



武汉纺织大学  
WUHAN TEXTILE UNIVERSITY

崇真尚美



# Linux系统编程

数学与计算机学院

教师：朱萍

# 本课程的主要内容

- **Linux**基础知识（4学时）
- **Linux**文件系统（6学时）
- **Linux**系统管理（2学时）
- **Linux**网络管理及应用（4学时）
- **Linux Shell**编程（8学时）
- **Linux**下C编程（6学时）
- **Linux**下进程通信（6学时）
- **Linux**下线程通信（6学时）
- **Linux**文件接口编程（6学时）

# Linux Shell编程

❖ **shell简介**

❖ **shell基础**

❖ **shell脚本**

❖ **shell变量**

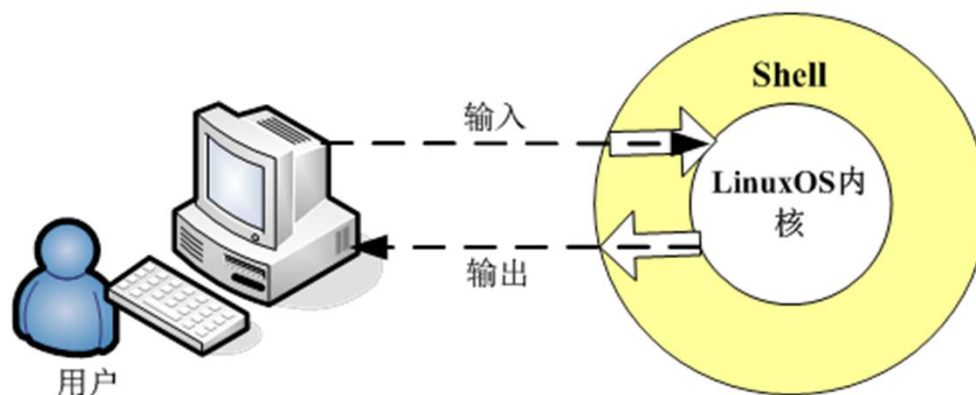
❖ **shell编程基础**

❖ **shell控制结构**

❖ **shell函数**

# Shell简介

- ❖ 在Linux操作系统中，**shell**是用户与操作系统内核打交道的接口



- ❖ Linux中的**shell**有多种类型，最常用的几种是 Bourne Again Shell (简称bash)、Bourne Shell (简称sh)、C-Shell (简称csh) 和Korn Shell (简称ksh)。

# 脚本语言比较

❖ Unix/Linux Shell（包括awk、 sed），是日常使用最多的。

- 个人感觉，shell更适合做流程化的系统管理工作，而对于逻辑较为复杂的应用级产品，shell的开发和维护难度都相当高。
- shell的学习难度大于大多数主流脚本语言，这和它的语法特点，调试、运行环境等等都有关系。

# 脚本语言比较

- ❖ **Python**最大的魅力之一，就是简单易学，一个新手可以短期内（有人说是十分钟，我认为几天是完全可能的）就写出高质量并且实用的程序，它甚至被称为非计算机专业人士学习编程的最好选择。
- ❖ **Python**的功能强大，标准库的丰富程度，估计连**Java**也望尘莫及。**Python**适合做系统管理，但这并不是说它不能做大型应用程序。
- ❖ **Python**应用的成功案例，已经数不胜数，尤其是在科学计算领域更是独树一帜。**Python**是一门优秀的面向对象语言，但更多的**Python**程序员，喜欢**Python**面向过程的部分。

## 脚本语言比较

❖ **Ruby** 是脚本语言的后起之秀，它的成功要得益于杀手级框架**Rails**，**Rails**社区早就喊出了干掉**Java**的口号，虽然这种口号没有什么实际意义，但 **Rails**的确已经成为眼下最为炙手可热的企业应用框架。我个人花了几个月的时间学习**Rails**，虽然没有实际的开发经验，但**Rails**敏捷开发上的突出表现，的确是当仁不让的，这也是它能迅速窜红的主要原因吧。

# 几种常见shell

(1) Bourne Shell(sh)是AT&T Bell实验室的 Steven Bourne为Unix开发的，它是Unix的默认Shell。在编程方面相当优秀，但在处理与用户的交互方面不如其它几种Shell。

(2) C Shell(csh)是加州伯克利大学的Bill Joy为BSD Unix开发的，与sh不同，它的语法与C语言很相似。C Shell与BourneShell并不兼容。

(3) Korn Shell(ksh)是AT&T Bell实验室的David Korn开发的，它集合了C Shell和Bourne Shell的优点，并且与Bourne Shell向下完全兼容。Korn Shell的效率很高，其命令交互界面和编程交互界面都很好。

(4) Bourne Again Shell (即bash)是自由软件基金会(GNU)开发的一个Shell，它是Linux系统中一个默认的Shell。Bash不但与Bourne Shell兼容，还继承了C Shell、Korn Shell等优点。



# Shell简介

- ❖ 查看该发行版本可用的shell类型以及当前正在使用的shell

```
jkx@ubuntu:~$ sudo cat /etc/shells
[sudo] password for jkx:
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
jkx@ubuntu:~$ echo $SHELL
/bin/bash
```

# Linux Shell编程

❖ **shell简介**

❖ **shell基础**

❖ **shell脚本**

❖ **shell变量**

❖ **shell编程基础**

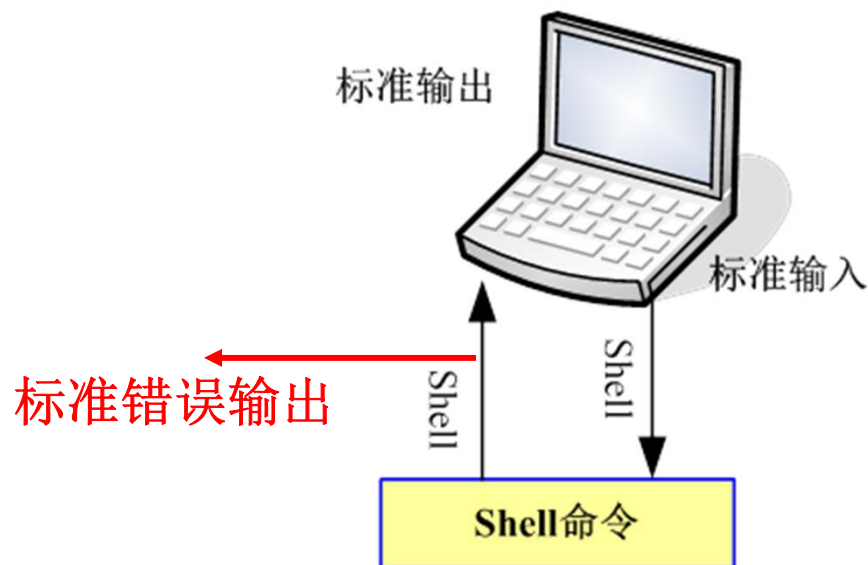
❖ **shell控制结构**

❖ **shell函数**

# Shell基础

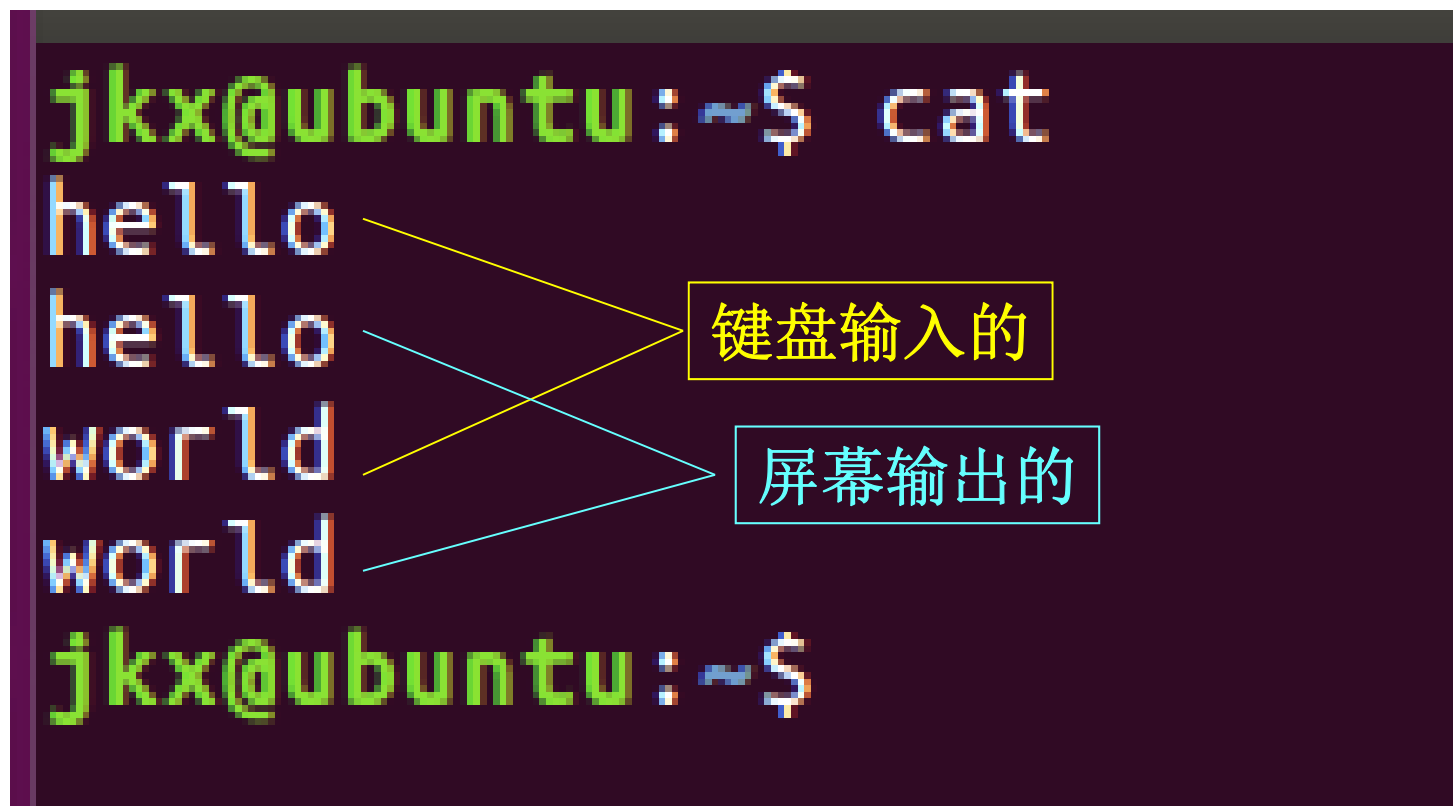
## ❖ 标准输入、输出、出错

- 执行一个**shell**命令时，**Linux**系统通常会自动打开三个文件：标准输入文件（**stdin**）、标准输出文件（**stdout**）和标准错误输出文件（**stderr**）。



# 标准输入

- ❖ 例如 **cat** 命令，当后面没有接文件时，系统会将标准输入（也就是键盘输入的内容）当做要 **cat** 的文件。



```
jkgx@ubuntu:~$ cat
hello
hello
world
world
jkgx@ubuntu:~$
```

The diagram illustrates the flow of data in a terminal. Two yellow boxes on the right, labeled "键盘输入的" (Keyboard Input) and "屏幕输出的" (Screen Output), have arrows pointing to the terminal text. The "键盘输入的" box has arrows pointing to the two "hello" lines. The "屏幕输出的" box has arrows pointing to the two "world" lines and the final prompt line.

# 直接使用标准输入输出存在的问题

- ❖ 输入数据只能用一次，下次还要重新输入，而且在输入过程中出现错误不好修改
- ❖ 输出到屏幕上的结果只能看，不能使用，当结果很多时，不方便浏览。
- ❖ 为此，Linux引入重定向机制。
- ❖ 重定向是指：改变来源（标准输入）和去向（标准输出），产生了输入重定向和输出重定向。

# 输入重定向

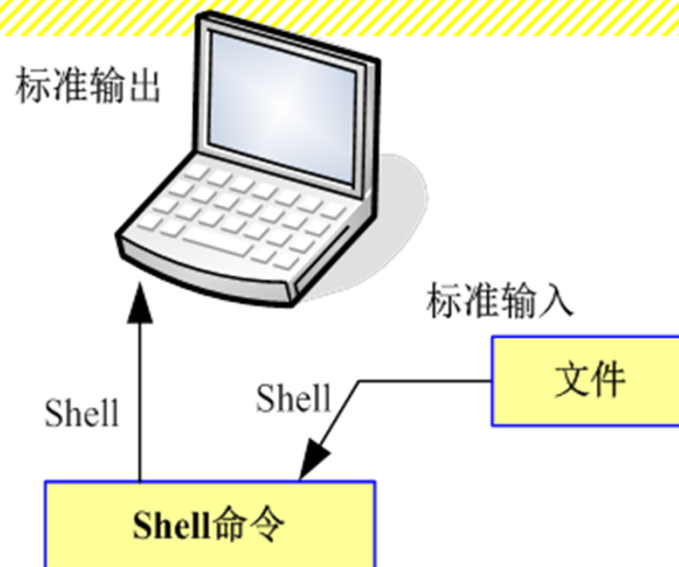
## ❖ 输入重定向符号 <

## ❖ 语法格式:

- **shell 命令 < 文件名**

含义: 该文件内容作为shell命令的输入

```
jlx@ubuntu:~$ wc < /etc/passwd
48   78 2565
```



- ❖ 实际上, 大多数命令都将输入文件当做参数, 所以上面的命令等效于 `wc /etc/passwd`
- ❖ 所以, 输入重定向< 使用不多

# 输出重定向

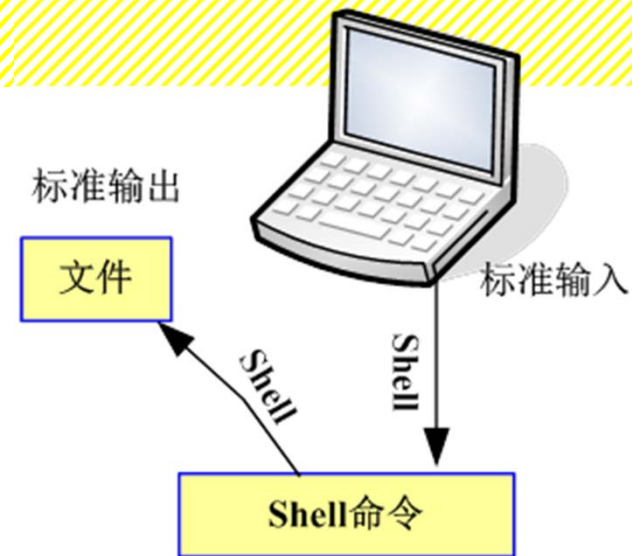
## ❖ 输出重定向符号 >

## ❖ 语法格式:

- **shell 命令 > 文件名**

含义：shell命令执行结果输出到该文件中

```
jkg@ubuntu:~$ date > date.log
jkg@ubuntu:~$ cat date.log
Wed Oct  5 19:52:21 PDT 2016
jkg@ubuntu:~$
```



如果该文件不存在，则会新建该文件。如果已经存在，这该文件会被重写

## 输出追加重定向

- ❖ 如果不想覆盖已有的内容，只是添加到文件后面，可以使用输出追加重定向 >>

```
jcx@ubuntu:~$ date >> date.log
jcx@ubuntu:~$ cat date.log
Wed Oct  5 19:52:21 PDT 2016
Wed Oct  5 19:55:15 PDT 2016
```



# 错误重定向

- ❖ 错误重定向符号 `2>`
- ❖ 错误追加重定向符号 `2>>`
- ❖ 语法格式:
  - **shell 命令** `2>` 文件名
  - **shell 命令** `2>>` 文件名

```
jkgx@ubuntu:~/work/chapt5$ gcc -o hello hello.c
hello.c:1:15: fatal error: std: No such file or directory
compilation terminated.
jkgx@ubuntu:~/work/chapt5$ gcc -o hello hello.c 2>err.out
jkgx@ubuntu:~/work/chapt5$ cat err.out
hello.c:1:15: fatal error: std: No such file or directory
compilation terminated.
```

# 管道

❖ 管道符号: |

❖ 语法格式:

- 命令1 | 命令2
- 将命令1的输出作为命令2的输入

```
jkx@ubuntu:~$ ls
date.log  Documents  examples.desktop  nfsdir  Public  Templates  work
Desktop   Downloads  Music             Pictures  software  Videos
jkx@ubuntu:~$ ls | wc -w
13
```

# 管道

❖ `ps aux | grep xxx`

xxx:可以是进程名、用户名、进程id等等

```
jkg@ubuntu:~$ ps aux | grep smbd
root      1350  0.0  1.0 42304 10920 ?        Ss   18:52   0:00 /usr/sbin/smbd
-D
root      1356  0.0  0.3 40468  4076 ?        S    18:52   0:00 /usr/sbin/smbd
-D
root      1362  0.0  0.4 42312  5092 ?        S    18:52   0:00 /usr/sbin/smbd
-D
jkg       3163  0.0  0.0   5108   844 pts/4    S+   20:14   0:00 grep --color=au
to smbd
```

# 特殊符号

❖ Shell中有一些特殊符号

## (1) 通配符

\*

?

[ ]

## (2) 引号

双引号

反引号

单引号

## (3) 注释符

#

## 特殊符号：通配符

通配符通常用于模式匹配，如文件名匹配、路径名搜索、字符串查找等

(1) \* : 任意个（包括0个）任意字符

(2) ? : 任意单个字符

(3) [] : 字符范围，中间可以使用连接符-

```
jkg@ubuntu:~/work/chapt2$ ls
display file1.c file2.c fruits fruits_1 fruits_2 product_i
jkg@ubuntu:~/work/chapt2$ ls fruits*
fruits fruits_1 fruits_2
jkg@ubuntu:~/work/chapt2$ ls file?.c
file1.c file2.c
jkg@ubuntu:~/work/chapt2$ ls fruits_[1-9]
fruits_1 fruits_2
jkg@ubuntu:~/work/chapt2$
```

## 特殊符号：引号

- ❖ 当命令中有包含空格的字符串时，需要将该字符串用单引号或双引号包起来。
- ❖ 单引号：
  - 单引号包住的符号都当做普通字符处理，也就是说单引号里面的特殊符号也失去了特殊性

```
jkgx@ubuntu:~/work/chapt2$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
al/games:/snap/bin
jkgx@ubuntu:~/work/chapt2$ echo '$PATH'
$PATH
```

**\$**表示获取变量的值

**PATH**是一个环境变量

# 特殊符号：引号

## ❖ 反引号

- 反引号位于键盘左上角与波浪号共用一个按键
- 不要与单引号混淆
- 反引号括起来的字符串需要先执行，其结果作为输出。

```
jkx@ubuntu:~/work/chapt2$ echo `date`  
Wed Oct 5 21:35:25 PDT 2016
```

# 特殊符号：引号

## ❖ 双引号

- 它关闭**shell**中大部分的特殊符号，但是保留了例如 **\$**，反引号```和转义字符`\`的特殊性。
  - **\$**:取变量的值
  - 反引号```：先执行反引号中包含的命令
  - **\**：当其后跟的是“**\$**”、反引号“```”、双引号“`""`”、“`\`”时，当普通字符处理。简单来说，当你要输出这几个字符时，需要前面加上`\`。



## 双引号中的\$和反引号举例

```
jkx@ubuntu:~/work/chapt2$ echo "$PATH"  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/snap/bin
```

```
jkx@ubuntu:~/work/chapt2$ echo "date"  
date  
jkx@ubuntu:~/work/chapt2$ echo "`date`"  
Wed Oct  5 21:17:39 PDT 2016  
jkx@ubuntu:~/work/chapt2$ echo '`date`'  
`date`
```

## 双引号中的转义字符\举例

```
jlx@ubuntu:~/work/chapt2$ echo "\$PATH = $PATH"  
$PATH = /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:  
/usr/local/games:/snap/bin
```

```
jlx@ubuntu:~/work/chapt2$ echo "\`date\` = `date`"  
`date` = Wed Oct  5 21:22:07 PDT 2016
```

## 双引号中的转义字符\举例

```
jkg@ubuntu:~/work/chapt2$ echo ""
> ^C
jkg@ubuntu:~/work/chapt2$ echo "\""
"
jkg@ubuntu:~/work/chapt2$ echo ""
jkg@ubuntu:~/work/chapt2$ echo "\"\""
""
```

```
jkg@ubuntu:~/work/chapt2$ echo "\""  
> ^C  
jkg@ubuntu:~/work/chapt2$ echo "\"\""  
\""
```

# Linux Shell编程

❖ shell简介

❖ shell基础

❖ shell脚本

❖ shell变量

❖ shell编程基础

❖ shell控制结构

❖ shell函数

# Shell脚本

## ❖ 1. 什么是Shell脚本

- Shell脚本是使用**shell**命令编写的文件，也称为**shell script**。
- 与结构化程序不同，**shell**不需要编译成目标程序，也不需要链接成可执行的目标码，**shell**是按行一条接着一行地**解释**并执行**shell**脚本中的命令。
- Shell脚本多用**vi**来编辑。

# Shell脚本

## ❖ 2.shell脚本执行方式

有3种方式可以执行一个**shell** 脚本。

（1）为脚本文件加上可执行权限，然后在命令行直接输入**shell**脚本文件名执行。

（2） **sh shell**脚本名

（3） **. shell**脚本名

# Shell 脚本

❖ vi编写脚本pwd\_script:

**#!/bin/bash**

#!告诉操作系统,  
此脚本的解释器  
为 /bin/bash

**#this script is to test shell running**

**date**

**cd /home/jkx**

**echo "The working directory is:"**

**pwd**

**#end**

# Shell 脚本执行方式

## ❖ 第一种执行方式:

- 为shell脚本文件加上可执行权限
- `./ shell脚本名`

```
jcx@ubuntu:~/work/chapt5$ chmod +x pwd_script
jcx@ubuntu:~/work/chapt5$ ./pwd_script
Wed Oct  5 22:53:05 PDT 2016
the working directory is
/home/jcx
```



# Shell 脚本执行方式

## ❖ 第二种执行方式

- sh shell脚本名
- sh和文件名之间需要空格，此时文件无需可执行权限

```
jcx@ubuntu:~/work/chapt5$ sh pwd_script
Wed Oct  5 22:54:04 PDT 2016
the working directory is
/home/jcx
```

# Shell 脚本执行方式

## ❖ 第三种执行方式

- . shell脚本名
- . 和文件名之间需要空格，此时文件无需可执行权限

```
jcx@ubuntu:~/work/chapt5$ . pwd_script
Wed Oct  5 22:57:42 PDT 2016
the working directory is
/home/jcx
```

# Linux Shell编程

❖ **shell**简介

❖ **shell**基础

❖ **shell**脚本

❖ **shell**变量

❖ **shell**编程基础

❖ **shell**控制结构

❖ **shell**函数

# Shell变量

- ❖ 在**shell**编程中也可以使用变量，一个变量就是内存中被命名的一块存储空间。
- ❖ 一个**Shell**变量的名字可以包含数字，字母和下划线，变量名的开头只准许是字母和下划线。变量名中的字母是大小写敏感的
- ❖ 变量名在理论上的长度没有限制。
- ❖ 在**shell**编程中可以使用四种变量：
  - 用户自定义变量
  - 环境变量
  - 位置变量
  - 特殊变量

# Shell变量：用户自定义变量

## ❖ 语法格式：

- 变量名=变量值
- 如果字符串里包含空格，就必须用引号把它们括起来。注意在等号两边不能有空格。
- 无论何时想要获取变量内容，必须在它前面加\$字符。

```
jcx@ubuntu:~$ v1=hello
jcx@ubuntu:~$ echo $v1
hello
jcx@ubuntu:~$ v1=1+2
jcx@ubuntu:~$ echo $v1
1+2
```

# Shell变量：用户自定义变量

## ❖清除变量

- 格式： **unset** 变量名
- 设置的变量不需要时可以清除。

```
jkx@ubuntu:~$ unset v1
jkx@ubuntu:~$ echo $v1

jkx@ubuntu:~$
```

# Shell变量：环境变量

- ❖ 它决定了用户的工作环境，通常用大写字母作为变量名，以便把它们和用户在脚本程序里定义的变量区分开来。
- ❖ 常用环境变量**P160/表6.2**

环境变量	含义说明
<b>HOME</b>	当前用户的主目录，即用户登录时默认的目录
<b>PATH</b>	以冒号分隔的用来搜索命令的路径列表
<b>PS1</b>	命令提示符， <b>root</b> 用户为#， <b>bash</b> 中普通用户通常是\$字符，在 <b>c shell</b> 中通常是%
<b>PS2</b>	二级提示符，用来提示后续的输入，通常是>字符
<b>IFS</b>	输入域分隔符。当 <b>shell</b> 读取输入时，用来分隔单词的一组字符，它们通常是空格、制表符和换行符
<b>TERM</b>	用来设置用户的终端类型，系统主控台（ <b>console</b> ）不用设置

# Shell变量：环境变量

## ❖ 查看环境变量：env命令

```
jkk@ubuntu:~$ env
XDG_VTNR=7
LC_PAPER=en_US.UTF-8
LC_ADDRESS=en_US.UTF-8
XDG_SESSION_ID=c2
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/jkk
LC_MONETARY=en_US.UTF-8
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GPG_AGENT_INFO=/home/jkk/.gnupg/s.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=56623114
LC_NUMERIC=en_US.UTF-8
OLDPWD=/home/jkk/work/chapt5
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=jkk
```



## Shell变量：位置变量

- ❖ 如果脚本程序在执行时带有参数，就会创建一些额外的变量，这些额外的变量因跟变量所在命令行位置有关，因此被称为位置变量或位置参数。

```
jkg@ubuntu:~/work/chapt5$ cat location_script
#!/bin/bash
echo "$0"
echo "$1"
echo "$2"
echo "$3"
jkg@ubuntu:~/work/chapt5$ ./location_script aa bb cc
./location_script
aa
bb
cc
```

- ❖ \$0, \$1, \$2, \$3就是位置参数

## Shell变量：位置变量

- ❖ **shell**提供的位置变量有**\$0**、**\$1**、**\$2**、**\$3**、**\$4**、**\$5**、**\$6**、**\$7**、**\$8**、**\$9**。位置变量**\$0**存放脚本名，**\$1**、**\$2**、**\$3**、**\$4**、**\$5**、**\$6**、**\$7**、**\$8**、**\$9**存放从左至右的命令行上的参数。
- ❖ 当命令行上命令参数超过**9**个时，**shell**提供了**shift**命令可以把所有参数变量左移一个位置，使**\$2**变成**\$1**，**\$3**变成**\$2**，依此类推。
- ❖ 使用格式如下：
  - **shift [n]**
  - 其中**n**表示向左移动参数的个数，默认值为**1**。

## Shell变量：位置变量

❖ 例如：

```
#!/bin/bash
# this script test shift
echo $0
echo "start"
echo $1,$2,$3,$4,$5,$6,$7,$8,$9
shift
echo "after shift"
echo $1,$2,$3,$4,$5,$6,$7,$8,$9
shift 2
echo "after shift 2"
echo $1,$2,$3,$4,$5,$6,$7,$8,$9
```

## Shell变量：位置变量

```
x@ubuntu:~/work/chapt5$ ./location_script aa bb cc dd ee ff
location_script
start
aa,bb,cc,dd,ee,ff,gg,hh,ii
after shift
bb,cc,dd,ee,ff,gg,hh,ii,jj
after shift 2
cc,dd,ee,ff,gg,hh,ii,jj,,
after shift 3
dd,ee,ff,gg,hh,ii,jj,,,
```

# Shell变量：特殊变量

- ❖ **shell**中有一些变量是系统定义的，有特殊的含义，变量值由系统指定，被称之为**特殊变量**。
- ❖ 使用特殊符号一般用双引号包起来
  - **\$#**：表示传递给脚本的实际参数个数。
  - **\$\$**：当前**shell**脚本的进程号。
  - **\$\***：位置参数的值，各个参数之间用环境变量**IFS**中定义的字符分隔开。
  - **\$@**：也表示位置参数的值，它不使用**IFS**环境变量，所以当**IFS**为空时，参数值不会结合在一起。
  - **!**：上一个后台命令的进程号。
  - **\$?**：执行最后一条命令的退出状态。

## 特殊变量举例

```
#!/bin/bash
# this script test special variable
echo "the script file name is : $0"
echo "the total arguments number is : $#"
```

echo "the arguments are \$@"

echo "the pid is : \$\$"

echo "end"

## 特殊变量举例

```
jcx@ubuntu:~/work/chapt5$ chmod +x specialvar_script
jcx@ubuntu:~/work/chapt5$ ./specialvar_script aa bb cc dd
the script file name is : ./specialvar_script
the total arguments number is : 4
the arguments are aa bb cc dd
the pid is : 5042
end
```

# Linux Shell编程

❖ **shell**简介

❖ **shell**基础

❖ **shell**脚本

❖ **shell**变量

❖ **shell**编程基础

❖ **shell**控制结构

❖ **shell**函数



# shell编程基础

- ❖ (1) Shell脚本的输入/输出
- ❖ (2) Shell的逻辑运算
- ❖ (3) Shell的算术运算

# Shell脚本的输入/输出

## (1) 输入命令: **read**

- **read**命令来将键盘输入赋值给一个变量, 有点类似**scanf**。
- 命令格式如下:
  - **read** 变量名1 [变量名2.....]

```
jkg@ubuntu:~/work/chapt5$ read v1
abc
jkg@ubuntu:~/work/chapt5$ echo $v1
abc
jkg@ubuntu:~/work/chapt5$ read v1 v2
abc,edf
jkg@ubuntu:~/work/chapt5$ echo $v1
abc
jkg@ubuntu:~/work/chapt5$ echo $v2
edf
```

# Shell脚本的输入/输出

## (2) 输入命令: **echo**

- **echo**输出多个空格时必须用单引号括起, 否则多个空格都认为是一个空格

## (3) **export**命令

- **export**命令可将在**shell**脚本中定义的变量导出到子**shell**中, 并使之在子**shell**中有效。

export1\_script

```
#!/bin/bash
var1="The first variable"
export var2="The second variable"
sh export2_script
```

export2\_script

```
#!/bin/bash
echo "$var1"
echo "$var2"
```

## Shell脚本的输入/输出

```
jcx@ubuntu:~/work/chapt5$ chmod +x export*  
jcx@ubuntu:~/work/chapt5$ ./export2_script  
  
jcx@ubuntu:~/work/chapt5$ ./export1_script  
The second variable  
jcx@ubuntu:~/work/chapt5$
```

# shell编程基础

- ❖ (1) Shell脚本的输入/输出
- ❖ (2) Shell的逻辑运算
- ❖ (3) Shell的算术运算

# Shell的逻辑运算

❖ 所有程序设计语言的基础是对条件进行测试判断，并根据测试结果（真 / 假）采取不同的操作

## （1）条件测试：

■ 有两种条件测试命令，语法格式如下：

1) **test** 条件表达式

2) **[ 条件表达式 ]**

• **注意：**使用第二种方法进行条件测试时，必须在**[ ]**前后保留空格，否则**shell**提示**error**

condition\_test\_script1

```
#!/bin/bash
if test -f hello.c
then
    echo "find file"
fi
```

condition\_test\_script2

```
#!/bin/bash
if [ -f hello.c ]
then
    echo "find file"
fi
```

```
kx@ubuntu:~/work/chapt5$ ls
condition_test_script1  export1_script  location_sc
condition_test_script2  export2_script  pwd_script
err.out                hello.c         specialvar_
kx@ubuntu:~/work/chapt5$ chmod +x condition_test_s
kx@ubuntu:~/work/chapt5$ ./condition_test_script1
ind file
kx@ubuntu:~/work/chapt5$ ./condition_test_script2
ind file
```

❖注意：如果自己写的文件命名为test而运行有误，有可能是与系统的test命令冲突。

# 条件测试的三种类型

## <1>字符串比较

字符串比较	结 果
<b>string1 = string2</b>	如果两个字符串相同则结果为真
<b>string1 != string2</b>	如果两个字符串不同则结果为真
<b>-n string</b>	如果字符串不为空则结果为真
<b>-z string</b>	如果字符串为空（一个空串）则结果为真

注意：字符串比较时=两边有空格



# 条件测试的三种类型

## <2>算术比较

算术比较	结 果
<b>expression1 -eq expression2</b>	如果两个表达式相等则结果为真
<b>expression1 -ne expression2</b>	如果两个表达式不等则结果为真
<b>expression1 -gt expression2</b>	如果expression1大于expression2则结果为真
<b>expression1 -ge expression2</b>	如果expression1大于或等于expression2则结果为真
<b>expression1 -lt expression2</b>	如果expression1小于expression2则结果为真
<b>expression1 -le expression2</b>	如果expression1小于或等于expression2则结果为真
<b>! expression</b>	如果表达式为假则结果为真，反之亦然

# 条件测试的三种类型

## <3>文件测试

文件条件 测试	结 果
<b>-d file</b>	如果文件是一个目录则结果为真
<b>-f file</b>	如果文件是一个普通文件则结果为真
<b>-s file</b>	如果文件的长度不为0则结果为真
<b>-r file</b>	如果文件可读则结果为真
<b>-w file</b>	如果文件可写则结果为真
<b>-x file</b>	如果文件可执行则结果为真

# Shell的逻辑运算

(2) 逻辑运算: **AND运算** 和 **OR运算**

<1>AND运算: &&

■语法格式为:

- 条件1 && 条件2 && 条件3.....

■从左到右执行, 与C语言类似, 某个条件为假则终止执行。所有条件为真, 返回值才为真。

## AND运算例子

```
#!/bin/bash
touch file1
rm file2
if [ -f file1 ] && echo "hello" && [ -f file2 ] && echo "world"
then
    echo "if condition is true"
else
    echo "if condition is false"
fi
exit 0
```

```
jlx@ubuntu:~/work/chapt5$ ./and_script
rm: cannot remove 'file2': No such file or directory
hello
if condition is false
```

# Shell的逻辑运算

(2) 逻辑运算: **AND运算** 和 **OR运算**

**<2>OR运算: ||**

语法格式为:

- 条件1 || 条件2 || 条件3.....
- 从左到右执行, 与C语言类似, 某个条件为真则终止执行。所有条件为假, 返回值才为假。

## OR运算例子

❖ 把上面的&& 改为 ||

```
#!/bin/bash
touch file1
rm file2
if [ -f file1 ] && echo "hello" && [ -f file2 ] && echo "world"
then
    echo "if condition is true"
else
    echo "if condition is false"
fi
exit 0
```

```
jkx@ubuntu:~/work/chapt5$ ./or_script
rm: cannot remove 'file2': No such file or directory
if condition is true
jkx@ubuntu:~/work/chapt5$ ./or_script 2>err.out
if condition is true
```

# shell编程基础

- ❖ (1) Shell脚本的输入/输出
- ❖ (2) Shell的逻辑运算
- ❖ (3) Shell的算术运算

# Shell的算术运算

❖ **bash**提供了3种方法对数值数据进行**算术运算**

:

- (1) 使用**expr**命令（用的少，基本被(2)取代）
- (2) 使用**shell**扩展**`$((expression))`**
- (3) 使用**let**命令



# Shell的算术运算

## ❖ (2) \$((表达式))

```
cx@ubuntu:~/work/chapt5$ a=2 b=3
cx@ubuntu:~/work/chapt5$ echo "a+b = $a+$b"
a+b = 2+3
cx@ubuntu:~/work/chapt5$ echo "$a+$b = $((a+b))"
2+3 = 5
```

## Shell的算术运算

### ❖ (3) let 表达式

```
jkx@ubuntu:~/work/chapt5$ let a=2 b=3  
jkx@ubuntu:~/work/chapt5$ let c=a*b  
jkx@ubuntu:~/work/chapt5$ echo $c  
6
```

# Linux Shell编程

- ❖ **shell简介**
- ❖ **shell基础**
- ❖ **shell脚本**
- ❖ **shell变量**
- ❖ **shell编程基础**
- ❖ **shell控制结构**
- ❖ **shell函数**

# Shell控制结构

## ❖ 1.if 语句

- 基本if语句
- elif语句
- if嵌套

## ❖ 2.case语句

## ❖ 3.for语句

## ❖ 4.while语句

## ❖ 5 until语句

## ❖ 6.break和continue语句

# 基本if语句

不要忘记结尾处fi，就是if反着写

❖ 语法格式:

if condition  
then  
statements  
else  
statements  
fi

```
#!/bin/bash
echo "abc is the user's name? please answer !"
read name
if [ $name = "yes" ]
then
    echo "hello abc"
else
    echo "abc is not user's name"
fi
```

```
jkg@ubuntu:~/work/chapt5$ ./if_script
abc is the user's name? please answer yes or no
no
abc is not user's name
jkg@ubuntu:~/work/chapt5$ ./if_script
abc is the user's name? please answer yes or no
yes
hello abc
```

# elif语句

## ❖ 语法格式

```
if condition1  
then  
    statements  
elif condition2  
then  
    statements  
.....  
else  
    statements  
fi
```

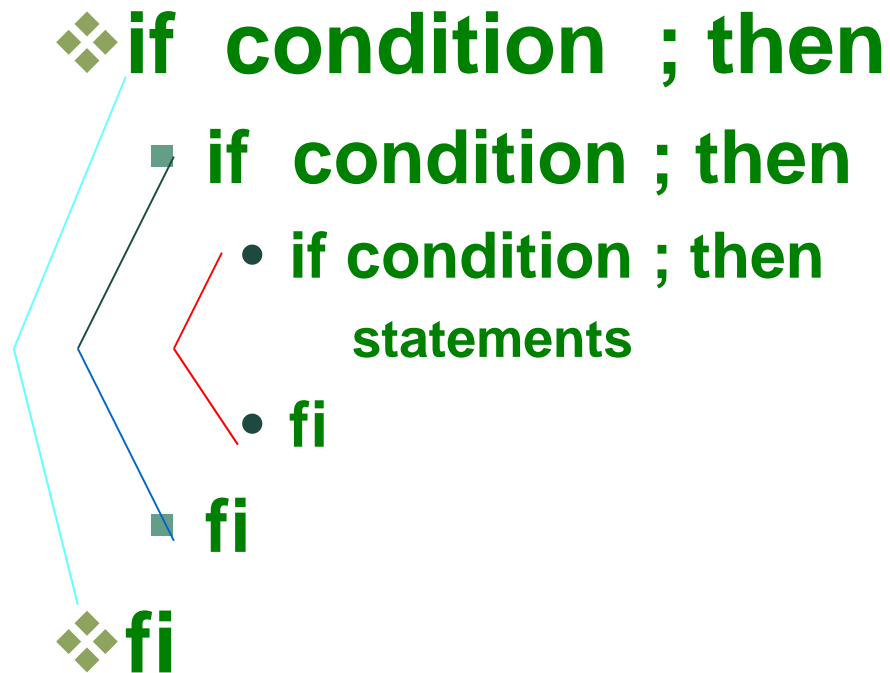
## elif语句

- ❖ 修改上面例子，将else改为elif，再增加else处理当输入的不是yes或no，则输出input error的提示信息。

```
#!/bin/bash
echo "abc is the user's name? please answer yes or no"
read name
if [ $name = "yes" ]
then
    echo "hello abc"
elif [ $name = "no" ]
then
    echo "abc is not user's name"
else
    echo "input error!"
fi
exit 0
```

```
jckx@ubuntu:~/work/chapt5$ ./elif_script
abc is the user's name? please answer yes or no
y
input error!
```

# if嵌套



❖ 分号；的作用：如果多个命令在一行，需要用分号隔开。更紧凑。



## 分号的作用

```
if condition1 ; then
    statements
elif condition2 ; then
    statements
elif condition3 ; then
    statements
.....
else
    statements
fi
```

# case语句

case variable in

pattern1) statements;;

pattern2) statements;;

.....

pattern) statements;;

~~\*) statements;;~~

esac

两个分号

\*类似C语言中的  
**default**

不要忘记结尾处的**esac**，就是  
**case**反着写

## case语句例子

```
#!/bin/bash
echo "abc is the user's name? please answer yes or no"
read name
case $name in
y|Y|yes|YES)
    echo "hello, abc!"
    echo "Welcome" ;;
[Nn][Oo])
    echo "abc is not your name" ;;
*)
    echo "sorry, your input isn't recognized"
    echo "please input yes or no"
    exit 1 ;;
esac
exit 0
```

## case语句例子

```
jkg@ubuntu:~/work/chapt5$ ./case1_script
abc is the user's name? please answer yes or no
y
hello, abc!
Welcome
jkg@ubuntu:~/work/chapt5$ ./case1_script
abc is the user's name? please answer yes or no
No
abc is not your name
jkg@ubuntu:~/work/chapt5$ ./case1_script
abc is the user's name? please answer yes or no
yesno
sorry, your input isn't recognized
please input yes or no
```

# for语句

❖ 语法格式:

**for variable in values**

**do**

**statements**

**done**

```
#!/bin/bash
for str in hello world 123
do
    echo $str
done
exit 0
```

```
jkx@ubuntu:~/work/chapt5$ ./for_script
hello
world
123
```

# for语句

❖也可以与通配符结合使用

```
#!/bin/bash
for file in $(ls *_script)
do
    echo $file
done
exit 0
```

**\$( )**等价  
于反引号

```
jkg@ubuntu:~/work/chapt5$ ./for1_script
and_script
case1_script
elif_script
export1_script
export2_script
for1_script
for_script
if_script
location_script
or_script
```

# for语句

❖ for 使用一个数值范围 { .. }

```
#!/bin/bash
sum=0
for i in {1..10}
do
    sum=$((sum+i))
done
echo $sum
exit 0
```

```
jkg@ubuntu:~/work/chapt5$ ./for2_script
55
```

# while语句

语法格式:

**while condition**

**do**

**statements**

**done**

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

```
echo "please enter password:"
```

```
read passwd
```

```
while [ "$passwd" != "123456" ]
```

```
do
```

```
    echo "sorry, try again"
```

```
    read passwd
```

```
done
```

```
exit 0
```

当条件满足时，进入循环

```
jkk@ubuntu:~/work/chapt5$ ./while_script
please enter password:
123
sorry, try again
123456
```



# while语句

❖ 将for语句求1~10之和改为while语句

```
#!/bin/bash
sum=0
var=1
while [ "$var" -le 10 ]
do
    let sum=sum+var
    let var=var+1
done
echo $sum
exit 0
```

```
jckx@ubuntu:~/work/chapt5$ ./while1_script
55
```

# until语句

语法格式:

```
until condition
do
    statements
done
```

```
#!/bin/bash
sum=0
var=1
until [ "$var" -gt 10 ]
do
    let sum=sum+var
    let var=var+1
done
echo $sum
exit 0
```

当条件满足时，退出循环

```
jcx@ubuntu:~/work/chapt5$ ./until_script
55
```

# break语句

❖ 与C语言一样，**break**用于跳出一层循环

```
var1=0
while [ "$var1" -le 5 ]
do
    var1=$((var1+1))
    if [ "$var1" -eq 3 ]
    then
        break
    else
        echo "$var1"
    fi
done
```

```
jlx@ubuntu:~/work/chapt5$ ./break_script
1
2
```

# continue语句

❖ 与C语言一样，`continue`用于直接跳到下一轮循环

```
var1=0
while [ "$var1" -le 5 ]
do
|   var1=$((var1+1))
|   if [ "$var1" -eq 3 ]
|   then
|       continue
|   else
|       echo "$var1"
|   fi
done
```

```
jkg@ubuntu:~/work/chapt5$ ./continue_script
1
2
4
5
6
```

# Linux Shell编程

❖ **shell**简介

❖ **shell**基础

❖ **shell**脚本

❖ **shell**变量

❖ **shell**编程基础

❖ **shell**控制结构

❖ **shell**函数

# Shell 函数

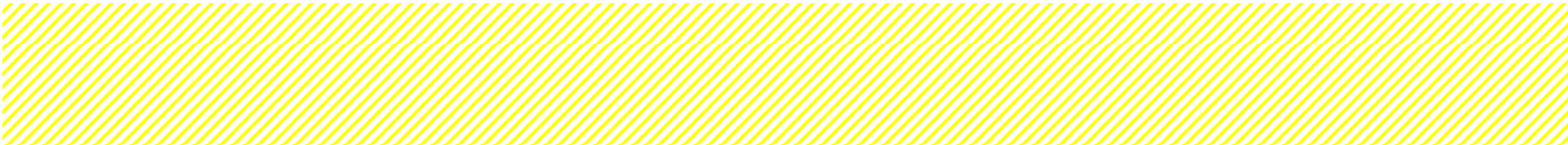
❖ **shell**除了可以定义变量外，还可以定义函数。

❖ 定义一个**shell**函数，语法格式如下：

```
函数名()  
函数体  
}
```

❖ 调用自定义函数：

- 在使用函数前必须先定义该函数，使用时利用函数名直接调用。



```
#!/bin/bash
func1(){
    echo "this is sub function"
}

echo "main function start"
func1
echo "main function end"
```

```
jkx@ubuntu:~/work/chapt5$ ./func_script
main function start
this is sub function
main function end
```

# 课堂作业

(1) 当前目录有一名为script的可执行脚本，不可以运行的方法是：

- ❖ A、 ./script
- B、 sh script
- C、 . script
- D、 script

(2) 命令ls /home/`whoami`/ 的执行结果是什么？

whoami:打印出当前正在操作的用户名

(3) 写出命令，查找当前用户主目录下所有以d开头的文件并排序，只查找一层，不递归



# 作业

❖ 课堂作业：P183/10

❖ P182/2, 4, 9

❖ 附加题：设计一个shell脚本：求命令行上整数的最大值和最小值。



武汉纺织大学  
WUHAN TEXTILE UNIVERSITY

崇真尚美



Thank You !

