《Linux 操作系统》第六讲：

Shell 编程

学时计划：4 学时 理论，4 学时 实验

（实验五：Shell 编程）

教学大纲：

1、Shell 基础

2、Shell 语法

3、Shell 编程案例一：用户管理

4、Shell 编程案例二：备份 MySQL 数据库

5、讨论与思考

Shell 是用户和 Linux 操作系统之间的接口。Linux 中有多种 shell，其中缺省使用的是 Bash。

本讲讲述 Shell 的工作原理、Shell 的种类、Shell 的一般操作及 Bash 的特性。最后通过两个案例介绍 Shell 的应用方法和 Shell 编程的一般思路。

本讲仅简单介绍 Shell 的基础知识，对于更多的 Shell 内容，本讲最后推荐了一些相关的专业教程和技术书籍。

一、Shell 基础

1.1Shell 介绍

Linux 系统的 Shell 作为操作系统的外壳，为用户提供使用操作系统的接口。它是命令语言、命令解释程序及程序设计语言的统称。

Shell 是用户和 Linux 内核之间的接口程序，如果把 Linux 内核想象成一个球体的中心，shell 就是围绕内核的外层。当从 shell 或其他程序向 Linux 传递命令时，内核会做出相应的反应。

1.1.1Shell 是一个命令语言解释器。

Shell 拥有自己内建的 Shell 命令集，Shell 也能被系统中其他应用程序所调用。用户在提示符下输入的命令都由 Shell 先解释然后传给 Linux 核心。有一些命令，比如改变工作目录命令 cd，是包含

在 shell 内部的。还有一些命令，例如拷贝命令 cp 和移动命令 rm，是存在于文件系统中某个目录下的单独的程序。对用户而言，不必关心一个命令是建立在 Shell 内部还是一个单独的程序。

Shell 运行时，首先检查命令是否是内部命令，若不是再检查是否是一个应用程序（这里的应用程序可以是 Linux 本身的实用程序，如 ls 和 rm，也可以是购买的商业程序，如 xv，或者是自由软件，如 emacs)。然后 Shell 在搜索路径里寻找应用程序（搜索路径就是一个能找到可执行程序的目录列表)。如果键入的命令不是一个内部命令并且在路径里没有找到这个可执行文件，将会显示一条错误信息。如果能够成功找到命令，该内部命令或应用程序将被分解为系统调用并传给 Linux 内核。

1.1.2Shell 自身就是一个解释型的程序设计语言。

Shell 程序设计语言支持绝大多数在高级语言中能见到的程序元素，如函数、变量、数组和程序控制结构。Shell 编程语言简单易学，任何在提示符中能键入的命令都能放到一个可执行的 Shell 程序中。

1.1.3Shell 的工作原理

当普通用户成功登录，系统将执行一个称为 Shell 的程序。正是 Shell 进程提供了命令行提示符。作为默认值，对普通用户用“$”作提示符，对超级用户（root）用“#”作提示符。一旦出现了 shell 提示符，就可以键入命令名称及命令所需要的参数。Shell 将执行这些命令。如果一条命令花费了很长的时间来运行，或者在屏幕上产生了大量的输出，可以从键盘上按 ctrl+c 发出中断信号来中断它（在正常结束之前，中止它的执行)。

当用户准备结束登录对话进程时，可以键入 logout 命令、exit 命令或文件结束符(EOF)(按 ctrl+d 实现),结束 Shell 的工作状态。

1.2Shell 与 Linux 内核的管理

Shell、用户与 Linux 操作系统内核之间的关系如下图 6-1 所示。

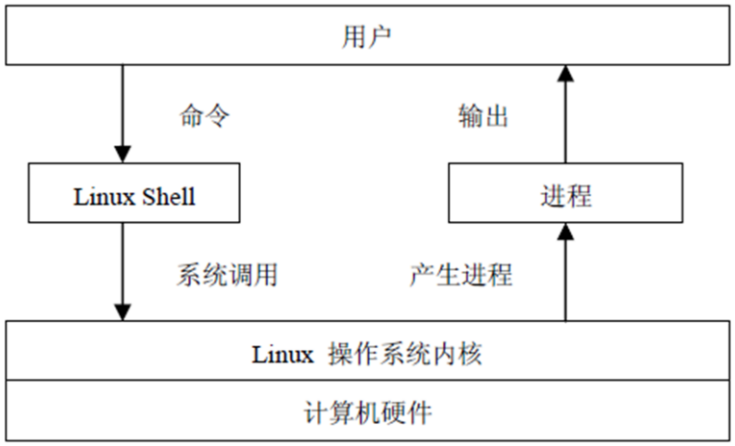

图 6-1 Shell、用户与 Linux 操作系统内核的关系

1.3 查看 Shell 版本

Linux 的 Shell 有多种类型，其中最常见的就是 Bourne Shell（sh）、C Shell（csh）和 Korn Shell（ksh）。

用户登录系统后，可以通过命令查看自己的 Shell 版本。

命令：

```
shell>echo $SHELL
```

如果需要临时的更换 Shell 版本的话，可以通过命令变更。

命令：

```
shell>csh
shell>echo $SHELL
shell>ksh
shell>echo $SHELL
shell>bash
shell>echo $SHELL
```

1.4Bash

Bash 是大多数 Ubuntu 系统的默认 Shell。Bash 的特点主要有下面几个方面。

（1）可自动补全命令。

（2）可以使用通配符。

（3）可以保存历史命令。

（4）支持命令别名。

Bash 中可以使用 alias 和 unalias 命令给命令或者可执行程序起别名和删除别名。

命令举例：

```
shell>lm
#lm 命令不存在，命令执行出错。通过 alias 让 lm 成为 ls -l 命令的别名。
shell>alias lm='ls -l'
#执行 lm 命令，相当于执行 ls -l 命令。
shell>lm
```

（5）支持输入/输出重定向。

输入/输出重定向用于改变命令的输入和输出的默认配置。Linux 系统默认的输入是键盘，默认的输出是显示器。

命令举例：

```
shell>ls > /home/zhaodongfeng/ls1.out
#ls 命令的结果不通过显示器显示而存放到文件中去。
shell>ls -l > /home/zhaodongfeng/ls1.out
#将 ls -l 命令的结果追加到 ls1.out 文件中，并覆盖以前的内容。
```

（6）支持管道功能。

管道用于将一系列的命令连接起来，也就是把前面命令的输出作为后面命令的输入。管道的命令是“|”。

```
shell>grep 211.69.32.33 /var/log/munin/munin-node.log
#查看日志中的内容。

shell>grep 211.69.32.33 /var/log/munin/munin-node.log |wc -l
#查看日志中的内容，其结果作为 wc 命令的输入，用来进行统计。
```

（7）支持两级提示符。

Bash 有两级提示符。第一级提示符是“$”。当 Bash 需要进一步提示以便补全命令时，会显示第二级提示符“>”。

1.5Shell 脚本的建立和执行

1.5.1Shell 脚本的建立

Shell 程序可以存放到文件中，这种被 Shell 解释执行的命令文件被称为 Shell 脚本。Shell 脚本程序与 Windows 的.bat 批处理文件

非常相似，但功能要比 bat 文件强大的多。

Shell 脚本程序可以在存放时不指定扩展名,通常为了便于记忆，总是习惯用.sh 来描述 Shell 脚本。

1.5.2 执行 Shell 脚本的方式

执行 Shell 脚本的方式有三种。

（1）输入重定向。

使用输入重定向的方式让 shell 从文件中读取命令并执行。

命令：

```
shell>bash<脚本文件路径与文件名
```

（2）以脚本文件为参数执行。

使用 bash 命令，将脚本文件作为 bash 命令的参数。

命令：

```
shell>bash 脚本文件路径与文件名
```

（3）将 Shell 脚本的权限设置为可执行后直接在提示符下执行。

通常情况下,用户是不能够直接执行由文本编辑器(vi 或者 nano)建立的 Shell 脚本的,因为建立的 Shell 脚本文件是没有执行权限的。通过 chmod 命令将 Shell 脚本的权限设置为执行权限后,可以直接执行 Shell 脚本文件。

命令：

```
shell>chmod +x 脚本文件路径与文件名
shell>./脚本文件路径与文件名
```

二、Shell 语法

2.1 建立一个 Shell 脚本程序

如同其他语言一样，使用任意一种文字编辑器，比如 vi、nano 等来编写 Shell 程序。

（1）程序必须以下面的行开始（必须放在文件的第一行）。

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

符号#!用来告诉系统它后面的参数是用来执行该文件的程序。在本例中使用/bin/bash 来执行程序。

（2）注释

在进行 Shell 编程时，以#开头的句子表示注释，直到这一行的

结束。建议在程序中使用注释。如果使用注释，那么即使相当长的时间内没有使用该脚本，也能在很短的时间内明白该脚本的作用及工作原理。

（3）第一个 Shell 脚本程序

在 Linux 中使用下述命令创建一个脚本程序。

```
shell>cd ~
shell>mkdir shell
shell>cd shell
#创建 Shell 脚本程序。
shell>nano hello.sh
```

在 hello.sh 中输入下述内容。

```
#!/bin/bash
#输出字符
echo “hello world!”
#end
```

保存后退出。执行下述命令，运行 Shell 脚本。

```
#执行 Shell 程序。
shell>bash hello.sh
#赋予 Shell 脚本执行的权限。
shell>chmod +x hello.sh
#直接执行 Shell 脚本。
shell>./hello.sh
```

2.2 变量

在 Shell 编程中，所有的变量都由字符串组成，并且不需要对变量进行声明。

变量名是以字母或下划线大头的字母、数字和下划线序列，并且大小写字母意义不同。变量名的长度不受限制。

要赋值给一个变量，可以直接写：变量名=值。需要注意的是“=”两边不能够有空格。取出变量值时可以加一个美元符号（$）在变量前面。

创建 Shell 脚本 printchar.sh，具体内容如下。

```
#!/bin/sh
#对变量赋值：
a="hello world"
# 现在打印变量 a 的内容：
echo "A is:"
echo $a
```

2.3 算数运算

Shell 是一种弱编程语言，算数运算的功能不算是很强大，一般使用((a=a+1))这种格式，或者使用 let a=a+1 形式进行算数运算。前者的执行效率最高。

Shell 进行算数运算的优先级和 C 语言的优先级是相同的，在此不再累述。

命令举例：

```
shell>let a=1
shell>let b=3
shell>let c=a+b
shell>echo “a is $a,b is $b,c is $c.”
```

2.4 条件语句：if

Shell 具有一般高级语言所具有的控制结构，如 if 语句、case 语句等。

if 语句根据表达式的值是真或者假来决定要执行的程序段落。最常用的 if 语句结构是 if…then…elseif…then…else…fi。

命令举例：判断当前的 Shell 是否是 bash。

```
#!/bin/bash
#判断当前的 Shell 是否为/bin/bash。
if [ "$SHELL" = "/bin/bash" ]
        then
                echo "your login shell is bash"
        else
                echo "your login shell in not $SHELL"
fi
#end
```

命令举例：判断用户输入的答案。

```
#!/bin/bash
#输出提示输入的对话。
echo -n "Please input the answer char:"
#等待用户数据内容。
read Input
if [ $Input = y ]
        then
                echo "The answer is right."
        elif [ $Input = n ]
                then
                        echo "The answer is wrong."
        else
                echo "The Input is bad."
fi
# end
```

2.5 条件语句：case

如果用户已经规划好几个项目类型，只要选择执行的类型就能够正确执行，虽然可以使用嵌套的 if 语句实现，但是 case 显然更为方便和容易。

命令举例：判断用户的选择。

```
#!/bin/bash
echo "what are your hobbies?"
case $1 in
        'table tennis')
                echo "your choose table tennis."
                        ;;
        basketball)
                echo "your choose basketball."
                        ;;
        football)
                echo "your choose football."
                        ;;
        *)
                echo "Usage {'table tennis'|basketball|football}"

        exit 1
esac
#end
```

可以通过两种方式来执行。

```
#带参数进行执行。
zhaodongfeng@TeachServer:~/shell$ bash checksports.sh football
what are your hobbies?
your choose football.
#不带参数进行执行，提示可以输入的参数。
zhaodongfeng@TeachServer:~/shell$ bash checksports.sh
what are your hobbies?
Usage {'table tennis'|basketball|football}
```

2.6 条件语句：select

select 表达式是一种 bash 的扩展应用，尤其擅长于交互式使用。用户可以从一组不同的值中进行选择。

命令举例：选择最喜欢的操作系统。

```
#!/bin/bash

echo "What is your favourite OS?"
select chooseos in "Linux" "Windows" "FreeBSD" "Mac Lion" "Other";do
        break;
done

echo "Your choose is $chooseos."

#end
```

执行该脚本的结果如下。

```
zhaodongfeng@TeachServer:~/shell$ bash chooseos.sh
What is your favourite OS?
1) Linux
2) Windows
3) FreeBSD
4) Mac Lion
5) Other
#? 4
Your choose is Mac Lion.
```

2.7 循环语句：for

for 循环语句是使用叫做的循环语句，在 Shell 中有两种方式。通过两个案例来进行说明。

命令举例：1+2+…+100=？

```
# !/bin/bash
let s=0
for ((i=1;i<=100;i=i+1))
do
        s=$(($s+$i))
done
echo "the count is $s"
#end
```

命令举例：循环列出数组中的数值。

```
# !/bin/bash
for m in 1 2 3 4 5 6
do
        echo $m
done
#end
```

2.8 循环语句：while 和 until

while 语句和 until 语句的语法结构和用途相似。while 语句会在测试条件为真时循环执行。

命令：

```
while expression
  do
      commands
  done
```

until 语句会在测试条件为假时循环执行。

命令：

```
until expression
  do
      commands
  done
```

命令举例：计算 1+2+3+4+5 的值。

```
#!/bin/bash
let s=0; p=1
#n1 -le n2 表示 n1 不大于 n2。
while test $p -le 5
        do
                let s=$s+$p
                let p=$p+1
```

```
        done
echo "1+2+3+4+5=$s"
#end
```

2.9break 和 continue

在 Shell 的 for、while、until 循环语句中，也可以使用 break 和 continue 语句以调离现有的循环。break 用于中断循环的执行，将程序流程移至循环语句结束之后的下一个命令。continue 语句则忽略之后的命令，将程序流程转移至循环开始的地方。break 和 continue 语句都可以加上数字，以指示要跳出的循环数目。

2.10 函数

Shell 脚本也有自定义函数的功能。当脚本变得很大时，可将脚本文件中常用的程序写成函数，这样使脚本更小、更易于维护。

命令举例：取出最大值。

```
#!/bin/bash
#定义函数
max()
{
        while test $1
        do
                if test $maxvalue
                        then
                                if test $1 -gt $maxvalue
                                        then
                                                maxvalue=$1
                                fi
                        #数据初始化。
                        else
                                maxvalue=$1
                        fi
                shift   //函数参数左移一位。
        done
        #返回值
        return $maxvalue
}

max $*
echo "max value is:$maxvalue."
#end
```

调用该函数的方式。

```
zhaodongfeng@TeachServer:~/shell$ bash choosemax.sh 34 68 90 12 45
max value is:90.
```

三、Shell 编程案例一：用户管理

目的：

编写 Shell 脚本，实现通过参数定义用户名、口令，实现用户的创建。

案例：usermanage.sh

```
#!/bin/bash

echo -n "Please input your username:"
read username

echo -n "your username is $username,are you sure?(y or n):"
read usernamesure

if [ "$usernamesure" = "y" ]
        then
                sudo mkdir /home/$username
                sudo useradd $username -d /home/$username
                echo "$username is create success."
                echo "Please change password for $username."
                sudo passwd $username
        elif
                [ "$usernamesure" = "Y" ]
                        then
                        sudo mkdir /home/$username
                        sudo useradd $username -d /home/$username
                        echo "$username is create success."
                        echo "Please change password for $username."
                        sudo passwd $username
        elif
                [ "$usernamesure" = "n" ]
                        then
                                echo "your operate is cancel."
                                break
        elif
                [ "$usernamesure" = "N" ]
```

```
                then
                        echo "your operate is cancel."
                        break
        else
                echo "your input is bad."
                break
fi
#end
```

脚本执行的情况，如下。

```
zhaodongfeng@TeachServer:~/shell$ bash usermanage.sh
Please input your username:zhaodongfeng
your username is zhaodongfeng,are you sure?(y or n):y
zhaodongfeng is create success.
Please change password for zhaodongfeng.
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

四、Shell 编程案例二：备份 MySQL 数据库

目的：

通过 Shell 脚本实现 MySQL 数据的自动备份。

案例：MySQL 数据库备份

```
#!/bin/bash

username="zhaodongfeng"
userpassword="zdf123456"

BKdatabase="student"
BKfoldername="/var/www/databasebackup/"

BKdatabasefilename="$BKdatabase-`date '+%y%m%d%H%M'`.sql"

cd $BKfoldername

mysqldump -u$username -p$userpassword $BKdatabase>$BKdatabasefilename

cd ~

echo "MySQL database $BKdatabase Backup success!"
```

```
echo "Backup file is $BKdatabasefilename."
```

通过 Shell 进行数据库备份需要首先在数据库中创建帐户并授予权限，在服务器上创建文件目录，并赋予权限。

可以将 Shell 和计划任务结合起来，实现定时和周期性的 MySQL 数据库备份，并可以删除之前备份的旧的档案数据。

五、讨论与思考

5.1 Shell 编程语言

（1）Shell 脚本虽然具备一定的高级编程语言，但是功能还是比较简单。是否可以使用其他高级开发语言编写 Shell 脚本？

（2）Shell 脚本是否可以完成所有的 Linux 操作？

5.2Shell 迷思

（1）听说 awk 是 Shell 编程的重点，他能够完成什么工作？

（2）Shell 脚本安全么？