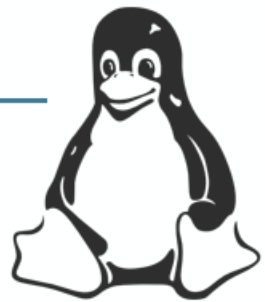


Linux系统管理

南通师范高等专科学校 朱亚林

本章导言

“欲穷千里目，更上一层楼”。信息技术领域的学习与创新永无止境。之前的章节，较为全面的介绍了以Debian为代表的Linux操作系统的基础知识，学习了使用命令操作计算机去完成相应的工作。但这些还只是基础，要想让Linux操作系统运转得更加智能化，就不得不去深入探究Shell。本章将系统的介绍Shell的基础情况，以及如何利用它进程编程，从而进一步提升系统管理的效率。

第9章 Shell编程

9.1.1 什么是Shell

- 人类与计算机，就像是拥有着两种不同语言的物种。人类讲述的是自然语言，计算机能够理解的是机器语言。自从操作系统出现之后，人类与计算机的交互才开始越来越便捷。这是因为，操作系统搭建起了人与计算机沟通的桥梁。
- 在整个计算机体系中，处于最基础部分的是计算机硬件，因为有了这些硬件（CPU、内存、硬盘、显示器、鼠标、键盘），才让用户有了可以操作和获取结果的设备；与硬件打交道是操作系统的内核；而在内核之上，则是真正与用户进行交互的Shell。
- 广义上来说，Shell就是用户向计算机内核传达指令、操作硬件的一个中间层，是人与机器进行交互的界面。一般来说，Shell分为两种：一种是图形界面（GUI），比如Windows Explorer；另一种是指令操作（CLI），也就是Linux中的命令行操作。

9.1.2 你用的是什么Shell

实例：查询SHELL信息

（1）查询本机安装Shell的情况

（2）查询当前用户使用的Shell

（3）临时切换Shell

（4）回到默认Shell

（5）正式切换到zsh

9.1.3 Shell的好伙伴——管道

- 管道是将程序的输出与另一程序的输入连接起来的操作。Ken Thompson在将管道机制添加到Shell时说，运行结果“很震撼”。作为亲历Unix发展的Brian Kernighan在其著作《Unix传奇：历史与回忆》中说到“管道也许是Unix中最引人注目的创新”。因此，在学习Shell时，用户必须要掌握管道的相关功能。

9.1.3 Shell的好伙伴——管道

- 管道，是在两条相关命令之前存在的一道竖杠——“|”。例如：

```
ls | wc
```

- 再如，要查看当前操作系统用户登录情况，则可以使用以下命令：

```
who     #谁登录了
who | wc -l  #有多少人登录了
who | grep zz  #zz是否登录了
```

9.2 Shell启动的奥秘

- Shell程序在启动时，会自动执行用户主目录下的.bashrc文件中的命令。默认情况下，.bashrc中会写有关于命令行样式的配置、用户环境变量的配置、命令别名的配置以及自定义启动运行的程序等等。

9.2.1 命令行历史记录的设置

- 在命令行中输入的命令都会被用户主目录中.bash_history这个文件记录。这些记录可以通过上下方向键进行浏览，也可以通过history命令来查看。
- 使用vim打开.bashrc，找到如下图所示的这节内容。

```
# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000
```

9.2.2 命令提示符样式的设置

- 命令提示符是启动Shell之后，默认在光标闪烁点之前的那一段提示信息。一般会包括用户名、主机名以及当前目录名。
- 以下这张截图就是.bashrc中配置命令提示符的一段。

```
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
```

9.2.3 别名的设置

- 对于网络上一些较长的网址，人们可以通过短网址网站对其进行设置，使其缩短长度，便于输入。这样的网站有诸如：https://www.urlc.cn/等。
- 在Linux中，用户也可以将一些常用的命令进行缩短，或者指定一个习惯的新名字，这就是别名的应用，而相关的设置只需要在.bashrc中进行即可。
- 使用vim打开.bashrc文件，可以看到有如下图所示的设置区域。

```
# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
```

9.3 Shell的变量

Shell的功能之一在于它是一个命令解释器，之前的章节中介绍的命令都充分体现了这一强大的功能。Shell还有一项特性是它可以进行编程。为了在编程过程中可以方便的将对象进行传递、访问，和其他编程语言一样，变量是必须存在的。Shell中的变量分为以下几类：

- 环境变量：保存和系统操作环境相关的数据；
- 本地变量：用户自定义的变量，只在当前的Shell中生效；
- 位置参数变量：向脚本当中传递参数或数据的变量。

9.3.1 环境变量

- 在Shell中，有一些预定义的环境变量，包含着系统中的一些特定信息。这些变量在命令行中可以直接使用。可以通过以下命令，将当前环境变量显示出来：

```
zz@zz-server:/$ env
```

```
zz@zz-server:/$ env
TERM=xterm-256color
SHELL=/bin/sh
SSH_CLIENT=192.168.3.71 63648 22
SSH_TTY=/dev/pts/1
LC_ALL=en_US.utf8
USER=zz
PAGER=more
MAIL=/var/mail/zz
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin:/usr/local/sbin
:/usr/local/bin
PWD=/
LANG=en_US.utf8
SHLVL=1
HOME=/var/services/homes/zz
TERMINFO=/usr/share/terminfo
LOGNAME=zz
SSH_CONNECTION=192.168.3.71 63648 192.168.3.99 22
PGDATA=/var/services/pgsql
_=/bin/env
```

变量名	含义
$HOME	用户主目录的完整路径名
$PATH	Shell搜索命令的路径
$USER	用户名
$SHELL	当前shell
$TERM	当前终端的类型

9.3.2 本地变量

本地变量是指由用户自行创建的变量，Shell中变量的创建格式如下：

变量名=变量值

变量的定义无需声明类型，且“=”两侧不能有空格。在定义过程中，如果变量不存在，就创建该变量；如果变量已经存在，则用新的变量值覆盖原来的值。如果所赋的值是一个连续的字符串，则将字符串直接输入；如果所赋的值是一个字符（串），则以引号将值括起来。

- 变量区分大小分，即fileName和FILENAME是两个变量；
- 变量必须以字母或下划线开头；
- 变量可以由数字、字母和下划线等字符组成。

9.3.2 本地变量

实例1:

（1）创建本地变量

（2）设置环境变量

（3）检测本地变量与环境变量的差异

（4）创建只读变量

（5）从命令行输入变量

9.3.3 Shell中变量的显示

在命令行中成功定义了变量后，可以通过echo命令，配合特殊意义的字符，将变量的相关属性进行显示。

实例：显示变量

（1）直接显示变量

（2）显示变量与其它对象的组合信息

（3）替换没有定义过的变量

（4）替换定义过的变量

（5）变量未设置时报错

（6）显示变量字符串长度

（7）截取变量指定部分

（8）删除变量中指定部分

9.3.4 变量的取消

取消变量是指将已经存在的变量从内存中释放。取消变量的基本格式如下：

unset 变量名

实例：取消变量

9.4 Shell的数组

- Shell中的数组是一个较为特殊的数据结构，一个数组是由一组元素组成，这组元素在数组中从0开始依次进行编号（下标），如下图所示。

9.4.1 数组的创建

- 数组的创建主要使用declare命令，一般格式如下：

```
declare -a 数组名
```

- 数组创建完成后，使用下标名给数组赋值，一般格式如下：

```
数组名[0]=值
```

9.4.2 数组的访问

数组的访问非常方便，是由数组名结合索引进行的。一般格式如下：

echo ${数组名[索引]}

如果要获取数组的其它信息，则可以通过特殊字符来进行：

- ${数组名[*]}或${数组名[@]}可访问全部元素
- ${#数组名[*]}或${#数组名[@]}可获取数组长度
- ${!数组名[*]}或${!数组名[@]}可获取数组内容的序号

- 切片

9.4.3 数组的连接

多个数组可以连接成为一个新的数组，一般格式如下：

```
新数组=(${数组1[*]} ${数组2[*]})
```

需要注意的是，数组1和数组2两个表达式间必须要有一个空格，否则数组1的最后一个元素会与数组2的第一个元素连接在一起成为一个元素。

9.4.4 数组元素的替换

可以在数组中查找指定的内容并进行替换，一般格式如下：

数组名=(${数组名[*]/查找元素/替换元素})

9.4.5 数组及元素的取消

- 和变量的取消一样，数组及元素的取消使用的命令也是unset。

9.5 Shell中的算术运算

- Shell命令行可以进行简单的算术运算，但与Python等语言的解释器不同，Shell中需要通过特定符号来指明运算部分，否则视为字符串的操作。另外，Shell只支持整数的计算，遇到小数时会将小数点向后的部分舍去。
- Shell中算术运算的一般格式如下：

((表达式)) 或 [表达式]

9.5.1 赋值

给变量赋值的方法有多种:

- a=10
- (($a=10))
- $[a=10]
- let a=10
- declare -i a=10 # -i表示声明整型变量 int

9.5.2 计算

- Shell中支持的计算类型有加、减、乘、除、模、幂等常规计算，还包括类似于“+=”等复合运算、自增自减运算，以及位运算等。

9.5.2 计算

例：Shell中的简单计算

步骤：

（1）声明变量并赋值

```
zz@debian:~$ ((a=9))
zz@debian:~$ ((b=2))
```

（2）进行简单运算

```
zz@debian:~$ echo $((a+b))    #加法
zz@debian:~$ echo $((a-b))    #减法
zz@debian:~$ echo $((a*b))    #乘法
zz@debian:~$ echo $((a/b))    #除法
zz@debian:~$ echo $((a%b))    #模
zz@debian:~$ echo $((a**b))   #幂
```

（3）进行复合运算

```
zz@debian:~$ ((a+=b))
```

a=a+b(赋值符号)

以上命令是指执行a=a+b运算。类似这样的复合运算还有：

a-=b 等价于a=a-b

a*=b 等价于a=a*b

a/=b 等价于a=a/b

a%=b 等价于a=a%b

（4）自增自减

```
zz@debian:~$ ((a=9))
zz@debian:~$ ((a++))    a++   a=a+1   a+=1
```

与之相对应的有自减，即a--，是指a=a-1。

9.6 Shell脚本的规范

- Shell有两种工作模式：一种是互动模式，即用户输入命令，系统即时予以反馈；另一种是脚本模式，是指将需要执行的命令以一定的规则写在文件中，由Shell读取并执行。脚本模式使得人机对话更为“自动化”，大大提升了工作效率，方便用户对于任务的部署。

9.6.1 Shell脚本的基本格式

编写Shell脚本无需借助于特定的编辑器，使用cat命令、VIM编辑器、Nano编辑器都可以，但是在编写时应该注意以下几点：

- 脚本文件名一般以.sh结束（为了方便辨识）
- 脚本第一行内容必须为“#!/bin/bash”（不包含引号）
- 脚本中所有以“#”开头的内容为注释内容
- 脚本编写完成后需要赋予执行权限（不赋予权限将会报错）

9.6.1 Shell脚本的基本格式

例：第一个Shell脚本

步骤：

（1）通过vim新建一个welcome.sh

（2）在welcome.sh中输入以下内容

（3）给welcome.sh赋予执行权限

（4）执行welcome.sh

```
#!/bin/bash
# It's first shell script
echo "Hello,Linux"
```

9.6.2 Shell脚本与参数

- shell脚本在执行时可以像互动模式那样接收用户输入的参数。一般格式如下：

```
./脚本文件名.sh 参数1 参数2 参数..
```

- 脚本在接收到这些参数后，是通过位置参数的方式来进行处理的。位置参数遵循以下规则。

位置参数	含义
$0	脚本文件名
$1-$9	接收到的第1到第9个参数
$(10+)	接收到的位置大于10的参数，位置需要使用括号
$#	计算参数的总个数
$*或$@	全部参数，以空格隔开

- 另外，可以使用shift命令将位于队列最前端的参数（$1）删除，之后其余参数顺次前移。

9.6.2 Shell脚本与参数

例：Shell脚本读取参数

步骤：

（1）创建读取参数的脚本

（2）在readParameter.sh中输入以下内容

（3）执行脚本并输入参数

```
#!/bin/bash
echo “文件名为：” $0
echo “参数个数为：” $#
echo “参数内容为：” $*
echo “第1个参数为：” $1
shift
echo “第1个参数为：” $1
```

9.7 Shell脚本的输入与输出

- 检测一个程序是否智能化，很重要的一点，体现在人机的友好交互上。输入与输出是这项交互的基础。Shell脚本中提供了文字化的输入输出以及图形化（dialog）的输入输出。

9.7.1 输入命令——read

- 之前的章节已经提到，read命令可以直接在命令行使用。作为一个重要的输入命令，read的功能极为丰富。
- read命令可以从键盘读取数据，也可以通过重定向从文件中读取数据。read命令的格式一般如下：

```
read [选项] 参数
```

9.7.1 输入命令——read

选项	说明
-a array	把读取的数据赋值给数组 array，下标从0计数
-d delimiter	用字符串delimiter作为读取结束的标志。
-e	在输入路径时，可以使用Tab命令补全功能
-n num	读取长度为num的字符串。
-p prompt	显示提示信息，提示内容为 prompt。
-r	原样读取（Raw mode），不把反斜杠字符解释为转义字符。
-s	静默模式（Silent mode），在输入字符时不再屏幕上显示（如输入密码）。
-t seconds	设置超时时间为seconds，单位为秒。
-u fd	使用文件描述符 fd 作为输入源，而不是标准输入，类似于重定向。

9.7.1 输入命令——read

例：使用提示信息输入数据

步骤：

（1）创建脚本inputText.sh，并在其中输入以下内容

（2）保存inputText.sh，并赋予执行权限

（3）执行脚本，查看运行结果

```
#!/bin/bash
read -p “请输入您的姓名：” name
read -p “请输入您的家乡：” hometown
echo “欢迎您，来自”$hometown“的”$name
```

9.7.1 输入命令——read

例：读取密码数据

步骤：

（1）创建脚本inputPasswd.sh，并在其中输入以下内容

（2）为inputPasswd.sh赋予执行权限，并查看运行结果

```
#!/bin/bash
read -t 5 -p “请输入您的用户名：” username
read -t 5 -s -p “请输入您的密码：” passwd
echo -e “\n您输入的密码是：”$passwd
```

9.7.1 输入命令——read

例：读取数组信息

步骤：

（1）创建脚本inputToArray.sh，并在其中输入以下内容

（2）为inputToArray.sh赋予执行权限，并查看运行结果

```
#!/bin/bash
read -p “请输入学生姓名：” -a student
echo “一共有”${#student[*]}“名学生信息，他们是：”
echo ${student[*]}
```

9.7.2 输出命令——echo与printf

- Shell中输出信息一般使用echo命令与printf命令。echo命令在之前的章节中已经作过详细的讲解，这里主要介绍printf命令。
- printf命令输出的文本或参数以空格分隔，可以自定义格式化字符串，以及制定字符串的宽度、左右对齐方式等。print的一般格式如下：

```
printf [-v var] format [arguments]
```

- [-v var]是指，printf可以将其后的信息输出到变量var中，而不是直接输出到屏幕；
- format是指，设置其后内容输入的格式，例如用制表符将字符串按照设定的样式输出；
- [arguments]是指参数列表。

9.7.2 输出命令——echo与printf

例：printf输出简单信息

步骤：

（1）使用printf在命令行中输出信息

（2）使用printf输出换行信息

9.7.2 输出命令——echo与printf

- 转义序列是指编程语言中有特殊意义的符号标记，比如上例中的“\n”代表的就是换行的意思。Shell中printf支持的常见转义序列如下表所示。

转义序列	作用
\"	双引号
\\	反斜杠
\a	响铃
\b	退格
\c	截断输出
\e	退出
\f	翻页
\n	换行
\r	回车
\t	水平制表符
\v	竖直制表符

9.7.2 输出命令——echo与printf

例：printf输出类表格信息

步骤：

（1）创建脚本printTable.sh，并在其中输入以下内容

（2）赋予printTable.sh执行权限，并执行

```
#!/bin/bash
printf “姓名\t性别\t年龄\t\n”
printf “张怡\t女\t18\t\n”
printf “李芹\t女\t19\t\n”
printf “王笑\t男\t19\t\n”
```

9.7.2 输出命令——echo与printf

printf还可以预设好输出格式，再将输出内容作为参数填入输出。在预设样式时，通常会用到以下格式替代符：

- %s：输出一个字符串，同时可以加入输出宽度的参数N。%-Ns指宽度为N个字符（- 表示左对齐，没有则表示右对齐），宽度不足N则自动以空格填充，超过也会将内容全部显示出来。
- %c：输出一个字符。
- %d：输出一个整数。
- %f：以小数形式输出实数，同时可以加入输出参数m与n。%-m.nf 指格式化为小数，其中.n指保留n位小数，m同字符串。

9.7.2 输出命令——echo与printf

例：printf定义格式输出信息

步骤：

（1）创建脚本printStyle.sh，并在其中输入以下内容

（2）赋予printStyle.sh执行权限，并执行

```
#!/bin/bash
printf “%-8s %-8s %-8s %-8s\n” 姓名 性别 年龄 “身高（M）”
printf “%-8s %-8s %-8s %-8.2f\n” 张怡 女 18 1.721
printf “%-8s %-8s %-8s %-8.2f\n” 李芹 女 19 1.685
printf “%-8s %-8s %-8s %-8.2f\n” 王笑 男 19 1.786
```

9.8 Shell编程的分支控制

- 流水式的平铺直叙只能解决线性的程序问题，在处理具体问题时，还需要进行条件判断等操作，这些就是流程分支控制的问题。Shell编程中提供了这些功能丰富的控制语句。

9.8.1 test命令

- test命令一般用来检查某个条件是否成立。这里的条件可以是进行数值、字符串的比较，也可以是检测某个文件是否存在。
- test命令使用的一般格式如下：

```
test [表达式] [选项]
```

9.8.1 test命令

- 不同于其他语言，Shell在比较数值和字符串关系时，其操作符有所不同，具体如下表所示。

数值比较	字符串比较	说明
-eq	=	等于则为真
-ne	! =	不等于则为真
-gt	>	大于则为真
-ge	>=	大于等于则为真
-lt	<	小于则为真
-le	<=	小于等于则为真
	-z string	字符串长度为零则为真
	-n string	字符串长度不为零则为真

9.8.1 test命令

1. 变量间的关系测试

数值和字符串都是存放在变量中的，使用test在进行变量关系检测时除了要注意使用什么样的操作符之外，还要注意变量的表达形式。 $? 返回上一条命令执行完的结果

SHELL中，返回值0表示真，返回值1表示假

例9-19：使用test测试数值与字符串关系

（1）在Shell中创建数值变量，并进行比较

（2）在Shell中创建字符串变量，并进行比较

9.8.1 test命令

2.文件测试

在使用Shell脚本进行操作系统管理时，经常会去检测一些文件是否存在，以及该文件具备哪些属性等等，test就具备这样的功能。

9.8.1 test命令

test在检测文件时，经常用到的选项如下表所示。

选项	说明
-b 文件名	如果文件存在且是块文件则为真
-c 文件名	如果文件存在且是字符设备则为真
-d 文件名	如果文件存在且是目录则为真
-e 文件名	如果文件存在则为真
-f 文件名	如果文件存在且为普通文件则为真
-x 文件名	如果文件存在且为可执行文件则为真
-w 文件名	如果文件存在且可写则为真
-r 文件名	如果文件存在且可读则为真
-l 文件名	如果文件存在且为链接时则为真
-s 文件名	如果文件存在且至少有一个字符则为真
文件1 -nt 文件2	当文件1比文件2新时则为真
文件1 -ot 文件2	当文件1比文件2旧时则为真

9.8.1 test命令

3. 另一种测试表达方法

在对上述的变量及文件情况进行测试时，除了可以使用test命令之外，也可以使用中括号来进行操作。

例如：

- test “$a” = “$b”等价于[“$a” = “$b”]。
- test -d test 等价于 [-d test]

相关规则与test别无二致。

9.8.2 if条件语句

- Shell程序中的条件分支是通过if语句来实现的。if语句的语法格式如下：

```
if 条件命令1; then
    条件命令1为真时执行的命令
[elif 条件命令2; then
    条件命令2为真时执行的命令...]
[else
    条件命令为假时执行的命令]
fi
```

9.8.2 if条件语句

例：使用if检查文件是否存在

脚本将会获取用户在脚本名称后输入的一个文件名，检测这个文件是否存在，并给出相应的提示。

9.8.2 if条件语句

if在进行判断时，其后的条件命令可以是一个表达式，也可以是一组表达式进行的组合判断。

Shell中的逻辑关系有“与”、“或”、“非”三种：

- 与：-a，连接多个表达式，且每个表达式都为真时，结果才为真
- 或：-o，连接多个表达式，且只要有一个表达式为真，结果即为真
- 非：!，这是单目操作符，加在某个表达式前，表示取相反的结果

9.8.2 if条件语句

例：用户登录检测

脚本将会提示用户输入用户名和密码，并对其进行判断。符合要求则登录，不符合要求则给出相应提示信息。

9.8.3 case条件选择

- if条件语句用于在两个选项中选定一项，而case条件选择则为用户提供了根据表达式的值从多个选项中选择匹配项的渠道，即多选一。
- case条件选择的语法结构如下：

```
case 表达式 in
匹配1) 语句1;;
匹配2) 语句2;;
匹配3) 语句3;;
...
*) 通配语句;;
esac
```

- 在以上语法结构中，表达式会得出一个明确的返回值，由此值和匹配1~匹配n相比较，如果有一致的，则执行其后的语句；如果所有的匹配都与表达式结果不一致，那么执行*)之后的通配语句。

9.8.3 case条件选择

例：使用case实现分数到等级的转换

本例执行时，会要求用户输入一个分数（0~100），脚本将分数转换成相应的等级。90及以上为“A”，80及以上为“B”，70及以上为“C”，60及以上为“D”，其余为“E”。

9.8.3 case条件选择

例：使用case实现正则表达式匹配

本例执行时，会要求用户输入一个字符，程序判断输入的是什么类型。

9.9 Shell编程的循环控制

- 循环是指让某一段程序按照指定规则反复执行，使用循环控制语句可以使编程效果大大提升。在Shell中，可用于循环的语句有for、while、until、select等几类。

9.9.1 for循环

1. 类C循环

C语言中的循环简洁明了，通过三段语句分别规定了循环的起始、终点和渐进，Shell中的for循环与之极为相似，其一般语法格式如下。

```
for ((表达式1;表达式2;表达式3));do
循环语句
done
```

其中：

- 表达式1是初始化语句，一般用于定义循环中关键变量，并对其初始化；
- 表达式2是判断表达式，当此判断成立时循环会进入下一轮，当判断不成立时，循环终止；
- 表达式3是普通语句，一般用来修改关键变量的值，其直接影响表达式2的判断成立与否。
- do与done之前写入需要循环的内容

9.9.1 for循环

- 例：使用类C循环批量创建和删除文件夹

9.9.1 for循环

2. for...in循环

这种for循环的结构与Python中的for循环非常类似，使用起来也非常直观，其语法结构一般如下：

```
for 变量 in 列表;do
循环语句
done
```

其中：

- 列表，包含着需要被循环的对象；这里的列表呈现形式多样，既可以是传统的数组，也可以是从m到n的简略写法{m..n}，还可以是一个通配符“*.doc”；
- 变量根据列表的内容而定，循环结构会将列表中的对象从第1个开始，逐一取出按照顺序赋值给变量，即如果列表中有3个对象，第1次循环时变量的值就是列表中第1个对象的值，第2次循环时变量的值就是第2个对象的值，以此类推；当最后一个对象赋值给变量后并执行循环语句后，循环中止。

9.9.1 for循环

- 例：使用for...in...循环批量创建文件夹

9.9.2 while循环

while循环的句式相对简单，即在while之后有一个判断语句作为循环条件，其语法格式如下：

```
while 判断条件; do
循环语句
done
```

这里的判断条件会产生一个返回值，当为真时，循环继续，当为假时，循环终止。

9.9.2 while循环

- 例：通过while循环实现1+2+3+...+100的求和

9.9.2 while循环

- 例：使用while读取文件

9.9.3 until循环

until也是循环的一种，其与while循环的基本要求恰好相反，即当判断条件失败时，就不断循环执行指定的语句。一旦符合判断条件，就退出循环。

until循环的语法格式如下：

```
until 判断条件; do
循环语句
done
```

对于until这样特殊的循环可用场景还是很多的，例如检测某项工作有没有完成，如果处于未完成状态，那么就会一直提示，直到完成为止。

9.9.3 until循环

- 例：使用until检测日志文件是否创建

9.9.4 select循环

select循环一个非常奇妙，它会把列表的内容像菜单一样呈现在屏幕上供用户选择。

select循环的语法结构如下：

```
select 变量 in 列表; do
循环语句
done
```

其中：

- 列表中的信息将会被条目式显示在屏幕上，并配以编号；
- 在按照编号对信息进行选择后，该条目的列表内容部分将被赋值给“变量”，而编号部分则会被环境变量REPLY接收；
- 在循环语句结束后，会等待用户的下一次输入，如果按下回车键，会继续将列表中的信息条目式显示；
- select循环只能借助于break语句才能中断。

9.9.4 select循环

- 例：使用select制作水果询价程序

```
zz@debian:~/LinuxStudy/chapter-9$ ./plu.sh
请选择商品序号：
1) 苹果
2) 香蕉
3) 梨
4) 桃子
5) 火龙果
6) 樱桃
7) 西瓜
8) 退出
#? 6
樱桃29元/斤
#? 5
火龙果9元/斤
#?
1) 苹果
2) 香蕉
3) 梨
4) 桃子
5) 火龙果
6) 樱桃
7) 西瓜
8) 退出
#? 8
已退出系统！
```

9.9.4 select循环

在循环体中经常会遇到需要中止循环的操作，这时就要用到break和continue语句了。break语句是彻底中止当前的循环；continue语句不会中止整体循环体，而是结束当前轮次的循环进入下一轮。

9.10 Shell编程的函数

- 函数是将特定程序功能进行打包命名的一种方法。例如，有一段代码专门用来判断素数，那么就可以将它独立出来，并且进行命名，在需要使用的地方进行调用即可。函数还有一大特点就是可以反复调用。

9.10.1 函数的定义

函数的定义语法有两种。

第一种：

```
function 函数名(){
        语句1
        语句2
        ...
}
```

第二种：

```
函数名(){
        语句1
        语句2
        ...
}
```

这两种方法定义出来的函数效果是一样的。

9.10.1 函数的定义

- 例：使用函数定义欢迎信息并调用

9.10.2 参数的传递

调用函数解决问题时，经常要向函数中传递相关的数据，也就是参数。Shell函数中参数的传递方法如下：

- 传递参数：调用函数时，在函数名后写上参数，以空格隔开；
- 使用参数：在函数中使用类似“$n”的方式来调用参数，第1个参数即为“$1”,第2个参数为“$2”，以此类推。

9.10.3 函数的返回值

- 调用函数进行问题处理，不只有以上显示信息这一种反馈形式，大部分情况下还是要将处理的结果返回到调用的位置的。函数的返回主要使用到return命令。
- return命令用于向调用函数的位置返回一个值（也可以只有return，没有返回值），当函数体遇到该命令后就不再继续执行了。return命令的使用格式如下。

```
function 函数名 {
        ...
        return 返回值
}
```

在函数调用的位置之后，可以使用

```
变量=$?
```

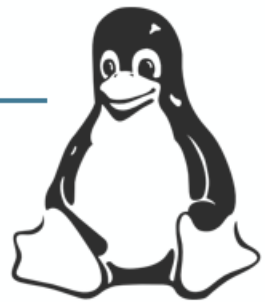
获取返回值。

9.10.4 函数的共享

当A文件调用B文件中的函数时，就叫做函数的共享。用户可以将常用功能封装成函数，并且放置在一个专门的文件中，在具体应用中，可以直接进行调用。

函数共享调用的方法如下：

- 在调用位置使用source命令加上函数所在的文件名
- 在调用位置使用“.”加上函数所在的文件名



Linux系统管理

南通师范高等专科学校 朱亚林