Buildroot 构建根文件系统(Jz2440)_jz2440 的 ubi 文件系统 - CSDN 博客

1. Buildroot 简介

Buildroot 是 Linux 平台上一个构建嵌入式 Linux 系统的框架。整个 Buildroot 是由 Makefile 脚本和 Kconfig 配置文件构成的。你可以和编译 Linux 内核一样，通过 buildroot 配置，menuconfig 修改，编译出一个完整的可以直接烧写到机器上运行的 Linux 系统软件 (包含 boot、kernel、rootfs 以及 rootfs 中的各种库和应用程序)。

下载地址： https://buildroot.org/download.html

2. 开发环境

Buildroot 版本：buildroot-2020.02.9.tar.gz
虚拟机：ubuntu 18.04.5 LTS
交叉编译器：arm-linux-gcc 4.4.3
Linux 内核：linux-4.15

3. buildroot 构建根文件系统

3.1 配置 buildroot

(1) 将 buildroot 源码 buildroot-2020.02.9.tar.gz 拷贝到 ubuntu 虚拟机中，解压源码：

```
tar xzf buildroot-2020.02.9.tar.gz
```

(2) 进入解压好的源码的目录中，配置 buildroot：

```
cd buildroot-2020.02.9/
```

```
make menuconfig
```

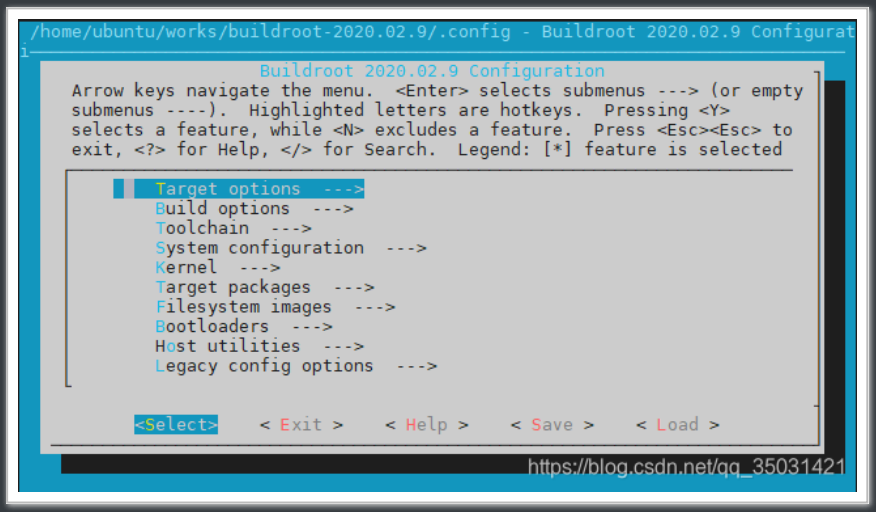
输入 `make menuconfig` 命令配置，最后会出现如下错误：

```
make[2]: *** [Makefile:253: /home/ubuntu/works/buildroot-2020.02.9/out
make[1]: *** [Makefile:960: /home/ubuntu/works/buildroot-2020.02.9/out
make: *** [Makefile:84: _all] Error 2
```

错误的原因：由于我的是新安装的 ubuntu 虚拟机，没有安装配置界面相关的库。

解决方法：`sudo apt install libncurses-dev`

(3) 重新执行 `make menuconfig` 命令后，弹出的配置界面如下图所示：



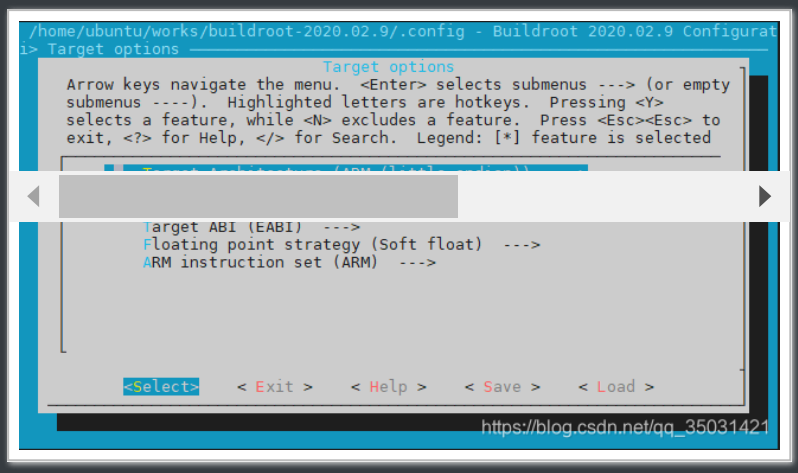
(4) 接下来我们一次配置 buildroot：

- ① 配置 Target options：首先配置 Target options 选项，需要配置的项目和其对应的内容如下（“=” 号后面是配置项
  要选择的内容，“//” 号后面是注释）：

```
Target options
    ->Target Architecture = ARM (little endian)  // 目标架构，这
    ->Target Binary Format = ELF                 // 二进制格式，
    ->Target Architecture Variant = arm920t      // s3c2440 的 C
    ->Target ABI = EABI                          // 应用程序二进
```

```
    ->Floating point strategy = Soft float        // 浮点数的策略,
    ->ARM instruction set = ARM                   // arm 汇编指令
```

配置完成后，如下图所示：

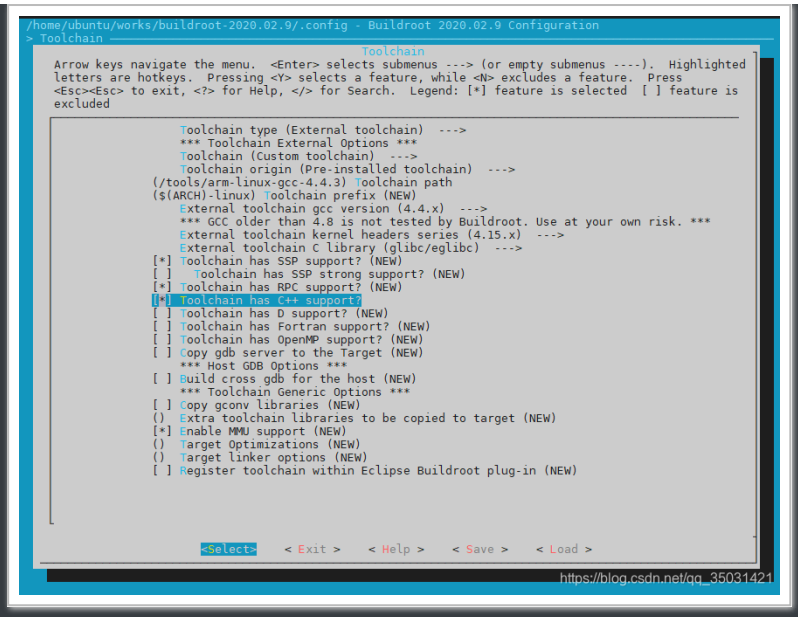


- ② 配置 Build option：主要是一些编译时用到的选项，比如 dl 的路径，下载代码包使用的路径，同时运行多个编译的上限，是否使能编译器缓冲区等等，这里按照默认就行了。

- ③ 配置 Toolchain：此配置项用于配置交叉编译工具链，也就是交叉编译器，这里设置为我们自己所使用的交叉编译器即可。 buildroot 其实是可以自动下载交叉编译器的，但是都是从国外服务器下载的，速度比较慢。配置内容如下：

```
Toolchain
    ->Toolchain type = External toolchain          // 工具链的类型
    ->Toolchain = Custom toolchain                 // 用户自己的交
    ->Toolchain origin = Pre-installed toolchain   // 预装的编译器
    ->Toolchain path = /tools/arm-linux-gcc-4.4.3 // 编译器的路径
    ->Toolchain prefix = $(ARCH)-linux             // 编译器的前缀
    -> External toolchain gcc version = 4.4.x      // 工具链的版本
    ->External toolchain kernel headers series = 4.15.x    // 内核
    ->External toolchain C library = glibc/eglibc
    ->[*] Toolchain has SSP support? (NEW)         // 选中
    ->[*] Toolchain has RPC support? (NEW)         // 选中
    ->[*] Toolchain has C++ support?               // 选中
    ->[*] Enable MMU support (NEW)                 // 选中
```

配置完后后，如下图所示：

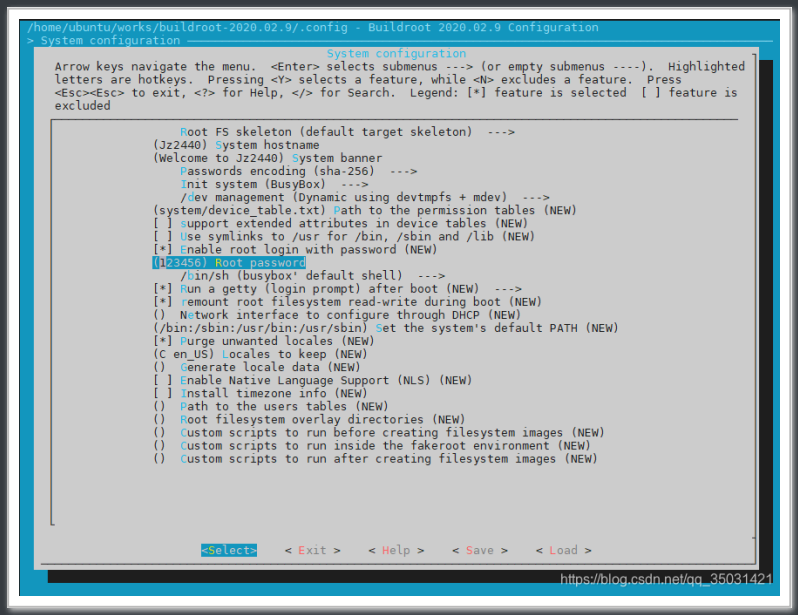


- ④ 配置 System configuration：此选项用于设置一些系统配置，比如开发板名字、欢迎语、用户名、密码等。需要配置的

  项目和其对应的内容如下：

```
System configuration
    ->System hostname = Jz2440                          // 系统名
    ->System banner = Welcome to Jz2440                 // 欢迎语
    -> Init system = BusyBox                            // 使用b
    -> /dev management = Dynamic using devtmpfs + mdev  // 使用
    -> [*] Enable root login with password (NEW)        // 使能登
        -> Root password = 123456                       // 登录密
    其他选项保持默认状态
```

配置完成如下图所示：

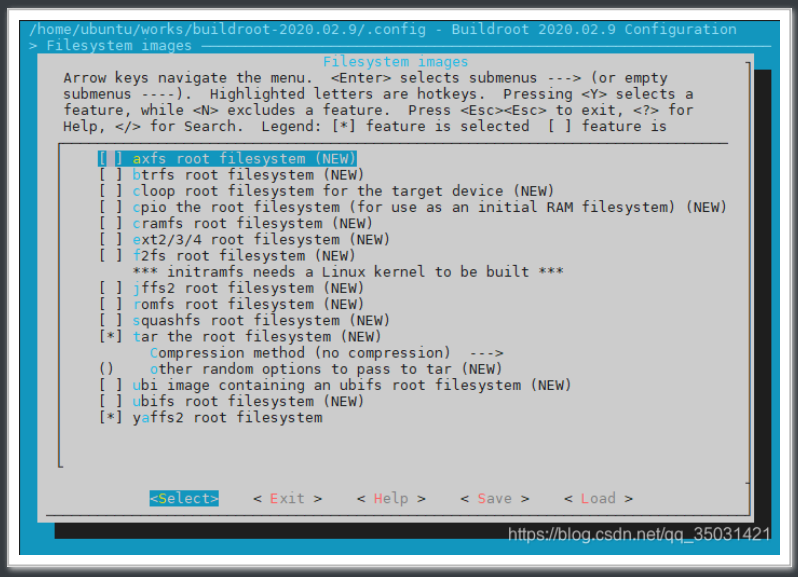


注：当我们配置时需要输入内容时，输错了，可以按 **ctrl + backspace** 组合键删除输错的信息。

- ⑤ 配置 Filesystem images：此选项配置我们最终制作的根文件系统为什么格式的，配置如下：

```
Filesystem images
    ->[*] yaffs2 root filesystem    // 配置根文件系统的格式为 yaf
```

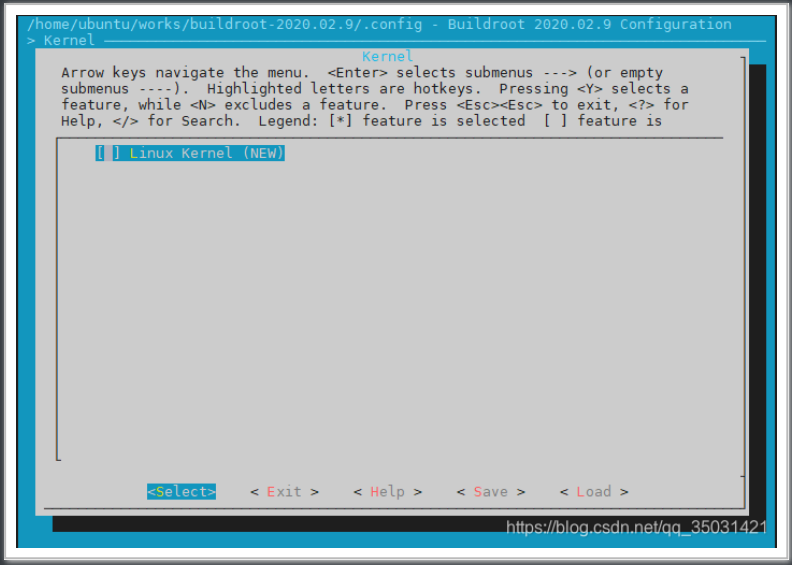
配置完成如下图所示：



- ⑥ 禁止编译 Linux 内核和 uboot：buildroot 不仅仅能构建根文件系统，也可以编译 linux 内核和 uboot。当配置 buildroot，使能 linux 内核和 uboot 以后 buildroot 就会自动下载最新的 linux 内核和 uboot 源码并编译。但是我们一般都不会使用 buildroot 下载的 linux 内核和 uboot，因为 buildroot 下载的 linux 和 uboot 官方源码，里面会缺少很多驱动文件，而且最新的 linux 内核和 uboot 会对编译器版本号有要求，可能导致编译失败。因此我们需要配置 buildroot，关闭 linux 内核和 uboot 的编译，只使用 buildroot 来构建根文件系统， 首先是禁止 Linux 内核的编译， 配置如下：

```
Kernel
    ->[ ] Linux Kernel (NEW)  // 不要选择编译 Linux Kernel 选项
```

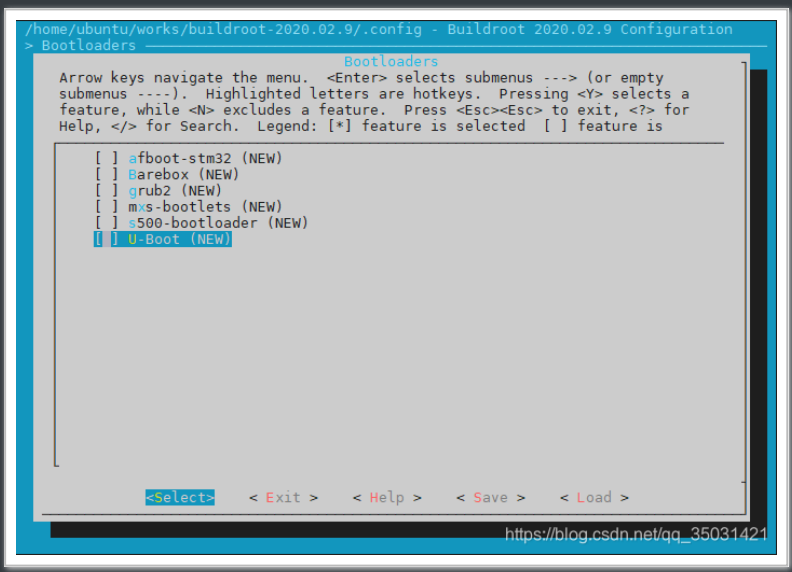
配置完成如下图所示：



禁止编译 uboot：

```
Bootloaders
    ->[ ] U-Boot // 不要选择编译 U-Boot 选项
```

配置完成如下图所示：



- ⑦ 配置 Target packages：此选项用于配置要选择的第三方库或软件、比如 alsa-utils、 ffmpeg、 iperf 等工具，但是现

  在我们先不选择第三方库，防止编译不下去！先编译一下最基本的根文件系统，如果没有问题

  的话再重新配置选择第三方库和软件。

3.2 编译

(1) 配置完成以后就可以编译 buildroot 了，编译完成以后 buildroot 就会生成编译出来的根文件系统压缩包，我们可以直接使用。输入如下命令开始编译：

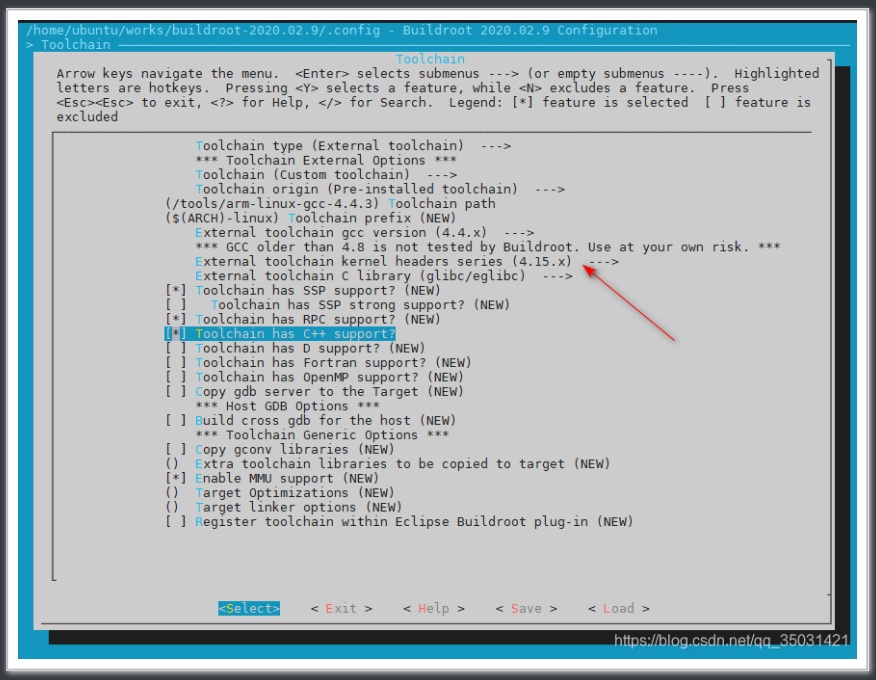
```
make    // 不能通过-jx 来指定多核编译
```

(2) 编译时产生了如下错误：

```
Incorrect selection of kernel headers: expected 4.15.x, got 2.6.x
package/pkg-generic.mk:254: recipe for target '/home/ubuntu/works/buil
make[1]: *** [/home/ubuntu/works/buildroot-2020.02.9/output/build/tool
Makefile:84: recipe for target '_all' failed

make: *** [_all] Error 2
```

意思是我们在配置 buildroot 的 Toolchain 时，选择的内核头为 4.15.x，而交叉编译器里的内核头为 2.6.x，如下图所示：



解决方法：

- ① 打开交叉编译器的 **version.h** 文件，本人的交叉编译器 **version.h** 文件的路径为：/tools/arm-linux-gcc-

4.4.3/arm-none-linux-gnueabi/sys-root/usr/include/linux/version.h，使用 vim 编辑器打开该文件：

```
sudo vim /tools/arm-linux-gcc-4.4.3/arm-none-linux-gnueabi/sys-r
```

- ② **version.h** 文件的内容如下图所示：

```
#define LINUX_VERSION_CODE  132640
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
~
~
```

如上图所示：`LINUX_VERSION_CODE` 为 `132640`，此值为十进制值，转为为十六进制值为 `20620`，对应的内核版本为 2.6.32；这个值需要改为与 buildroot 中配置的一致，即 4.15.x，转换为十六进制为 `40F00`，对应的十进制为 `265984`，所以 `LINUX_VERSION_CODE` 改为 `265984`：

```
#define LINUX_VERSION_CODE 265984
```

version.h 文件修改完后，重新 make 编译，错误消失。

3.3 测试

(1) 编译完成后，在 buildroot-2020.02.9/output/images 目录下可以看到以下文件：

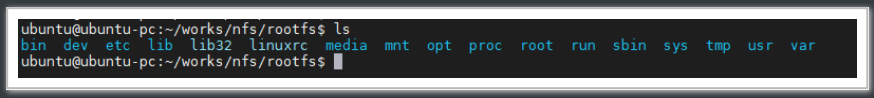
```
ubuntu@ubuntu-pc:~/works/buildroot-2020.02.9/output/images$ ls
rootfs.tar  rootfs.yaffs2
ubuntu@ubuntu-pc:~/works/buildroot-2020.02.9/output/images$
```

其中，rootfs.tar 是打包好的根文件系统，rootfs.yaffs2 是用于烧写到 Fash 里的 yaffs2 格式的根文件系统。我们使用 rootfs.tar 进行测试，测试没问题我们就可以把 rootfs.yaffs2 文件系统烧写到 Flash。

(2) 把 rootfs.tar 拷贝到 nfs 网络文件系统，并解压：

```
cp rootfs.tar ~/works/nfs/rootfs
cd ~/works/nfs/rootfs
tar xf rootfs.tar
rm -rf rootfs.tar
```

解压完成后，目录下的内容如下：



(3) 在 uboot 修改内核启动参数 bootargs，让开发板通过 nfs 挂载根文件系统，可以在 uboot 输入一些命令修改启动参数：

```
setenv bootargs "console=ttySAC0,115200 rw root=/dev/nfs nfsroot=192.1
saveenv
```

当然，本人的是在 设备树 文件通过修改 chosen 节点来修改 bootargs 内核启动参数的，该节点内容如下：

```
/*设置内核启动参数*/
    chosen {
        bootargs = "console=ttySAC0,115200 rw root=/dev/nfs nfsroot=19
    };
```

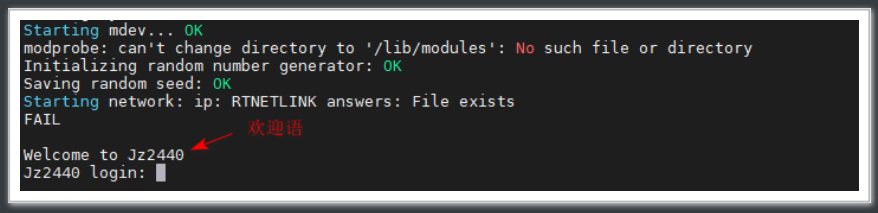
注：关于通过 nfs 挂载根文件系统，在内核 Documentation/filesystems/nfs/nfsroot.txt 有如下说明：

> root=/dev/nfs
> nfsroot=[<server-ip>:]< root-dir >[,< nfs-options >]
> ip=<client-ip>:< server-ip >:< gw-ip >:< netmask >:< hostname >:< device >:< autoconf >

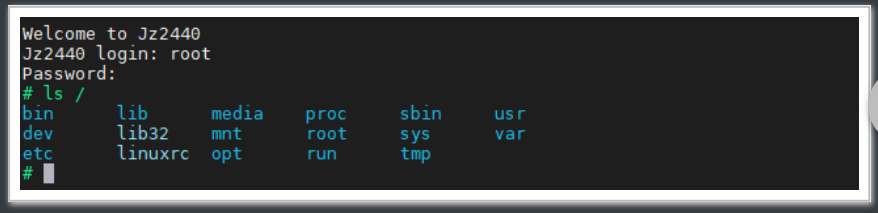
其中：

- [<server-ip>:] 服务器 ip 地址；
- <root-dir> 服务器上哪个目录设置成被单板挂载；
- [<nfs-options>] 用中括号表示的参数可以省略。尖括号的不可省略。
- <client-ip> 表示单板的 ip 地址；
- <server-ip> 服务器 ip 地址；
- <gw-ip> 网关，单板和服务器同一网段；
- <netmask> 子网掩码；
- <hostname> 不关心这个，不要；
- <device> 网卡，如 eth0\eth1；
- <autoconf> 自动配置，这个不需要，写成 off。

(4) 设置好内核启动参数后，重启开发板，进入根文件系统后，如下图所示：



使用 root 用户，输入密码 123456 登录后，进入到文件系统中，如下图所示：



从上图可以看出的 buildroot 构建的根文件系统运行基本没有问题，但是这个根文件系统是最简单的，我们并没有在 buildroot 里面配置任何第三方的库和软件，后续我们需要再继续添加。

附：从上图可以发现，输入命令的时候命令行前面一直都是 “#” ，如果我们进入到某个目录后，前面并不会显示当前目录的路径，这样不利于我们自己查看当前所处的路径。最好能像 Ubuntu 一样，可以指出当前登录的用户名，主机名以及所处的目录，如下图所示：

这需要通过 “PS1” 这个这个环境变量来设置，PS1 用于设置命令提示符格式，格式如下：

```
PS1 = '命令列表'
```

命令列表中可选的参数如下：

```
\! 显示该命令的历史记录编号。
\# 显示当前命令的命令编号。
\$ 显示$符作为提示符，如果用户是 root 的话，则显示#号。
\\ 显示反斜杠。
\d 显示当前日期。
\h 显示主机名。
\n 打印新行。
\nnn 显示 nnn 的八进制值。
\s 显示当前运行的 shell 的名字。
\t 显示当前时间。
\u 显示当前用户的用户名。
\W 显示当前工作目录的名字。
\w 显示当前工作目录的路径
```

我们打开 / etc/profile 文件，找到如下所示内容：

```
if [ "$PS1" ]; then
        if [ "`id -u`" -eq 0 ]; then
                export PS1='# '
        else
                export PS1='$ '
        fi
fi
```

修改为：

```
if [ "$PS1" ]; then
        if [ "`id -u`" -eq 0 ]; then
                export PS1='[\u@\h]:\w# '
        else
                export PS1='[\u@\h]:\w$ '
        fi
fi
```

/etc/profile 文件修改完成以后重启开发板，这个时候我们跳转到某个目录的时候命令行就会有提示，如下图所示：

4. Buildroot 实用技巧与指令

在 buildroot 的顶层目录输入以下命令，可以查看到一些帮忙信息：

```
make help
```

打印出来的信息如下：

```
ubuntu@ubuntu-pc:~/works/buildroot-2020.02.9$
ubuntu@ubuntu-pc:~/works/buildroot-2020.02.9$ make help
Cleaning:
  clean                   - delete all files created by build
  distclean               - delete all non-source files (including .config)

Build:
  all                     - make world
  toolchain               - build toolchain
  sdk                     - build relocatable SDK

Configuration:
  menuconfig              - interactive curses-based configurator
  nconfig                 - interactive ncurses-based configurator
  xconfig                 - interactive Qt-based configurator
  gconfig                 - interactive GTK-based configurator
  oldconfig               - resolve any unresolved symbols in .config
  syncconfig              - Same as oldconfig, but quietly, additionally update deps
  olddefconfig            - Same as syncconfig but sets new symbols to their default value
  randconfig              - New config with random answer to all options
  defconfig               - New config with default answer to all options;
                               BR2_DEFCONFIG, if set on the command line, is used as input
  savedefconfig           - Save current config to BR2_DEFCONFIG (minimal config)
  update-defconfig        - Same as savedefconfig
  allyesconfig            - New config where all options are accepted with yes
  allnoconfig             - New config where all options are answered with no
  alldefconfig            - New config where all options are set to default
  randpackageconfig       - New config with random answer to package options
  allyespackageconfig     - New config where pkg options are accepted with yes
  allnopackageconfig      - New config where package options are answered with no

Package-specific:
  <pkg>                   - Build and install <pkg> and all its dependencies
  <pkg>-source            - Only download the source files for <pkg>
  <pkg>-extract           - Extract <pkg> sources
  <pkg>-patch             - Apply patches to <pkg>
  <pkg>-depends           - Build <pkg>'s dependencies
  <pkg>-configure         - Build <pkg> up to the configure step
  <pkg>-build             - Build <pkg> up to the build step
  <pkg>-show-info         - generate info about <pkg>, as a JSON blurb
  <pkg>-show-depends      - List packages on which <pkg> depends
  <pkg>-show-rdepends     - List packages which have <pkg> as a dependency
  <pkg>-show-recursive-depends
                          - Recursively list packages on which <pkg> depends
  <pkg>-show-recursive-rdepends
                          - Recursively list packages which have <pkg> as a dependency
  <pkg>-graph-depends     - Generate a graph of <pkg>'s dependencies
  <pkg>-graph-rdepends    - Generate a graph of <pkg>'s reverse dependencies
  <pkg>-dirclean          - Remove <pkg> build directory
  <pkg>-reconfigure       - Restart the build from the configure step
  <pkg>-rebuild           - Restart the build from the build step

busybox:
  busybox-menuconfig      - Run BusyBox menuconfig

Documentation:
  manual                  - build manual in all formats
  manual-html             - build manual in HTML
  manual-split-html       - build manual in split HTML
  manual-pdf              - build manual in PDF
  manual-text             - build manual in text
  manual-epub             - build manual in ePub
  graph-build             - generate graphs of the build times
  graph-depends           - generate graph of the dependency tree
  graph-size              - generate stats of the filesystem size
  list-defconfigs         - list all defconfigs (pre-configured minimal systems)

Miscellaneous:
  source                  - download all sources needed for offline-build
  external-deps           - list external packages used
  legal-info              - generate info about license compliance
  show-info               - generate info about packages, as a JSON blurb
  printvars               - dump internal variables selected with VARS=...

  make V=0|1              - 0 => quiet build (default), 1 => verbose build
  make O=dir              - Locate all output files in "dir", including .config

For further details, see README, generate the Buildroot manual, or consult
it on-line at http://buildroot.org/docs.html

ubuntu@ubuntu-pc:~/works/buildroot-2020.02.9$ https://blog.csdn.net/qq_35031421
```

从上图可以看出，使用 make 可以做很多的事情，例如：

命令 / 目标	描述
make <pkg>	单独编译某个 pkg 模块以及其依赖的模块，例如 make demo_app
make <pkg>-source	只下载某 pkg，然后不做任何事情

命令 / 目标	描述
make <pkg>-extract	只下载解压 pkg, 不编译，pkg 解压后放在 output/build / 目录对应的 pkg-dir 目录下
make <pkg>-patch	应用补丁，如果有的话添加补丁
make busybox-menuconfig	进入 busybox 配置选项

剩余的指令，可以那上面截图对应的英文解释。

全文完