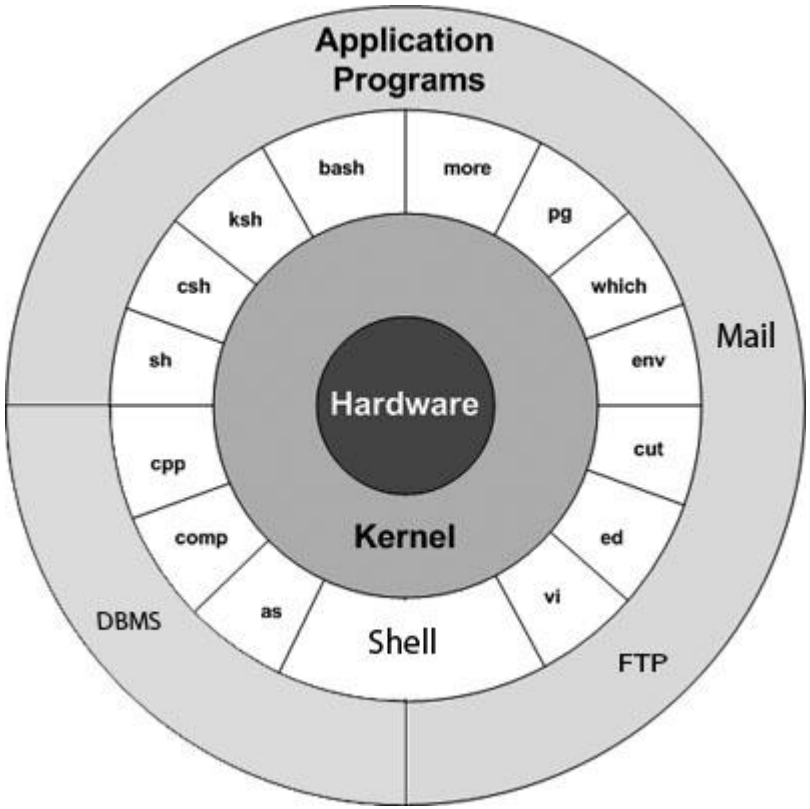
Shell脚本编程基础



内容

①

- Shell概述
  - Shell的概念
  - Shell的特点
  - Shell版本
  - Shell程序示例
  - Shell程序执行方法
- Linux命令帮手
  - 命令行编辑
  - screen工具

②

- Shell特殊字符
  - 通配符
  - 正则表达式
  - 引号
  - 转义符与路径符
  - 输入输出重定向
  - 注释和后台命令
  - 命令组合符
  - 成组命令

③

- Shell变量
  - 用户自定义变量
  - 位置变量
  - Shell预定义变量
  - 环境变量

④

- 算术运算
- 控制结构
- 函数

⑤

- 作业控制
- Shell内置命令
- Shell脚本调试
- Shell脚本示例

Shell概述

- Shell的概念
- Shell的特点
- Shell版本
- Shell程序示例
- Shell程序的执行

Shell概念

- Linux shell是Linux系统的用户界面，是连接Linux内核与用户之间的桥梁，是Linux命令的解释程序。同时，Shell也是一种高级程序设计语言，有变量、关键字、控制结构和函数等程序设计语言要素。

Shell的特点

- 利用控制结构（如判断、循环等）和管道将Linux命令组合起来
- 使用正则表达式
- 可以直接使用shell内置命令，而无须创建新的进程
- 灵活使用数据流，提供管道和输入/输出重定向机制，方便数据处理
- 提供后台执行命令方式
- 用户可以配置环境

Shell版本

- **Bourne shell (sh)**
  - Steven Bourne (AT&T Bell)
- **C shell (csh)**
  - Bill Joy (美国加州大学伯克利分校)
- **Korn shell (ksh)**
  - David Korn (AT&T Bell)
- **Bourne Again shell (bash)**
  - GNU
- **Debian Almquist Shell (dash)**
  - 简化版的bash，Ubuntu默认的shell

Shell程序示例

```
$ cat shell_example.sh        #显示脚本内容
mkdir animal
cd animal
mkdir bird
touch fox

$ sh shell_example.sh         #运行脚本
```

Shell程序的执行

4种方式（shell脚本文件名为example.sh）：

- sh <example.sh 或 bash <example.sh
- sh example.sh 或 bash example.sh
- path/example.sh （需有执行权限并在脚本文件第一行指定解释器的路径，如：#!/bin/bash）
- . example.sh 或 source example.sh

CentOS中，sh是bash的符号链接；
Ubuntu中，sh是dash的符号链接

Linux命令帮手：命令行编辑

- 命令行自动补齐
- 命令历史
- 光标跳转
- 命令编辑

自动补齐

- 命令自动补齐

  $ history　　#输入到his时按Tab键，看看会发生什么

- 文件名自动补齐

  $ cat hello.c　　#输入到cat h时按Tab键，看看会发生什么。如果没有什么发生，就再按一次Tab键看看！

命令历史

- 显示历史命令：
  - history [num]
  - ↑/ctrl+p, ↓/ctrl+n
  - ctrl+r
- 执行历史命令：
  - !!或!-1　重复上一条命令
  - !n　重新执行第n条历史命令
  - !-n　重新执行倒数第n条历史命令
  - !string　重新执行以字符串string开头的最近的历史命令
  - !?string?　重新执行包含string的最近的历史命令

光标跳转

- Ctrl + a　跳转至命令行首　（Ahead of line）
- Ctrl + e　跳转至命令行尾　（End of line）
- Ctrl + f　向前跳转一个字符　（Forward）
- Ctrl + b　向后跳转一个字符　（Backward）
- Alt + f　向前跳转到下一个字的第一个字符
- Alt + b　向后跳转到下一个字的第一个字符

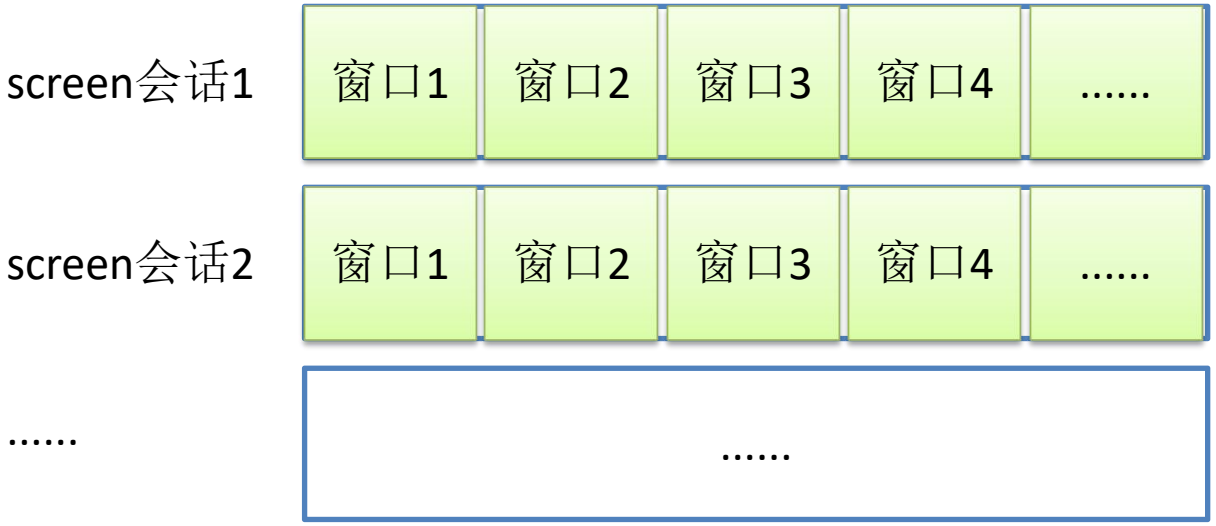
命令编辑

- Ctrl + w　　删除一个词
- Ctrl + u　　删除光标前面的所有字符
- Ctrl + k　　删除光标所在位置及其后的所有字符
- Ctrl + d　　删除光标所在位置的字符
- Ctrl + c　　取消本次命令输入
- Alt + d　　删除从光标当前位置，到当前词的结尾字符
- Ctrl + y　　粘贴最后一次删除的内容

Linux命令帮手：screen工具

- 我们常需要SSH 或者telent 远程登录到Linux 服务器，运行一些需要很长时间才能完成的任务，在此期间不能关掉窗口或者断开连接，否则这个任务就会被杀掉，半途而废。
- 我们可以用screen命令解决这个问题。screen命令可以实现当前窗口与任务分离，即使我们离线了，服务器仍在后台运行任务。当我们重新登录服务器，可以读取窗口线程，重新连接任务窗口。

screen的会话与会话窗口




```
$ screen -ls
There are screens on:
        166.tty1.DESKTOP-QONARUO        (11/14/22 09:07:58)     (Attached)
        157.tty1.DESKTOP-QONARUO        (11/14/22 09:07:45)     (Detached)
2 Sockets in /run/screen/S-xiezy.
```

screen的使用方法

会话操作：

- 创建screen会话：screen
- 列出screen会话：screen -ls
- 恢复screen会话：screen -r [n]
- 退出screen会话：screen -d [n]　#在会话外脱离已绑定（attached）的会话
- 脱离screen会话：Ctrl+a, d　　#在会话中脱离（detach）该会话

会话中的窗口操作：

- 创建会话窗口：Ctrl+a, c
- 会话窗口切换：
  - Ctrl+a, w　显示所有窗口列表
  - Ctrl+a, n　切换到下一个窗口
  - Ctrl+a, p　切换到前一个窗口
  - Ctrl+a, 0..9　切换到第0..9个窗口
  - Ctrl+a, [Space]　由窗口0循序切换到窗口9
  - Ctrl+a, Ctrl+a　在两个最近使用的窗口间切换
- 退出会话窗口：exit

内容

①

- Shell概述
  - Shell的概念
  - Shell的特点
  - Shell版本
  - Shell程序示例
  - Shell程序执行方法
- Linux命令帮手
  - 命令行编辑
  - screen工具

②

- Shell特殊字符
  - 通配符
  - 引号
  - 转义符与路径符
  - 输入输出重定向
  - 注释和后台命令
  - 命令组合符
  - 成组命令

③

- Shell变量
  - 用户自定义变量
  - 位置变量
  - Shell预定义变量
  - 环境变量

④

- 算术运算
- 控制结构
- 函数

⑤

- 作业控制
- Shell内置命令
- Shell脚本调试
- Shell脚本示例

Shell特殊字符

- 通配符
- 正则表达式
- 引号
- 转义符与路径符
- 输入输出重定向
- 注释和后台命令
- 命令组合符
- 成组命令

通配符

*（星号）	如file*、*.fa	#任意个任意字符
?（问号）	如file?.txt	#1个任意字符
[字符集合]	如[13579] [agh]	#集合中的任意一个字符
[字符范围]	如[0-9] [a-z]	#范围中的任意一个字符
!或^	如file[!3-9]	#除了某范围或集合中的任意一个字符

通配符实例

$ ls

ant antilope bat bear camel cat cow crab dog duck fish fox frog hog pig wolf

$ ls *g

dog frog hog pig

ant antilope bat bear camel cat cow crab dog duck fish fox frog hog pig wolf

$ ls ?og

dog hog

ant antilope bat bear camel cat cow crab dog duck fish fox frog hog pig wolf

$ ls [a-c]*

ant antilope bat bear camel cat cow crab

ant antilope bat bear camel cat cow crab dog duck fish fox frog hog pig wolf

$ ls [!a-c]*

dog duck fish fox frog hog pig wolf

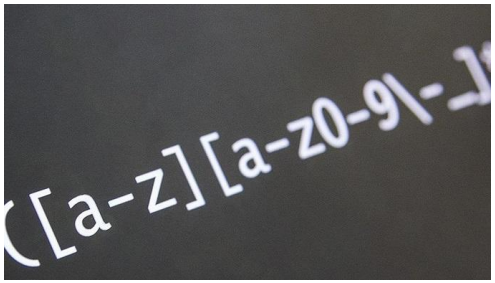
$ ls [dhp]*g

dog hog pig

ant antilope bat bear camel cat cow crab dog duck fish fox frog hog pig wolf

正则表达式

- 正则表达式（Regular Expression）是一种文本模式，包括普通字符和特殊字符（称为“元字符”）。正则表达式使用单个字符串来描述、匹配一系列符合某个规则的字符串。
- grep、sed、vi、less等命令都支持正则表达式，另外Perl、Python、PHP、JavaScript、R、Ruby、AWK等脚本语言及Java、C#、Delphi等编译语言也都支持正则表达式。
- 正则表达式包括基本正则表达式和扩展的正则表达式。



基本正则表达式

元字符	含义
*	匹配之前的项0次或多次
.	匹配除了换行符外的任意一个字符
^	匹配行首，如^M 即 匹配以M开头的行
$	匹配行尾，如G$ 即 匹配以G结尾的行
[]	匹配中括号中指定的任意一个字符，如[aeiou]匹配任意一个元音字母，[a-z][0-9]匹配一个小写字母和一位数字构成的两个字符
[^]	匹配除中括号内的字符以外的任意一个字符
\	转义符，用于取消特殊符号的含义

基本正则表达式示例

```
$ cat nucleic_acid
DNA
mRNA
tRNA
rRNA
snRNA
snoRNA
miRNA
lncRNA
```

```
$ grep ‘[mt]RNA’ nucleic_acid
mRNA
tRNA
```

```
$ grep ‘[^mtr]RNA’ nucleic_acid
snRNA
snoRNA
miRNA
lncRNA
```

扩展的正则表达式

元字符	含义
?	匹配之前的项0次或1次
+	匹配之前的项1次或多次
()	匹配表达式，创建一个用于匹配的字串
{n}	匹配其前面的字符恰好出现n次，如[0-9]\{2\} 匹2位数字
{n,}	匹配其前面的字符出现不小于n次
{n,m}	匹配其前面的字符至少出现n次，最多出现m次，如[a-z]\{6,8\} 匹配6到8个小写字母
|	匹配|两边的任意一项

扩展的正则表达式示例

```
$ cat nucleic_acid
DNA
mRNA
tRNA
rRNA
snRNA
snoRNA
miRNA
lncRNA
```

```
$ grep -E ‘^.?RNA’ nucleic_acid
mRNA
tRNA
rRNA

$ grep -E "[a-z]{3}RNA" nucleic_acid
snoRNA
lncRNA
```

通配符与正则表达式的区别

- **通配符**是用来匹配文件名的，是完全匹配；
- **正则表达式**是用来匹配文件中的文本的，是包含匹配，如grep查找一个正则表达式时，任意包含该表达式的行都会显示出来。

引号

- **双引号：**
  - –由双引号括起来的字符，除$、`和\外都作为普通字符处理
- **单引号：**
  - –由单引号括起来的字符除\外都作为普通字符处理
- **反引号：**
  - –反引号括起来的字符shell作为命令处理

双引号

双引号中的内容：

- $表示变量引用（关于Shell变量，我们在后面还要学习），就是用变量值替换$及后面的变量名；
- 反引号（`）表示命令替换，反引号及其中的内容用反引号中的命令的执行结果替换；
- 反斜杠（\）表示转义，其后如果是$、"、`、\时，它们会被当做普通字符对待，另外\t表示制表符，\n表示换行。

双引号示例

```
$ echo "Hello $LOGNAME"
Hello peter
$ echo "Hello \$LOGNAME"
Hello $LOGNAME
$ echo "Current time is `date`"
Current time is Thu Sep  3 21:10:25 CST 2020
$ echo "Ross is \"good at marriage\"."
Ross is "good at marriage".
$ echo "Current time is \`date\`"
Current time is `date`
```

单引号

- 单引号的使用要简单一些，其中的字符除反斜杠（转义符，\）外都被Shell作为字符本身对待
- 与双引号相比，$符号在单引号中将作为其自身输出而不是变量开始的标志；反引号（``）也是其自身而不作为命令替换的标志；但反斜杠（\）与在双引号中相同，也解释为转义符

单引号示例

```
$ echo 'Hello $LOGNAME'                    # $原样输出
Hello $LOGNAME
$ echo 'Hello `echo Peter`'                # 反引号原样输出
Hello `echo Peter`
$ echo -e 'Hello\tPeter'                   # 转义符，\t输出制表符
Hello      Peter
$ echo -e 'Hello\nPeter'                   # 转义符，\n输出换行符
Hello
Peter
$ echo -e 'Hello\\tPeter'                  # 转义符，\\输出\
Hello\tPeter
```

反引号

- 反引号表示命令替换，Shell在执行命令行之前，先用反引号中的命令的执行结果替换反引号及其中的内容。
- 例如：

```
$ echo "The current time is `date`"
The current time is 2020年 11月 25日 星期三 19:39:09 CST
```

转义符与路径符

- **转义符\：**
  - –放在特殊符号之前，该特殊符号表示符号本身
  - –表示换行、制表符等：\n，\t
  - –用在命令行的末端，换行后继续输入（相当于转义了换行符）
- **路径符/：**
  - –路径中跟在目录的后面，在路径的开始则表示根目录
  - –在算术运算中还表示除法运算

输入输出重定向

- 标准输出重定向（>）：ls >file
- 标准输出添加重定向（>>）：ls >>file
- 标准错误输出重定向（2>）：ls genome 2>file
- 标准错误输出添加重定向（2>>）：ls genome 2>>file
- 标准输入重定向（<）：tr b B <file
- 即时输入重定向（<<）：cat >file <<eof

注释和后台命令

- **注释（#）：**

  # This is an annotation line

  #！/bin/bash　#特殊的注释行，脚本解释器路径

- **后台命令（&）：**

  sort file &

命令组合符

- 无条件命令组合符
  - 管道：

    cat fern | grep sinensis　　#命令1的输出作为命令2的输入
  - 顺序执行：

    cd plant; touch rose　　#依次执行命令1和命令2
- 条件命令组合符
  - 逻辑与：

    test -f cow && rm -f cow　　#如果命令1执行成功则执行命令2
  - 逻辑或：

    test -d animal || mkdir animal　#如果命令1执行失败则执行命令2

成组命令

- **{ }**：成组命令（不新建进程，在当前Shell中运行命令），如：

```
{ echo "My working path is:";pwd;}
```

空格

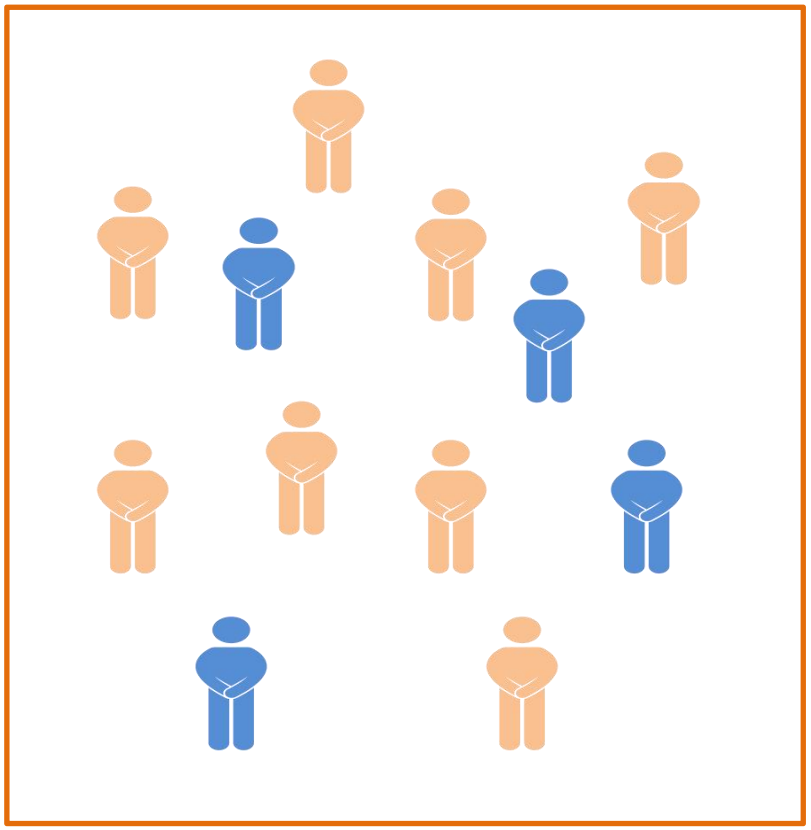
分号

  - 另外，{}还可用来组合字符串，如：

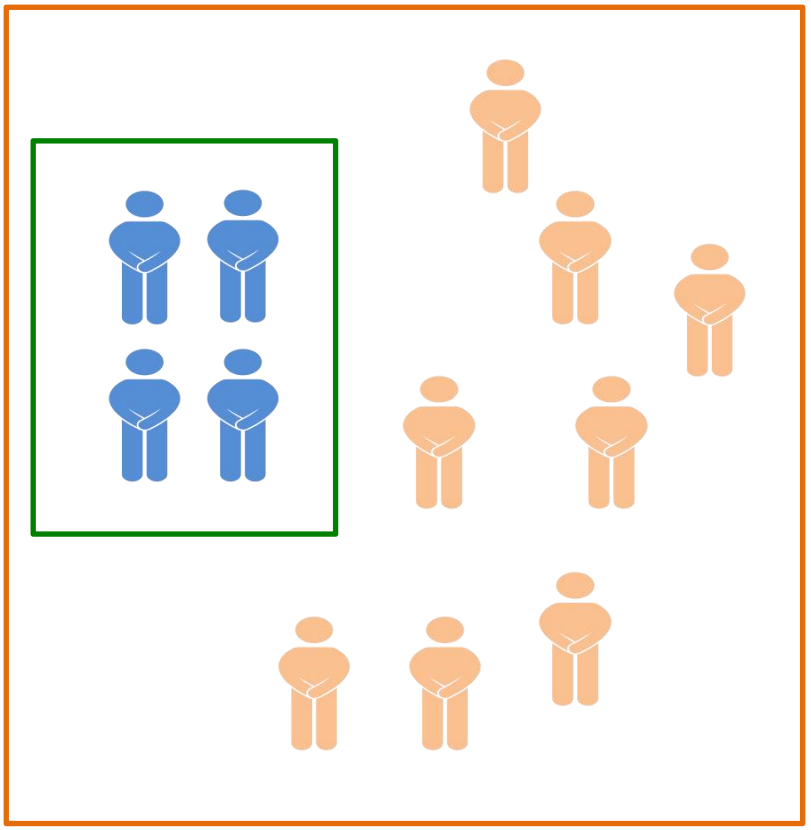
```
touch {file1,2}_{1,2,3}
```

- **( )**：成组命令（新建子进程，在子Shell中运行命令），如：

```
(echo "My current path is:";pwd)
```



{ }



( )

{}组合命令就像是在大房间里把几个人编成一组（左图），但他们只是逻辑上的一组，在物理空间上与其他人还是在一个房间中。

()组合命令就像在大房间中建一个小房间（右图），把几个人编成一组并放到这个小房间，他们做的事不会影响大房间中的其他人。


未完待续
LOADING