Linux操作系统

5 Shell编程基础

主讲：杨东平
中国矿大计算机学院

了解 Shell

- Shell
  - Shell 提供了用户与内核进行交互操作的一种接口，它接收用户输入的命令并把它送入内核去执行
    - 既接收以命令行方式输入的命令(包括系统提供的内部命令、独立存在于某个目录下的程序)，也能执行由 Shell 命令编写的 Shell 程序
- Shell 最主要的功能是解释执行各种命令

Shell 脚本

- Shell 是一种应用程序，它提供了一个界面，用户通过这个界面访问操作系统内核的服务
  - (1) Shell 既是一种命令解释器(交互模式)
    - 能解释并执行命令行提示符下输入的命令
  - (2) Shell又是一种脚本编程语言(非交互模式)
    - 提供了将命令组合成复杂功能的机制，这就是Shell 脚本(Shell Script)，是一种为 Shell 编写的脚本程序
    - Shell 脚本不仅是命令的简单组合，它还具有高级程序设计语言的特点：支持变量、命令行参数、交互式输入、函数、数组及程序的控制结构等
    - Shell脚本以文本方式保存，并由Shell进行解析执行，但Shell脚本文件必须具有可读和可执行权限
  - 业界所说的 Shell 通常都是指 Shell 脚本，但要知道，Shell 和 Shell Script 是两个不同的概念
  - 由于习惯的原因，简洁起见，后文出现的“Shell 编程" 都是指 Shell 脚本编程，不是指开发 Shell 自身

Shell 的种类

- Shell 编程跟其它语言编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了
- bsh(Bourne Shell)——Unix，1979
  - 最基本、较简单、编程能力强、但操作使用不够方便，主要用于系统管理任务的自动化
- csh(C Shell)
  - Bill Joy编写，采用”类C”语法
- ksh(Korn Shell)
  - 完全兼容bsh并且包含了csh的很多特性，功能更强大
- Bash(Bourne Again Shell)——1987,Linux 默认的 Shell
  - 继承 bsh 的标准、扩充人机交互的特性
  - 提供命令历史查阅功能
  - 命令补全、命令编辑

用户Shell

- 用户可以选择自己喜欢的 Shell(在系统管理员为用户开帐号时指定)
  - 在 /etc/passwd 文件中可以看到用户使用的 Shell 的名称
- 查看系统支持哪些 Shell：
  - 语法：cat /etc/shells
- 查看自己的Shell类型：
  - 语法：# echo $SHELL
  - 说明：$SHELL是一个环境变量，它记录用户所使用的Shell类型

Shell 的启动和退出

- 用户在成功登录进入系统后，系统产生一个特定的 Shell，这是用户的第一个进程
- 用户希望中止命令或脚本的执行，可以用 “Ctrl+C”
- 退出Shell：
  - (1) 快捷键：Ctrl+D
  - (2) 命令：exit

Shell 命令

Ø Shell 可运行的命令：

- v 内部命令
  - F 出于运行效率的考虑，将一些命令构造在 Shell 的内部，这些命令比非内部命令执行速度快，如：read、cd、echo、eval、exec、exit、export
- v 外部命令：
  - F 文件系统中的命令，程序可执行文件，可在 $PATH 环境变量的目录中搜索到
- v Shell 脚本命令
  - F (1) Linux 别名
    - u 通过Shell内置命令alias创建的用户自定义命令，例：alias ll='ls -l --color=auto'
  - F (2) Linux Shell 保留字：如if、then、fi、for、while、case、esac、else、until等在 shell 中有特殊的含义
  - F (3) Linux Shell 函数

Shell 对命令的解释执行过程

二个 Shell 例子

Ø 例1：

- v 可以在一个命令行上键入多个命令 ,本例中的命令列表用分号(“;”)做间隔符，如：

$ cd /boot; ll

二个 Shell 例子(续)

Ø 例2：第一个shell程序

- v 1 在 vi 编辑器输入以下源代码并存储到 hello.sh 文件中：

```
#! /bin/bash
echo "Hello World !"
```

  - F #! /bin/bash 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell
  - F echo 命令用于向窗口输出文本

二个 Shell 例子(续)

Ø 例2：第一个shell程序

- v 2 赋予Shell脚本执行权限
  - F 命令：chmod 权限 脚本文件名
  - F 赋予 hello.sh 执行权限：
    chmod 755 hello.sh
- v 3 执行Shell脚本
  - F (1) 直接执行，语法：./脚本文件名
    - u 例： ./hello.sh
    - u 注意：文件的第一行需要“#! /bin/bash”标记
  - F (2) 用 bash 执行，语法：bash 脚本文件名
    - u 例：bash hello.sh
    - u 此方式不要求脚本的第一行为解释器信息

例2 视频

bash 的基本功能

- (1) 命令别名与快捷键
- (2) 历史命令
- (3) 输出重定向
- (4) 多命令顺序执行
- (5) Shell 中的特殊符号

别名

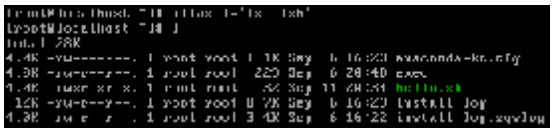
- 命令别名通常是其他命令的缩写，用来减少键盘输入

- 查看系统中所有的别名命令
  - 语法：alias

```
[root@localhost ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-ti
lde'
```

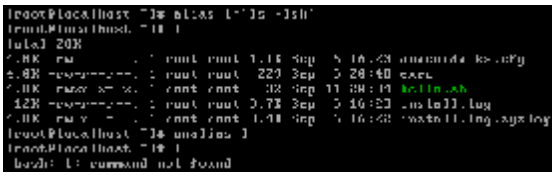

别名(续)

- 定义命令别名
  - 语法：alias 别名=原命令
    alias 别名="原命令 选项和参数"
  - 说明：
    - 别名的内容可以包括命令名、选项和参数
    - 这个别名命令是临时的，系统重启后失效



别名(续)

- 取消别名
  - 语法：unalias 别名



别名(续)

- 让别名永久生效
  - 将命令的别名写入环境变量配置文件(~/.bashrc)，就不需每次开机都重新定义别名了
    - ~/.bashrc 文件可以用 vi 编辑、修改和保存

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```


命令的执行顺序

- 命令执行时的顺序：
  - (1) 第一顺位执行用路径或相对路径执行的命令
  - (2) 第二顺位执行别名
  - (3) 第三顺位执行 Bash 的内部命令
  - (4) 第四顺位执行按照 $PATH 环境变量定义的目录查找顺序找到的命令

常用快捷键

ctrl+c	强制终止当前命令
ctrl+l	清屏
ctrl+a	光标移动到命令行首
ctrl+e	光标移动到命令行尾
ctrl+u	从光标所在位置删除到行首
ctrl+z	把命令放入后台
ctrl+r	在历史命令中搜索

命令和文件名的自动补全

- 当输入到足够多的字符时，按“Tab”键就可以帮你补全一个指令，也可以帮你补全一个路径或者一个文件名
- 连续按两次 “Tab” 键，系统会把所有以你输入的字符为开始的命令或者文件名都列出来

历史命令

- Linux 会记录已执行过的命令
  - 历史命令保存在用户家目录的 ~/.bash_history 文件中
  - 只有当用户正常退出当前 Shell 时，在当前 Shell 中运行的命令才会保存至 .bash_history 文件中
- 历史命令
  - 语法：history [选项] [历史命令保存文件]
  - 选项：
    - -c 清空历史命令
    - -w 把缓存中的历史命令写入历史命令保存文件

历史命令(续)

- 默认记录 1000 条历史命令，历史命令记录条数在环境变量配置文件 /etc/profile 文件中的 HISTSIZE 变量中设置

```
HOSTNAME=`/bin/hostname 2>/dev/null`
HISTSIZE=1000
if [ "$HISTCONTROL" = "ignorespace" ] ; then
    export HISTCONTROL=ignoreboth
else
    export HISTCONTROL=ignoredups
fi
```

```
[root@localhost ~]# echo $HISTSIZE
1000
```


历史命令的调用

- (1) 使用上、下箭头调用以前的历史命令
- (2) 使用 !n 重复执行第 n 条历史命令
- (3) 使用 !! 重复执行上一条命令
- (4) 使用 !字串 重复执行最后一条以该字串开头的命令

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


输入/输出的重定向

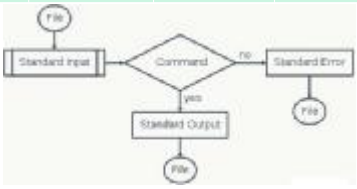
- Linux系统中程序分类
  - 图形程序：
    - 为图形环境设计，通常希望用户使用鼠标，并且使用常用的用的图形组件，比如弹出窗口和按键作为用户输入
  - 屏幕程序：
    - 基于屏幕的程序使用文本控制台，这种程序使用整个屏幕处理文本定位和屏幕刷新，不需要使用鼠标，也适用于终端和虚拟控制台，如vi
  - 终端程序：
    - 以流的形式聚集输入和显示输出，很少刷新屏幕，终端程序以其简明的特点经常被称为命令
- 这里使用的是终端程序

输入/输出的重定向(续)

Ø标准输入输出的重定向：把原从标准输入/输出的数据改为从另一个文件输入/输出

Ø标准输入输出

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



标准输出重定向

类型	符号	作用
标准输出重定向	命令 > 文件	以覆盖方式把命令的正确输出结果输出到指定的文件中，文件不存在则创建文件，存在则清空
	命令 >> 文件	以追加方式把命令的正确输出结果输出到指定的文件中，文件不存在则创建文件，存在则追加到尾部
标准错误输出重定向	错误命令 2> 文件	以覆盖方式把命令的错误信息输出到指定的文件中
	错误命令 2>> 文件	以追加方式把命令的错误信息输出到指定的文件中
正确输出和错误输出同时保存	命令 > 文件 2>&1	以覆盖方式把正确输出和错误信息同时保存到同一个文件中，& 表示等同于，2>&1 表示 2 的输出重定向等同于 1，即和1重定向的文件在系统上的位置是一样的
	命令 >> 文件 2>&1	以覆盖方式把正确输出和错误信息同时保存到同一个文件中
	命令 > 文件1 2> 文件2	以覆盖方式把正确输出出到文件1中，把错误信息输出到文件2中
	命令 >> 文件1 2>> 文件2	以追加方法把正确输出出到文件1中，把错误信息输出到文件2中

输入重定向

Ø标准输入的重定向：

- v 语法：命令 < 文件
  - F < 是输入重定向符，表示命令原从标准输入读入数据，现改为从“文件”读入数据
- v 例：

wc [选项] [文件名]

- F 英文原文：Word Count

-c：统计字节数 -w：统计单词数 -l：统计行数

- F 接收键盘输入时以ctrl+d 结束输入
- F 若不指定文件名称，或文件名为“-”，则 wc 指令从标准输入设备读取数据(以 ctrl+d 结束)，并显示统计结果

wc < test.log

- F 接收文件输入

多命令顺序执行(命令列表)

Ø命令列表

- v 一次提供多个命令执行，有三种实现方式：

多命令执行符	格式	作用
;	命令1; 命令2	多个命令顺序执行，命令之间没有任何逻辑联系
&&	命令1&& 命令2	逻辑与 当命令1正确执行，则命令2才会执行 当命令1执行不正确，则命令2不会执行
||	命令1|| 命令2	逻辑或 当命令1执行不正确，则命令2才会执行 当命令1正确执行，则命令2不会执行

管道符

Ø语法：命令1 | 命令2

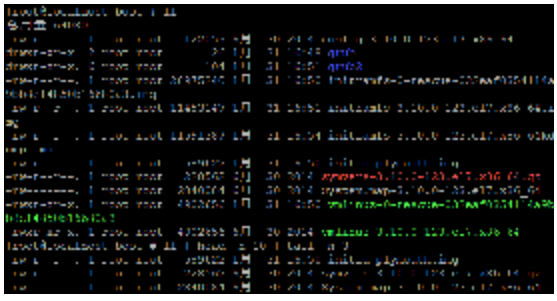
Ø功能：命令1 的正确输出作为命令2 的操作对象(输入)

Ø注意：管道命令操作符对错误信息没有直接处理能力

Ø管道是一个非常有用的功能，Linux中有很多命令会输出大量的信息，为了方便阅读可以通过管道对显示的内容进行过滤

管道符(续)

Ø# ll | head -n 10 | tail -n 3



通配符

通配符	作用
?	匹配一个任意字符
*	匹配 0 个或多个任意字符，也就是可以匹配任何内容
[]	匹配括号中任意一个字符。如：[abc] 代表一定匹配其中的一个字符，或者是 a 或者是 b 或者是 c
[-]	匹配括号中任意一个字符，“-” 代表范围。例如：[a-z] 代表匹配一个小写字母。
[^]	逻辑非，表示匹配不是括号中的一个字符。例如：[^0-9] 代表匹配一个不是数字的字符。

Bash 中其他特殊符号

符号	作用
''	在单引号中所有的特殊符号，如'$'和``(反引号)都不表示特殊含义
“”	在双引号中的特殊符号都没有特殊含义，但是 "$"、``(反引号)和"\"是例外，它们具有"调用变量的值"、"引用命令"和"转义符"的特殊含义
``	反引号括起来的内容是系统命令，在 Bash 中会先执行，它和 $() 作用相同，推荐使用 $()，因为反引号非常容易看错。
$()	和反引号作用相同，用来引用系统命令
#	在 Shell 脚本中，# 开头的行代表注释
$	调用变量的值，如需要调用变量 name 的值时，需要用 $name 的方式得到变量的值
\	转义符，跟在 \ 之后的特殊符号将失去特殊含义，变为普通字符。如 \$ 将输出 "$" 符号，而不当作是变量引用。