Linux shell编程常用方法总结

- **不因虚度年华而悔恨，不因碌碌无为而羞耻**
- **不积硅步，无以至千里；不积小流，无以成江海**
- **关注我，公众号：Linux兵工厂，后台回复C++获得更多实例代码，获取更多干货知识（Linux、网络、驱动、C/C++、后台服务、Qt、Python等）**

1. shell是什么

- shell是通过c语言编写的，是是用户和Linux之间的接口程序。编写shell脚本方便于系统管理。
- shell命令有两种形式：内部命令:内置在源码中，即存在内存中，比如:cd、echo；外部命令:存在于文件系统中某个目录下的单独的程序。
- shell按登录分：交互式非登录shell和非交互式shell。

> Unix shell，一种壳层与命令行界面，是UNIX操作系统下传统的用户和计算机的交互界面。第一个用户直接输入命令来执行各种各样的任务。普通意义上的shell就是可以接受用户输入命令的程序。它之所以被称作shell是因为它隐藏了操作系统低层的细节。

2. 各种shell

- 熟知的shell有： `Bourne shell` 、 `C shell` 、 `Korn shell` 等。
  `Bourne shell` ： 史蒂夫·伯恩在贝尔实验室时编写，1978年随Version7Unix首次发布。
  `C shell` ： C shell（csh）比尔·乔伊在加州大学伯克利分校时编写，1979年随BSD首次发布。
  `Korn shell` ： 是一款由大卫·科恩（David Korn）于二十世纪八十年代早期在贝尔实验室开发的

Unix shell，并在1983年7月14日的USENIX年度技术会议（英语：USENIX Annual Technical Conference）上发布

3. shell编程

1.shell变量

shell变量是shell设置的特殊变量,也是shell正确运行所必须的。分为局部变量和环境变量。

- 局部变量
  以字母、数字、_组成，以字母和下划线开头。其中数字开头的变量保留为shell本身使用。

```
MYVAR="hello"

# 访问变量
echo $MYVAR

#清除变量
unset MYVAR
```

终端下查看所有变量： `set`
说明一个变量为只读： `readonly` (无法用unset清除)

- 全局变量
  用export声明为全局变量，用unset清楚，终端下用env查看所有的环境变量。 `export设置的全局变量只是临时的，重启后失效。`
- 常用的环境变量
  `HOME` 保存用户目录
  `PATH` 保存用冒号分割的目录路径名
  `TERM` 终端类型(xterm图形终端 linux文本终端)
  `UID` 当前用户的标识符 取值是由数字构成的字符串
  `PWD` 当前工作目录的绝对路径名
  `PS1` 主提示符 # $
  `PS2` 辅助提示符 在输入行末尾 \ 输出该提示符
  `IFS` shell指定的缺省域分割符
  `LOGNAME` 保存登录名
  `SHELL` 保存缺省shell
  `RANDOM` 产生随机数

```
ubuntu@ubuntu:~/workspace_ex$ echo $HOME
/home/ubuntu
ubuntu@ubuntu:~/workspace_ex$ echo $PATH
/home/ubuntu/bin:/home/ubuntu/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu@ubuntu:~/workspace_ex$ echo $TERM
xterm-256color
ubuntu@ubuntu:~/workspace_ex$ echo $UID
1000
ubuntu@ubuntu:~/workspace_ex$ echo $PWD
/home/ubuntu/workspace_ex
ubuntu@ubuntu:~/workspace_ex$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
ubuntu@ubuntu:~/workspace_ex$ echo $PS2
>
ubuntu@ubuntu:~/workspace_ex$ echo $IFS

ubuntu@ubuntu:~/workspace_ex$ echo $LOGNAME
ubuntu
ubuntu@ubuntu:~/workspace_ex$ echo $SHELL
/bin/bash
ubuntu@ubuntu:~/workspace_ex$
```

- 数组变量

```
# 数组变量
ARRAY=(1 2 3)

# 显示第一个数组元素
echo ${ARRAY[0]}

# 显示数组长度
echo ${#ARRAY[*]}
echo ${#ARRAY[@]}

# 显示所有元素
echo ${ARRAY[@]}
echo ${ARRAY[*]}
```

2. 变量替换

变量=${var:-word} var为空或未设置，用word代替var进行替换，var值不变
变量=${var:=word} var为空或未设置，用word代替var进行替换，var值为word
变量=${var:=word} var不为空，变量值为var值，var值不变
变量=${var:?message} var为空或未设置，message作为标准错误打印出来
变量=${var:?message} var不为空，用message进行替换变量值，var值不变
变量=${var:+word} var设置，用word代替var进行替换，var值不变
变量=${var:+word} var未设置，变量值为空，var值不变仍未空

- 变量替换

${var} 或 $var: 变量值。
${var:-value}: 如果$var不为空(test -n $var)，使用$var；否则使用value
${var:=value}: 如果$var不为空，使用$var；否则将value赋给$var，并使用value
${var:?value}: 如果$var不为空，使用$var的值。否则，打印value并退出(exit)。类似断言$var不为空
${var:+value}: 如果$var不为空，则使用value，否则使用空
${#var}: 返回变量$var的长度。

- 删除指定字符串

${var#pattern} 将 $var 的值从左侧删除与模式 pattern 匹配的最短字符串并返回
${var##pattern} 将 $var 的值从左侧删除与模式 pattern 匹配的最长字符串并返回
${var%pattern} 将 $var 的值从右侧删除与模式 pattern 匹配的最短字符串并返回
${var%%pattern} 将 $var 的值从右侧删除与模式 pattern 匹配的最长字符串并返回

其中：#表示左侧 %表示右侧。一个#或%表示最短；两个#或%表示最长。

- 变量值替换

${var/pattern/replace} 将$var的值中第一个与模式pattern匹配的串替换为 replace 并返回。
${var/pattern} 将$var的值中第一个与模式pattern匹配的串删除并返回。
${var//pattern/replace} 将$var的值中所有与模式pattern匹配的串替换为replace*并返回。
${var/#pattern/replace} 将$var的值开头与模式pattern匹配的串替换为replace并返回。
${var/%pattern/replace} 将$var的值结尾与模式pattern匹配的串替换为replace并返回。

- 变量值大小写转换

${var^pattern} 将$var值开头与模式pattern匹配的串转换为大写。如果模式pattern省略，则将首字母转换为大写。
${var^^pattern} 将$var值中所有与模式pattern匹配的串转换为大写。如果模式pattern省略，则将整个$var转换为
${var,pattern} 将$var值开头与模式pattern匹配的串转换为小写。如果模式pattern省略，则将首字母转换为小写。
${var,,pattern} 将$var值中所有与模式pattern匹配的串转换为小写。如果模式pattern省略，则将整个$var转换为

- 变量值截取

${var:pos} 返回 $var 值从 pos 开始（到结尾）的子字符串。
${var:pos:len} 返回 $var 值从 pos 开始长度为 len 的子字符串。

- 命令算数替换

```
# 使用算数运算的结果替换算数表达式所在位置的内容$((1+1))

var=`expr 4 \ * 9`
let var=2**3 (mi yun suan)
myvar=`date`
echo $myvar
```

3. 特殊字符引用

- 关掉一个字符的特殊意义：" ", '', \
  单引号将消除被扩在单引号中的所有特殊字符的含义
  双引号以下字符的特殊含义不能删除：

> $,`,$,`,",\

4. 常用shell语句

if语句

```
if
  then

  elif
  then

  else

 fi
```

- test 测试命令 test expression 或 [空格expression空格]

```
if test $[num1] -eq $[num2]
then
    echo '两个数相等！'
else
    echo '两个数不相等！'
fi
```

- 文件测试

> -e 文件存在则为真
> -r 文件存在且可读为真
> -w 文件存在且可写为真
> -x 文件存在且可执行为真
> -s 文件存在且非空为真
> -d 文件存在且为目录为真
> -f 文件存在且为普通文件为真
> -c 文件存在且为字符型特殊文件
> -b 文件存在且为块特殊文件

- 字符串测试

> = 等于则为真
> != 不相等则为真
> -z 字符串为空串为真
> -n 字符串为非空串为真

- 数值测试

> -eq 等于
> -ne 非等于
> -lt 小于
> -gt 大于
> -le 小于等于
> -ge 大于等于

- 测试时使用逻辑操作符

> -a 逻辑与 操作两边均为真
> -o 逻辑或 操作两边一边为真则为真
> ! 逻辑非 条件为假则为真
> 优先级: ! -a -o
> if [-x file1 -a -x fiel2]
> fiel1 file2 都存在且可执行则为真
> 混合条件: && ||

case语句

```
case $变量名 in
      模式1)
      命令序列1
      ;;
      模式2)
      命令序列2
      ;;
      *)
      默认执行的命令序列
      ;;
esac
```

for循环

```
for i in 1 2 3 4
do
    echo $i
done
```

while循环

```
x=0
while [ $x -lt 10]
do

     echo $x
     x=$(($x+1))

done
```

select 循环

```
select c in c1 c2 all none
do
   case $ in
   )

   ;;
   )

   ;;


   *)

    ;;

    esac
done
```

特殊变量

用户只能根据shell的定义来使用这些变量，而不能重定义

```
$#     位置参数的数量
$*     所后位置参数的内容
$?     命令执行后返回的状态  获取函数的返回值
$$     当前进程的进程号
$!     后台运行的最后一个进程号
```

内置符号命令

```
. 执行文件
: 空操作
& 后台工作
$() 命令替换 a=$(date) echo $a
(()) 算数表达式计算 同let
$(())  算数扩展（不用于被括起来的值中包含=的情形）
[] 同 test
[[]] 同上
```

bash调试

> 启动调试 bash -选择项 shell程序文件名

- -n：不会执行该脚本，仅查询脚本语法是否有问题，并给出错误提示。
- -v：在执行脚本时，先将脚本的内容输出到屏幕上，然后执行脚本，如果有错误，也会给错误提示。
- -x：将执行的脚本内容及输出显示到屏幕上。