**shell脚本基础**

shell脚本：可以执行文件，可以实现某种功能

shell是门语言

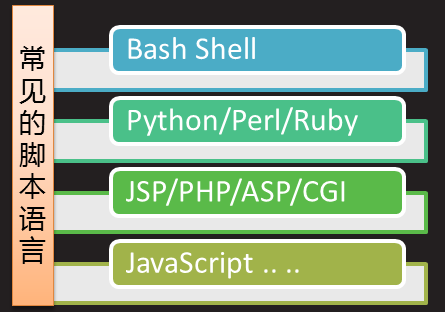
认识shell环境

什么是shell脚本：

提前设计可执行语句，用来完成特定任务的文件

解释型程序

顺序、批量执行



规范shell脚本的一般组成

#!环境申明

#注释文本

可执行代码

bash shell的使用方式

交互式

人工干预、智能化程度高

逐条解释执行、效率低

非交互式

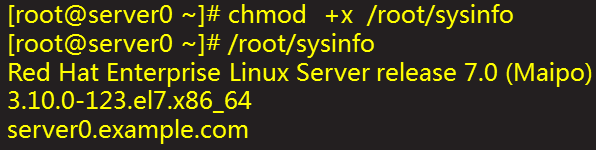
需要提前设计、智能化难度大

批量执行、效率高

方便后台悄悄运行

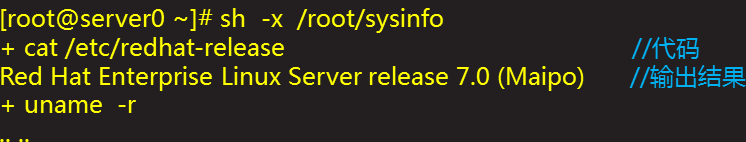
脚本运行及调试

调试执行权限，独立运行



通过sh执行脚本代码

结合-x选项，可以调试脚本代码



简单脚本编写技巧

管道传递

使用|管道操作

将前一条命令的标准输出交给后一条命令处理

免交互及输出处理

>：将前面命令的正确输出，写入到文本文件中，只收集正确信息

1. ：将前面命令的错误输出，写入到文本文件中，只收集错误信息

&>：将前面命令的正确与错误输出，写入到文本文件中，正确与错误都收集

免交互处理

脚本一般在后台执行，要尽量减少人工交互的语句

脚本输出信息的处理

记录有价值的信息（>>/var/log/foo.log）

屏蔽无价值的、干扰性的信息（&> /dev/null）

自定义输出：echo ‘文本字符串’

重定向输出

屏幕输出文本的类别

标准输出（1）：命令行执行正常的显示结果

标准错误（2）：命令行执行出错或异常时的显示结果

将屏幕信息保存到文件

cmd > file cmd >> file

cmd 2> file cmd 2>>file

cmd &> file cmd 2 > file1 >&2

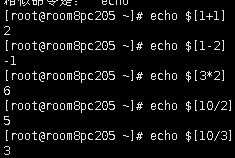
补充：

‘’（单引）：取消所有特殊字符意义

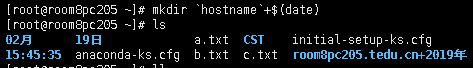


$[]：运算+ - \* / %（取余，求模）

定律：余数一定小于除数



$()与``反撇号：将命令的输出结果，作为参数



**使用变量**

变量的定义和引用

什么是变量：

以不变的名称存放的可能变化的值

变量名=变量值

方便以固定名称使用某个值

提高对任务需求、运行环境变化的适应能力

定义/赋值变量：

设置变量时的注意事项

若指定的变量名已存在，相当于为此变量重新赋值

等号两边不要有空格

变量名由字母/数字/下划线组成，区分大小写

变量名不能以数字开头，不要使用关键字和特殊字符

查看/引用变量

引用变量值：$变量名

查看变量值：echo $变量名 echo ${变量名}

变量的种类

运维角度的变量区分

根据变量的用途不同区分

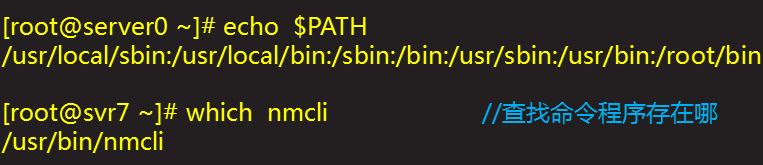
|  |  |
| --- | --- |
| 类型 | 说明 |
| 环境变量 | 变量名一般都大写，用来设置用户/系统环境 |
| 位置变量 | bash内置，存储执行脚本时提供的命令行参数 |
| 预定义变量 | bash内置，可以直接调用的特殊值，不能直接修改 |
| 自定义变量 | 用户自主设置、修改及使用 |

环境变量

常见的环境变量

PWD、PATH、USER、LOGNAME

SHELL、HOME



位置变量：由系统定义赋值完成，用户直接调用，方便向脚本中传递命令行参数

在执行脚本提供的命令行参数

表示为$n，n为序号

$1、$2、...${10}、${11}

预定义变量：由系统定义完成

用来保存脚本程序的执行信息

直接使用这些变量

不能直接为这些变量赋值

|  |  |
| --- | --- |
| 变量名 | 含义 |
| $# | 已加载的位置变量个数（后头加几个显示就有几个） |
| $\* | 所有位置的变量的值 |
| $? | 程序退出后的状态值，0表示正常，其他值异常（判断命令是否成功） |

**条件测试及选择**

条件测试

shell测试的依据

命令行/程序的退出状态值$?

值为0，表示执行成功

值不为0，表示执行异常或失败

在脚本职工自定义退出状态值

以退出之前最后最后一条命令的$?作为脚本的退出状态值

也可以自行替换，添加exit整数

常用的测试选项

检查文件状态

-e：文档是否存在 ，存在为真，不存在为假

-d： 存在且为目录，存在为真，不存在为假

-f：存在且为文件，存在为真，不存在假

-r： 存在且具备读权限，存在为真，不存在为假

-w： 存在且具备写权限，存在为真，不存在为假

-x：存在且具备执行权限，存在为真，不存在为假

比较整数大小（带e字母都有等于二字，g代表大于，l代表小于）

-gt： 大于

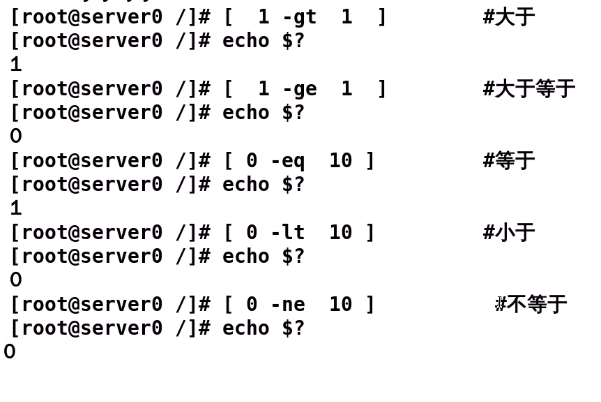
-ge：大于等于

-eq：等于

-ne：不等于

-lt：小于

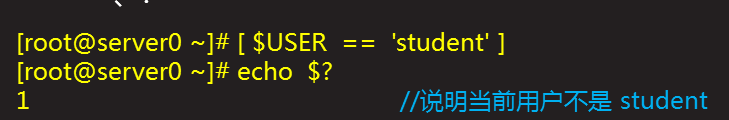
-le：小于等于



字符串比对

==：字符串相等为真

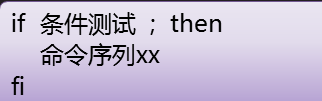
!=：字符串不相等为真



If选择结构

if单分支处理

单分支：当条件满足时，作xx处理



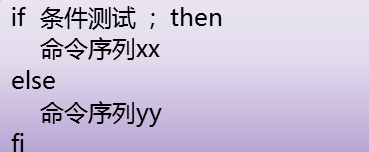
if [条件测试]; then

命令序列xx

fi

if双分支处理

双分支：当条件满足/不满足时，分别做xx、yy处理



if [条件判断];then

命令序列xx

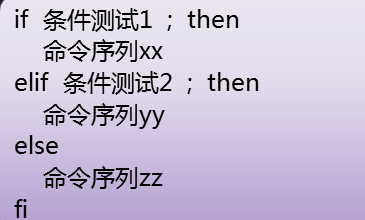
else

命令序列yy

fi

if多分支处理

多分支：当条件1满足时，作xx处理；否则继续检查条件2，若成立则作yy处理；否则，作zz处理



if [条件判断1]; then

命令序列xx

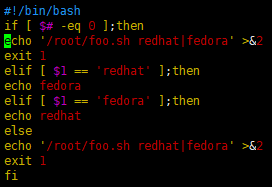
elif [条件判断2]; then

命令序列yy

else

命令序列zz

fi



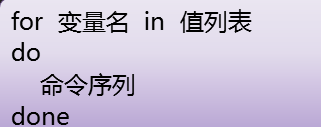
**列表循环**

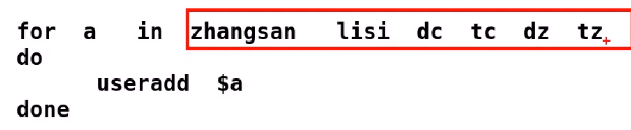
for循环结构

for循环处理

遍历/列表循环

根据变量的不同取值，重复执行xx处理





利用命令替换取值

使用$(命令行)操作

先执行括号内的命令，提取此命令行的标准输出

然后将标准输出结果替换整个$()表达式

可以作为参数嵌入到其他命令行

**总结和答疑**

问题现象

编写的脚本运行异常

问题1：若提供参数则正常，否则会报错 line xx:[: =: unary operator expected]

问题2：执行脚本时报错linue xx:[:missing`]’

故障分析及排除

原因分析

问题1：[]测试操作缺少必要的参数，很可能是参与运算的变量未加双引号，当变量为空值时会报错

问题2：若[]操作符与参数之间无空格则语法有错误

解决办法

问题1：添加双引号，比如改成if [ $1 = “redhat” ]

问题2：[ 参数1 ..... 参数2 ]

造数工具｛20..80｝：列出20-80的所有数字

｛起始..结束｝

>&2把正确的转换为错误