系统开发工具基础第二次实验报告

张烨 23020007162

2024 年 9 月 11 日

目录

一、实验目的

二、实验内容

（一）Shell

1.脚本格式和执行命令

脚本以#!/bin/bash开头，下面是一个例子：

```
#!/bin/bash

echo "Hello Shell"
~
```

图 1: shell 脚本

shell 脚本执行命令：./test.sh或sh test.sh，但是两者存在差异。

./test.sh：这种方式通过直接调用脚本文件来执行。要使用这种方式，脚本文件必须具有执行权限。使用 chmod 命令赋予脚本执行权限。

sh test.sh：这种方式通过指定的 shell 解释器来执行脚本。脚本文件不需要有执行权限，只要脚本文件可读即可。

下面是一个例子：

```
[root@localhost ~]# vim helloshell.sh
[root@localhost ~]# ./helloshell.sh
bash: ./helloshell.sh: 权限不够
[root@localhost ~]# chmod 777 helloshell.sh
[root@localhost ~]# ./helloshell.sh
Hello Shell
[root@localhost ~]# sh helloshell.sh
Hello Shell
[root@localhost ~]#
```

图 2: 脚本执行

2.shell 中的变量——自定义变量、静态变量、全局变量

变量定义规则：

（1）变量名称可以由字母，数字和下划线组成，不能以数字开头，环境变量名建议大写

（2）等号两侧不能有空格

（3）在 bash 中，变量默认类型都是字符串类型，无法直接进行数值运算

（4）变量的值如果有空格，需要使用双引号或单引号括起来

1. 自定义变量

格式：变量 = 值等号两边不能留有空格

撤销变量：unset 变量

输出变量：echo $变量

```
[root@localhost ~]# A=1
[root@localhost ~]# echo $A
1
[root@localhost ~]# unset A
```

图 3: 变量

2. 静态变量

格式：readonly 变量

注意：静态变量不能撤销

```
[root@localhost ~]# readonly B=3
[root@localhost ~]# echo $B
3
[root@localhost ~]# unset B
bash: unset: B: 无法反设定: 只读 variable
```

图 4: 静态变量

3. 全局变量

格式：export 变量

全局变量可以在其他 shell 程序使用

```
#!/bin/bash
echo $LZJ
```

图 5: 全局变量

```
[root@localhost ~]# vim test1.sh
[root@localhost ~]# export LZJ=6666
[root@localhost ~]# sh test1.sh
6666
```

图 6: 全局变量

3.特殊变量

1.$n

n 为数字，$0代表脚本名称，10 以内参数用$1-9 表示，10 以上的需要用大括号包含

```
#!/bin/bash

echo "Shell 传递参数实例！";
echo "执行的文件名：$0";
echo "第一个参数为：$1";
echo "第二个参数为：$2";
echo "第三个参数为：$3";
```

图 7: 1

```
[root@localhost ~]# vim test3.sh
[root@localhost ~]# chmod +x test3.sh
[root@localhost ~]# ./test3.sh 1 2 3
Shell 传递参数实例！
执行的文件名：./test3.sh
第一个参数为：1
第二个参数为：2
第三个参数为：3
[root@localhost ~]#
```

图 8: 1

2.$#

$#：用于获取所有输入参数个数，常用于循环。

```
#!/bin/bash
echo "$1 $2 $3"
echo $#
```

图 9: 2

```
[root@localhost ~]# vim test4.sh
[root@localhost ~]# sh test4.sh 1 2 3
1 2 3
3
[root@localhost ~]#
```

图 10: 2

3.$* $@ $?

$*：代表命令行中所有的参数，把所有参数看成一个整体
$@：代表命令行中所有的参数，不过把每个参数区分对待
$?：最后一次执行状态命令的状态 0 表示正确执行

```
#!/bin/bash

echo "参数0："$0

echo "输出参数1"

echo "参数1：$1

echo "输出参数2"

echo "参数2："$2

echo $#

echo $*

echo $@

echo $?
~
```

图 11: 3

```
[root@localhost ~]# vim test5.sh
[root@localhost ~]# sh test5.sh 1 2
参数0：test5.sh
输出参数1
参数1：1
输出参数2
参数2：2
2
1 2
1 2
0
[root@localhost ~]#
```

图 12: 3

4.运算符

基本语法：

1.expr +、-、*、/、% 运算符之间要有空格

2.$（（运算式））或$[运算式]

```
[root@localhost ~]# expr 3 + 2
5
[root@localhost ~]# expr 4 - 2
2
[root@localhost ~]# s=$[(2+3)*4]
[root@localhost ~]# echo $s
20
[root@localhost ~]#
```

图 13: 运算符

5.条件判断

两个整数之间比较：

表 1: 整数

符号	描述
-lt	小于
-le	小于等于
-eq	等于
-gt	大于
-ge	大于等于
-ne	不等

文件权限判断：

表 2: 文件

符号	描述
-r	有读的权限
-w	有写的权限
-x	有执行的权限

文件类型判断：

表 3: 文件类型

符号	描述
-f	文件存在并且是一个常规文件
-e	文件存在
-d	文件存在并且是一个目录

```
[root@localhost ~]# [ 23 -le 24 ]
[root@localhost ~]# #?
[root@localhost ~]#
[root@localhost ~]# [ 23 -le 24 ]
[root@localhost ~]# echo $?
0
[root@localhost ~]# [ 23 -ge 22 ]
[root@localhost ~]# echo $?
0
[root@localhost ~]# [ 23 -eq 24 ]
[root@localhost ~]# echo $?
1
[root@localhost ~]# [ -w 111.txt ]
[root@localhost ~]# echo $?
1
```

图 14: 比较

多条件判断：

&&表示前一条命令执行成功时，才执行后一条命令

|| 表示上一条命令执行失败后，才执行下一条命令

```
[root@localhost ~]# [ 23 -eq 24 ] && echo ok || echo notok
notok
[root@localhost ~]#
```

图 15: 多条件判断

6.if 判断

1. 基本语法

if [条件判断式];then

　　程序

fi

或者

if [条件判断式]

then

程序

fi

多条件判断

if [条件判断式]

then

程序

elif [条件判断式]

then

程序

fi

2. 注意事项：

（1）中括号和条件判断式之间必须有空格

（2）if 后要有空格

下面是一个例子：

```
#! bin/bash

if [ $1 -eq 1 ]
 then
    echo "xuezhiqian"
elif [ $1 -eq 2 ]
 then
        echo "tanjianci"
fi
```

图 16: if 判断

```
[root@localhost ~]# vim test6.sh
[root@localhost ~]# sh test6.sh 1
xuezhiqian
[root@localhost ~]# sh test6.sh 2
tanjianci
[root@localhost ~]#
```

图 17: if 判断

7.case 语句

1. 基本语法

case $变量名 in

” 值 1”)
程序 1
;;
” 值 2”)
程序 2
;;
*)
变量都不是以上值，则执行此程序
;;
esac

2. 注意事项

（1）case 行尾必须为单词“in”，每个模式匹配必须以“)”结束

（2）双分号“;;”表示命令序列结束，相当于 break

（3）最后的“*)”表示默认模式，相当于 default

下面是一个例子：

```
#!/bin/bash
case $1 in
1)
        echo "xuezhiqian"
        ;;
2)
        echo "tanjianci"
        ;;
*)
        echo "zyzyzy"
        ;;
esac
```

图 18: case 语句

```
[root@localhost ~]# vim test7.sh
[root@localhost ~]# sh test7.sh
zyzyzy
[root@localhost ~]# sh test7.sh 1
xuezhiqian
[root@localhost ~]# sh test7.sh 2
tanjianci
[root@localhost ~]#
```

图 19: case 语句

8.for 循环

基本语法

for((初始值; 循环控制; 条件变量变化))

do

程序

done

或

for 变量 in 值 1 值 2 值 3

do

程序

done

下面是一个例子：

```
#!/bin/bash
s=0
for i in $*
do
        echo "zyzyzy $i"
done
~
```

图 20: for 循环

```
[root@localhost ~]# vim test8.sh
[root@localhost ~]# sh test8.sh 1 2 3
zyzyzy 1
zyzyzy 2
zyzyzy 3
[root@localhost ~]#
```

图 21: for 循环

9.while 循环

基本语法

while [条件判断式]

do

程序

done

下面是一个例子：

```
#!/bin/bash
s=0
i=1
while [ $i -le 100 ]
do
        s=$[$s+$i]
        i=$[$i+1]
done

echo $s
~
```

图 22: while 循环

```
[root@localhost ~]# vim test9.sh
[root@localhost ~]# sh test9.sh
5050
[root@localhost ~]#
```

图 23: while 循环

10. 函数

基本语法

[function] funname[()]

Action;

[return int;]

funname

下面是一个例子：

```
#!/bin/bash
function sum()
{
        s=0;
        s=$[$1+$2]
        echo $s
}

read -p "输入第一个参数：" P1
read -p "输入第二个参数：" P2

sum $P1 $P2
~
~
```

图 24: 函数

```
[root@localhost ~]# vim test10.sh
[root@localhost ~]# sh test10.sh
输入第一个参数：1
输入第二个参数：2
3
[root@localhost ~]#
```

图 25: 函数

11. 不同的 ls 命令

1. 阅读 `man ls`，然后使用 `ls` 命令进行如下操作：

   - 所有文件（包括隐藏文件）
   - 文件打印以人类可以理解的格式输出 (例如，使用 454M 而不是 454279954)
   - 文件以最近访问顺序排序
   - 以彩色文本显示输出结果

   典型输出如下：

```
-rw-r--r--   1 user group 1.1M Jan 14 09:53 baz
drwxr-xr-x   5 user group  160 Jan 14 09:53 .
-rw-r--r--   1 user group  514 Jan 14 06:42 bar
-rw-r--r--   1 user group 106M Jan 13 12:12 foo
drwx------+ 47 user group 1.5K Jan 12 18:08 ..
```

图 26: 问题

```
[root@localhost ~]# man ls
[root@localhost ~]# ls -a
.                .cshrc          helloworld.sh          .tcshrc     test8.sh  文档
..               .dbus           httpd_2.4_install.sh   test10.sh   test9.sh  下载
anaconda-ks.cfg  .esd_auth       .ICEauthority          test1.sh    tets3.sh  音乐
.bash_history    game.sh         initial-setup-ks.cfg   test2.sh    .vim      桌面
.bash_logout     hello2.txt      last-modified.txt      test3.sh    .viminfo
.bash_profile    hello.sh        .local                 test4.sh    公共
.bashrc          helloshell.sh   marco_history.log      test5.sh    模板
.cache           hello.txt       .mozilla               test6.sh    视频
.config          Helloword.sh    .pki                   test7.sh    图片
[root@localhost ~]# ls -h
anaconda-ks.cfg  hello.txt             last-modified.txt  test3.sh  test8.sh  视频  桌面
game.sh          Helloword.sh          marco_history.log  test4.sh  test9.sh  图片
hello2.txt       helloworld.sh         test10.sh          test5.sh  tets3.sh  文档
hello.sh         httpd_2.4_install.sh  test1.sh           test6.sh  公共      下载
helloshell.sh    initial-setup-ks.cfg  test2.sh           test7.sh  模板      音乐
[root@localhost ~]# ls -t
test10.sh  test4.sh       hello.sh              game.sh  下载
图片       test3.sh       last-modified.txt     桌面     initial-setup-ks.cfg
test9.sh   tets3.sh       marco_history.log     视频     anaconda-ks.cfg
test8.sh   test2.sh       httpd_2.4_install.sh  音乐
test7.sh   test1.sh       Helloword.sh          公共
test6.sh   helloshell.sh  hello2.txt            模板
test5.sh   helloworld.sh  hello.txt             文档
[root@localhost ~]# ls --color=auto
anaconda-ks.cfg  hello.txt             last-modified.txt  test3.sh  test8.sh  视频  桌面
game.sh          Helloword.sh          marco_history.log  test4.sh  test9.sh  图片
hello2.txt       helloworld.sh         test10.sh          test5.sh  tets3.sh  文档
hello.sh         httpd_2.4_install.sh  test1.sh           test6.sh  公共      下载
helloshell.sh    initial-setup-ks.cfg  test2.sh           test7.sh  模板      音乐
[root@localhost ~]#
```

图 27: 解答

12. 编写 marco 和 polo 函数

2. 编写两个 bash 函数 `marco` 和 `polo` 执行下面的操作。每当你执行 `marco` 时，当前的工作目录应当以某种形式保存，当执行 `polo` 时，无论现在处在什么目录下，都应当 `cd` 回到当时执行 `marco` 的目录。为了方便 debug，你可以把代码写在单独的文件 `marco.sh` 中，并通过 `source marco.sh` 命令，（重新）加载函数。

图 28: 问题

```
#!/bin/bash
marco(){
    echo "$(pwd)" > $HOME/marco_history.log
    echo "save pwd $(pwd)"
}
polo(){
    cd "$(cat "$HOME/marco_history.log")"
}
```

图 29: 函数

```
[root@localhost ~]# mkdir missing
[root@localhost ~]# cd missing
[root@localhost missing]# vim marco.sh
[root@localhost missing]# source marco.sh
[root@localhost missing]# marco
save pwd /root/missing
[root@localhost missing]# cd
[root@localhost ~]# polo
[root@localhost missing]#
```

图 30: 执行

为了让结果更清晰，新建了 missing 目录，可以看到，当执行 polo 函数时，回到了当时执行 marco 的目录。

（二）Vim

1.三种模式

1. 命令模式：不能对文件直接编辑，只能通过快捷键进行一些操作 (如移动光标、复制、粘贴等)，打开 vim 后默认进入命令模式

2. 末行模式：可在末行输入一些命令对文件进行操作 (如搜索、替换、保存、退出、高亮等)

3. 编辑模式：可对文件内容进行编辑

2.vim 的进入和退出

1. 进入

（1）vim 文件路径：直接打开指定文件

（2）vim + 数字文件路径：打开指定文件并将光标移动到指定行

```
[root@localhost ~]# vim +4 test3.sh
```

图 31: 指定第四行

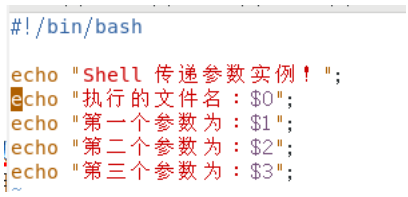

图 32: 查看

可以看到，打开文件后光标在第四行

（3）vim +/关键词文件路径：打开指定文件并高亮显示关键词

图 33: 关键词 echo

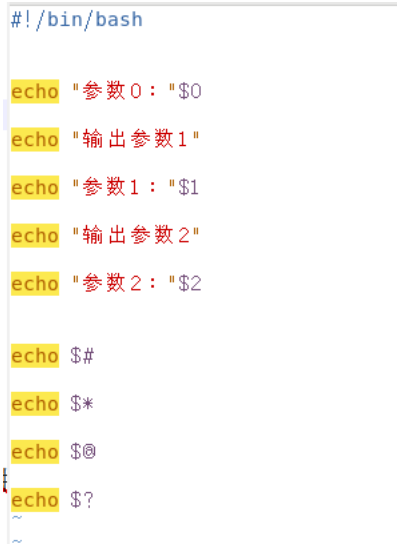

图 34: 查看

设置关键词为 echo，可以看到，打开文件后 echo 被高亮显示

（4）vim 文件路径 1 文件路径 2 文件路径 3 ：同时打开多个文件，文件之间可切换操作

3.移动光标

虽然可以直接使用 ↑↓← → 进行移动，但正规的是：

表 4: 移动光标

符号	描述
h	左
j	下
k	上
l	右

4.简单计算器

在编辑模式下 ctrl+r，然后点击 =, 光标就会移动到末行，输入要计算的式子，按下 enter，计算结果就会出现在原光标位置处

图 35: 计算器

图 36: 计算器

5.查找

当我们想要查找指定字符时，可以切换到底行模式下，输入? 和字符，下面是一个查找 hello 的例子：

图 37: hello

可以看到，在底行模式下输入：?hello，按下 enter 键后 hello 就会被高亮显示。

6.替换

切换到底行模式下，输入`%s/string1/string2/g`, 把 string1 替换成 string2, 下面把 hh 替换为 hello

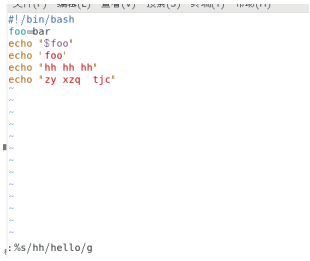

图 38: hh

图 39: hello

可以看到，hh 被换成了 hello

7.保存及退出

下面是不同的指令对应的不同功能：

表 5: 移动光标

符号	描述
q	直接退出
q!	强制退出
w	保存
wq	保存并退出
wq!	保存并强制退出

8.调用外部命令

切换到底行模式下，输入:! 和外部命令，比如说外部命令 ls, 就是:!ls

图 40: ls

enter 回车后就可以看到：

```
请按 ENTER 或其它命令继续
anaconda-ks.cfg  helloworld.sh          test1.sh   test7.sh   模板
game.sh          httpd_2.4_install.sh   test1.txt  test8.sh   视频
hello2.txt       initial-setup-ks.cfg   test2.sh   test9.sh   图片
hello.sh         last-modified.txt      test3.sh   tets3.sh   文档
helloshell.sh    marco_history.log      test4.sh   text1.txt  下载
hello.txt        missing                test5.sh   text2.sh   音乐
Helloword.sh     test10.sh              test6.sh   公共       桌面

请按 ENTER 或其它命令继续
```

图 41: ls

三、心得体会

通过学习，掌握了 shell 和 vim 的基础功能，同时对数据整理也有了一定的了解。这次学习丰富了自身技能，并激发了我的学习热情。

四、相关练习、报告和代码查看链接

本次报告相关练习、报告和代码均可以在 https://kkgithub.com/zhangtantan77/work 查看