第03章 Shell操作

讲师：武永亮

课程目标

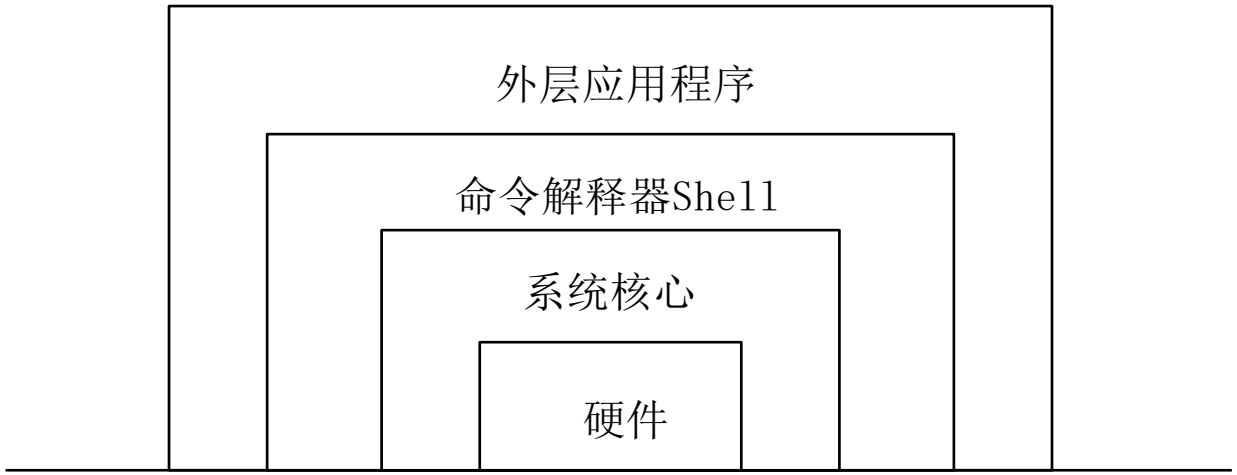
- 了解Shell的基本概念
- 掌握Shell的基本操作
- 掌握Shell环境及环境

课程内容

- Shell简介
- Shell基本用法
- Shell变量及环境

Shell

- Shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口(命令解释器)。它接收用户输入的命令并把它送入内核去执行。起着协调用户与系统的一致性和在用户与系统之间进行交互的作用。
- Shell在Linux系统上具有极其重要的地位。

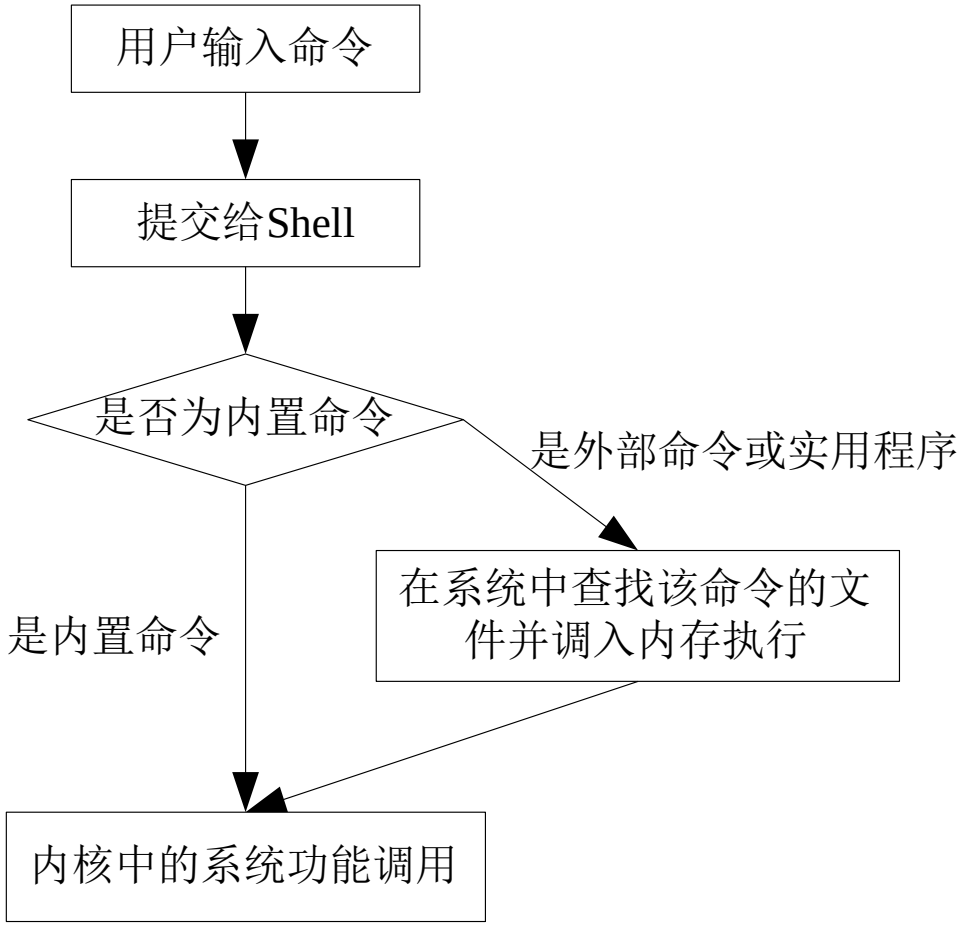


Shell的重要功能

- 命令行解释
- 命令的多种执行顺序
- 通配符（ wild-card characters ）
- 命令补全、别名机制、命令历史
- I/O重定向（ Input/output redirection ）
- 管道（ pipes ）
- 命令替换（``或$( )）
- Shell编程语言（ Shell Script ）

命令解释过程

- Shell可以执行
  - ✓内部命令
  - ✓应用程序
  - ✓Shell脚本



Shell的主要版本

Bash（Bourne Again Shell）	**bash是大多数Linux系统的默认Shell。 bash与bsh完全向后兼容，并且在bsh的基础上增加和增强了很多特性。 bash也包含了很多C Shell和Korn Shell中的优点。 bash有很灵活和强大的编程接口，同时又有很友好的用户界面**
Ksh（Korn Shell）	Korn Shell（ksh）由Dave Korn所写。它是UNIX系统上的标准Shell。 在Linux环境下有一个专门为Linux系统编写的Korn Shell的扩展版本，即Public Domain Korn Shell（pdksh）。
tcsh （csh 的扩展）	tcsh是C Shell的扩展。tcsh与csh完全向后兼容，但它包含了更多的使用户感觉方便的新特性，其最大的提高是在命令行编辑和历史浏览方面

Linux的元字符

- 在 Shell 中有一些具有特殊的意义字符，称为 Shell 元字符（ shell metacharacters ）。
- 若不以特殊方式（使用转义字符）指明，Shell并不会把它们当做普通文字符使用。

字符	含义	字符	含义
‘	强引用	*、?、！	通配符
“	弱引用	<、>、>>	重定向
\	转义字符	-	选项标志
$	变量引用	#	注释符
;	命令分离符	空格、换行符	命令分隔符

课程内容

- Shell简介
- Shell基本用法
- Shell变量及环境

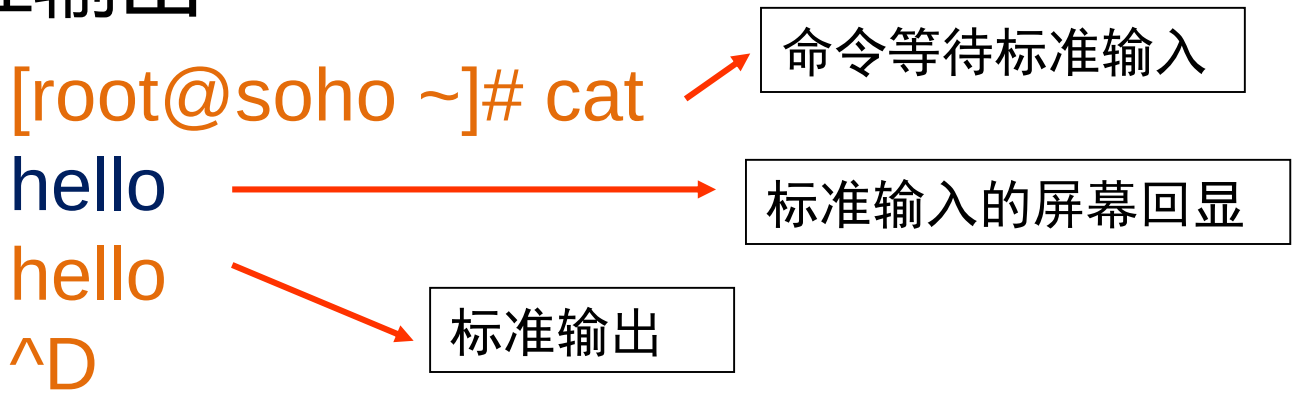
标准输入/输出设备

- Linux命令执行时接收输入数据，命令执行后将产生的数据结果输出
- Linux的大部分命令都具有标准的输入/输出设备端口

名称	文件描述符	含义	设备	说明
STDIN	0	标准输入	键盘	命令在执行时所要的输入通过它来取得
STDOUT	1	标准输出	显示器	命令执行后的输出结果从该端口送出
STDERR	2	标准错误	显示器	命令执行时的错误信息通过该端口送出

标准输入/输出举例

- 标准输入和标准输出



- 标准错误输出

重定向（Redirection ）

- 重定向：不使用系统的标准输入输出端口或标准错误端口，而进行重新的指定，所以重定向分为输出重定向、输入重定向和错误重定向。通常情况下重定向到一个文件
- 在Shell中通过重定向符实现重定向

重定向符

重定向符	说明
<	输入重定向
<<! …… !	输入重定向的特例，即HERE文件，通常用于Shell脚本中。其中“!”可以使用任何字符或字符串替换，只要其没在……中出现过即可。
>	覆盖式的输出重定向
>>	追加式的输出重定向
2>	覆盖式的错误输出重定向
2>>	追加式的错误输出重定向
&>	同时实现输出重定向和错误重定向（覆盖式）

重定向举例

```
$ tr 'a-z' 'A-Z' <a.txt
$ ls -l /tmp >myfile
$ ls -l /etc >>myfile

$ myprogram 2> err_file
$ myprogram &> output_and_err_file

$ find  ~  -name *.mp3 > ~/cd.play.list
$ echo “Please call me : 68800000”>message
```

管道的引入

- UNIX 系统的一个基本哲学是：一连串的小命令能够解决大问题。其中每个小命令都能够很好地完成一项单一的工作。现在需要有一些东西能够将这些简单的命令连接起来，这样管道就应运而生。
- Linux命令具有过滤特性，即一条命令通过标准输入端口接受一个文件中的数据，命令执行后产生的结果数据又通过标准输出端口送给后一条命令，作为该命令的输入数据。后一条命令也是通过标准输入端口而接受输入数据。

管道（ Pipe ）

- 管道（使用符号“|”表示）用来连接命令
  - ✓命令1 | 命令2
  - ✓将命令1的STDOUT发送给命令2的STDIN
  - ✓STDERR不能通过管道转发
  - ✓例如：ls| tr 'a-z' 'A-Z‘ | wc -w

由于管道线中的命令总是从左到右顺序执行的，因此管道线是单向的

管道应用举例

管道应用举例（1）

```
$ ls -lR /etc | less
$ tail +15 myfile | head -3
$ man bash | col -b > bash.txt
# echo "p4ssW0rd" | passwd --stdin user1

$ ls -l | grep "^d"
$ cat /etc/passwd | grep username
$ dmesg | grep eth0
$ rpm –qa | grep httpd

$ echo "test email" | mail -s "test" user@example.com
$ echo "test print" | lpr
```

管道应用举例（2）

- 统计磁盘占用情况

  ✓统计当前目录下磁盘占用最多的10个一级子目录

  $ du . --max-depth=1 | sort -rn | head -11

  ✓以降序方式显示使用磁盘空间最多的普通用户的前十名

  $ du * -cks | sort -rn | head -11

  ✓以排序方式查看当前目录（不包含子目录）的磁盘占据情况。

  $ du -S | sort -rn | head -11

管道应用举例（3）

- 统计进程

  ✓按内存使用从大到小排列输出进程。

  # ps -e -o "%C : %p : %z : %a"|sort -k5 -nr

  ✓按CPU使用从大到小排列输出进程。

  # ps -e -o "%C : %p : %z : %a"|sort -nr

管道应用举例（4）

- 列出YUM仓库中所有可用的 Apache 模块并按升序输出

```
# yum list | grep ^mod_ | cut -d'.' -f 1 | sort
# yum list | grep ^mod_ | awk -F\. '{print $1}' | sort
```

- 以排序方式列出YUM仓库中在 /etc/httpd/conf.d/ 目录下生成配置文件的所有 Web 应用软件包（不包含 Apache 模块）

```
# repoquery --queryformat="%{NAME}\n"  \
  --whatprovides “/etc/httpd/conf.d/*” |  \
 egrep -v  "(^$|^mod)" | sort | uniq
```

管道应用举例（5）

- 从ifconfig 命令的输出过滤出 eth0 网络接口当前的IPv4地址

  ✓# ifconfig eth0 | awk -F\: '/inet / {print $2}'|awk '{print $1} '

  ✓# ifconfig eth0 | grep 'inet ' | awk -F '[:]+' ' {print $4}'

  ✓# ifconfig eth0 | grep -i 'inet[^6]' | sed 's/[a-zA-Z:]//g' | awk '{print $1}‘

- 从ip命令的输出过滤出 eno16777736网络接口当前的IPv4地址

  ✓# ip a s eno16777736|grep 'inet '| awk -F '[/]+' '{print $3}'

T型管道（tee）

- 格式
  - ✓命令1 | tee　文件名 | 命令2
- 功能
  - ✓将命令1的STDOUT保存在文件名中，然后管道输入给命令2
- 用于
  - ✓保存不同阶段的输出
  - ✓复杂管道的故障排除
  - ✓同时查看和记录输出

命令替换（ Command Substitution ）

- 使用命令的输出，常用于
  - ✓在文本中嵌入命令的执行结果
  - ✓命令参数是另一个命令执行的结果
- 使用方法

  $(command)　　或　　`command`

  cmd1 $(cmd2)　　或　　cmd1 `cmd2`
- 使用举例

  $ echo The present time is `date`

  $ rpm -qi $(rpm -qf $(which date))　　**# 嵌套**

命令组合

命令行形式	说明	举例
CMD1 ; CMD2	顺序执行若干命令	pwd;date;ls
CMD1 && CMD2	当CMD1运行成功时才运行CMD2	gzip mylargefile && echo “OK.”
CMD1 || CMD2	当CMD1运行失败时才运行CMD2	write osmond || mail -s test osmond < my.log
(CMDLIST)	在子Shell中执行命令序列	(date; who | wc -l) > ~/login-users.log
{CMDLIST}	在当前Shell中执行命令序列	{ cd /home/jjh; chown jjh:bin s* ;}

课程内容

- Shell简介
- Shell基本用法
- Shell变量及环境

Shell 变量

- Shell 变量大致可以分为三类
  - ✓**内部变量**：由系统提供，用户只能使用不能修改。
  - ✓**用户变量**：由用户建立和修改，在 shell 脚本编写中会经常用到。
  - ✓**环境变量**：这些变量决定了用户工作的环境，它们不需要用户去定义，可以直接在 shell 中使用，其中某些变量用户可以修改。

用户自定义变量

- 变量赋值（定义变量）
  - ✓varName=Value
  - ✓export varName=Value
- 引用变量 $varName

- 一般地，所有的Shell变量都是字符串。
- 当变量的值仅仅包含数字时才允许进行数值计算。
- 在较新的 bash 中，可是使用 declare 或 typeset 命令声明变量及其属性，但一般不需要声明。而且为了使脚本兼容于不同的 shell，在没有必要的情况下尽量不使用变量声明。

引用

- 在 bash 中，有些字符具有特殊含义，如果需要忽略这些字符的特殊含义，就必须使用引用技术。
- 引用可以通过下面三种方式实现
  - ✓ 使用转义字符：\
  - ✓ 使用单引号： ‘ ’
  - ✓ 使用双引号： “ ”
- 转义字符的引用方法就是直接在字符前加反斜杠。例：\$，\‘，\“，\\，\ ，\!

强引用和弱引用

- 强引用
  - ✓单引号对是强引用
  - ✓单引号对中的字符都将作为普通字符，但不允许出现另外的单引号
- 弱引用
  - ✓双引号对是弱引用
  - ✓双引号对中的部分字符仍保留特殊含义
    - $（美元符号） - 变量扩展
    - `（反引号） - 命令替换
    - \（反斜线） - 禁止单个字符扩展
    - !（叹号） - 历史命令替换

命令行执行过程

1. 将命令行分成单个命令词
2. 展开别名
3. 展开大括号中的声明（{}）
4. 展开颚化声明（~）
5. 命令替换（ $() 或 ``）
6. 再次把命令行分成命令词
7. 展开文件通配（*、?、[abc]等等）
8. 准备I/0重定向（<、>）
9. 运行命令！

Shell 变量的作用域

- 局部变量的作用范围仅仅限制在其命令行所在的Shell或Shell脚本文件中；
- 全局变量的作用范围则包括本Shell进程及其所有子进程。
- 可以使用 export 内置命令将局部变量设置为全局变量。
- 可以使用 export 内置命令将全局变量设置为局部变量。

export 命令

- 显示当前Shell可见的全局变量
  - ✓export [-p]
- 定义变量值的同时声明为全局变量
  - ✓export <变量名1=值1> [<变量名2=值2> ...]
- 声明已经赋值的某个（些）局部变量为全局变量
  - ✓ export <变量名1> [<变量名2> ...]
- 声明已经赋值的某个（些）全局变量为局部变量
  - ✓ export -n <变量名1> [<变量名2> ...]

Shell环境变量

- 环境变量定义 Shell 的运行环境,保证 Shell 命令的正确执行.
- Shell用环境变量来确定查找路径、注册目录、终端类型、终端名称、用户名等。
- 所有环境变量都是全局变量(即可以传递给 Shell 的子进程),并可以由用户重新设置。

常见的 Shell 环境变量

变量名	含义
HOME	用户主目录
LOGNAME	登录名
USER	用户名，与登录名相同
PWD	当前目录/工作目录名
MAIL	用户的邮箱路径名
HOSTNAME	计算机的主机名
INPUTRC	默认的键盘映像
SHELL	用户所使用的 shell 的路径名
LANG	默认语言
HISTSIZE	history 所能记住的命令的最多个数
PATH	shell 查找用户输入命令的路径 (目录列表)
PS1、PS2	shell 一级、二级命令提示符

Shell变量的查询、显示和取消

- 显示当前已经定义的所有变量
  - ✓所有环境变量：env
  - ✓所有变量和函数（包括环境变量）：set
- 显示某（些）个变量的值
  - ✓echo $NAME1 [$NAME2 ……]
- 取消变量的声明或赋值
  - ✓unset <NAME>

用户工作环境

- 用户登录系统时，Shell为用户自动定义唯一的工作环境并对该环境进行维护直至用户注销。
  - ✓该环境将定义如身份、工作场所和正在运行的进程等特性。这些特性由指定的环境变量值定义。
- 用户工作环境有**登录环境**和**非登录环境**之分。
  - ✓**登录环境**是指用户登录系统时的工作环境，此时的Shell对登录用户而言是主Shell。
  - ✓**非登录环境**是指用户再调用子Shell时所使用的用户环境。

设置用户工作环境

- 对所有用户进行设置
  - ✓/etc/profile
  - ✓/etc/bashrc
- 只对当前用户进行设置
  - ✓~/.bash_profile
  - ✓~/.bashrc

通常，个人bash 环境设置都定义在 ~/.bashrc 文件里

登录 shell 和非登录 shell的启动过程

- Login shell

  /etc/profile → /etc/profile.d/*.sh

  $HOME/.bash_profile

  $HOME/.bashrc → /etc/bashrc

- Non-Login shell

  $HOME/.bashrc → /etc/bashrc

课程总结

- Shell简介
- Shell基本用法
- Shell变量及环境

本章思考题

- 常用的文件和目录操作命令有哪些？各自的功能是什么？
- 常用的信息显示命令有哪些？各自的功能是什么？
- 打包和压缩有何不同？常用的打包和压缩命令有哪些？
- 简述在Shell中可以使用哪几种方法提高工作效率。
- Linux下的隐含文件如何标识？如何显示？
- Linux下经常使用-f和-r参数，它们的含义是什么？
- Vi的3种运行模式是什么？如何切换？
- 什么是重定向？什么是管道？什么是命令替换？
- Shell变量有哪几种？如何定义和引用Shell变量？
- 登录Shell和非登录Shell的启动过程？
- 如何设置用户自己的工作环境？

THANK YOU!