编译内核

配置内核：

下载源码包：

```
zrz@Zhao:~/Project/phytium_kernel$ git clone
git@gitee.com:phytium_embedded/phytium-linux-kernel.git
```

自定义内核：

- 拷贝一份原系统中：
  - `/proc`目录下`config.gz`文件，解压得到`config`文件
  - `/usr/src`目录下的`linux-headers-xxx-embeded`文件夹
- 修改配置：
  - 进入源码文件夹：

    ```
    zrz@Zhao:~/Project/phytium_kernel$ cd phytium-linux-kernel/
    ```

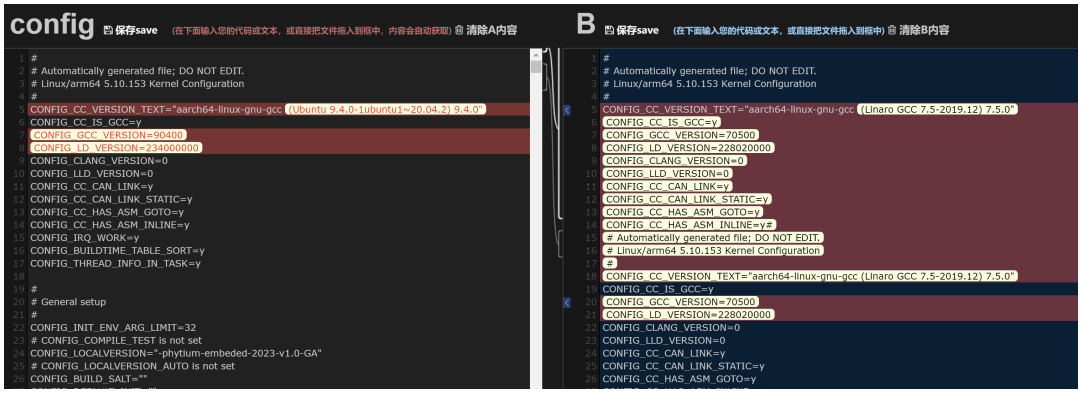
  - 先生成一份萤火工场飞腾派的`defconfig`配置：

    ```
    zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ make
    phytiumpi_firefly_defconfig
    ```

  - 修改生成的`.config`文件(就位于源码文件夹目录下)

    按照原系统中解压得到的`config`文件内容，替换生成的`.config`文件中除**编译环境**外的所有内容：

    如下图：左边是`config`文件内容，右边是位于源码文件夹目录下的`.config`文件内容

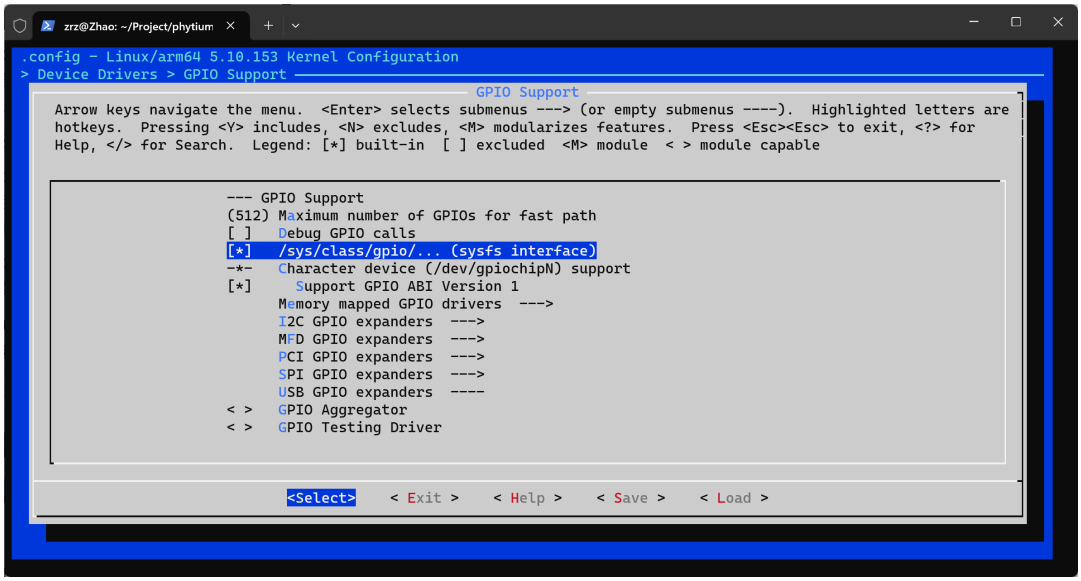
    这些地方的差异不用改变，因为自己的编译是在交叉编译环境中执行的。

  - 在`linux-headers-xxx-embeded`**文件夹**、**源码目录文件夹**执行`make menuconfig`命令：

    按照`linux-headers-xxx-embeded`**文件夹**下的menuconfig逐一配置**源码目录文件夹**下自定义内核的配置

    ```
    zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ make menuconfig
    ```

- 自定义配置中额外配置`Device Drivers`下的`gpio，pwm，i2c，spi`模块内的飞腾开发板支持



1. 替换内核方法

编译：

`-j12`使用12线程加速：

```
zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ make -j12
```

生成ko 的安装目录和文件：

由于内核很多模块编译成ko，所以需要手工生成ko的安装目录和文件：

```
zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ mkdir build
zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ export
INSTALL_MOD_PATH=`pwd`/build
zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ make modules_install
zrz@Zhao:~/Project/phytium_kernel/phytium-linux-kernel$ ls build/lib/modules/
5.10.153-phytium-embeded-2023-v1.0-GA
```

其中，模块名称分2 部分：

- 第一部分“5.10.153-rt76-phytium-embeded”不会变化
- 第二部分“-v1.0-GA”会随着版本的不断更新而持续更新，本章节后续章节描述内核模块以不变部分为准。

替换内核：

- **在编译完成的文件夹中找到如下文件：**
  - `arch/arm64/boot/Image`
  - `arch/arm64/boot/dts/phytium/phytiumpi_firefly.dtb`

- `build/lib/modules/5.10.153-phytium-embeded-2023-v1.0-GA`

- 备份并删除原内核，设备树，模块：

  压缩：`tar -czvf archive_name.tar`

  解压：`tar -xvf archive_name.tar`

```
user@Phytium-Pi:~/Downloads/temps$ sudo cp /boot/Image ./
user@Phytium-Pi:~/Downloads/temps$ sudo cp /boot/phytium-pi-board.dtb ./
user@Phytium-Pi:~/Downloads/temps$ sudo cp /lib/modules/5.10.153-phytium-
embeded-2023-V1.0-GA+ ./ -R

user@Phytium-Pi:~/Downloads/temps$ sudo rm -rf /boot/Image
user@Phytium-Pi:~/Downloads/temps$ sudo rm -rf /boot/phytium-pi-board.dtb
user@Phytium-Pi:~/Downloads/temps$ sudo rm -rf /lib/modules/5.10.153phytium-
embeded-2023-V1.0-GA+
```

- 将内核和设备树安装到SD 卡的/boot 目录(注意提前备份旧的内核和设备树以及内核模块)。
  - 将`arch/arm64/boot/Image`复制到SD卡的`/boot`
  - 将`arch/arm64/boot/dts/phytium/phytiumpi_firefly.dtb`复制到SD卡的`/boot`

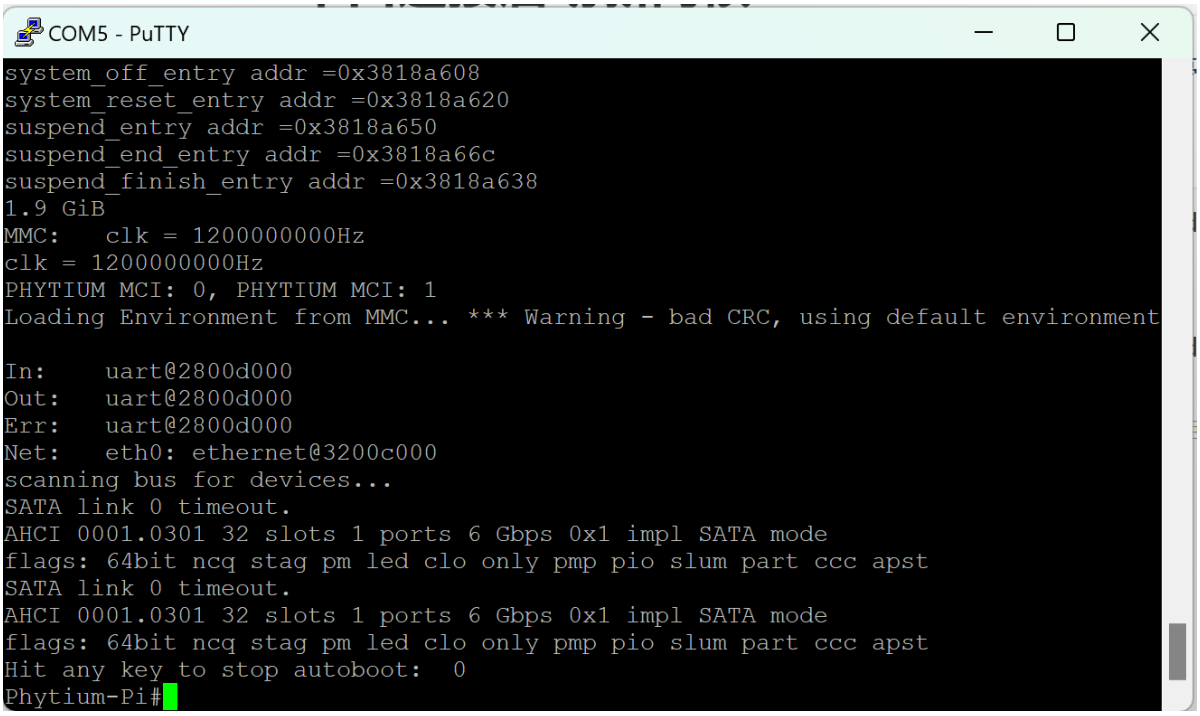
```
user@Phytium-P:-/Downloads/config$ sudo cp ./Image /boot/
user@Phytium-P:-/Downloads/config$ sudo cp ./phytiumpi_firefly.dtb /boot/
```

- 然后将配套的内核模块安装到/lib/modules：
  - 将`build/lib/modules/5.10.153-rt76-phytium-embeded`复制到SD卡的`/lib/modules`
  - 重命名其为`5.10.153-phytium-embeded-2023-V1.0-GA+`

```
user@Phytium-P:-/Downloads/config$ sudo cp ./5.10.153-phytium-embeded-2023-
V1.0-GA /lib/modules -R
user@Phytium-P:-/Downloads/config$ sudo mv ./5.10.153-phytium-embeded-2023-
V1.0-GA ./5.10.153-phytium-embeded-2023-V1.0-GA+
```

串口连接启动新内核

启动开发板，然后在Uboot 启动阶段敲击键盘的回车键，这时系统会停留在Uboot 的Shell 界面，如下所示。



在Uboot 的shell 菜单按照如下步骤引导内核和设备树启动。

- **设置启动参数，其中，嵌入式Linux 文件系统：**

```
E2000# setenv bootargs 'console=ttyAMA1,115200 earlycon=pl011,0x2800d000
root=/dev/mmcblk0p1 rootfstype=ext4 rootwait rw cma=256m;';
```

- **设置启动命令行，其中，嵌入式Linux 文件系统：**
  - 加载内核到内存：`ext4load mmc 0:1 0x90100000 boot/Image;`

    这时，串口会打印如下信息提示加载成功：

    `28692992 bytes read in 6293 ms (4.3 MiB/s)`
  - 加载设备树到内存：`ext4load mmc 0:1 0x90000000 boot/phytiumpi_firefly.dtb;`

    这时，串口会打印如下信息提示加载成功：

    `28692992 bytes read in 6293 ms (4.3 MiB/s)`
  - 引导启动内核：`booti 0x90100000 - 0x90000000;`

  使用如下命令直接修改bootcmd，方便之后的启动(不需要再手敲一遍上述配置)：

```
E2000# setenv bootcmd 'ext4load mmc 0:1 0x90100000 boot/Image; ext4load mmc
0:1 0x90000000 boot/phytiumpi_firefly.dtb; booti 0x90100000 - 0x90000000;';
```

这是修改之前的 `bootenv`：

```
Phytium-Pi#printenv
arch=arm
baudrate=115200
board=e2000
board_name=e2000
boot_os=bootm $kernel_addr -:- $ft_fdt_addr
bootargs=console=ttyAMA1,115200 earlycon=pl011,0x2800d000 root=/dev/mmcblk0p1 rootfstype=ext4 rootwait  rw cma=256m ;
bootcmd=ext4load mmc 0 0x90100000 boot/Image;ext4load mmc 0 0x90000000 boot/phytium-pi-board.dtb; booti 0x90100000 -:- 0x90000000
bootdelay=2
cpu=armv8
distro_bootcmd=run load_kernel; run load_initrd; run load_fdt; run boot_os
eth0addr=00:11:22:33:44:55
eth1addr=10:22:33:44:55:66
eth2addr=10:11:22:33:44:55
eth3addr=00:22:33:44:55:66
ethaddr=00:11:22:33:44:55
fdt_addr=0x381BC000
fdtcontroladdr=f9c3f700
ft_fdt_addr=0x90000000
ft_fdt_name=boot/phytium-pi-board.dtb
gatewayip=202.197.67.1
initrd_addr=0x95000000
ipaddr=202.197.67.2
kernel_addr=0x90100000
load_fdt=ext4load scsi 0:2 $ft_fdt_addr $ft_fdt_name
load_initrd=ext4load scsi 0:2 $initrd_addr initrd.img-4.19.0.e2000
load_kernel=ext4load scsi 0:2 $kernel_addr boot/uImage-2004
loadaddr=0x90000000
netdev=eth0
netmask=255.255.255.0
serverip=202.197.67.3
stderr=uart@2800d000
stdin=uart@2800d000
stdout=uart@2800d000
vendor=phytium

Environment size: 1105/4092 bytes
```

这是修改之后的 `bootenv`：

```
Phytium-Pi#printenv
arch=arm
baudrate=115200
board=e2000
board_name=e2000
boot_os=bootm $kernel_addr -:- $ft_fdt_addr
bootargs=console=ttyAMA1,115200 earlycon=pl011,0x2800d000 root=/dev/mmcblk0p1 rootwait rw;
bootcmd=ext4load mmc 0:1 0x90100000 boot/Image;ext4load mmc 0:1 0x90000000 boot/phytiumpi_firefly.dtb; booti 0x90100000 - 0x90000000
bootdelay=2
cpu=armv8
distro_bootcmd=run load_kernel; run load_initrd; run load_fdt; run boot_os
eth0addr=00:11:22:33:44:55
eth1addr=10:22:33:44:55:66
eth2addr=10:11:22:33:44:55
eth3addr=00:22:33:44:55:66
ethaddr=00:11:22:33:44:55
fdt_addr=0x381BC000
fdtcontroladdr=f9c3f700
ft_fdt_addr=0x90000000
ft_fdt_name=boot/phytium-pi-board.dtb
gatewayip=202.197.67.1
initrd_addr=0x95000000
ipaddr=202.197.67.2
kernel_addr=0x90100000
load_fdt=ext4load scsi 0:2 $ft_fdt_addr $ft_fdt_name
load_initrd=ext4load scsi 0:2 $initrd_addr initrd.img-4.19.0.e2000
load_kernel=ext4load scsi 0:2 $kernel_addr boot/uImage-2004
loadaddr=0x90000000
netdev=eth0
netmask=255.255.255.0
serverip=202.197.67.3
stderr=uart@2800d000
stdin=uart@2800d000
stdout=uart@2800d000
vendor=phytium

Environment size: 1081/4092 bytes
Phytium-Pi#
```

- **保存配置：**

```
E2000# saveenv;
```

- **reset即可。**

2. OS源码内修改内核配置编译

> **通过Phytium开发者给出的 `<PKG>_OVERRIDE_SRCDIR` 机制：**
>
> 更改默认OS的内核配置构建整个系统：内核镜像+设备树+文件系统
>
> 参考：https://gitee.com/phytium_embedded/phytium-pi-os里的readme.md文档README.md

系统要求

Buildroot被设计为在x86 Linux系统上运行，结合其他因素，本仓库只支持在ubuntu20.04、ubuntu22.04、debian11这三种x86系统上进行开发，不支持其他系统。

安装必要依赖

```
• Build tools:
- which
- sed
- make (version 3.81 or any later)
- binutils
- build-essential (only for Debian based systems)
- gcc (version 4.8 or any later)
- g++ (version 4.8 or any later)
- bash
- patch
- gzip
- bzip2
- perl (version 5.8.7 or any later)
- tar
- cpio
- unzip
- rsync
- file (must be in /usr/bin/file)
- device-tree-compiler
- bc
• Source fetching tools:
- wget
- git
```

下载5.10内核和OS编译源码

```
zrz@Zhao:~/Project$ git clone git@gitee.com:phytium_embedded/phytium-linux-kernel.git
zrz@Zhao:~/Project$ git clone git@gitee.com:phytium_embedded/phytium-pi-os.git
```

配置环境变量

由于WSL中PATH包含了windows的PATH，而linux编译时不允许PATH中含有spaces, TABs, and/or newline (\n) characters，所以使用`export`命令手动配置：

> Windows Subsystem for Linux (WSL) 是一种允许在Windows系统上运行Linux发行版的技术。WSL在底层实现中确实会共享某些系统资源，包括环境变量（如PATH）。这可能导致WSL中的Linux发行版的PATH受到Windows系统环境变量的影响。
>
> 主要原因是WSL的设计理念是为了提供与本机系统集成的Linux体验。这使得您可以在WSL中轻松访问Windows文件系统，并且可以在Windows和Linux之间共享一些配置和资源。但正如您提到的，这也可能导致PATH中存在空格等问题，因为Windows系统允许路径中有空格，而Linux则不允许。
>
> 为了解决这个问题，您可以在WSL中适当地调整PATH环境变量，以确保它不包含空格。您可以编辑Linux发行版中的shell配置文件，如`.bashrc`，并将PATH设置为正确的值。在这个文件中，您可以使用`export`命令设置PATH，确保它不包含空格，例如：

```
export
PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/user/bin"
```

这样，您可以在WSL中自定义您的PATH，以适应您的需求，而不受Windows环境变量的影响。当然，确保PATH不包含空格是一种好的做法，以避免潜在的问题。

```
zrz@Zhao:~/Project/phytium-pi-os$ echo $PATH
/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib
```

只截取linux自己的PATH：执行下述命令：

```
zrz@Zhao:~/Project/phytium-pi-os$ export PATH="/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:/opt/toolchains/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib"
```

设置系统区域：

```
zrz@Zhao:~/Project/phytium-pi-os$ sudo apt-get install language-pack-zh-hans
zrz@Zhao:~/Project/phytium-pi-os$ sudo dpkg-reconfigure locales
zrz@Zhao:~/Project/phytium-pi-os$ export LC_ALL="zh_CN.UTF-8"
zrz@Zhao:~/Project/phytium-pi-os$ export LC_CTYPE="zh_CN.UTF-8"
zrz@Zhao:~/Project/phytium-pi-os$ export LC_MESSAGES="zh_CN.UTF-8"
```

config配置

- **编译默认配置的文件系统：**

  加载defconfig：`$ make phytiumpi_xxx_defconfig`
  其中`phytiumpi_xxx_defconfig`为以下文件系统之一：

  ```
  phytiumpi_defconfig
  phytiumpi_desktop_defconfig
  ```

  ```
  zrz@Zhao:~/Project/phytium-pi-os$ make phytiumpi_desktop_defconfig
  ```

- **支持Phytium-optee：**

  本项目还支持编译Phytium-optee，关于Phytium-optee的信息请参考：`https://gitee.com/phytium_embedded/phytium-optee` defconfig默认不编译Phytium-optee，如果需要编译Phytium-optee请执行：
  使用phytiumpi_xxx_defconfig作为基础配置项，合并支持optee的配置：
  `$ ./support/kconfig/merge_config.sh configs/phytiumpi_xxx_defconfig configs/phytiumpi_optee.config`

```
zrz@Zhao:~/Project/phytium-pi-os$ ./support/kconfig/merge_config.sh
configs/phytiumpi_desktop_defconfig configs/phytiumpi_optee.config
```

- **修改内核进行编译：**

> 默认的内核源码目录是`output/build/linux-<version>`，如果在该目录对内核进行了修改（例如修改内核配置或源码）， 当运行`make clean`后该目录会被删除，所以在该目录中直接修改内核是不合适的。
> 因此，对于这种情况提供了一种机制：`<PKG>_OVERRIDE_SRCDIR`机制。
> 操作方法是，创建一个叫做local.mk的文件，其内容是：
>
> ```
> $ cat local.mk
> LINUX_OVERRIDE_SRCDIR = /home/xxx/linux-kernel
> ```
>
> 将local.mk文件和.config文件放在同一目录下，对于树内构建是源码根目录，对于树外构建是树外构建的输出目录。
> LINUX_OVERRIDE_SRCDIR指定了一个本地的内核源码目录，这样就不会去下载、解压、打补丁内核源码了，而是使用LINUX_OVERRIDE_SRCDIR 指定的内核源码目录。
> 这样开发人员首先在LINUX_OVERRIDE_SRCDIR指定的目录对内核进行修改，然后运行`make linux-rebuild`或者 `make linux-reconfigure`即可。 该命令首先将LINUX_OVERRIDE_SRCDIR中的内核源码同步到`output/build/linux-custom`目录，然后进行**配置、编译、安装**。
> 如果想要编译、安装内核，并重新生成系统镜像，请运行`make linux-rebuild all`。

注意： `make linux-rebuild`或者 `make linux-reconfigure`执行时不会复制自定义内核下之前的配置文件，只复制源码！

上述机制说明指定内核目录后，`make linux-rebuild`或者 `make linux-reconfigure`命令会将自定义内核目录下的源码(不包含内核目录下的配置.config)复制到：`phytium-pi-os/output/build/linux-custom`目录下，因此我们需要在`phytium-pi-os/output/build/linux-custom`目录下对内核的配置进行修改：

- 创建`local.mk`文件：

  ```
  zrz@Zhao:~/Project/phytium-pi-os$ sudo vim local.mk
  ```

  修改路径为自定义内核所在目录：`LINUX_OVERRIDE_SRCDIR = /home/zrz/Project/phytium-linux-kernel`

- 先执行`make linux-rebuild`或`make linux-reconfigure`，然后
  - 在`phytium-pi-os/output/build/linux-custom`目录下配置内核：
  - 或者直接在`phytium-pi-os/`目录下使用`make linux-menuconfig`命令配置内核：

```
zrz@Zhao:~/Project/phytium-pi-os$ make linux-reconfigure
//zrz@Zhao:~/Project/phytium-pi-os$ cd output/build/linux-custom/
//zrz@Zhao:~/Project/phytium-pi-os/output/build/linux-custom$ make menuconfig
zrz@Zhao:~/Project/phytium-pi-os$ make linux-menuconfig
```

注意：在`menuconfig`中配置`Device Drivers`下的`gpio，pwm，i2c，spi`模块内的飞腾开发板支持

- 重新安装：

```
zrz@Zhao:~/Project/phytium-pi-os$ make linux-rebuild all
```

执行后提示增加编译一些模块，默认选y/1/m即可。

3. 问题汇总：

1. 编译生成的镜像文件烧录并运行在别人的系统上时出现问题：无法提取root权限：

```
user@phvtiumpi:~s sudo su
sudo: /usr/bin/sudo must be owned by uid 0 and have the setuid bit set
user@phytiumpi:~s su
Password:
su: Authentication failure
user@phytiumpi:~S
```

解决办法：

`exit`退出当前user用户，使用root用户登录(密码为：root)修改user用户权限：

```
root@phytiumpi:~# chmod 775 /usr
root@phytiumpi:~# chmod 4755 /usr/bin/sudo
```

之后即可在user用户中提取root权限。