Linux——shell脚本基础3：shell函数【参数传递及输入输出】&内置函数

[函数定义 1](#_Toc323507224)

[退出状态 1](#_Toc323507225)

[参数传递 2](#_Toc323507226)

[标准IO 2](#_Toc323507227)

[脚本调试 2](#_Toc323507228)

[AND&OR 3](#_Toc323507229)

[内置命令补充 3](#_Toc323507230)

#### 函数定义

|  |
| --- |
| 函数定义： |
| 在Shell 中，函数就是一组命令集或语句形成一个可用块  函数增强了shell的可编程能力 |
| 组成 |
| function\_name(){  statements  }  函数名（在一个脚本中必须唯一）  函数体（命令集合） |
| 注意 |
| shell中不存在向前声明(先定义后使用)  必须在调用一个函数之前对它进行定义  函数可以定义自己内部使用的变量：local  像使用命令一样调用函数 |
| eg |
| function copyfile()  {  local srcfile=$1  local dstfile=$2  cp $srcfile $dstfile  return 0 # always return success  }  copyfile /tmp/myconf /etc/sysconf |
| 调用 |
| function\_name params  可以通过return命令让函数返回数字值，或者echo一个字符串返回 |

#### 参数传递

|  |
| --- |
| 参数传递： |
| 参数使用与位置参数一样  $1, $2, ……  $@, $\*  $#  func $var1 $var2  注：变量中间用空格、Tab分割 |
| 返回值 |
| return:主要用来返回退出状态,即$?  0 <= 返回值（整数） <= 255；  0 为无错误；  非0 为有错误；  注：如果返回其他数据，可以使用echo或者全局变量  return\_var=`func $var1 $var2` |

#### 退出状态

|  |  |
| --- | --- |
| 退出状态(exit status)： | |
| 函数返回一个被称为退出状态的值. 退出状态可以由return 来指定statement, 否则函数的退出状态是函数最后一个执行命令的退出状态(0 表示成功,非0 表示出错代码).  可以在脚本中由$? 引用  退出值为 0 表示成功  退出值非 0 表示失败  当一个命令因为严重的信号 N 退出时，bash 将使用 128+N 作为它的退出状态 | |
| 出现一个信号 | 128及以上  严重的信号 N 退出 |
| 命令没找到： | 127 |
| 文件找到但不可执行： | 126 |
| 展开或重定向中的错误：脚本程序可使用的错误代码 | 1-125 |
| 所有的内建命令在用法错误时返回 | 2 |
| 成功 | 0 |

#### 内置命令补充

|  |
| --- |
| 内置命令补充： |
| :命令 |
| 冒号(:)是一个空命令，偶尔用于简化条件逻辑，相当于true的一个别名。  例如 while :  也用于在变量的条件设置中  例如 : ${var:=value}  if ….;then  :  fi |
| .命令 |
| 用于在当前shell中执行命令  . ./shell\_script  相当于source命令  在执行脚本程序中列出的命令时，使用的是调用该脚本程序的同一个shell |
| 续行符\[enter] |
| 必须是行的最后一个字符 用于统一命令跨多行 |
| echo命令 |
| 输出结尾带有换行符的字符串  echo –n “string to output” bash风格去换行符 |
| eval命令 |
| 允许对参数进行求值，是shell的内置命令  即允许代码被随时生成和运行 |
| exec命令 |
| 两种用法  典型：将当前shell替换为一个不同的程序  exec wall “Thanks for all the fish”  第二种用法修改当前文件描述符  exec 3 < afile |
| exit n命令 |
| 使脚本程序以退出码n结束运行  若是在脚本退出时不指定退出状态，那么该脚本最后一条被执行命令状态将被用作返回值 |
| export命令 |
| 功能说明：设置或显示环境变量。  语　　法：export [-fnp][变量名称]=[变量设置值]  参　　数：  　-f 　代表[变量名称]中为函数名称。  　-n 　删除指定的变量。变量实际上并未删除，只是不会输出到后续指令的执行环境中。  　-p 　列出所有的shell赋予程序的环境变量  注意：  1、执行脚本时是在一个子shell环境运行的，脚本执行完后该子shell自动退出；  2、一个shell中的系统环境变量才会被复制到子shell中（用export定义的变量）；  3、一个shell中的系统环境变量只对该shell或者它的子shell有效，该shell结束时变量消失（并不能返回到父shell中）。  4、不用export定义的变量只对该shell有效，对子shell也是无效的。  补充说明：  1.在shell中执行程序时，shell会提供一组环境变量。export可新增，修改或删除环境变量，供后续执行的程序使用。export的效力仅及于该此登陆操作。  2.  export把自己的参数创建为一个环境变量，而这个环境变量可以被当前程序调用的其他脚本和程序看见  被导出变量构成从该shell衍生的任何子进程的环境变量 |
| expr命令 |
| 格式; expr argu operator argu [必带空格]  将它的参数当做一个表达式来求值，最常见用法简单数学运算  x=`expr $x + 1` 注意，运算符两边空格必须  x=$(expr $x + 1)  例子：   1. 直接计算   $expr 10 + 10  $expr 30 / 3  $expr 30 \\* 3 【必须转义】   1. 增量计算   $LOOP=10  $LOOP=`expr $LOOP + 1` //反引号   1. 数值测试   可用expr测试是否为数字  $VALUE=12  $expr $VALUE + 10 > /dev/null 2>&1  $echo $?  0   1. expr本身的退出状态   注：返回值与系统退出命令恰好相反  $VALUE=”hello”  $expr $VALUE = “hello”  1  $echo $?  0   1. 模式匹配及抽取   $VALUE=”hello”  $expr $VALUE : ‘.\*’  5  抽取  $VALUE=”account.doc”  $expr $VALUE : ‘\(.\*\).doc’  account |
| let命令 |
| 系统默认“＋”是作为字符串处理的，因此它不会被作为一个操作符，而只是被识别为一个普通的字符串  使用let强制为数字  $ d=111  $ echo $d+1  111+1  $ let d=$d+1; echo $d  112 |
| printf命令 |
| 格式化输出  d 十进制  C 一个字符  S 一个字符串  % 一个%字符  printf “%s\n” hello  printf “%s %d” “Hi” 6 |
| return命令 |
| 使函数返回，可有一个数值参数，作为该函数的返回值，如果不带参数，return命令默认返回最后一条命令的退出码 |
| set命令 |
| 为shell设置参数变量  set $(date)  echo The month is $2  即参数的值作为当前脚本参数列表了 |
| shift命令 |
| 把所有参数变量左移一个位置，是$2->$1 $3->$2原来$1被丢弃，$0保持不变  可带参数表示左移一个数值参数 |
| trap命令 |
| 用于指定接收到信号后要采取的行动  常见用途：在脚本程序被中断时完成清理工作  trap command signal  注：脚本程序通常是从上到下顺序解释执行的，必须在想保护那部分代码之前指定trap命令  HUP(1) 挂起，中断掉线或用户退出 引发  INT(2)中断 ctrl+c组合键发生  QUIT(3)退出，ctrl+\组合键发生  ABRT(6)中止，默写严重的执行错误引发  ALRM(14)报警，用来处理超时  TERM(15)终止，系统关机时发出 |
| unset命令 |
| 从环境变量中删除变量或函数  不能删除shell本身定义的只读变量 |