**Shell脚本基础**

# 第一行#!/bin/bash用于指定哪一个程序来编译和执行脚本，以下“#”开头独占一行用于注释来说明

# echo [-ne] [字符串]用于回显文字：

# echo $?：回显结果

# 1. -n：不进行换行

# 2. -e：文字输出，若出现以下字符则特殊处 理：

## \a 发出警告声

## \b 删除前一个字符

## \c 行尾不加换行符号

## \f 换行后光标停留在原来的位置

## \n 换行后光标移到下一行行首

## \r 光标移至行首，不换行

## \t 插入Tab 键

## \v 与\f相同

## \\ 插入\字符

## \nnn插入nnn（八进制）所代表的ASCII字符

## --help 显示帮助

## --version 显示版本信息

# read命令：自动读给REPLY

### read variable #读取变量给variable

### read #可同时读取多个变量

### read -p “Please input： ” #自动读给REPLY

# test命令可以用方括号[]来测试表达式的值

### 字符串测试

|  |  |
| --- | --- |
| [ -z str ] | 如果字符串str长度为0，返回结果为真 |
| [ -n str ] | 如果字符串str长度不为0，返回结果为真 |
| [ str1 = str2 ] | 两个字符串相等 |
| [ str1 != str2 ] | 两个字符串不相等 |

### 整数测试

1. 比较大小：（操作符两边必须留空格！）

|  |  |
| --- | --- |
| [ int1 -eq int2 ] | int1 等于 int2 |
| [ int1 -ne int2 ] | int1 不等于 int2 |
| [ int1 -gt int2 ] | int1 大于 int2 |
| [ int1 -ge int2 ] | int1 大于或等于 int2 |
| [ int1 -lt int2 ] | int1 小于 int2 |
| [ int1 -le int2 ] | int1 小于或等于 int2 |

1. let命令或（）只能用于整数测试

|  |  |  |
| --- | --- | --- |
| 相应的操作符为： | | |
| ==、!= | >、>= | <、<= |
| 1、x=1；let “$x==1”; echo $? | | |
| 2、x=1；(($x+1>= 2)); echo $? | | |
| 两种测试方法区别： | | |
| 使用的操作符不同 | | |
| let 和 （）中可以使用算术表达式，而[ ]不能 | | |
| let 和（）中，操作符两边可以不留空格 | | |

### 逻辑测试

|  |  |
| --- | --- |
| [ expr1 -a expr2 ] | 逻辑与，都为真时，结果为真 |
| [ expr1 -o expr2 ] | 逻辑或，有一个为真时，结果为真 |
| [ ! expr ] | 逻辑非 |
| [[ pattern1 && pattern2 ]] | 逻辑与，都为真时，结果为真 |
| [[ pattern1 || pattern2 ]] | 逻辑或，有一个为真时，结果为真 |
| [[ ! pattern ]] | 逻辑非 |

主：不能随便添加括号

### 文件测试

|  |  |
| --- | --- |
| -f fname | fname 存在且是普通文件时，返回真（0） |
| -L fname | fname 存在且是链接文件时，返回真 |
| -d fname | fname 存在且是一个目录时，返回真 |
| -e fname | fname（文件或目录）存在时，返回真 |
| -s fname | fname 存在且大小大于0时，返回真 |
| -r fname | fname（文件或目录）存在且可读时，返回真 |
| -w fname | fname（文件或目录）存在且可写时，返回真 |
| -x fname | fname（文件或目录）存在且可执行时，返回真 |

### 检查空值

|  |
| --- |
| [ “$name” = “” ] |
| [ ! “$name” ] |
| [ “X${name}” != “X” ] |

# if条件语句

|  |  |
| --- | --- |
| 语法结构： | |
| if expr1 | #如果expr1为真（返回值为0） |
| then | #那么 |
| commands1 | #执行语句块 commands1 |
| elif expr2 | #若expr1不真，而expr2为真 |
| then | #那么 |
| commands2 | #执行语句块 commands2 |
| … … | #可以有多个elif语句 |
| else | # else最多只能有一个 |
| commands4 | #执行语句块 commands4 |
| fi | # if语句必须以fi终止 |

|  |
| --- |
| 几点说明： |
| 1、elif可以有任意多个（0个或多个）。 |
| 2、else只能有一个（0个或多个）。 |
| 3、if语句必须以fi结束。 |
| 4、expr通常为条件测试表达式；可以是多个命令，以最后一个命令退出状态为条件值。 |
| 5、commands为可执行语句，如果为空，需空命令“：”，即冒号。不做任何事情，只返回一个退出状态0。command -v 判断命令是否正确 |
| 6、if语句可以嵌套使用。 |

# case选择语句

|  |  |
| --- | --- |
| 语法结构： | |
| case expr in | # expr为表达式，关键字in不要忘！ |
| pattern1) | # expr与pattern1匹配，注意右半括号 |
| commands1 | # 执行语句块commands1 |
| ;; | #跳出case结构 |
| pattern2) | # expr与pattern2匹配 |
| commands2 | # 执行语句块commands2 |
| ;; | #跳出case结构 |
| … … | #可以有任意多个模式匹配 |
| \*) | # expr与上面模式都不匹配 |
| commands | # 执行语句块commands |
| ;; | #跳出case结构 |
| esac | #case语句必须以esac终止 |

case只能做字符串语句比较等同于if语句

|  |
| --- |
| 几点说明： |
| 1、表达式expr按顺序匹配每个模式，一旦有一个模式匹配成功，则执行该模式后面的所有命令，退出case语句。 |
| 2、如果expr没有找到匹配的模式，则执行“\*）”的命令块（类似于if中的else）；“\*）”可以不出现。 |
| 3、所给的匹配模式pattern可以含有通配符和“|” |
| 4、每个命令块的最会必须有一个双分号，独占一行，或放在最后一个命令的后面 |

# for循环语句

### 语法结构：

|  |  |
| --- | --- |
| for variable in list #每一次循环依次将list中的一个值赋给循环变量 | |
| do | #循环开始标志 |
| commands | #循环变量每取一次值，循环体就执行一遍 |
| done | #循环结束标志 |

### 几点说明：

|  |
| --- |
| a．列表list可以是命令替换、变量名替换、字符串和文件名列表（可含通配符） |
| b．for循环执行的次数取决于列表list中单词个数 |
| c．for循环体中一般要出现循环变量，但也可以不出现 |

### 循环执行过程：

|  |
| --- |
| 执行第一轮循环时，将list中的第一个词赋给循环变量，并将该词从list中删除，然后进入循环体，执行do和done之间的命令。下一次进入循环体时，则将第二个词赋给循环变量，并将该词从list中删除，再往后的循环也将以此类推。当list中的词全部被移走后，循环就结束了。 |

### 位置参量的使用：$\*与$@

### 可以省略in list，此时使用位置参量

# until循环结构语句

### 语法结构

|  |  |
| --- | --- |
| until expr | #执行expr |
| do | #若expr的退出状态非0，进入循环，否则退出until |
| commands | #循环体 |
| done | #循环结束标志，返回循环顶部 |

### 执行过程

|  |
| --- |
| 与while循环类似，只是当expr退出状态非0时才执行循环体，直到expr为0时退出循环。 |

# 死循环语句while ：

### 语法结构

|  |  |
| --- | --- |
| while expr | #执行expr |
| do | #若expr的退出状态为0，进入循环，否则退出while |
| commands | #循环体 |
| done | #循环结束标志，返回循环顶部 |

### 执行过程

|  |
| --- |
| 先执行expr，如果其退出状态为0，就执行循环体。执行到关键字done后，回到循环顶部，while命令再次检查expr的退出状态。以此类推，循环将一直继续下去，直到expr的退出状态非0时为止。 |

# break [n]

|  |
| --- |
| 1.用于强行退出当前循环。 |
| 2.如果是嵌套循环，则break命令后面可以跟一数字n，表示退出第n重循环（最里面的为第一重循环） |

# continue [n]

|  |
| --- |
| 1.用于忽略本次循环的剩余部分，回到循环的顶部，继续下一次循环。 |
| 2.如果是嵌套循环，continue命令后面也可以跟一数字n，表示回到第n重循环的顶部。 |

# exit [n]：

|  |
| --- |
| 用于退出脚本或当前进程。n是一个从0到255的整数，0表示成功退出，非零表示遇到某种失败而非正常退出。该整数被保存在状态变量$?中。 |

# sleep [n]：暂停n秒钟

# select循环与菜单

### 语法结构

|  |  |
| --- | --- |
| select variable in list | |
| do | #循环开始标志 |
| commands | #循环变量每取一次值，循环体就执行一遍 |
| done | #循环结束标志 |

### 说明

|  |
| --- |
| a．select循环主要用于创建菜单，按数字顺序排列的菜单项将显示在标准错误上，并显示PS3提示符，等待用户输入。 |
| b．用户输入菜单列表中的某个数字，执行相应的命令。 |
| c．用户输入被保存在内置变量REPLY中 |

### select与case

|  |
| --- |
| a．select是个无限循环，因此要记住用break命令退出循环，或用exit命令终止脚本。也可以按ctrl+c退出循环。 |
| b．select经常和case联合使用 |
| c．与for循环类似，可以省略in list，此时使用位置参量 |

# shift [n] 循环控制命令

|  |
| --- |
| 1.用于将参数列表list左移指定次数，默认为左移一次。 |
| 2.参数列表list一旦被移动，最左端的那个参数就从列表中删除。while循环遍历位置参量列表时，常用到shift。 |

# 随机数和expr命令

### 生成随机数的特殊变量：

|  |  |
| --- | --- |
| echo $RANDOM | 范围是：【0，32767】 |

### expr：通用的表达式计算命令

|  |
| --- |
| 表达式中参数与操作符必须以空格分开，表达式中的运算可以是算数运算，比较运算，字符串运算和逻辑运算。 |

# 字符串操作

|  |  |
| --- | --- |
| ${ #var } | 返回字符串变量var 的长度。 |
| ${ var:m } （#m取值从0到${ #var }-1） | 返回${ var }中第m个字符到最后的部分 |
| ${ var:m:1en } （#m取值从0到${ #var }-1） | 返回${ var }中第m个字符开始，长度为1en的部分 |
| ${ var#pattern } | 删除${ var }中开头部分与pattern匹配的最小部分 |
| ${ var##pattern } | 删除${ var }中开头部分与pattern匹配的最大部分 |
| ${ var%pattern } | 删除${ var }中结尾部分与pattern匹配的最小部分 |
| ${ var%%pattern } | 删除${ var }中结尾部分与pattern匹配的最大部分 |
| ${ var/old/new } | 用new替换${ var }中第一次出现的old |
| ${ var//old/new } | 用new替换${ var }中所有的old（全局替换） |

注意：pattern，old中可以使用通配符。

# 脚本调试

### sh -x 脚本名

|  |
| --- |
| 该选项可以使用户跟踪脚本的执行，此时shell对脚本中每条命令的处理过程为：先执行替换，后显示，再执行它。 |
| shell显示脚本中的行时，会在行首添加一个加号“+” |

### sh -v 脚本名

|  |
| --- |
| 在执行脚本之前，按输入的原样打印脚本各行，打印一行执行一行。 |

### sh -n 脚本名

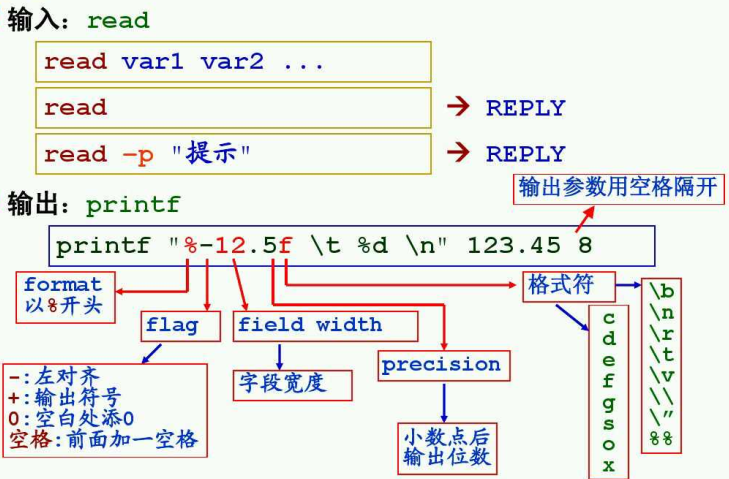
|  |
| --- |
| 对脚本进行语法检查，但不执行脚本。如果存在语法错误，shell会报错，如果没有错误，则不显示任何内容。 |

# 编程小结：

### 变量

|  |
| --- |
| 局部变量、环境变量（export、declare -x） |
| 只读变量、整型变量  举例：declare -i x；x=“hello”；echo $x |
| 位置参量（$0,$1,... ,$\*,$@,$#,$$,$?） |
| 变量的间接引用（eval,${ !str }）  举例：name=“hello”；x=“name”；echo ${ !x } |
| 命令替换（‘cmd’、$(cmd)） |
| 整数运算  declare定义的整型变量可以直接进行运算，否则需要let命令或$[...]、$((...))进行整数运算。 |

### 输入输出



### 条件测试

### 字符串测试

|  |  |
| --- | --- |
| [ -z str ] | 如果字符串str长度为0，返回结果为真 |
| [ -n str ] | 如果字符串str长度不为0，返回结果为真 |
| [ str1 = str2 ] | 两个字符串相等 |
| [ str1 != str2 ] | 两个字符串不相等 |

如果使用双方括号，可以使用通配符进行模式匹配

|  |  |
| --- | --- |
| [[ str1 = str2 ]] | 两个字符串相等(也可以使用==) |
| [[ str1 != str2 ]] | 两个字符串不相等 |
| [[ str1 > str2 ]] | str1 > str2,按ASCII码比较 |
| [[ str1 < str2 ]] | str1 < str2, 按ASCII码比较 |

### 整数测试

|  |  |
| --- | --- |
| [ int1 -eq int2 ] | int1 等于 int2 |
| [ int1 -ne int2 ] | int1 不等于 int2 |
| [ int1 -gt int2 ] | int1 大于 int2 |
| [ int1 -ge int2 ] | int1 大于或等于 int2 |
| [ int1 -lt int2 ] | int1 小于 int2 |
| [ int1 -le int2 ] | int1 小于或等于 int2 |
| [[ int1 == int2 ]] | int1 等于 int2 |
| [[ int1 != int2 ]] | int1 不等于 int2 |
| [[ int1 > int2 ]] | int1 大于 int2 |
| [[ int1 >= int2 ]] | int1 大于或等于 int2 |
| [[ int1 < int2 ]] | int1 小于 int2 |
| [[ int1 <= int2 ]] | int1 小于或等于 int2 |

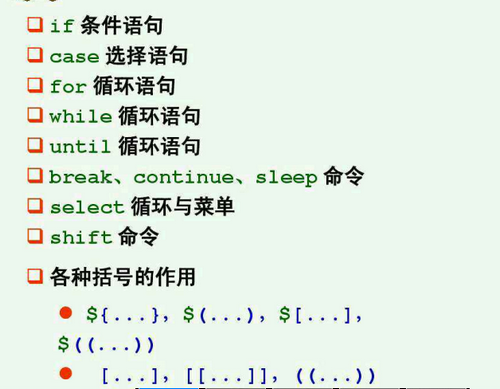
### c.逻辑测试

|  |  |
| --- | --- |
| [ expr1 -a expr2 ] | 逻辑与，都为真时，结果为真 |
| [ expr1 -o expr2 ] | 逻辑或，有一个为真时，结果为真 |
| [ ! expr ] | 逻辑非 |
| [[ pattern1 && pattern2 ]] | 逻辑与，都为真时，结果为真 |
| [[ pattern1 || pattern2 ]] | 逻辑或，有一个为真时，结果为真 |
| [[ ! pattern ]] | 逻辑非 |

### d.文件测试

|  |  |
| --- | --- |
| -f fname | fname 存在且是普通文件时，返回真（0） |
| -L fname | fname 存在且是链接文件时，返回真 |
| -d fname | fname 存在且是一个目录时，返回真 |
| -e fname | fname（文件或目录）存在时，返回真 |
| -s fname | fname 存在且大小大于0时，返回真 |
| -r fname | fname（文件或目录）存在且可读时，返回真 |
| -w fname | fname（文件或目录）存在且可写时，返回真 |
| -x fname | fname（文件或目录）存在且可执行时，返回真 |

### 控制结构



### 函数

