

# 第四章 Linux shell程序设计

---

# Shell简介



## ✓ 功能

- ✓ 命令解释器

- ✓ 程序设计语言

它有变量、关键字、各种控制语句和自己的语法结构

易于编写、修改和调试

解释性语言，无需编译

# 主要内容



- ✓ shell概述
- ✓ 命令历史
- ✓ 名称补全
- ✓ 别名
- ✓ shell特殊字符
- ✓ shell变量
- ✓ 算术运算
- ✓ 控制结构
- ✓ 函数

# shell概述

---

shell的种类

shell脚本的建立和执行

# shell的种类



## ✓ Bourne shell (sh)

- ✓ 由Bell实验室的*Steven Bourne*为UNIX开发
- ✓ 是UNIX的默认shell
- ✓ 是其他shell的开发基础

## ✓ C shell (csh)

- ✓ 加州伯克利大学的*Bill Joy*为BSD UNIX开发
- ✓ C shell与Bourne shell不兼容

# shell的种类



## ✓ Korn shell (ksh)

- ✓ Bell实验室的*David Korn*开发的
- ✓ 集合了C shell和Bourne shell的优点，与sh完全兼容

## ✓ Bourne Again shell (bash)

- ✓ 自由软件基金会为GNU计划开发的shell
- ✓ Linux系统中默认的shell
- ✓ 与sh完全兼容

# shell脚本的建立和执行

## ✓ shell脚本

✓ 由shell执行的命令序列构成

✓ 例1

```
echo "my working directory is:"  
pwd  
echo "today is:"  
date  
~
```

# shell脚本的建立和执行



## ✓ shell脚本的执行

- ✓ 以脚本名作为bash参数

形式一： `bash 脚本名 [参数]`

形式二： `./脚本名 [参数]`



# shell脚本的建立和执行

## ✓ shell脚本的执行

- ✓ 将shell脚本的权限设置为可执行，直接执行

形式一：

```
chmod a+x ex1  
PATH = $PASH :.  
ex1
```

形式二：

```
chmod a+x ex1  
./ex1
```

# shell脚本的建立和执行

## ✓ 例2：带控制结构的shell程序

```
#!/bin/bash
# If no arguments, then listing the current directory.
# Otherwise, listing each subdirectory.

if test $# = 0
then ls .
else
    for i
    do
        ls -l $i | grep '^d'
    done
fi
~
```

# 命令历史



## ✓ 功能

- ✓ bash 可以记录输入过的命令

## ✓ 说明

- ✓ 可以使用 “↑” 键, “↓” 键重复输入过的命令
- ✓ 环境变量

**HISTFILE**: 系统默认将命令历史记录保存在用户主目录下面的  
“.bash\_history” 文件中

**HISTSIZE**: 历史文件中能够保留的命令个数, 默认值为1000

# 显示历史命令

---

## ✓ history命令

### ✓ 功能

列出以前输入的命令

### ✓ 语法

history [n]

### ✓ 例子

history //显示所有命令

history 5 //显示最后 5 个命令

# 执行历史命令

---

## ✓ 功能

✓ 是一种命令替换，它以字符 “!” 开头

## ✓ 例子

!132 //执行第132条历史命令

!-1 //重复执行上一条命令

!! //重复执行上一条命令

# 名称补全

---

## ✓ 功能

- ✓ 指当键入的字符足以确定目录中一个唯一的文件时，只须按Tab键就可以自动补齐该文件名的剩下部分

## ✓ 例子

hist Tab

# 别名

## ✓ 定义别名

### ✓ 格式

alias name=value

### ✓ 例如

alias

//显示别名清单

alias ll

//显示别名ll的内容

alias h= 'history 5'

//设置别名

### ✓ 说明

定义别名时，在命令中包含空格或特殊字符时需要引号，防止产生歧义

# 别名

---

## ✓ 取消别名

### ✓ 格式

`unalias name`

### ✓ 例子

`unalias h`

### ✓ 说明

`unalias`命令可以在一个命令中同时取消多个别名的定义



# shell特殊字符



- ✓ 通配符
- ✓ 引号
- ✓ 输入/输出重定向
- ✓ 注释
- ✓ 管道操作符
- ✓ 后台命令
- ✓ 命令操作符

# 通配符



## ✓ 星号 (\*)

### ✓ 匹配任意字符

try\*c      //匹配try1.c try.c try.basic

## ✓ 问号 (?)

### ✓ 匹配任意一个字符

p?.c      //匹配p1.c p2.c pa.c

# 通配符

## ✓ 方括号 ([ ])

- ✓ 匹配括号内所限定的任何一个字符

[Mm]akefile      //匹配Makefile makefile

file[1-5].c      //匹配file1.c file2.c file3.c file4.c file5.c

file[1-20]

## ✓ 叹号 (!)

- ✓ 如果在方括号内，表示不匹配括号内所限定的任何单个字符

file[!23].c      //不匹配文件：file2、file3

file[!2-4]      //不匹配文件：file2、file3、file4

# 引号

## ✓ 双引号 (")

- ✓ 消除元字符的特殊含义

## ✓ 单引号 (')

- ✓ 消除元字符的特殊含义

## ✓ 倒引号 (`)

- ✓ 命令替换

## ✓ 元字符

- ✓ 指的是在Shell中有特殊含义的字符

< > | ; ! ? \* [ ] \$ \ " ' ` ( ) { } ^

# 双引号



## ✓ 功能

- ✓ 除 \$ ` " 三个元字符外，均作为普通字符
- ✓ 保留空白字符（空格、制表符、换行符）

# 双引号

## ✓ 例3

```
echo "curent directory is `pwd`"  
echo "home directory is $HOME"  
echo "file *.*"  
echo "directory '$HOME'"  
echo "filename is No\$\**"
```

```
[wuhua@localhost ~]$ ./ex3.sh  
curent directory is /home/wuhua  
home directory is /home/wuhua  
file *.*  
directory '/home/wuhua'  
filename is No$\**
```

# 单引号



## ✓ 功能

- ✓ 除单引号自身以外，其他元字符都作为普通字符

# 单引号

## ✓ 例4

```
echo `curent directory is `pwd`  
echo 'home directory is $HOME'  
echo 'file *.*'  
echo 'directory '$HOME'  
echo 'filename is No\$\*'
```

```
[wuhua@localhost ~]$ ./ex4.sh  
curent directory is `pwd`  
home directory is $HOME  
file *.*  
directory /home/wuhua  
filename is No\$\*
```



# 倒引号



## ✓ 功能

- ✓ 将一个命令的输出作为另外一个命令的参数

## ✓ 格式

- ✓ `command1 `command2``

## ✓ 例子

`echo "current directory is `pwd`"`

# 输入/输出重定向

---

## ✓ 输出重定向

✓ 把命令的标准输出重定向到指定文件中

✓ 格式

命令 > 文件名

命令 >> 文件名

✓ 说明

输出附加定向符 (>>) 的作用是把命令/程序的输出附加到指定文件的末尾

✓ 例子

who > userlog

date>>userlog

# 输入/输出重定向

---

## ✓ 输入重定向

✓ 让命令从指定文件中取得输入数据

✓ 格式

命令 < 文件名

✓ 例子

`bash < ex1` //从脚本文件ex1中取得命令

`wc < fileinfo` //将fileinfo作为wc命令的输入

# 输入/输出重定向

---

## ✓ 输入重定向

### ✓ 说明

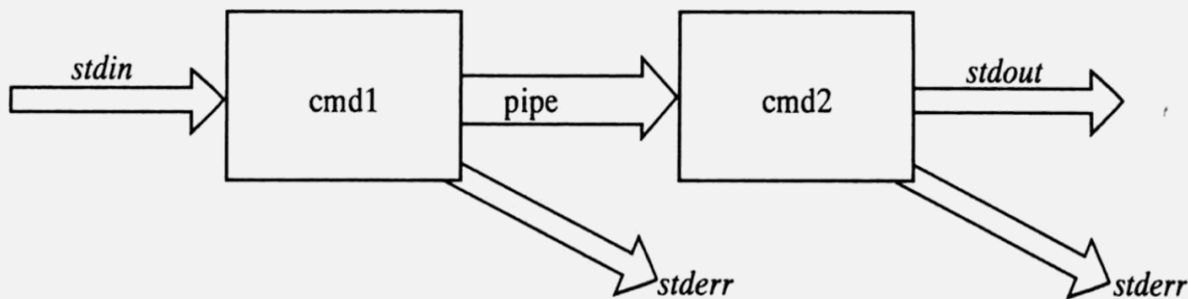
- ✓ here文档：输入重定向操作符 “<<” 后面的输入用一个分隔符开始，用同样的另一个分隔符结束。两个分隔符之间的内容就是要执行的命令的输入

# 管道

## ✓ 管道操作符 (|)

- ✓ 通过管道把一个命令的输出传递给另一个命令作为输入
- ✓ 格式

`command1 | command2 | ... | commandn`



# 管道



## ✓ 管道操作符 (|)

### ✓ 例子

ls -l | more

ls -l | grep '^d'

ps | sort > passort

# 后台命令

---

## ✓ 后台命令符 (&)

✓ 将当前命令送到后台执行

✓ 例子

```
find / -name file2.txt &
```

```
find / -name file > /root/testfile &
```

```
gcc m1.c m2.c -o prog&
```

# 命令执行操作符

---

## ✓ 操作符

- ✓ 顺序执行 ;
- ✓ 逻辑与 &&
- ✓ 逻辑或 ||



# 顺序执行



## ✓ 格式

✓ 命令1; 命令2; ...; 命令n


## ✓ 功能

✓ 各条命令从左到右依次执行

## ✓ 例子

```
pwd ; who | wc -l ; cd /home/bin
```

# 逻辑与



## ✓ 格式

✓ 命令1 && 命令2

## ✓ 功能

✓ 先执行命令1，如果成功，才执行命令2；否则不执行命令2

## ✓ 例子

`cp ex1 ex10 && rm ex1`

# 逻辑或



## ✓ 格式

✓ 命令1 || 命令2

## ✓ 功能

✓ 先执行命令1，如果不成功，则执行命令2；否则，不执行命令2

## ✓ 例子

```
cat abc || pwd
```

# 成组命令

---

## ✓ 格式

✓ ( 命令序列 )

✓ { 命令序列 }

## ✓ 功能

✓ 配对括号之间的所有命令作为一个整体执行

## ✓ 例子

```
date;who>tmp1
```

```
(date;who)>tmp2
```

```
{ date;who;}>tmp
```

# shell变量



## ✓ shell支持两种类型的变量

- ✓ 环境变量：由系统定义，用来保留系统的内容
- ✓ 局部变量：由用户定义、修改或删除

## ✓ 说明

- ✓ shell变量没有存储类和类型的限制
- ✓ 使用时可以“边定义，边使用”

# 环境变量



## ✓ 功能

- ✓ 系统会自动设置一些变量，这些变量的值决定了用户的工作环境和外观，因此被称为环境变量

## ✓ 说明

- ✓ 习惯上，环境变量的变量名用大写字母表示
- ✓ 环境变量一般在用户登陆时由系统设置，也可以由用户修改。当用户注销后，环境变量会随之复原

# 环境变量

## ✓ 常用的环境变量

| 名称       | 用途              |
|----------|-----------------|
| HOME     | 用户主目录的路径名       |
| LOGNAME  | 用户的登录名          |
| PATH     | 命令搜索路径，以冒号作为分隔符 |
| PS1      | 主命令提示符          |
| PWD      | 当前的工作路径         |
| SHELL    | shell的路径名       |
| TERM     | 当前登陆的终端类型       |
| HISTFILE | 命令历史文件          |
| HISTSIZE | 命令历史最多可包含的命令条数  |

# 环境变量



✓ 查看环境变量


✓ 例子

```
echo $HOME
```

```
echo $PATH
```



# 环境变量



✓ 修改环境变量

✓ 例子

HISTSIZE=500

PS1=here:

# 环境变量

## ✓ 修改环境变量

✓ 例子：修改命令提示符

| 转义字符             | 含义                        |
|------------------|---------------------------|
| <code>\\$</code> | 显示\$符号，如果当前用户是root，则显示#符号 |
| <code>\u</code>  | 显示当前用户的用户名                |
| <code>\h</code>  | 显示运行该shell的计算机主机名         |
| <code>\W</code>  | 显示当前工作目录基准名               |
| <code>\w</code>  | 显示当前工作目录                  |
| <code>\d</code>  | 显示当前日期                    |

# 用户定义变量

---

## ✓ 命名规则

- ✓ 以字母或下划线开头
- ✓ 由字母、数字、下划线组成
- ✓ 变量名区分大小写

# 用户定义变量

## ✓ 变量赋值

### ✓ 格式

变量名 = 字符串                      //等号两边不能有空格

### ✓ 例子

a=1

b=`date`                      //将命令的结果作为变量值

c=\$a                      //将一个变量的值赋给另一个变量

# 用户定义变量

---

## ✓ 引用变量值

- ✓ 变量名前面加\$符号

- ✓ 例子

echo \$a \$b

# 用户定义变量

---

✓ 查看所有变量--set命令

✓ 例子

set

✓ 清除变量--unset命令

✓ 例子

unset a

# 用户定义变量

---

## ✓ 说明

- ✓ shell默认将任何赋给变量的值都解释为字符串
- ✓ 如果值中包含空格，赋值时必须用引号括起来

例子

```
x="hello world"
```

# 位置变量

## ✓ 功能

✓ 一种特殊的变量，用于存放命令行的参数

✓ 例子

```
cat file1 file2 file3 file4 ...
```

```
$0 $1 $2 $3 $4 ....
```

| 位置变量            | 含义               |
|-----------------|------------------|
| \$0             | 表示命令名或脚本文件名      |
| \$1 \$2 ... \$9 | 命令行参数            |
| \$#             | 命令行参数的个数（不包含命令名） |
| \$*             | 将所有参数看成是一个整体     |
| \$@             | "\$*" 的一种变体      |



# 位置变量

## ✓ 例2：带控制结构的shell程序

```
#!/bin/bash
# If no arguments, then listing the current directory.
# Otherwise, listing each subdirectory.

if test $# = 0
then ls .
else
    for i
    do
        ls -l $i | grep '^d'
    done
fi
~
```

# 位置变量

✓ 例5:

```
#!/bin/bash
echo "I was called with $# parameters"
echo "My name is $0"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are $*"
~
```

✓ 练习:

编写一个脚本，将当前目录下的某个文件（以参数的形式）复制到用户主目录（\$HOME）下

# 位置变量

## ✓ shift命令--移动位置参数

✓ 当脚本的参数多于9个时，我们用shift命令来使用多于9个的参数

✓ 格式

shift [n]                      n表示移动的位数

✓ 说明

若不指定n的值，则系统默认n为1

| 命令      | cat | f1         | f2         | f3         | f4  | f5  | f6  |
|---------|-----|------------|------------|------------|-----|-----|-----|
| 原位置参数   | \$0 | <b>\$1</b> | \$2        | \$3        | \$4 | \$5 | \$6 |
| shift   | \$0 |            | <b>\$1</b> | \$2        | \$3 | \$4 | \$5 |
| shift 2 | \$0 |            |            | <b>\$1</b> | \$2 | \$3 | \$4 |

# 位置变量

✓ 例6:

```
echo $0: $1 $2 $3 $4  
shift  
echo $0: $1 $2 $3 $4  
shift 2  
echo $0: $1 $2 $3 $4
```

# 输入/输出命令

## ✓ echo命令

### ✓ 功能

在屏幕上显示字符串或变量的值

### ✓ 选项

-n            不自动换行

-e            解释转义字符

| 转义字符 | 含义   |
|------|------|
| \a   | 响铃   |
| \b   | 回退   |
| \c   | 禁止换行 |
| \n   | 回车换行 |

# 输入/输出命令

---

## ✓ read命令

### ✓ 功能

从键盘读入数据赋值给变量

### ✓ 格式

read 变量1 变量2 ...

### ✓ 说明

变量个数与给定数据个数相同，则依次对应赋值

变量个数少于数据个数，则从左至右对应赋值，最后一个变量被赋予剩余的所有数据

# 输入/输出命令

✓ 例8:

```
echo  
echo -e "enter name:\c"  
read name  
echo "your name is $name"  
~
```

✓ 练习:

编写一个脚本，将当前目录下的某个文件（用read命令读入文件名）复制到用户主目录（\$HOME）下

# export命令

---

## ✓ 功能

- ✓ 用户所创建的shell变量，默认为局部变量。export命令可以扩展变量的使用环境

## ✓ 格式

- ✓ export 变量名

## ✓ 例子

x=1

export x



# export命令

---

## ✓ 说明

✓ 在一个进程内部，同名局部变量的值优先使用

✓ 例9

```
string="Linux"  
echo "the result is $string"
```

# 算术运算



## ✓ expr命令

### ✓ 功能

处理整数运算

### ✓ 格式

expr 算术表达式

### ✓ 说明

可以使用的操作符有+、-、\*、/（取整）、%（取余）等

# 算术运算

## ✓ expr命令

### ✓ 例子

expr 5 + 1

expr 5 - 1

expr 5 \\* 2

expr 5 / 2

### ✓ 说明

表达式的元素之间必须有空格

字符\*（乘）在shell中有特殊含义，因此前面必须有转义字符“\”

# 算术运算

---

## ✓ expr命令

### ✓ 说明

当有变量参与运算时，需要在变量名前面加 “\$”

### ✓ 例子

```
x=10
```

```
expr $x + 1
```

```
y=`expr $x + 1`
```

```
z=$(expr $x + 1)
```

# 算术运算



## ✓ let命令

### ✓ 功能

处理整数算术运算

### ✓ 格式

let 算数表达式

((算术表达式))

### ✓ 说明

可以使用的操作符有+、-、\*、/(取整)、% (取余) 等

# 算术运算



## ✓ let命令

### ✓ 例子

```
let x=9*9
```

### ✓ 说明

表达式的操作符两侧没有空格

对于字符\*（乘）不需要转义字符 “\”

# 算术运算

---

## ✓ let命令

### ✓ 例子

`let x=x+1`

`((x=x+1))`

### ✓ 说明

当有变量参与运算时，不需要在变量名前面加 “\$”

# 条件测试



## ✓ test命令

### ✓ 功能

用来判断表达式的值为真还是为假

### ✓ 格式

`test` 测试条件

`[ 测试条件 ]`                      //方括号内需要有空格

`[[ 测试条件 ]]`                      //方括号内需要有空格



# 条件测试



## ✓ test命令

### ✓ 测试类型

文件属性的判断

字符串的比较

整数的比较

逻辑关系运算

# 条件测试

## ✓ 文件属性的测试

| 操作符 | 功能            |
|-----|---------------|
| -r  | 文件是否存在，并且可读   |
| -w  | 文件是否存在，并且可写   |
| -x  | 文件是否存在，并且可执行  |
| -s  | 文件是否存在，并且长度非0 |

# 条件测试

## ✓ 文件属性的测试

| 操作符 | 功能               |
|-----|------------------|
| -f  | 文件是否存在，并且是普通文件   |
| -d  | 文件是否存在，并且是目录文件   |
| -b  | 文件是否存在，并且是块设备文件  |
| -c  | 文件是否存在，并且是字符设备文件 |
| -L  | 文件是否存在，并且是符号链接文件 |

# 条件测试

---

## ✓ 文件属性的测试

### ✓ 例子

test -f file1

[ -f file1]

test -d dir1

[ -d dir1 ]

# 条件测试

- ✓ 例10：判断指定的一个文件是否是普通文件

```
#!/bin/bash
if test -f "$1"
then
    echo "$1 is an ordinary file."
else
    echo "$1 is not an ordinary file."
fi
```

# 条件测试

## ✓ 字符串的测试

| 操作符               | 示例                           | 功能                  |
|-------------------|------------------------------|---------------------|
| <code>-z</code>   | <code>-z str1</code>         | 字符串str1长度为0，测试结果为真  |
| <code>-n</code>   | <code>-n str1</code>         | 字符串str1长度大于0，测试结果为真 |
| <code>==</code>   | <code>str1 = str2</code>     | 字符串相等               |
| <code>!=</code>   | <code>str1 != str2</code>    | 字符串不相等              |
| <code>&gt;</code> | <code>str1 \&gt; str2</code> | 按照字典顺序str1在str2后    |
| <code>&lt;</code> | <code>str1 \&lt; str2</code> | 按照字典顺序str1在str2前    |

# 条件测试

---

## ✓ 字符串的测试

### ✓ 例子

`test -n $str2`

`[ -n $str2 ]`

`[ $str1 = $str2 ]`

//等号两边加空格

`[ $str1 \< $str2 ]`

# 条件测试

## ✓ 整数的测试

| 操作符              | 示例                         | 功能               |
|------------------|----------------------------|------------------|
| <code>-eq</code> | <code>num1 -eq num2</code> | num1等于num2, 测试为真 |
| <code>-ne</code> | <code>num1 -ne num2</code> | num1不等于num2      |
| <code>-gt</code> | <code>num1 -gt num2</code> | num1大于num2       |
| <code>-ge</code> | <code>num1 -ge num2</code> | num1大于等于num2     |
| <code>-lt</code> | <code>num1 -lt num2</code> | num1小于num2       |
| <code>-le</code> | <code>num1 -le num2</code> | num1小于等于num2     |



# 条件测试

---

## ✓ 整数的测试

### ✓ 例子

[ "\$x" -lt 1 ]

[ "\$y" -gt 10 ]

# 条件测试

## ✓ 整数的测试

### ✓ 说明

如果使用 “<、>、<=、>=”，则需要加双括号

### ✓ 例11

```
#!/bin/bash
if (( $1 > 0 ))
then
    echo "$1 is positive"
else
    echo "$1 is not"
fi
```

# 条件测试

## ✓ 逻辑运算符

| 操作符           | 示例                  | 功能                |
|---------------|---------------------|-------------------|
| <b>!</b> (非)  | <b>!</b> exp        | 如果表达式为假，则测试为真     |
| <b>-a</b> (与) | exp1 <b>-a</b> exp2 | 两个表达式都为真，则测试为真    |
| <b>-o</b> (或) | exp1 <b>-o</b> exp2 | 两个表达式中有一个为真，则测试为真 |

# 条件测试

---

## ✓ 逻辑运算符

### ✓ 例子

[ ! "\$x" -lt 5 ]

[ "\$x" -lt 1 -o "\$x" -gt 10 ]

[[ "\$x" -lt 1 || "\$x" -gt 10 ]]

# 控制结构



- ✓ 顺序结构
- ✓ 分支结构
  - ✓ if then else
  - ✓ case
- ✓ 循环结构
  - ✓ while
  - ✓ until
  - ✓ for

# if语句

## ✓ if - then 结构

```
if (test expression)
then
  commands
  ...
fi
```

## ✓ 说明

条件测试语句两边的括号( )可以省略

可以使用条件测试语句的第二种形式: `if [ expression ]`

# if语句



✓ if - then - else结构

✓ 语法格式

```
if test expression
then
    true - commands
else
    false - commands
fi
```

# if语句

## ✓ 例12

```
#!/bin/bash
echo "Is it morning?Please answer yes or no"
read answer
if test $answer = "yes"
then
    echo "Good Morning"
else
    echo "Good afternoon"
fi
~
```



# if语句



## ✓ if - then - elif结构

```
if test expression1
then
    commands_1
elif test expression2
then
    commands_2
...
else
    commands_n
fi
```

# if语句

## ✓ 例13

```
#!/bin/bash
echo "Is it morning?Please answer yes or no"
read answer
if [ $answer = "yes" ]
then
    echo "Good Morning"
elif [ $answer = "no" ]
then
    echo "Good afternoon"
else
    echo "Enter yes or no"
fi
~
```

# 练习



- ✓ 输入的命令行参数必须是hello，才会正确显示；否则，显示错误提示
- ✓ 检测某个文件是否是一个普通文件
- ✓ 比较两个字符串str1和str2是否相等
- ✓ 判断一个数字是否是正数
- ✓ 判断给定的数字是否介于1到10之间

# if语句

## ✓ 说明

- ✓ if的测试部分通常是用test命令实现，也可以利用命令执行是否成功来判断

如果命令正常结束，则测试条件为真

如果命令执行不成功，则测试条件为假

```
if commands
then
    true - commands
else
    false - commands
fi
```

# if语句

## ✓ 例14：判断给定用户名是否已登陆系统

```
#!/bin/bash
echo "input user name:"
read user
if who | grep $user
then
    echo "$user has logged in the system"
else
    echo "$user has not logged in the system"
fi
```

# case语句

## ✓ 功能

- ✓ 实现多分支结构

## ✓ 格式

```
case string in
pattern 1)  command-list ;;
pattern 2)  command-list ;;
pattern n)  command-list ;;
esac
```

## ✓ 说明

- ✓ 每个模式必须以“ )” 结束
- ✓ 每个模式字符串后可以跟多条命令，其最后一条必须以 “;;” 结束

# case语句

## ✓ 例15

```
#!/bin/bash
echo "please chose either 1,2,3:"
echo "[1]print the current working directory"
echo "[2]chang the current working directory"
echo "[3]list the contents of the current directory"
read x
case $x in
1) pwd;;
2) cd $HOME; pwd;;
3) ls .; pwd;;
*) echo "wrong";;
esac
```

# case语句



## ✓ 说明

- ✓ 可以用 \* 作为最后一个模式字符串，匹配其他任何情况
- ✓ 模式字符串可以使用通配符：\*、?、[ ]
- ✓ 如果一个模式字符串中包含多个模式，那么各模式之间用 “|” 分隔



# case语句

## ✓ 例16

```
#!/bin/bash
echo "Is it morning?Enter yes or no"
read time
case $time in
yes | y | Yes | YES | Y ) echo "Good Morning";;
n* | N* ) echo "Good Afternoon";;
* ) echo "Please answer yes or no";;
esac
```

# 循环结构



- ✓ while循环
- ✓ until循环
- ✓ for循环

# while循环



## ✓ 功能

- ✓ 循环条件为真就进入循环，测试条件为假，退出循环

## ✓ 格式

```
while test expression  
do  
    commands  
done
```

```
while [ expression ]  
do  
    commands  
done
```

# while循环

✓ 例17：在屏幕上输出1-10

```
count=1
while [ $count -le 10 ]
do
    echo $count
    let count=count+1
done
```

✓ 练习：求1-10的和

# while循环



## ✓ 说明

- ✓ 当脚本的参数很多时，我们用shift命令来访问多个参数

# while循环

## ✓ 例18：求所有参数的累加和

```
if [ $# -le 0 ]
then
    echo "Not enough parameters"
    exit 0
fi
sum=0
while [ $# -gt 0 ]
do
    let sum=sum+$1
    shift
done
echo $sum
```

# while循环

- ✓ 例19：判断给定的参数（多个）是否为普通文件，如果是，则显示内容；否则，显示不是普通文件的提示信息

```
#!/bin/bash
while [ $1 ]
do
    if [ -f $1 ]
    then
        echo "display:$1"
        cat $1
    else
        echo "$1 is not a file name."
    fi
    shift
done
```

# until循环



## ✓ 功能

✓ 循环条件为假时，执行循环体。当测试条件为真时，循环结束

## ✓ 格式

```
until test expression  
do  
    commands  
done
```

```
until [ expression ]  
do  
    commands  
done
```



# until循环

✓ 例20：在屏幕上输出1-10

```
#!/bin/bash
count=1
until [ $count -gt 10 ]
do
    echo $count
    ((count=count+1))
done
```

✓ 练习：求1-10的和

# until循环

## ✓ 说明

✓ until语句可以与while语句相互替换

✓ until语句适用于这样的情况：

如果想让循环不停的执行，直到发生某件事为止

✓ 例21

```
#!/bin/bash
until who|grep "$1"
do
    sleep 10
done
echo "$1 has logged in the system"
~
```

# for循环



## ✓ 算术表达式方式

### ✓ 格式

```
for ((exp1; exp2; exp3))  
do  
    commands  
done
```

### ✓ 说明

三个条件表达式任何一个都可以缺少，但是分号不能缺少

# for循环

✓ 例22：在屏幕上输出1-10

```
#!/bin/bash
for ((i=1; i<=10; i=i+1))
do
    echo $i
done
```

✓ 练习：求1-10的和

# for循环

## ✓ 循环的嵌套

✓ 例23：打印给定行数的 \*

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****  
*****
```

# for循环



## ✓ 值表方式

### ✓ 格式

```
for variable in list - of - values  
do  
    commands  
done
```

### ✓ 说明

variable: 是循环变量

list - of - values : 是参数列表

in后列出n个参数, 则循环体执行n次

# for循环

✓ 值表方式

✓ 例24

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Number:$i"
done
~
```

# for循环

## ✓ 值表方式

### ✓ 说明

for循环语句中的参数列表允许使用通配符

### ✓ 例25：显示当前目录下所有.c文件

```
#!/bin/bash
for file in *.c
do
    echo "===== $file ====="
    cat $file
done
```



# for循环

## ✓ 值表方式

### ✓ 例26

```
#!/bin/bash
if [ $# = 0 ]
then
    echo "input parameters."
else
    for i in $@
    do
        echo "$i"
    done
fi
```

# for循环



## ✓ 值表方式

### ✓ 说明

如果在for语句中没有参数列表，则循环变量的取值范围是全部参数

# for循环



## ✓ 练习

- ✓ 判断给定的参数（多个）是否为普通文件，如果是，则显示内容；  
否则，显示不是文件的信息

# break和continue



## ✓ break命令

### ✓ 功能

使程序跳出for、while、until循环

### ✓ 格式

break n

### ✓ 说明

n表示要跳出的循环的层数，默认值为1

# break和continue



## ✓ continue命令

### ✓ 功能

跳到下一次循环继续执行

### ✓ 格式

`continue n`

### ✓ 说明

n表示要跳出的循环的层数，默认值为1

# break和continue

✓ 例27:

```
for i in 1 2 3 4 5
do
    if [ $i -eq 3 ]
    then
        break
    else
        echo "$i"
    fi
done
```

# break和continue

## ✓ 例28:

```
while true
do
    echo "please select your operation:"
    echo "1 copy"
    echo "2 delete"
    echo "3 backup"
    echo "4 quit"
    read op
    case $op in
        1) echo "your selection is copy";;
        2) echo "your selection is delete";;
        3) echo "your selection is backup";;
        4) echo "exit";break;;
        *) echo "invalid section,try again...";continue;;
    esac
done
```

# exit命令

---

## ✓ 功能

- ✓ 退出正在执行的脚本

## ✓ 格式

- ✓ `exit n`



# exit命令

## ✓ 例18:

```
if [ $# -le 0 ]
then
    echo "Not enough parameters"
    exit 0
fi
sum=0
while [ $# -gt 0 ]
do
    let sum=sum+$1
    shift
done
echo $sum
```

# select语句



## ✓ 功能

✓ 生成菜单列表

## ✓ 格式

```
select item in list - of - values  
do  
    commands  
done
```

# select语句

## ✓ 说明

- ✓ select是无限循环，应在循环体内提供结束循环的语句

## ✓ 例29:

```
echo "what is your favourite os"  
select var in "Linux" "UNIX" "Windows" "Others"  
do  
    break  
done  
echo "You have selected $var"
```

# select语句

## ✓ 说明

- ✓ select语句中的提示符默认为#?, 用户可以进行修改

## ✓ 例30:

```
_PS3="Choice?"
select choice in add delete update exit
do
    case $choice in
        add) echo "call add";;
        delete) echo "call delete";;
        update) echo "call update";;
        exit) echo "call exit";break;;
    esac
done
```

# 函数



## ✓ 格式

```
function func_name ()  
{  
    commands  
}
```

## ✓ 说明

- ✓ 关键字function可以省略

# 函数

## ✓ 说明

- ✓ 使用函数前必须先定义，即必须将函数定义放在脚本开始部分
- ✓ 调用函数时，直接利用函数名，不带括号

## ✓ 例31:

```
#!/bin/bash
fun1()
{
    echo "Function is executing."
}
echo "start...."
fun1
echo "end."
~
```

# 函数

## ✓ 说明

✓ 函数的参数利用位置变量传递

## ✓ 例32:

带参数的  
函数调用

```
#!/bin/bash
fun2()
{
    echo "Let's begin"
    echo $1 $2 $3
    echo "The end"
}
fun2 a b c
```

# shell内置命令

---

## ✓ 外置命令

- ✓ 存放在/bin、/sbin等目录下的命令
- ✓ 执行时，系统需要创建子进程

date touch mkdir cat more mv ls link rm rmdir cp  
chmod chown ps

## ✓ 内置命令

- ✓ 在shell的地址空间内执行，不再创建新进程

break continue cd echo exit export kill let pwd read  
return shift test umask wait



# shell内置命令

---

## ✓ type命令

✓ 显示命令是内置命令还是外部命令

✓ 格式

type 命令名

✓ 例子

type cd echo who

# shell内置命令

## ✓ eval命令

✓ 利用别的命令作为参数，进行变量或命令替换

✓ 格式

eval 参数

✓ 例子

a= "ls"

b= "|wc"

eval \$a\$b

# shell内置命令

## ✓ trap命令

✓ 捕捉信号

✓ 格式

trap 命令 信号名

✓ 例子

trap "commands" 2

//捕捉到信号2就执行命令

trap "" 2

//忽略信号2

trap 2

//还原为信号2的默认处理

# shell内置命令

---

## ✓ trap命令

### ✓ 说明

命令表要用引号括起来

信号名可以用数字或信号名表示（信号名忽略SIG前缀）

# shell内置命令

## ✓ trap命令

✓ 例33:

```
#!/bin/bash
trap "my_exit" 2
loop=0
my_exit()
{
    echo "you just hit <Ctrl-C>,at number $loop"
    echo "I will now exit"
    exit 1
}
while true
do
    sleep 1
    let loop=loop+1
    echo $loop
done
```