# 第5章 chapter 5

# Linux的Shell和自动化程序

Shell的原意是外壳，用来形容物体外部架构。各种操作系统都有自己的Shell，在DOS系统中，它的Shell是command.com程序，而Windows操作系统的程序Shell是explorer.exe程序。与Windows等操作系统不同，Linux系统中将Shell独立于操作系统核心程序之外，使得用户可以在不影响操作系统本身的情况下进行修改，更新版本或添加新的功能。Shell在不同操作系统中可能有不同的实现和特性，但它们的基本目的都是为用户提供一种与计算机系统交互的方式。

## 5.1 Shell入门和基础知识

### 5.1.1 Shell的概念

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

### 5.1.2 Shell的类型

#### 1. Shell的类型及功能特点

在Linux系统中，有多种不同的Shell可供选择，每种都有其独特的特性和用途。常用的几种Shell是Bourne Shell(sh)、C Shell(csh)、Ash Shell(ash)、Korn Shell(ksh)和Bourne Again Shell(bash)等。

#### 2. Shell的查看方法

可以使用echo和chsh或者cat /etc/shells命令查看Shell，具体方法如表5.1所示。

表5.1 查看目前系统可以使用的Shell

| 方　法 | 命　令 | 描　述 |
| --- | --- | --- |
| echo $0 | echo $0 | 查看当前使用的Shell |

续表

| 方　法 | 命　令 | 描　述 |
| --- | --- | --- |
| ps -p $ $ | ps -p $ $ | 查看当前使用的 Shell |
| 查看 /etc/shells 文件 | cat /etc/shells | 显示 /etc/shells 文件的内容，其中包含系统中可用 Shell 的路径 |
| 使用 chsh 命令 | chsh -l | 列出系统中所有可用的 Shell，并提示选择要更改为的 Shell |
| 查看可执行文件 | ls /bin<br>或<br>ls /usr/bin | 列出 /bin 和 /usr/bin 目录中的可执行文件，其中包括各种 Shell 的执行文件 |
| 查看 /etc/passwd 文件 | cat /etc/passwd | 显示 /etc/passwd 文件的内容，包含系统中所有用户的信息，其中包括默认 Shell 的信息 |

示例：分别使用 cat /etc/shells 命令和 chsh -l 命令，查看目前系统可以使用的 Shell，如图 5.1 所示。

### 3. Shell 的切换方法

Shell 的切换方法有多种，用户可以通过 exec 或 chsh 命令来切换 Shell，使用 echo 命令查看 Shell。

示例：查看当前使用的 Shell 进程(默认是 Bash)和目前系统中能使用的 Shell，使用 exec 命令替换当前 Shell 进程(替换成另一个)。然后再替换回原来的 Shell 进程，具体操作如图 5.2 所示。

```
[root@localhost ~]# more /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
[root@localhost ~]# chsh -l
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
[root@localhost ~]#
```

图 5.1　查看目前系统可以使用的 Shell

```
[root@localhost ~]# chsh -l
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
[root@localhost ~]# echo $0
bash
[root@localhost ~]# exec sh
sh-5.1# echo $0
sh
sh-5.1# bash
[root@localhost ~]#
[root@localhost ~]# ps -p $$
    PID TTY          TIME CMD
   2873 pts/0    00:00:00 bash
[root@localhost ~]#
```

图 5.2　Shell 的切换方法

## 5.1.3　创建和执行简单的 Shell 程序

创建和执行一个 Shell 程序非常简单，一般需要以下三个步骤。

(1) 利用文本编辑器创建脚本内容。

(2) 使用“chmod”命令设置脚本的可执行属性。

(3) 执行脚本。

一个合法的 Shell 脚本程序,都是以 Shell 解释器声明开始的,即在 Shell 程序的第一行。其中,“#!”后面的“/bin/bash”表示实际使用的解释器。

### 1. 创建 Shell 脚本

1) 选择 Shell

决定要使用的 Shell,如 Bash、Zsh 等。在脚本的第一行添加 Shebang 行,指定使用的 Shell,例如:

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

2) 编写脚本

使用文本编辑器 gedit 命令或 vi 命令创建 Shell 脚本文件,添加要执行的命令。例如,创建一个简单的脚本 myscript.sh,如下。

```
#!/bin/bash
echo "Hello, World! Hello, Linux! Hello, China!"
ls -l
```

在终端执行 gedit myscript.sh 命令,编写上述脚本,并保存脚本,具体操作如图 5.3 所示。

```
[root@localhost ~]#gedit myscript.sh
```

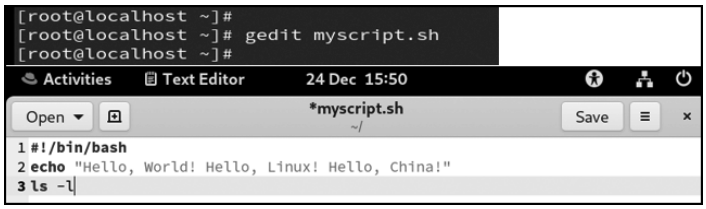


图 5.3 编写脚本

### 2. 添加脚本执行权限

保存文件后,需确保文件有执行权限,可以使用 chmod 命令添加执行权限,具体操作如图 5.4 所示。

```
[root@localhost ~]#chmod a+x myscript.sh
```


图 5.4 添加脚本执行权限

### 3. 执行 Shell 脚本

1）在终端中执行

使用终端进入脚本所在的目录，并执行脚本 myscript.sh。如果脚本不在当前目录，可以使用绝对路径或相对路径执行。具体操作如图 5.5 所示。

```
[root@localhost ~]#
[root@localhost ~]# ./myscript.sh
Hello, World! Hello, Linux! Hello, China!
total 32
drwxr-xr-x. 9 root root 146 Nov 16 21:49 Desktop
drwxr-xr-x. 2 root root   6 Dec 10 10:02 f1
drwxr-xr-x. 2 root root   6 Dec 10 10:02 f2
-rwsr--r--. 1 root root  61 Dec 10 10:27 file
-rw-r-sr--. 1 root root  24 Dec 10 10:05 file1
-rw-r--r--. 1 root root  24 Dec 10 10:05 file2
-rwxr-xr-x. 1 root root  67 Dec 24 15:51 myscript.sh
```

图 5.5 执行 Shell 脚本

2）添加脚本路径到环境变量

将脚本所在的目录添加到系统的 PATH 环境变量中，可以在任何地方执行该脚本，具体操作如图 5.6 所示。

```
[root@localhost ~]#export PATH=$PATH:/root
[root@localhost ~]#cd /
[root@localhost /]#myscript.sh
```

```
[root@localhost ~]# export PATH=$PATH:/root
[root@localhost ~]# cd /
[root@localhost /]# myscript.sh
Hello, World! Hello, Linux! Hello, China!
total 28
dr-xr-xr-x.   2 root root    6 Aug 10  2021 afs
lrwxrwxrwx.   1 root root    7 Aug 10  2021 bin -> usr/bin
dr-xr-xr-x.   5 root root 4096 Oct 10 08:13 boot
drwxr-xr-x.  20 root root 3540 Dec 24 14:20 dev
drwxr-xr-x. 134 root root 8192 Dec 24 13:43 etc
drwxr-xr-x.  10 root root  110 Dec 23 09:12 home
```

图 5.6 添加脚本路径到环境变量

其中，在命令 export PATH＝＄PATH:/root 中，/root 目录是 myscript.sh 脚本存放的路径。

## 5.2 Bash Shell

Bash 是一种功能强大的命令行解释器，是操作系统的外壳。Bash 不仅有非常灵活和强大的编程接口，同时又有非常友好的用户界面。它内建 40 个 Shell 命令和 12 个命令行参数。Red Hat Linux 9 中默认使用的 Shell 是 Bash，它为用户提供使用操作系统的接口，承担着用户与操作系统内核之间进行沟通的任务。

在 Linux 中可以直接浏览.bash_history 文件，或使用 history 命令来查看目前的命

令记录,如图 5.7 所示。

```
[root@localhost /]#history | tail -5
```

系统提供的 history 命令可以列出完整的系统在该用户登录时执行过的所有命令,并以命令执行的先后顺序列出记录的号码。如果要查看最近执行的命令,可以使用“history n”命令,其中,n 表示需要查看的最近执行的命令的条数。例如,列出系统最近执行的 5 条命令,如图 5.8 所示。

```
[root@localhost /]#history 5
```

```
[root@localhost /]#
[root@localhost /]# history | tail -5
  823  export PATH=$PATH:/root
  824  cd /
  825  myscript.sh
  826  history
  827  history | tail -5
[root@localhost /]#
```

图 5.7 history

```
[root@localhost /]# history 5
  824  cd /
  825  myscript.sh
  826  history
  827  history | tail -5
  828  history 5
[root@localhost /]#
```

图 5.8 history 前 n 条记录

在命令记录中,每条用户执行过的命令都会被赋予一个记录号码,用户可以利用这些记录号码来执行指定的要执行的旧命令。其语法如下。

```
!<记录号>
```

示例:要执行 824 条记录标记的命令,可以在命令行提示符下执行如下命令,结果如图 5.9 所示。

```
[root@localhost /]#!824
```

## 5.2.1 交互式处理

从用户登录系统开始,Shell 程序就是在系统终端中显示不同的命令行提示符(root 用户登录系统则提示符显示“#”,普通用户登录则显示“$”),然后等待用户输入命令,如图 5.10 所示。在接收来自用户输入的命令后,Bash 会根据命令的不同类型(包括程序或 Shell 内置命令)来执行,在执行完毕后,Bash 将结果回传给用户,并且再次回到命令提示符,以等待用户的下一次输入。这种模式会一直继续下去,直到用户执行 exit 命令或是按 Ctrl+D 组合键来注销,Bash 才会结束,Bash 的这种与用户沟通的方式称为“交互式处理”。

```
[root@localhost /]#
[root@localhost /]# !824
cd /
[root@localhost /]#
```

图 5.9 使用! 执行命令

```
[root@localhost /]# su zgh
[zgh@localhost /]$ su root
Password:
[root@localhost /]#
```

图 5.10 Bash 交互式处理

### 5.2.2 命令补全功能

在Shell终端使用Tab键(按一次或两次),可查询或补全命令,如图5.11所示。

```
[root@localhost /]#
[root@localhost /]# ls
ls          lshw        lslocks    lsmod      lsusb
lsattr      lsiio       lslogins   lsns       lsusb.py
lsblk       lsinitrd    lsmcli     lsof
lscpu       lsipc       lsmd       lspci
lsgpio      lsirq       lsmem      lsscsi
[root@localhost /]#
```

图5.11 Shell的补全功能

### 5.2.3 别名功能

alias命令用于创建、显示或删除命令别名。通过使用别名,用户可以为常用命令创建简短且易记的替代名称。

alias命令的基本用法和一些示例如表5.2所示。

表5.2 alias命令的基本用法和示例

| 命　令 | 解　释 | 示　例 |
|---|---|---|
| alias shortname='full command' | 创建别名,将shortname映射到full command | alias ll='ls -l' |
| alias | 显示当前会话中定义的所有别名 | alias |
| unalias shortname | 删除指定别名 | unalias ll |
| source ~/.bashrc | 永久保存别名 | # ~/.bashrc文件内容<br>alias ll='ls -l'<br>alias cls='clear' |
| source ~/.bash_profile | 永久保存别名 | |

**示例:** 对于熟悉DOS和Windows的用户来说,dir命令可以方便地显示当前目录的内容,但是在Linux中完成该功能的命令是"ls -l"。如果希望使用dir来代替"ls -l",则可以使用alias功能来创建一个到"ls -l"的别名,如图5.12所示。

```
[root@localhost ~]#alias dir='ls -l'
[root@localhost ~]#dir
```

```
[root@localhost ~]# alias dir='ls -l'
[root@localhost ~]# dir
total 32
drwxr-xr-x. 9 root root 146 Nov 16 21:49 Desktop
drwxr-xr-x. 2 root root   6 Dec 10 10:02 f1
drwxr-xr-x. 2 root root   6 Dec 10 10:02 f2
-rwsr--r--. 1 root root  61 Dec 10 10:27 file
-rw-r-sr--. 1 root root  24 Dec 10 10:05 file1
-rw-r--r--. 1 root root  24 Dec 10 10:05 file2
-rwxr-xr-x. 1 root root  67 Dec 24 15:51 myscript.sh
```

图5.12 alias的使用

如果希望查看当前 Linux 系统中使用的别名命令,可以直接输入“alias”命令。如果需要取消特定的别名命令,可以使用 unalias 命令。例如,取消 dir 别名可使用如下命令。

```
[root@localhost ~]#unalias dir
```

要使别名在每次登录时都可用,可以将别名定义添加到 Shell 配置文件中,如~/.bashrc 或~/.bash_profile。例如,在~/.bashrc 文件中添加以下行:

```
alias dir ='ls -l'
```

然后运行以下命令应用更改:

```
source ~/.bashrc
```

### 5.2.4 作业控制

作业控制是指在 UNIX/Linux 操作系统中对在终端中运行的进程(作业)进行管理和控制的一套机制。这包括在前台和后台执行命令,以及对作业的挂起、恢复等操作。

作业控制的常用命令和示例如表 5.3 所示。

表 5.3 作业控制的常用命令和示例

| 命 令 | 示 例 | 解 释 |
|---|---|---|
| & | command & | 将命令放置在后台运行 |
| bg | bg %1 | 将编号为 1 的作业切换到后台运行 |
| fg | fg %1 | 将编号为 1 的作业切换到前台运行 |
| jobs | jobs | 显示当前终端上运行的作业列表 |
| Ctrl+Z | Ctrl+Z | 暂停当前前台作业,并将其放入后台 |

**示例:** 将 sleep 命令放置在后台运行,等待 10s,如图 5.13 所示。

```
[root@localhost ~]#sleep 10 &
```

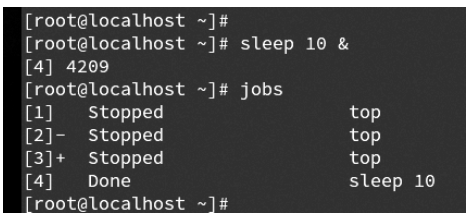


图 5.13 后台运行

当前某个任务在前台运行之后,就无法使用“&”将它投入后台运行,但是可以先使用 Ctrl+Z 组合键暂停该程序,然后在命令提示符下输入“bg”命令,即可将该任务投入后

台执行。如果要查看目前系统中正在运行的后台程序,可以使用 jobs 命令。

### 5.2.5 输入/输出重定向

输入/输出重定向是一种控制命令输入和输出的机制。

在 Linux 系统中,标准输入和输出有以下三种形态。

标准输入(stdin):通常是指键盘。

标准输出(stdout):通常是指将命令执行的结果输出到终端机或屏幕上。

标准错误输出(stderr):是指在命令发生错误时,将其错误信息输出到屏幕上。

重定向(>和<)是改写重定向,就是会删除原来的文件,而重定向(>>和<<)是追加重定向,就是新的内容将被添加到文件原来内容的后面。输入/输出重定向的常用命令以及相关示例和解释如表 5.4 所示。

表 5.4 输入/输出重定向的常用命令

| 命令 | 示 例 | 解 释 |
|---|---|---|
| > | ls > file_list.txt | 输出重定向:将 ls 命令的标准输出重定向到名为 file_list.txt 的文件中,如果文件不存在则创建,存在则覆盖 |
| >> | echo "Additional content" >> file_list.txt | 输出追加:将文本追加到名为 file_list.txt 的文件的末尾,如果文件不存在则创建 |
| < | wc -l < file_list.txt | 输入重定向:将 file_list.txt 文件的内容作为 wc -l 命令的标准输入,统计行数 |

示例:将 ls 命令的标准输出重定向到当前目录下的名为 file_list.txt 的文件中,如果文件不存在则创建,存在则覆盖,如图 5.14 所示。

```
[root@localhost ~]#ls > file_list.txt
[root@localhost ~]#more file_list.txt
```

```
[root@localhost ~]# ls > file_list.txt
[root@localhost ~]# more file_list.txt
Desktop
f1
f2
file
file1
file2
file_list.txt
ls_res
myscript.sh
```

图 5.14 重定向到 ls_res

### 5.2.6 管道

管道(|)是用于将一个命令的输出传递给另一个命令作为输入。通过使用管道,可以将多个命令串联起来,实现复杂的数据处理任务。

基本语法:

```
command1 | command2
```

示例：使用 cat 命令查看 file_list.txt 的内容输出，然后通过管道 | 将这些内容传递给 head -2 命令。具体操作如图 5.15 所示。

```
[root@localhost ~]#cat file_list.txt | head -2
```

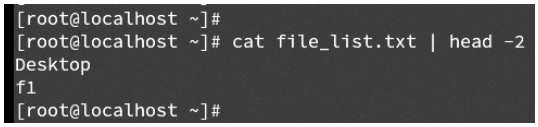


图 5.15 管道

## 5.2.7 Bash 中的特殊字符

在 Linux 下有一些符号会被 Shell 特殊对待，这些符号可以用来指定特殊的范围或功能，除了前面介绍的以外，如>、>>、<、<<、|和!，还有以下可以在 Shell 中使用的特殊字符。

### 1. 通配符

通配符是一种用于匹配文件名或路径的字符模式，通常用于命令行中执行操作或查询文件系统。在 UNIX/Linux 系统中，通配符可以帮助用户快速匹配一组文件或目录，具体如表 5.5 所示。

表 5.5 通配符

| 通配符 | 含　义 | 示　例 | 匹配的文件名示例 |
|---|---|---|---|
| * | 匹配任意长度的任意字符(包括零个字符) | *.txt | file.txt，document.txt，backup.txt |
| ? | 匹配任意单个字符 | file?.txt | file1.txt，fileA.txt |
| [ ] | 匹配方括号内列举的任意一个字符 | [aeiou] | apple，orange，umbrella |
| [ - ] | 匹配指定范围内的字符 | [0-9] | file0.txt，file9.txt |
| { } | 创建一个模式集，匹配其中任意一个模式 | {*.txt，*.md} | document.txt，report.md，note.txt |

“*”和“?”是 Linux 系统中最常用的两个通配符，在查找字符串的时候，通配符可以代替任意的字符。其中，“?”可以代替任意一个字符，“*”可以代替任意多个字符。

示例：执行“find /etc/ini*”命令就会列出/etc 目录下所有以 ini 开头的文件名，如图 5.16 所示。

```
[root@localhost ~]#find /etc/ini*
```

```
[root@localhost ~]#
[root@localhost ~]# find /etc/ini*
/etc/inittab
[root@localhost ~]#
```

图 5.16 通配符“*”

示例：使用通配符“?”查找当前目录的文件，如图 5.17 所示。

```
[root@localhost ~]# ls
Desktop  file   file_list.txt  passwd_inf          shadow_info
f1       file1  ls_res         password_file.txt
f2       file2  myscript.sh    shadow_inf
[root@localhost ~]# find fi??
file
[root@localhost ~]# find fi???
file1
file2
[root@localhost ~]#
```

图 5.17 通配符“?”

## 2. 命令取代符

命令取代符是一种用于嵌套执行命令并将其输出插入其他命令或上下文中的机制，这个机制也被称为命令替换。命令取代符的基本语法、用途及示例如表 5.6 所示。

表 5.6 命令取代符

| 基本语法 | 示例 | 用途 |
| --- | --- | --- |
| result=$(command) | current_date=$(date) | 获取命令的输出并将其赋值给变量 |
| result=command`` | current_date=date`` | 与 $(command) 相同的语法形式 |
| "$(command)" | "Today is $(date)" | 在字符串中嵌套执行命令并插入其输出 |
| $(command1 $(command2)) | result=$(ls; wc -l) | 执行多个命令并将最后一个命令的结果赋值给变量 |
| $(command) | file_count=$(ls; wc -l) | 执行命令并获取结果 |
| result=$(echo "Hello $(whoami)") | Hello $(whoami) | 在命令中包含复杂的嵌套结构 |

命令取代符“`”在 Esc 键下方，与“~”符号在同一个键上。两个“`”符号包围的命令，是该命令行中首先被执行的命令。

示例：“echo `date`”命令，首先执行 date 命令，然后使用 echo 来显示 date 命令的结果，而不是显示字符串 date，如图 5.18 所示。

```
[root@localhost ~]#
[root@localhost ~]# echo `date`
Sun 24 Dec 23:15:45 CST 2023
[root@localhost ~]# date
Sun 24 Dec 23:15:47 CST 2023
[root@localhost ~]#
```

图 5.18 命令取代符“`”

```
[root@localhost ~]#echo `date`
```