands.

Demo: Edit profile to run ./hello & color ls

Summary

- Boot sequence
  - UBoot --> Kernel --> Root File System
- Makefiles automate building software.
  - Create targets for different products/actions.
- CMake: cross-platfrom meta build system
  - Process defined in CMakeLists.txt
- Text-based Editors
  - Nano
  - vi