

# 实验四.shell 编程

实 验 名 称	实验四.shell 编程
班 级	软工 182
学 号	1813031054
姓 名	陈建
日 期	2020.6.14

## 一、实验目的

1. 了解 shell 的特点和主要种类。
2. 掌握 shell 脚本的建立和执行方式。
3. 掌握 bash 的基本语法。
4. 学会编写 shell 脚本。

## 二、实验内容

1. shell 脚本的建立和执行。
2. 历史命令和别名定义。
3. shell 变量和位置参数、环境变量。
4. bash 的特殊字符。
5. 一般控制结构。
6. 算数运算及 bash 函数。

## 三、主要实验步骤

1. 利用 vi 建立一个脚本文件，其中包括 date、cal、pwd、ls 等常用命令；然后以不同方式执行该脚本。
2. 运行 history 命令，配置历史命令环境。
3. 体会 bash 的命令补齐功能。
4. 用 alias 定义别名，然后执行。
5. 对习题 4.8 中的 shell 脚本进行编辑，然后执行。
6. 按习题 4.14 要求编写脚本，然后执行。
7. 按习题 4.18 要求编写脚本，然后执行。
8. 运行例 4.20 的程序。若取消其中的 "eval"，则会出现什么情况。

## 四、具体实现

1. 分析：在脚本文件中输入所有命令，控制格式，执行文件时就会全部输出。  
关于执行方式，我所会的方式大概有四种：(1) sh (2) ./file (3) bash<file (4) source。

代码(date\_cal\_pwd\_ls.sh):

```
#!/bin/bash
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin //设置环境变量。  
//此题可不需要
```

```
export PATH
```

```
echo -e "following show command:\n"
```

```
echo -e "date:"
```

```
date
```

```
echo -e "\ncal:"
```

```
cal
echo -e "\npwd:"
pwd
echo -e "\nls:"
ls
echo -e "\n"
```

截图：

(1) `sh date_cal_pwd_ls.sh`：最常用、直接执行的方式，脚本创建子进程，在子进程 `bash` 中执行。

```
[gua@localhost mysh]$ sh date_cal_pwd_ls.sh
following show command:

date:
Sun Jun 14 10:45:10 EDT 2020

cal:
      June 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

pwd:
/home/gua/mysh

ls:
~  3      bingfa.c  date_cal_pwd_ls.sh  fibo_10.sh  fp.c      pcp  PI.sh  SHM.c
0  4      cpu      ex20             fk          haha.sh    pcp.c  shm   show123.sh
1  9      cpu.c    ff              fk.c        hello.sh   pcv    SHM   showname.sh
2  bingfa  cuts.sh  ff.c           fp          multiplying.sh pcv.c  shm.c  test_48.sh
```

(2) `./date_cal_pwd_ls.sh`：脚本文件必须可执行（默认 `rw-`），用 `"chmod u+x date_cal_pwd_ls.sh"` 命令实现，之后即可用此方式执行脚本。

```
-rwxr--r-- 1 gua gua 242 Jun 14 10:36 date_cal_pwd_ls.sh
```

```
[gua@localhost mysh]$ ./date_cal_pwd_ls.sh
following show command:

date:
Sun Jun 14 10:45:30 EDT 2020

cal:
      June 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

pwd:
/home/gua/mysh

ls:
~  3      bingfa.c  date_cal_pwd_ls.sh  fibo_10.sh  fp.c      pcp  PI.sh  SHM.c
0  4      cpu      ex20             fk          haha.sh    pcp.c  shm   show123.sh
1  9      cpu.c    ff              fk.c        hello.sh   pcv    SHM   showname.sh
2  bingfa  cuts.sh  ff.c           fp          multiplying.sh pcv.c  shm.c  test_48.sh
```

(3) `bash<date_cal_pwd_ls.sh`：数据重定向的标准输入方式执行脚本文件：

```
[gua@localhost mysh]$ bash<date_cal_pwd_ls.sh
following show command:

date:
Sun Jun 14 10:45:49 EDT 2020

cal:
      June 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

pwd:
/home/gua/mysh

ls:
~ 3      bingfa.c  date_cal_pwd_ls.sh  fibo_10.sh  fp.c      pcp  PI.sh  SHM.c
0 4      cpu      ex20             fk          haha.sh    pcp.c  shm   show123.sh
1 9      cpu.c    ff              fk.c       hello.sh   pcv    SHM   showname.sh
2 bingfa  cuts.sh  ff.c          fp         multiplying.sh  pcv.c  shm.c  test_48.sh
```

(4) source date\_cal\_pwd\_ls.sh: 在父进程中执行脚本，和直接执行区分，主要是能量表的问题，在此题中没有区别。

```
[gua@localhost mysh]$ source date_cal_pwd_ls.sh
following show command:

date:
Sun Jun 14 10:48:18 EDT 2020

cal:
      June 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

pwd:
/home/gua/mysh

ls:
~ 3      bingfa.c  date_cal_pwd_ls.sh  fibo_10.sh  fp.c      pcp  PI.sh  SHM.c
0 4      cpu      ex20             fk          haha.sh    pcp.c  shm   show123.sh
1 9      cpu.c    ff              fk.c       hello.sh   pcv    SHM   showname.sh
2 bingfa  cuts.sh  ff.c          fp         multiplying.sh  pcv.c  shm.c  test_48.sh
```

- 分析: histroy 命令用来查看历史指令，可以用”cat ~/.bash\_history”命令查看历史文件。  
history 命令一次显示 1000 条，可在/etc/profile 中修改上限。  
配置历史环境命令: /etc/profile 文件中的内容与环境变量有关，可添加/修改关于历史指令的内容，比如显示记录时间。  
但是一般想用历史命令最方便的就是”↑”，使用极其频繁。

截图:

若 history 命令后加参数，指定显示的命令数目。

```
[gua@localhost mysh]$ history 10
1035 cat showname.sh
1036 gcc pcp.c
1037 gcc pcp.c -o pcp
1038 ./pcp
1039 ll
1040 ./pcv
1041 c
1042 history
1043 c
1044 history 10
```

修改/etc/profile 文件需要 root 权限，切换后在文件末尾添加如下环境变量，最后”source /etc/profile”指令使其生效。

```
export HISTTIMEFORMAT="%y-%m-%d %H:%M:%S "
```

可以看到配置历史环境命令后 **history** 命令同时显示了记录时间。

```
File Edit View Search Terminal Help
[gua@localhost mysh]$ history 10
1043 20-06-14 11:10:14 c
1044 20-06-14 11:10:17 history 10
1045 20-06-14 11:14:20 su - root
1046 20-06-14 11:15:59 c
1047 20-06-14 11:16:03 history
1048 20-06-14 11:16:26 source /etc/profile
1049 20-06-14 11:16:28 c
1050 20-06-14 11:16:31 history
1051 20-06-14 11:17:34 c
1052 20-06-14 11:17:38 history 10
[gua@localhost mysh]$
```

3. 分析：Tab 键的命令补全也是很实用、常用的功能，不仅可以补全命令甚至可以补全文件名。双击 Tab 列出所有可能选项。

截图：

输入”ali”+Tab，即可补全为”alias”；

目录下含有 **pcp.c**、**pcv.c** 和 **pcv** 三个文件，输入”vim pc”+Tab 即可显示所有相关文件：

```
[gua@localhost mysh]$ vim pc
pcp.c  pcv  pcv.c
```

4. 分析：**alias** 是很重要的命令，很多常用的命令如”ll”、”vi”其实都是别名。我最常用的是”alias c=clear”，这样只要一个按键即可清空屏幕，很方便。

截图：

**alias** 命令直接可看系统的所有别名，可见 **vi** 命令其实都是 **vim**，”ll”就是”l-ls”的缩写。

```
[gua@localhost mysh]$ alias
alias c='clear'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[gua@localhost mysh]$
```

添加别名 **h=history**，**c=clear**。

再次 **alias** 查看时可见新的别名已经添加。

```
[gua@localhost mysh]$ alias h=history c=clear
[gua@localhost mysh]$ alias
alias c='clear'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias h='history'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[gua@localhost mysh]$
```

此时输入 **h** 即可查看历史命令，**c** 键一键清屏。

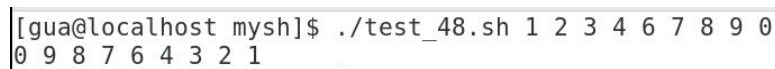
5. 分析: `count=$#`获取脚本参数个数; `cmd=echo` 将 `cmd` 赋值为 `echo`; `while [ $count -gt 0 ]` 循环条件为 `count>0`; 循环内: `cmd` 重新赋值为 `echo ${count 的值}`, 同时 `count-1`; 循环后 `eval` 命令扫描两次 `cmd`, 执行 `cmd` 命令。  
最终实现参数逆序输出。

代码(test\_48.sh):

```
count=$#
cmd=echo
while [ $count -gt 0 ]
do
    cmd="$cmd \${count}"
    count=`expr $count - 1`
done
eval $cmd
```

截图:

可见确实是逆序输出所有参数。



```
[gua@localhost mysh]$ ./test_48.sh 1 2 3 4 6 7 8 9 0
0 9 8 7 6 4 3 2 1
```

6. 分析: 斐波那契数列, 经典的编程题目。最初学 C++ 时利用数组完成, 而 shell 也可利用数组完成此题。从第三个元素起为前两元素和。

代码(fibo\_10.sh):

```
#!/bin/sh
```

```
f[0]=1;
f[1]=1;
sum=$((f[0]+f[1]));
echo -e "${f[0]}\n${f[1]}"
```

```
for((i=2;i<10;++i))
do
    f[i]=$((f[i-1]+f[i-2]))
    sum=$((sum+f[i]))
    echo -e ${f[i]}
done
echo -e ${sum}
```

截图:

```
gua@promote:~/mysh
File Edit View Search Terminal Help
[gua@promote mysh]$ sh fibo_10.sh
1
1
2
3
5
8
13
21
34
55
143
```

7. 分析: 很简单的脚本。`read` 命令读取从键盘输入为变量, 可一次读多个, 符合题意。`-p` 是提示语。输入的的第一个变量是整个字符串, 对字符串行处理用 `cut` 命令最方便, `-b` 参数意味着以字节区分, 这个参数不可少, 因为默认是 `Tab`, 无法按要求区分。要求截取第一个参数到第二个参数之间。最后用管道连接 `echo` 和 `cut` 语句。

代码(cuts.sh):

```
/bin/bash
```

```
read -p "input a string, beginning and ending:" s a b
echo $s | cut -b $a-$b
```

截图:

```
[gua@localhost mysh]$ sh cuts.sh
input a string, beginning and ending:12345678 2 7
234567
```

8. 分析: 脚本实现输入一个字符, 系统读取并显示输入的字符, 然后按键退出的过程。`eval` 命令利用其它命令作为自己的参数进行变量或命令替换并将结果组成新的命令行并执行。`eval` 扫描命令 2 次, 最终读取执行。

代码(ex20):

```
#!/bin/bash
```

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

getc()

```
{
    stty raw
    tmp=`dd bs=1 count=1 2>/dev/null`
    eval $1=$tmp
}
```

```
        stty cooked
    }

    press_any_key()
    {
        echo -n "Strike any key to continue ..."
        getc anychar
    }
}
```

```
echo -n "Enter a character:"
getc char
echo
echo "You entered $char "
press_any_key char
echo
```

截图：  
正确运行。

---

```
[gua@localhost mysh]$ sh ex20
Enter a character:A
You entered A
Strike any key to continue ...^M
```

去掉 eval，运行失败。

```
[gua@localhost mysh]$ sh ex20
Enter a character:Aex20: line 10: char=$tmp: command not found

You entered
Strike any key to continue ...
```