10 Shell基础

10.1 Shell概述

1. **Shell是什么**

- Shell是一个命令行解释器，它为用户提供了一个向Linux内核发送请求以便运行程序的界面级程序，用户可以用Shell来启动、挂起、停止甚至是编写一些程序
- 外层应用程序->Shell命令解释器->内核->硬件
- Shell还是一个功能相当强大的编程语言，易编写，易调试，灵活性强。
- Shell是**解释执行**的脚本语言，在Shell中可以直接调用Linux系统命令。

2. **Shell的分类**

- Bourne Shell：从1979起Unix就开始使用Bourne Shell，Bourne Shell的主文件名为sh。
  - Bourne家族主要有：sh、ksh、**Bash**、psh、zsh
  - Bash：Bash与sh兼容，Linux就是使用Bash作为用户的基本Shell。
- C Shell：C Shell主要是在BSD版的Unix系统当中使用，因其语法和C语言相类似而得名。
  - C家族主要包括：csh、tcsh
- Shell的两种主要语法类型有Bourne和C，彼此并不兼容。

3. **Linux支持的Shell**

- vim /etc/shells

10.2 Shell脚本的执行方式

1. **输出命令echo**

- 命令格式：echo [选项] [输出内容]
- 选项：
  - -e 支持反斜杠线控制的字符转换 控制字符 作用 \ 输出\ \a 输出警告音 \b 退格键，也就是向左删除键 \c 取消输出行末的换行符。和“-n”选项一致 \e ESCAPE键 \f 换页符 \n 换行符 \r 回车键 \t 制表符，也就是Tab键 \v 垂直制表符 \0nnn 按照八进制ASCII码表输出字符。 \xhh 按照十六进制ACSII码输出字符。
  - echo -e "\e[1;31m abcd \e[0m" #输出abcd(带颜色)
  - 30m=黑色，31m=红色，32m=绿色，33m=黄色
  - 34m=蓝色，35m=洋红，36m=青色，37m=白色

2. **第一个脚本**

- vi hello.sh `#!/bin/bash #每个脚本文件第一行都必须是这个 #注释(养成良好的注释习惯) #The first program #Author: WadeFrank(E-mail: wadefrank@163.com) echo -e “hello world”`

3. **脚本执行**

- 法一：赋予执行权限，直接运行
  - chmod 755 hello.sh
  - ./hello.sh
- 法二：通过Bash调用执行脚本
  - bash hello.sh

4. window下的sh文件转化为Linux下的sh文件

- dos2unix

10.3 Bash的基本功能

10.3.1 历史命令与命令补全

1. **历史命令**

- 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 /root/.bash_history
- 历史命令默认保存1000条，可以在环境变量配置文件/etc/profile中进行修改
- 历史命令的调用

- 用上下箭头来调用以前的历史命令
- 使用"!n"重复执行第n条历史命令
- 使用"!!"重复执行上一条命令
- 使用"!字串"重复执行最后一条以该字串开头的命令

2. **命令与文件补全**

- 在输入命令或文件时，按“Tab键”就会自动补全

10.3.2 命令别名与常用快捷键

1. **命令别名(临时生效)**

- alias 别名 = ‘原命令’ #设定命令别名
- alias #查询命令别名

2. **命令执行顺序**

- 第一顺位执行用绝对路径或相对路径执行的命令
- 第二顺位执行别名
- 第三顺位执行Bash的内部命令
- 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第一个命令

3. **让别名永久生效**

- vi /root/.bashrc

4. **删除别名**

- unalias 别名

5. **Bash常用快捷键** 快捷键 作用 ctrl+A 把光标移到命令行开头 ctrl+E 把光标移到命令行结尾 ctrl+C 强制终止当前命令 ctrl+L 清屏(=clear命令) ctrl+U 删除或剪切光标之前的命令 ctrl+K 删除或剪切光标之后的命令 ctrl+Y 粘贴ctrl+U或ctrl+K剪切的内容 ctrl+R 在历史命令中搜索，按下ctrl+R之后就会出现搜索界面，只需输入搜索内容，就会从历史命令中搜索 ctrl+D 退出当前终端 ctrl+Z 暂停，并放入后台 ctrl+S 暂停屏幕输出 ctrl+Q 恢复屏幕输出

10.3.3 输入输出重定向

1. **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
2. **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**的方式，把命令的正确输出保存到指定的文件或者设备中 命令 >> 文件 以**追加**的方式，把命令的正确输出保存到指定的文件或者设备中 标准错误输出重定向 错误命令 2> 文件 以**覆盖**的方式，把命令的错误输出保存到指定的文件或者设备中 错误命令 2>> 文件 以**追加**的方式，把命令的错误输出保存到指定的文件或者设备中 **正确输出和错误输出同时保存** 命令 > 文件 2>&1 以**覆盖**的方式，把命令的正确输出和错误输出保存到同一个文件中 命令 >> 文件 2>&1 以**追加**的方式，把命令的正确输出和错误输出保存到同一个文件中 命令 &>文件 以**覆盖**的方式，把命令的正确输出和错误输出保存到同一个文件中 命令 &>>文件 以**追加**的方式，把命令的正确输出和错误输出保存到同一个文件中 命令 >>文件1 2>>文件2 把正确的输出追加到文件1中，把错误的输出追加到文件2中

- /dev/null 垃圾箱

3. **输入重定向**

- wc [选项] [文件名](ctrl+D结束)
- 选项：
  - -l 统计行数
  - -c 统计字节数
  - -w 统计单词数
- 命令 < 文件 #把文件作为命令的输入

10.3.4 多命令顺序执行与管道符

1. **多命令顺序执行** 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，命令1正确执行时命令2执行，命令1执行不正确，则命令2不执行 || 命令1||命令2 逻辑或，命令1执行不正确时命令2执行，命令1正确执行时命令2不执行

- 命令 && echo yes ||echo no #判断命令是否报错

2. **管道符**

- 命令格式：命令1丨命令2 #命令1的正确输出作为命令2的操作对象
- 示例：
  - ll -a /etc/ 丨more #让命令结果分页表示
  - netstat -an丨grep ESTABLISHED #在命令结果中查找已经建立连接的

10.3.5 通配符与其他特殊符号

1. 通配符 通配符 作用 ？ 匹配一个任意字符 * 匹配0个或任意多个任意字符，也就是可以匹配任何内容 [] 匹配中括号中任意一个字符。例如：[abc]代表一定匹配一个字符，或者是a，或者是b，或者是c [-] 匹配中括号中任意一个字符，-代表一个范围。例如：[a-z] [^] 逻辑非，表示匹配不是中括号内的一个字符。例如[\^0-9]
2. Bash中的其他常用特殊符号 符号 作用 " 单引号。在单引号中所有的特殊符号，如$和`都没有特殊含义 "" 双引号。在双引号中的特殊符号没有特殊含义，但是$`和\除外。 `` 反引号。反引号括起来的内容是系统命令，在Bash中会先执行它。 $() 和反引号作用一样，用来引用系统命令。 # 注释 $ 用于调用变量的值，如$name \ 转义符

10.4 Bash的变量

10.4.1 用户自定义变量

1. **什么是变量**

- 变量是计算机内存的单元，其中存放的值可以改变。
- 变量可以用于保存有用信息。

2. **变量设置规则**

- 变量名称可以由字母、数字和下划线组成，但是不能用数字开头。
- 在Bash中，变量的默认类型都是字符串类型，如果要进行数值运算，则必须指定变量类型为数值型。
- 变量用等号连接值，等号左右不能有空格。
- 变量的值如果有空格，需要用单引号或者双引号包起来。
- 在变量的值中，可以使用“\“转义符。
- 如果需要增加变量的值，那么可以进行变量值的叠加。不过变量需要用双引号包起来”$变量名“或用\${变量名}包含。
- 如果是把命令的结果作为变量值赋予变量，则需要使用反引号或$()包含
- 环境变量名建议大写，便于区分

3. **变量分类**

- 用户自定义变量
- 环境变量：这种变量中主要保存的是和系统操作环境相关的数据。
- 位置参数变量：这种变量主要是用来向脚本当中传递参数或者数据的，变量名不能自定义，变量作用是固定的。
- 预定义变量：是Bash中已经定义好的变量，变量名不能自定义，变量作用也是固定的。

4. **本地变量**(用户自定义变量)

- 变量定义
  - name="feng xian"
- 变量叠加
  - aa=123
  - aa="$aa"456
  - aa=${aa}789
- 变量调用：echo $name
- 变量查看：set
- 变量删除：unset name

10.4.2 环境变量

1. **环境变量是什么？**

- 用户自定义变量只在当前的Shell中生效，而环境变量会在当前Shell和这个Shell的所有子Shell当中生效。
- 如果把环境变量写入相应的配置文件，那么这个环境变量就会在所有的Shell中生效。

2. **设置环境变量**

- export 变量名=变量值 #声明变量
- env #查询变量
- unset 变量名 #删除变量

- pstree #查询继承树

3. **系统常见环境变量**

- PATH：系统查找命令的路径
  - echo $PATH
  - PATH="$PATH":/root/sh #PATH变量叠加(临时生效)
- PS1:定义系统**提示符**的变量
  - \d：显示日期，格式为“星期 月 日”
  - \h：显示简写主机名。如默认主机名“localhost”
  - \t：显示24小时制时间，格式为“HH：MM：SS“
  - \T：显示12小时制时间，格式为“HH：MM：SS“
  - \A：显示24小时制时间，格式为”HH：MM“
  - \u：显示当前用户名
  - \w：显示当前所在目录的完整名称
  - \W：显示当前所在目录的最后一个目录
  - \#：执行的第几个命令
  - \$：提示符。如果是root，会显示提示符为“#”，如果是普通用户会显示提示符为“\$“
- 示例：PS1="[\u@\h \w]\$ "
- Mac：\h:\W \u\$

10.4.3 预定义变量

1. **位置参数变量** 位置参数变量 作用 $n n为数字，\$0代表命令本身，\$1-\$9代表第一到第9个参数，十以上的参数需要用大括号包含，如${10} $* 这个变量代表命令行中的所有参数，$*把所有的参数看成一**个整体** $@ 这个变量也代表命令行中所有的参数，不过$@把每个参数**区分对待** $# 这个变量代表命令行中所有参数的个数

- 示例1：vi sum `#!/bin/bash num1=$1 num2=$2 sum=$(($num1 + $num2)) #变量sum的和是num1加num2 echo $sum #打印变量sum的值`

2. **其他预定义变量** 预定义变量 作用 $? 最后一次执行的命令的返回状态。如果这个变量的值为0，证明上一个命令正确执行；如果这个变量的值为非0(具体是哪个数由命令自己来决定)，则证明上一个命令执行不正确 $$ 当前进程的进程号(PID) $! 后台运行的最后一个进程的进程号(PID)
3. **接收键盘收入**

- read [选项] [变量名]
- 选项：
  - -p “提示信息”：在等待read输入时，输出提示信息
  - -t 秒数：read命令会一直等待用户输入，使用此选项可以指定等待时间
  - -n 字符数：read命令只接受指定的字符数，就会执行
  - -s 隐藏输入的数据，适用于机密信息的输入
- 示例：`#!/bin/bash #Author:fengxian read -t 30 -p "Please input your name:" name #提示“输入姓名”并等待30秒，把用户的输入保存入变量name中 echo "Name is $name" read -s -t 30 -p "Please input your age:" age #年龄是隐私，所以我们用“-s“选项隐藏输入 echo "Age is $age" echo -e "\n" read -n 1 -t 30 -p "Please input your gender[M/F]:" gender #使用“-n 1”选项只接收一个输入字符 echo -e "\n" echo "Sex is $gender"`

10.5 Bash的运算符

10.5.1 数值运算与运算符

1. **declare声明变量类型**

- declare [+/-] [选项] 变量名
- 选项：
  - - 给变量设定类型属性
  - + 取消变量的类型属性
  - -i 将变量声明为整数型(integer)
  - -x 将变量声明为环境变量
  - -p 显示指定变量的被声明的类型

2. **数值运算**

- 方法1 `aa=11 bb=22 declare -i cc=$aa+$bb`
- 方法2：expr或let数值运算工具 `dd=$(expr $aa + $bb)`
- 方法3："$((运算式))"或"$[运算式]" `ff=$(($aa+$bb)) #最常用 gg=$[$aa+$bb]`

3. **运算符** 优先级 运算符 说明 13 -，+ 单目负、单目正 12 ！，～ 逻辑非、按位取反码或补码 11 *，/，% 乘、除、取模 10 +，- 加、减 9 <<，>> 按位左移、按位右移 8 <=，>=，<，> 小于或等于、大于或等于、小于、大于 7 ==，!= 等于、不等于 6 & 按位与 5 ^ 按位异或 4 | 按位或 3 && 逻辑与 2 || 逻辑或 1 =,+=,-=,*=,/=,%=,&=,^=,|=,<<=,>>= 赋值、运算后赋值
4. **变量测试与内容替换** 变量置换方式 变量y没有设置 变量y为空值 变量y设置值 x=${y-新值} x=新值 x为空 x=$y x=${y:-新值} x=新值 x=新值 x=$y x=${y+新值} x为空 x=新值 x=新值 x=${y:+新值} x为空 x为空 x=新值 x=${y=新值} x=新值；y=新值 x为空；y值不变 x=$y；y值不变 x=${y:=新值} x=新值；y=新值 x=新值；y=新值 x=$y；y值不变 x=${y?新值} 新值输出到标准错误输出(就是屏幕) x为空 x=$y x=${y:?新值} 新值输出到标准错误输出 新值输出到标准错误输出 x=$y

- 例子1:测试x=${y-新值} `unset y #删除变量 x=${y-new} # 进行测试 echo $x #因为变量y不存在，所以x=new`

10.6 环境变量配置文件

10.6.1 环境变量配置文件简介

1. **source命令**

- 让配置文件立即生效
- source 配置文件(=. 配置文件)

2. **环境变量配置文件简介**

- 环境变量配置文件中主要是定义对系统的操作环境生效的默认环境变量，比如PATH、HISTSIZE、PS1、HOSTNAME等默认环境变量

3. **环境变量** /etc目录下的对所有用户生效；~/目录下的只对当前用户生效；越在后面优先级越高。

- /etc/profile
- /etc/profile.d/*.sh(一组文件)
- ~/.bash_profile
- ~/.bashrc
- /etc/bashrc

10.6.2 环境变量配置文件作用

1. **/etc/profile的作用**

- USER变量
- LOGNAME变量
- MIAL变量
- PATH变量
- HOSTNAME变量
- umask变量
- 调用/etc/profile.d/*.sh文件

2. **~/.bash_profile的作用**

- 调用~/.bashrc文件
- 在PATH变量后面加入了":$HOME/bin"这个目录

3. **~/.bashrc的作用**

- 系统别名
- 调用/etc/bashrc文件

4. **/etc/bashrc的作用**

- PS1变量
- umask变量
- PATH变量
- 调用/etc/profile.d/*.sh文件

10.6.3 其他配置文件和登录信息

1. 注销时生效的环境变量配置文件

- ~/.bash_logout

2. 其他配置文件

- ~/bash_history(历史命令)

3. Shell登录信息

- 本地终端欢迎信息：/etc/issue 转义符 作用 \d 显示当前系统日期 \s 显示操作系统名称 \l 显示登录的终端号(比较常用) \m 显示硬件体系结构，如i386、i686等 \n 显示主机名 \o 显示域名 \r 显示内核版本 \t 显示当前系统时间 \u 显示当前登录用户的序列号
- 远程终端欢迎信息：/etc/issue.net
  - 转义符在/etc/issue.net文件中不能使用
  - 是否显示此欢迎信息，由ssh的配置文件/etc/ssh/sshd_config决定，加入“Banner /etc/issue.net“行才能显示(记得重启SSH服务)
- 登录后欢迎信息：/etc/motd
  - 不管是本地登陆还是远程登录，都可以显示

11 Shell编程

11.1 基础正则表达式(多练)

1. **正则表达式和通配符**

- 正则表达式用来**在文件中**匹配符合条件**字符串**，正则是**包含匹配**。grep、awk、sed等命令可以支持正则表达式。
- 通配符用来**在系统中**匹配符合条件的**文件名**，通配符是**完全匹配**。ls、find、cp这些命令不支持正则表达式，所以只能使用shell自己的通配符来进行匹配。

2. **基础正则表达式** 元字符 作用 * 前一个字符匹配**0次**或任意多次 . 匹配除了换行符外任意一个字符 ^ 匹配行首。例如：^hello会匹配以hello开头的行 $ 匹配行尾。例如：hello$会匹配以hello结尾的行 [] 匹配中括号指定的**任意一个字符**，只匹配一个字符。例如：[aeiou]匹配任意一个元音字母，[0-9]匹配任意一位数字，[a-z][0-9]匹配小写字和一位数字构成的两位字符。不匹配换行符(即空白行)。 [^] 匹配中括号的字符以外的**任意一个字符**。例如：[\^0-9]匹配任意一位非数字字符。不匹配换行符(即空白行)。 \ 转义符。 \{n\} 表示其前面的字符恰好出现n次。例如：[0-9]\{4\}匹配4位数字，[1][3-8][0-9]\{9\}匹配手机号码 \{n,\} 表示其前面的字符出现不小于n次。例如：[0-9]\{2,\}表示两位及以上的数字。 \{n,m\} 表示其前面的字符至少出现n次，最多出现m次。例如[a-z]\{6,8\}匹配6到8位的小写字母。

- 示例：
  - *
  - grep "a*" test_rule.txt #匹配所有内容，包括空白行
  - grep "aa*" test_rule.txt #匹配至少含有一个a的行
  - .
  - grep "s..d" test_rule.txt #匹配在s和d这两个字母之间一定有两个字符的单词
  - grep "s.*d" test_rule.txt #匹配在s和d这两个字母之间有任意字符的单词

11.2 字符截取命令

1. **cut**字段提取命令

- cut [选项] 文件名
- 选项：
  - -f 列号 #提取第几列
  - -d 分隔符 #按照指定分隔符分割列
- 示例：
  - cut -f 2 student.txt
- cut命令的局限
  - 只适用于制表符或者：和，和。分隔的比较标准的列。
  - 不适用于空格作为分隔符的文件

2. **printf**命令

- printf'输出类型输出格式' 输出内容
- 输出类型：
  - %ns #输出字符串。n是数字，表示输出几个字符。
  - %ni #输出整数。n是数字，表示输出几个数字。
  - %m.nf #输出浮点数。m和n是数字，表示输出的整数位数和小数位数。
- 输出格式：
  - \a #输出警告声音
  - \b #输出退格键(backspace)
  - \f #清除屏幕

    - \n #换行
    - \r #回车(enter)
    - \t #水平输出退格键(tab)
    - \v #垂直输出退格键(tab)
- print会自动加上换行符，而printf则需要手动添加

3. **awk**命令

- awk '条件1{动作1} 条件2 {动作2} …' 文件名
- 条件(pattern)：
    - 一般使用关系表达式作为条件(如x>10,x<=10等)
    - BEGIN
    - 在处理之前先执行一个动作
    - awk 'BEGIN{printf"This is a transcript \n"} {printf $2 "\t" \$6"\n"}' student.txt
    - END
    - 在所有动作完成之后，在执行一个动作
- 动作(action)：
    - 格式化输出
    - 流程控制语句
- 示例
    - awk '{printf $2 "\t" \$6"\n"}' student.txt #无条件完成动作输出第2列和第6列

4. **sed**命令

- sed是一种存在于几乎所有Unix平台上的轻量级流编辑器。
- sed主要是用来将数据进行选取、替换、删除、新增的命令。
- sed [选项] '[动作]' 文件名
- 选项：
    - -n 一般sed命令会把所有数据都输出到屏幕，如果加入此选择，则**只会把经过sed命令处理的行输出到屏幕**。
    - -e 允许对输入数据应用多条sed命令编辑，用；隔开
    - -i 用sed的修改结果**直接修改读取数据的文件**，而不是由屏幕输出
- 动作：
    - a 追加，在当前行后添加一行或多行。添加多行时，除最后一行外，每行末尾需要用"\"代表数据未完结。
    - c 行替换，用c后面的字符串替换原数据行，替换多行时，除最后一行外，每行末尾需用"\"代表数据未完结
    - i 插入，在当期行前插入一行或多行。插入多行时，除最后一行外，每行末尾需要用"\"代表数据未完结。
    - d 删除，删除指定的行。
    - p 打印，输出指定的行。
    - s 字串替换，用一个字符串替换另外一个字符串。格式为“行范围s/旧字串/新字串/g”(和vim中的替换格式类似)。
- 示例：
    - sed -n '2p' student.txt #查看文件的第二行
    - df -h |sed -n '2p'
    - sed '2a hello' student.txt #在第二行后追加hello
    - sed '2i hello \ world' #在第二行前插入两行数据

11.3 字符处理命令

1. 排序命令**sort**

- sort [选项] 文件名
- 选项：
    - -f 忽略大小写
    - -n 以数值型进行排序，默认使用**字符串型**排序
    - -r 反向排序
    - -t 指定分隔符，默认分隔符是制表符
    - -k n[,m] 按照指定的字段范围排序。从第n字段开始，m字段结束(默认到行尾)
- 示例：
    - sort /etc/passwd #排序用户信息文件
    - sort -r /etc/passwd #反向排序
    - sort -t ":" -k 3,3 /etc/passwd #指定分隔符是”：“，用第三字段开头和结尾排序

2. 统计命令wc

- wc [选项] [文件名](ctrl+D结束)
- 选项：
    - -l 统计行数
    - -c 统计字节数

- -w 统计单词数

11.4 条件判断

1. 按照**文件类型**进行判断 测试选项 作用 -b 文件 判断该文件是否存在，并且是否为块设备文件(是为真) -c 文件 判断该文件是否存在，并且是否为字符设备文件(是为真) **-d 文件** 判断该文件是否存在，并且是否为目录文件(是为真) **-e 文件** 判断该文件是否存在(是为真) **-f 文件** 判断该文件是否存在，并且是否为普通文件(是为真) -L 文件 判断该文件是否存在，并且是否为符合链接文件(是为真) -p 文件 判断该文件是否存在，并且是否为管道文件(是为真) -s 文件 判断该文件是否存在，并且是否为非空(是为真) -S文件 判断该文件是否存在，并且是否为套接字文件(是为真)

- 两种判断格式：
  - test -e /root/install.log #判断文件是否存在
  - [-e /root/install.log] #判断文件是否存在，shell脚本中更常用
  - $? #判断上一条命令是否执行正确，正确为0，不正确为非0
- 示例：
  - [-d /root] && echo "yes" || echo "no" #第一个判断命令如果正确执行，则打印“yes”，否则打印“no”

2. 按照**文件权限**进行判断 测试选项 作用 **-r 文件** 判断该文件是否存在，并且是否该文件拥有读权限(是为真) **-w 文件** 判断该文件是否存在，并且是否该文件拥有写权限(是为真) **-x 文件** 判断该文件是否存在，并且是否该文件拥有执行权限(是为真) -u 文件 判断该文件是否存在，并且是否该文件拥有SUID权限(是为真) -g 文件 判断该文件是否存在，并且是否该文件拥有SGID权限(是为真) -k 文件 判断该文件是否存在，并且是否该文件拥有SBit权限(是为真)
3. 两个文件之间进行比较 测试选项 作用 文件1 -nt 文件2 判断文件1的修改时间是否比文件2的新(是为真) 文件1 -ot 文件2 判断文件1的修改时间是否比文件2的旧(是为真) 文件1 -ef 文件2 判断文件1是否和文件2的Inode号一致，可以理解为两个文件是否为同一个文件。可以用于判断硬链接。
4. 两个整数之间比较 测试选项 作用 整数1 -eq 整数2 判断整数1是否和整数2相等(是为真) 整数1 -ne 整数2 判断整数1是否和整数2不相等(是为真) 整数1 -gt 整数2 判断整数1是否大于整数2(是为真) 整数1 -lt 整数2 判断整数1是否小于整数2(是为真) 整数1 -ge 整数2 判断整数1是否大于等于整数2(是为真) 整数1 -le 整数2 判断整数1是否小于等于整数2(是为真)

- 示例：[23 -gt 22] && echo "yes" || echo "no"

5. 字符串的判断 测试选项 作用 -z 字符串 判断字符串是否为空(是为真) -n 字符串 判断字符串是否为非空(是为真) 字符串1 == 字符串2 判断字符串1是否和字符串2相等(是为真) 字符串1 != 字符串2 判断字符串1是否和字符串2不相等(是为真)

- 示例：
  - name =sc
  - [-z "$name"] && echo "yes" || echo "no"

6. 多重条件判断 测试选项 作用 判断1 -a 判断2 逻辑与，判断1和判断2都成立，最终的结果才为真 判断1 -o 判断2 逻辑或，判断1和判断2有一个成立，最终的结果就为真 ! 判断 逻辑非，使原始的判断式取反

11.5 流程控制

11.5.1 if语句

1. **单分支if条件语句**

```
if [ 条件判断式 ];then
    程序
fi
```

或者：

```
if [ 条件判断式 ]
    then
        程序
fi
```

- 示例1：判断分区使用率 `#!/bin/bash #统计根分区使用率 #Author: fengxian rate=$(df -h | grep "/dev/vda1" | awk '{print $5}' | cut -d "%" -f1) #把根分区使用率作为变量值赋予变量rate if [ $rate -ge 10 ] then echo "Warning! /dev/sda3 is full!!" fi`

2. **双分支if语句**

```
if [ 条件判断式 ]
    then
        程序
```

```
  else
      程序
fi
```

- 示例2：备份mysql数据库 `#!/bin/bash #备份mysql数据库 #Author: Fengxian ntpdate asia.pool.ntp.org $>/dev/null #同步系统时间 date=$(date +%y%m%d) #把当前系统时间按照"年月日"格式赋予变量date size=$(du -sh /var/lib/mysql) #统计mysql数据库的大小，并把大小赋予size变量 if [-d /tmp/dbbak] then echo "Date:$date!" > /tmp/dbbak/dbinfo.txt echo "Date size:$size" >> /tmp/dbbak/dbinfo.txt cd /tmp/dbbak tar -zcf mysql-lib-$date.tar.gz /var/lib/mysql dbinfo.txt &>/dev/null rm -rf /tmp/dbbak/dbinfo.txt else mkdir /tmp/dbbak echo "Date:$date!" > /tmp/dbbak/dbinfo.txt echo "Date size:$size" >> /tmp/dbbak/dbinfo.txt cd /tmp/dbbak tar -zcf mysql-lib-$date.tar.gz /var/lib/mysql dbinfo.txt &>/dev/null rm -rf /tmp/dbbak/dbinfo.txt fi`
- 示例3：判断apache是否启动 `#!/bin/bash #判断apache是否启动 #Author: Fengxian port=$(nmap -sT 192.168.1.156 | grep tcp | grep http | awk '{print $2}') #使用nmap命令扫描服务器，并截取apache服务的状态，赋予变量port if ["$port" == "open"] then echo "$(date) httpd is ok!" >> /tmp/autostart-acc.log else /etc/rc.d/init.d/http start &>/dev/null echo "$(date) restart httpd !!" >> /temp/autostart-err.log fi`

3. **多分支if条件语句**

```
if [ 条件判断式1 ]
    then
        程序
elif [ 条件判断式2 ]
  then
      程序
…
else
  程序
fi
```

- 示例4： `#!/bin/bash #判断用户输入的是什么文件 #Author: Fengxian read -p "Please input a filename: " file #接收键盘的输入，并赋予变量file if [-z "$file"] #判断file变量是否为空 then echo "Error, please input a filename!" exit 1 elif [! -e "$file"] #判断file的值是否存在 then echo "Your input is not a file!" exit 2 elif [-f "$file"] #判断file是否为普通文件 then echo "$file is a regular file!" elif [-d "$file"] #判断file的值是否为目录文件 then echo "$file is a directory!" else echo "$file is an other file!" fi`

11.5.2 case语句

1. 多分支case条件语句

- case语句和if…elif…else语句一样都是多分支条件语句，不过和if多分支语句不同的是，case语句只能判断一**种条件关系**，而if语句可以判断多种条件关系。

2. case格式

```
case $变量名 in
    "值1")
        如果变量的值等于值1，则执行程序1
        ;;
  "值2")
    如果变量的值等于值2，则执行程序2
    ;;
  …省略其他分支…
  *)
    如果变量的值都不是以上的值，则执行此程序
    ;;
esac
```

- 示例1 `#!/bin/bash #判断用户输入 #Author: Fengxian read -p "Please choose yes/no " -t 30 cho case $cho in "yes") echo "Your choose is yes!" ;; "no") echo "Your choose is no!" ;; *) echo"Your choose is error!" ;; esac`

11.5.3 for循环

1. **语法1**

```
for 变量 in 值1 值2 值3
    do
    程序
  done
```

- 示例1: `#!/bin/bash #打印时间 #Author: Fengxian for time in morning noon afternoon evening do echo "This time is $time" done`
- 示例2 `#!/bin/bash #批量解压缩脚本 #Author: Fengxian cd /lamp ls *.tar.gz > ls.log for i in $(cat ls.log) do tar -zxf $i &>/dev/null done rm -rf /lamp/ls.log`

2. **语法2**

```
for ((初始值;循环控制条件;变量变化))
  do
    程序
  done
```

- 示例3: `#!/bin/bash #从1加到100 #Author: Fengxian s=0 for((i=1;i<=100;i=i+1)) do s=$(($s+$i)) done echo "The sum of 1+2+...+100 is $s"`
- 示例4 `#!/bin/bash #批量添加用户 #Author: Fengxian read -p "Please input user name: " -t 30 name read -p "Please input the number of users: " -t 30 num read -p "Please input the password: " -t 30 pass if [! -z "$name" -a ! -z "$num" -a ! -z "$pass"] then y=$(echo $num|sed's/[0-9]//g') if [-z "$y"] then for((i=1;i<=$num;i=i+1)) do /usr/sbin/useradd $name$i &>/dev/null echo $pass|/usr/bin/passwd --stdin $name$i &>/dev/null done fi fi`

11.5.4 while循环和until循环

1. while

- while循环是不定循环，也称作条件循环。
- 只要条件判断式成立，循环就会一直继续，直到条件判断式不成立，循环才会停止。

2. while格式

```
while [ 条件判断式 ]
  do
    程序
  done
```

- 示例1: `#!/bin/bash #从1加到100 #Author: Fengxian i=1 s=0 while [ $i -le 100 ] #如果变量i的值小于等于100，则执行循环 do s=$(($s+$i)) i=$(($i+1)) done echo "The sum is: $s"`

3. until循环

- 和while循环相反。
- until循环只要条件判断式不成立则进行循环，一旦循环条件成立，则终止循环。

4. until格式

```
until [ 条件判断式 ]
  do
    程序
```

```
done
```