第一次实验

181250090 刘育麟

使用的环境是Ubuntu18.04，阿里云的服务器，操作环境是使用termius对阿里云服务器进行ssh连接。

rename指令需要apt install rename才能使用。

第1题

用命令行打印HOME、PATH、SHLVL、LOGNAME变量的值。

show.sh

```
#!/bin/sh
#显示HOME变量
echo $HOME
#显示PATH变量
echo $PATH
#显示SHLVL变量
echo $SHLVL
#显示LOGNAME变量
echo $LOGNAME
```

```
root@iZuf6gkr8qr5rsx08reaomZ:~/test# ./show.sh
/root
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/local/maven3/bin
1
root
root@iZuf6gkr8qr5rsx08reaomZ:~/test#
```

第2题

请用中文解释Shell脚本程序，并说明运行结果：

```
#!/bin/sh
clear
select item in Continue Finish
do
case "$item" in
    Continue) ;;
    Finish) break ;;
    *) echo "Wrong choice! Please select again!" ;;
esac
done
```

```
#!/bin/sh
# #!/bin/sh是指此脚本使用/bin/sh来解释执行，#!是特殊的表示符，其后面根的是此解释此脚本的shell的路径。

clear
# 进行清屏

select item in Continue Finish
# select 表达式是一种bash的扩展应用，动作包括：
```

```
# (1)、自动用1,2,3,4列出菜单 （没有echo指令，自动显示菜单）
# (2)、自动read输入选择 （没有 read指令，自动输入）
# (3)、赋值给变量 （没有赋值指令，自动输入数字后，赋值字符串给变量）

do
case "$item" in
    Continue) ;;
    Finish) break ;;
    *) echo "Wrong choice! Please select again!" ;;
esac
# switch/case语句，输入1则继续，输入2则跳出循环，其他指令是输出echo的语句

done
# 一个循环

```

```
1) Continue
2) Finish
#? 3
Wrong choice! Please select again!
#? 1
#? 1
#? 1
#? s
Wrong choice! Please select again!
#? 2
root@iZuf6gkr8qr5rsx08reaomZ:~/test#
```

第3题

阅读下列Makefile 并用中文说明其含义。

```
=========Makefile1=========
export Top:=${shell pwd}
export Src:=$(Top)/src/
export Include:=$(Top)/include/
export Build:=$(Top)/build/
all:
    @$(MAKE) -C $(Src)
install:
    @cp $(Build)/test $(Top)
clean:
    @-rm -rf $(Build) $(Top)/test
========= Makefile2=========
all:main.o test4.o
    @mkdir -p $(Build)
    @mv *.o $(Build)
    $(MAKE) -C $(Src)/dir1
    $(MAKE) -C $(Src)/dir2
    $(CC) -o $(Build)/test $(Build)/*.o $(Build)/dir1/*.o $(Build)/dir2/*.o
main.o : $(Include)/func.h
    $(CC) -c main.c -I$(Include)
```

```
=========Makefile1=========
export Top:=${shell pwd}
export Src:=$(Top)/src/
export Include:=$(Top)/include/
export Build:=$(Top)/build/
```

```
# 设置变量。方便向后来发起的make进程（这些后起的make进程由当前的make进程启动）传送变量。
# ${shell pwd}：用pwd命令来获取当前所执行命令的目录，在makefile中要在pwd前面加shell，然后把shell pwd当一个变量来引用

all:
    @$(MAKE) -C $(Src)
# @：通常makefile会将其执行的命令行在执行前输出到屏幕上。如果将‘@’添加到命令行前，这个命令将不被make回显出来。
# $(make)：预设的 make 这个命令的名称（或者路径），这个指令就是执行指定位置的makefile。
# -C $(Src)：指定makefile的位置

install:
    @cp $(Build)/test $(Top)
# 复制$(Build)/test文件到$(Top)目录下

clean:
    @-rm -rf $(Build) $(Top)/test
# 删除$(Build)和$(Top)/test底下所有文件及目录
========= Makefile2=========
all:main.o test4.o
    @mkdir -p $(Build)
    @mv *.o $(Build)
    $(MAKE) -C $(Src)/dir1
    $(MAKE) -C $(Src)/dir2
    $(CC) -o $(Build)/test $(Build)/*.o $(Build)/dir1/*.o $(Build)/dir2/*.o
# main.o test4.o：执行这个target需要的参数prerequisites
# 下面是执行的几个步骤
# 1. mkdir：递归创建$(Build)目录
# 2. mv：将prerequisites移动到创建的$(Build)目录下
# 3. 对$(Src)/dir1和$(Src)/dir2执行make指令，也就是执行目录下的makefile
# 4. 环境变量中的编译器，默认是gcc。将后面所有文件进行静态链接，并将输出生成为$(Build)/test可执行文件

main.o : $(Include)/func.h
    $(CC) -c main.c -I$(Include)
# 因为前面需要main.o这个prerequisites，这个会早于all执行。
# 将main.c进行编译但是不链接。并且指定头文件路径$(Include)
```

makefile1是执行某个项目的源文件底下makefile（这里是makefile2），在执行结束后将编译和链接后的可执行文件test放到当前目录下，并且将编译时的.o文件进行删除。

makefile2是对某个项目进行编译与链接，将main.c文件编译成.o文件后并执行dir1和dir2目录下的文件makefile，最后将所有.o文件静态链接出一个可执行文件test放到build目录。