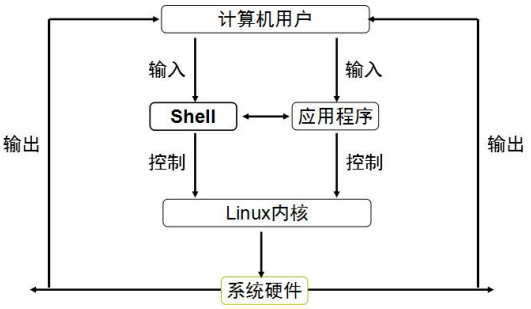
第九天　shell 编程

一．shell 的作用和历史
二．shell 的常用功能
三．shell 变量种类
四．shell 脚本的运行
五．编写简单的 shell 脚本

一．Shell 的作用及常见种类
Shell 的作用 -- 命令解释器，“翻译官”



vim　/etc/shells

二．Bash 的常用功能

1. Tab 键自动补齐

2.Bash 的命令历史
命令历史
　　保存用户曾经执行过的命令操作
　　存放位置：~/.bash_history 文件
查看历史命令
　　使用↑、↓按键逐条翻看，允许编辑并重复执行
　　执行：history
清除历史命令
　　执行：history -c

3.调用历史命令
!n：执行历史记录中的第 n 条命令
!str：执行历史记录中以“str”开头的命令
设置记录历史命令的条数
修改 HISTSIZE 参数（默认为 1000 条）

4.命令别名

为使用频率较高的复杂命令行设置简短的调用名称

存放位置：~/.bashrc

查看命令别名

格式：alias [别名]

设置命令别名

执行：alias 别名='实际执行的命令'

取消已设置的命令别名

格式：unalias 别名

5.Bash 的标准输入输出

交互式硬件设备

标准输入：从该设备接收用户输入的数据

标准输出：通过该设备向用户输出数据

标准错误：通过该设备报告执行出错信息

类型	设备文件	文件描述编号	默认设备
标准输入	/dev/stdin	0	键盘
标准输出	/dev/stdout	1	显示器
标准错误输出	/dev/stderr	2	显示器

6.Bash 的重定向操作

改变标准输入、标准输出、标准错误的方向

类型	操作符	用途
重定向标准输入	<	将命令中接收输入的途径由默认的键盘更改为指定的文件
重定向标准输出	>	将命令的执行结果输出到指定的文件中，而不是直接显示在屏幕上
	>>	将命令执行的结果追加输出到指定文件
重定向标准错误	2>	清空指定文件的内容，并将标准错误信息保存到该文件中
	2>>	将标准错误信息追加输出到指定的文件中
重定向标准输出和标准错误	&>	将标准输出、标准错误的内容全部保存到指定的文件中，而不是直接显示在屏幕上
	2>&1	

7.Bash 的管道操作

管道操作符号“|”
连接左右两个命令，将左侧的命令输出的结果，作为右侧命令的输入（处理对象）
格式：cmd1 | cmd2 [... | cmdn]

三．Shell 变量的应用

1.Shell 变量含义
为灵活管理 Linux 系统提供特定参数，有两层意思：
1）变量名：使用固定的名称，由系统预设或用户定义
2）变量值：能够根据用户设置、系统环境变化而变化

2.Shell 变量的种类
用户自定义变量：由用户自己定义、修改和使用
环境变量：由系统维护，用于设置用户的 Shell 工作环境，只有极少数的变量用户可以修改
预定义变量：Bash 预定义的特殊变量，不能直接修改
位置变量：通过命令行给程序传递执行参数

3.变量的赋值与引用
定义新的变量
变量名要以英文字母或下划线开头，区分大小写
格式：变量名=变量值
查看变量的值
格式：echo $变量名
查看所有变量：set
清除变量
unset 变量名

[root@localhost ~]# DAY=Sunday
[root@localhost ~]# echo $DAY
Sunday

在查看变量时，如果变量名容易和后边的字符串连在一起导致混淆，则应该使用大括号将变量名括起来，使用形式为：${变量名} ，例如：
若已知变量 Var 的值为 lamp，则执行“echo $Var3.0”命令后将显示结果“.0”而不是“lamp3.0”，因为在该命令中，会将“Var3”当成变量名（默认未定义此变量）。若希望正确显示“lamp3.0”的输出结果，则需要执行“echo ${Var}3.0”

从键盘输入内容为变量赋值
格式： read [-p "信息"] 变量名
结合不同的引号为变量赋值

双引号 “ ” ：允许通过$符号引用其他变量值
单引号 ‘ ’ ：禁止引用其他变量值，$视为普通字符
反撇号 `` ：将命令执行的结果输出给变量

4.数值变量的运算
计算整数表达式的运算结果
格式：expr 变量 1 运算符 变量 2 ...[运算符 变量 n]
expr 的常用运算符
加法运算：+
减法运算： -
乘法运算： *
除法运算： /
求模（取余）运算： %

Bash 程序并不适合进行强大的数学运算，例如小数或指数运算的，一般只能进行简单的整数运算
对 Shell 变量进行数值运算时，更多的时候是用于脚本程序的过程控制，如控制程序的循环次数
在 expr 命令的使用格式中，变量与运算符间是有空格的，可以同时使用多个运算符、多个变量
由于星号“*”作为 Bash 环境中的通配符使用，因此乘法运算符需要使用“*”的特殊形式（转义字符）

```
#!/bin/bash
read -p "please input num1:" -t 30 test1
read -p "input num2:" -t 30 test2
declare -i sum=”$test1+$test2”
echo “num1 + num2 = $sum”
```

5.环境变量赋值
设置变量的作用范围
格式：export 变量名...
export 变量名=变量值 [...变量名 n=变量值 n]
查看环境变量
env 或 export
清除用户定义的变量
格式：unset 变量名

对于用户自行定义的变量，默认只能在当前的 Shell 环境中使用，因此称为局部变量
局部变量在新开启的子 Shell 环境中是无效的（无法引用定义的变量），因此需要使用 export 命令将变量输出为全局变量
只有对于全局变量，在当前 Shell 的子 Shell 环境（例如 zsh）中，才能够被正确引用
变量不存在或者值为空时，通过 echo 命令查看时将显示一个空行

6.系统环境变量
环境变量配置文件
全局配置文件：/etc/profile
/etc/bashrc
用户配置文件：~/.bash_profile
~/.bashrc

有完整登陆流程时，加载环境变量顺序
先读/etc/profile
再读~/.bash_profile
再读~/.bashrc
再读/etc/bashrc
开始 Bash 界面

用 set 查看环境变量
[root@localhost root]# set
SHELL=/bin/bash
TERM=xterm
UID=0
USER=root
consoletype=pty

环境变量
常见的环境变量：
$USER 、$LOGNAME
$UID 、 $SHELL 、$HOME
$PWD、 $PATH
$PS1、$PS2

查看环境变量
[root@localhost ~]# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@localhost ~]# PATH="/opt/bin:$PATH"
[root@localhost ~]# echo $PATH
/opt/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

命令执行时查找顺序
1、以相对/绝对路径执行

2、由 alias 找到的执行
3、bash 内部命令执行
4、按$PATH 路径执行

环境变量 PS1
echo $PS1
\d 日期 \t 时间（24） \T 时间（12）
\H 完整主机名 \h 简写主机名
\u 用户名 \v bash 版本
\w 完整目录 \W 最后一个目录
\# 执行了第几个命令 \$ 提示符
PS1=‘[\u@\h \W \t #\#]\$’

位置变量

表示为 $n，n 为 1~9 之间的数字
$0 为第 0 个参数，脚本本身

[root@localhost ~]# ./exam01.sh one two three four five six

预定义变量

表示形式如下
$#：命令行中位置参数的个数
$*：所有位置参数的内容
$?：上一条命令执行后返回的状态，当返回状态值为 0 时表示执行正常，非 0 值表示执行异常或出错
$$：当前所在进程的进程号
$!：后台运行的最后一个进程号
$0：当前执行的进程/程序名

[root@localhost ~]# bash
[root@localhost ~]# echo $0 $$
bash 5887
[root@localhost ~]# exxit
bash: exxit: command not found
[root@localhost ~]# echo $?
127
[root@localhost ~]# exit
exit
[root@localhost ~]# echo $?

0

输出位置参数变量，脚本后要接参数
#!/bin/bash

echo "the command is $0"
echo "canshu1 is $1"
echo "canshu2 is $2"

输出预定义变量
#!/bin/bash

echo "canshu zongshu $#"
echo "canshu libiao: $*"
echo $?

多命令运行
; 命令顺序执行。
&& 前后命令的执行存在逻辑与关系，只有&&前面的命令执行成功后，它后面的命令才被执行。
|| 前后命令的执行存在逻辑或关系，只有||前面的命令执行失败后，它后面的命令才被执行。

通配符与特殊符号
通配符
* 任意多个
? 任意一个
[] 括号内任一个 [^0-9]非数字
特殊符号
\ 转义符
& 后台
！ 非

四．Shell 脚本的概念

Shell 脚本
1.用途：完成特定的、较复杂的系统管理任务
2.格式：集中保存多条 Linux 命令，普通文本文件
3.执行方式：按照预设的顺序依次解释执行

编写可执行的 Shell 脚本

建立包含执行语句的脚本文件

#脚本文件中包括的内容

运行环境设置：#!/bin/bash

注释信息：以#开始的说明性文字

可执行的 Linux 命令行

为脚本文件添加可执行权限

在脚本文件的各组成部分中，只有可执行语句是必不可少的（否则就不叫程序了）

当缺省运行环境设置时，会自动由当前加载该脚本的 Shell 解释器负责解释执行

Shell 脚本文件的扩展名并无严格的约束，不一定非得是“.sh”结尾的

给 shell 脚本增加执行权限

```
[root@localhost ~]# vi repboot.sh
#!/bin/bash
# To show usage of /boot directory and mode of kernel file.
echo "Useage of /boot: "
du -sh /boot
echo "The mode of kernel file:"
ls -lh /boot/vmlinuz-*
[root@localhost ~]# chmod a+x repboot.sh
```

运行 Shell 脚本程序

1.直接执行具有“x”权限的脚本文件

例如：./repboot.sh

2.使用指定的解释器程序执行脚本内容

例如：bash repboot.sh

3.通过 source 命令（或 . ）读取脚本内容执行

例如：souce repboot.sh 或 . hello.sh

脚本程序可以通过多种方式运行：

为脚本文件设置了可执行属性后，在 Shell 命令行中可以直接通过脚本文件的路径执行脚本程序

在调试阶段可以使用 Shell 程序直接调用脚本文件，不要求脚本具有可执行权限，格式是：bash 脚本名

使用 Bash 的内部命令“.”脚本文件执行时，将不会开启新的 Shell 环境。使用这种方式时，脚本文件作为“.”命令的参数，因此同样不要求具备可执行权限。

五．Shell 脚本应用示例

示例 1：

每周五 17:30 清理 FTP 服务器的公共共享目录

检查 /var/ftp/pub/ 目录，将其中所有子目录及文件的详细列表、当时的时间信息追加保存到 /var/log/pubdir.log 日志文件中，然后清空该目录

```
[root@localhost ~]# vi /opt/ftpclean.sh
#!/bin/bash
date  >>  /var/log/pubdir.log
ls  -lhR  /var/ftp/pub  >>  /var/log/pubdir.log
rm  -rf  /var/ftp/pub/*

[root@localhost ~]# crontab -e
30 17 * * 5  /opt/ftpclean.sh
```

Shell 脚本应用示例

示例 2:
每隔 3 天对数据库目录做一次完整备份
统计 /usr/local/mysql/var 目录占用的空间大小、查看当前的日期，并记录到临时文件 /tmp/dbinfo.txt 中
将 /tmp/dbinfo.txt 文件、/usr/local/mysql/var 目录进行压缩归档，备份到/opt/dbbak/目录中
备份后的包文件名中要包含当天的日期信息
最后删除临时文件/tmp/dbinfo.txt

```
[root@localhost ~]# vi /opt/dbbak.sh
#!/bin/bash
DAY=`date +%Y%m%d`
SIZE=`du -sh /usr/local/mysql/var`
echo "Date: $DAY" >> /tmp/dbinfo.txt
echo "Data Size: $SIZE" >> /tmp/dbinfo.txt
mkdir  /opt/dbbak
cd /opt/dbbak
tar -zcPf mysqlbak-${DAY}.tar.gz /usr/local/mysql/var /tmp/dbinfo.txt
rm -f /tmp/dbinfo.txt
```

```
[root@localhost ~]# crontab -e
55 23 */3 * *    /opt/dbbak.sh
```