**Shell01概述变量数值运算**

**一 Shell**

**1.1 什么是Shell**

在Linux内核与用户之间的解释器程序

通常指/bin/bash

负责向内核翻译及传达用户/程序指令

相当于操作系统的”外壳”



**1.2 Shell的使用方式**

交互式 ---命令行

人工干预\智能化程度高

逐条解释执行\效率低

非交互式 ---脚本

需要提前设计\智能化程度高

批量执行,效率高

方便在后台悄悄的运行

**1.3常见的Shell程序种类**

[root@desktop0 ~]# **cat /etc/shells**

/bin/sh

/bin/bash

/sbin/nologin

/usr/bin/sh

/usr/bin/bash

/usr/sbin/nologin

/bin/tcsh

/bin/csh

[root@desktop0 ~]# sh #直接输入shell名称,切换解释器

sh-4.2# #sh解释器的命令行界面

sh-4.2# exit #从当前解释器中退出到bash

exit

[root@desktop0 ~]#

**1.4 安装ksh解释器**

[root@desktop0 ~]# yum -y install ksh

[root@desktop0 ~]# cat /etc/shells

/bin/sh

......

/bin/ksh #出现ksh,安装成功

[root@desktop0 ~]# ksh #切换到ksh解释器

# #ksh解释器命令行界面

# ls / #测试列出根下目录

bin dev home lib64 mnt proc run srv tmp var

boot etc lib media opt root sbin sys usr

ksh原生的解释器, 不支持history 和 tab 光标左右挪动

**1.5 bash 基本特性**

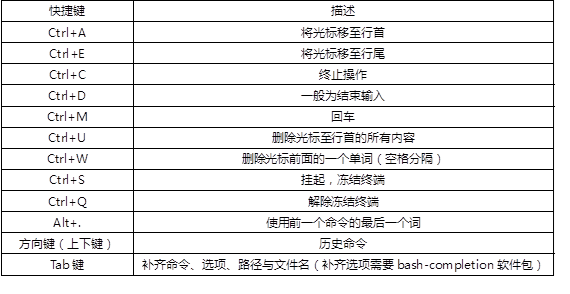
命令历史 history 快捷键(上下左右等) tab键

命令别名 alias 标准输入输出

重定向 管道操作

pstree: 查看进程树 1.8 中可用此命令查看进程树

**1.6 快捷键**



**ctrl + d** : 有exit功能 **ctrl + u** : 删除光标至行首的主要内容

**1.7 一个规范的脚本格式**

1.7.1声明解释器,作者信息

#!/bin/bash

#作者:

1.7.2 编写注释,解释脚本功能,步骤,变量含义等...

#xxxxxxxxxxxxx

1.7.3 编写代码 #代码和注释可穿插编写

案例: 编写一个脚本,输出hello word

[root@desktop0 opt]# vim test1.sh #创建脚本

#!/bin/bash #声明解释器

#这是一个测试脚本 #编写注释

echo "hello world" #编写代码

[root@desktop0 opt]# chmod u+x test1.sh #赋予可执行权限

[root@desktop0 opt]# ls #查看,**绿色**

test1.sh

[root@desktop0 opt]# ./test1.sh

#**以绝对路径或相对路径运行脚本**

hello world

**1.8 运行脚本方法**

1.8.1 赋予脚本执行权限,以绝对路径或相对路径运行脚本

1.8.2 调用解释器

[root@desktop0 opt]# **bash** test1.sh #除bash外,其他解释器也可以

hello world

#**调用解释器会开启子进程,脚本运行完成后退出,回到当前解释器下**

1.8.3 使用当前解释器(推荐方法: 不会开启子进程)

[root@desktop0 opt]# **source** test1.sh #使用当前解释器运行脚本

hello world

**在命令行中, . (点)=source, 当要注意空格**

案例: 创建目录abc.并进入目录abc

[root@desktop0 opt]# vim abc.sh

#!/bin/bash

mkdir ./abc

cd ./abc

[root@desktop0 opt]# source abc.sh #运行测试

案例: 测试调用解释器和使用当前解释器的效果

[root@desktop0 abc]#

[root@desktop0 opt]# vim test3.sh

#!/bin/bash

echo 123

exit

[root@desktop0 opt]# bash test3.sh

123

#调用bash运行test3.sh ,开启新的bash进程,执行echo 123,执行exit退出到当前bash 下

[root@desktop0 opt]# source test3.sh

123

Connection to 172.25.0.10 closed.

#在当前bash运行test3.sh,执行echo 123,执行exit退出远程连接ssh进程

**1.9 编写为本机快速配置yum的shell脚本**

[root@desktop0 opt]# vim yum.sh

#!/bin/bash

rm -rf /etc/yum.repos.d/\*.repo

echo "[abc]

name=test

baseurl=http:#classroom.example.com/content/rhel7.0/x86\_64/dvd/

enabled=1

gpgcheck=0" > /etc/yum.repos.d/abc.repo

[root@desktop0 opt]# source yum.sh #运行脚本

[root@desktop0 opt]# yum repolist #检查结果

已加载插件：langpacks

源标识 源名称 状态

abc test 4,305

repolist: 4,305

**1.10 编写为本机快速配置ftp的脚本**

[root@desktop0 ~]# vim ftp.sh

#!/bin/bash

yum -y install vsftpd &> /dev/null

systemctl restart vsftpd

systemctl enable vsftpd

[root@desktop0 ~]# systemctl stop firewalld

#模拟环境下关闭防火墙

[root@desktop0 ~]# source ftp.sh

ln -s '/usr/lib/systemd/system/vsftpd.service' '/etc/systemd/

system/multi-user.target.wants/vsftpd.service'

#出现此提示信息表示systemctl enable vsftpd设置成功

[root@desktop0 ~]# echo abc > /var/ftp/pub/abc.txt

[root@desktop0 ~]# cat /var/ftp/pub/abc.txt

abc

验证: 真机访问ftp:#172.25.0.10

[root@desktop0 ~]# rpm -q vsftpd #确认脚本执行结果

vsftpd-3.0.2-9.el7.x86\_64

[root@desktop0 ~]# systemctl status vsftpd

**二 变量**

**常量:**不会变化的量

**变量:**以固定名称存放,可能变化的值

提高脚本对任务需求\运行环境变化的适应能力

方面在脚本中重复使用

**2.1变量的定义**

**变量名称 = 变量的值**

由字母/数字/下划线组成,区分大小写; **不能以数字开头,不能用关键字和特殊字符**

**unset 变量名称** #取消变量定义,取消后变量全局消失

**2.2变量的种类**

自定义变量 由用户自主设置\修改及使用

环境变量 变量名通常大写,由系统维护,用来设置工作环境,个别变量用户可修改

位置变量 bash内置,存储执行脚本时提供的参数

预定义变量 bash内置,一般有特殊用途的变量,可直接调用,不能修改或赋值

**2.3 环境变量**

配置文件 /etc/profile /.bash\_profile

**env** : 列出所有环境变量 可与 | grep 配套使用

**set** : 列出所有变量 可与 | grep 配套使用

常见环境变量:PWD PATH USER LOGNAME UID SHELL HMOE PS1 PS2 HOSTNAME

PS1: 一级提示符 示例:[root@server0 ~]#

PS2: 二级提示符 用于折行

[root@desktop0 ~]# ls \

>

anaconda-ks.cfg ftp.sh

PATH: 存放系统命令的路径

**2.4 预定义变量**

用来保存脚本程序的执行信息

可直接使用这些变量 不能直接为这些变量赋值

变量名 含义

**$0 当前所在的进程名或脚本名**

**$$ 当前运行进程的PID**

**$? 命令执行后的返回状态, 0表示正常, 1或其他值表示异常**

**$# 已加载的位置变量的个数**

**$\* 所有位置变量的值**

案例: 编写脚本,创建用户tom,配置密码789

[root@desktop0 opt]# vim useradd.sh

#!/bin/bash

useradd tom &> /dev/null

echo 789 | passwd --stdin tom &> /dev/null

[root@desktop0 opt]# source useradd.sh

[root@desktop0 opt]# id tom

uid=1001(tom) gid=1001(tom) 组=1001(tom)

案例: 在执行脚本命令后输入用户和密码,然后创建输入的用户,并设置输入的密码

[root@desktop0 opt]# vim useradd.sh

#!/bin/bash

useradd $1 &> /dev/null

echo $2 | passwd --stdin $1 &> /dev/null

[root@desktop0 opt]# bash useradd.sh abcd 1234

[root@desktop0 opt]# id abcd

uid=1002(abcd) gid=1002(abcd) 组=1002(abcd)

**2.5 位置变量**

在执行交时提供的命令行参数

表示为$n, n为序号: $1 $2 ......${n}

**2.6 扩展赋值**

区分三种定界符:

双引号””: 允许扩展,以$引用其他变量  **#编写代码时建议使用此定界符**

单引号’’: 禁用扩展,即便$也视为普通字符,**屏蔽特殊符号的作用**

反撇号``: 将命令执行后的输出作为变量值 #与$( )等效,后者更方便嵌套使用

**2.7 read标准输入取值**

read从键盘读入变量值完成赋值

**格式: read [-p “提示信息”] 变量名**

-p 可选, -t 可指定超时秒数

终端显示控制

**stty -echo** : 关闭终端输出(无显示)

**stty echo**: 恢复终端输出(显示)

案例:编写脚本,弹出提示信息提示输入用户名和密码,输入完成后创建用户

[root@desktop0 opt]# vim useradd1.sh

#!/bin/bash

read -p "请输入用户名:" u

useradd ${u} &> /dev/null

read -p "请输入密码:" p

echo ${p} | passwd --stdin ${u} &> /dev/null

[root@desktop0 opt]# bash useradd1.sh

请输入用户名:xyz

请输入密码:123

[root@desktop0 opt]# id xyz

uid=1003(xyz) gid=1003(xyz) 组=1003(xyz)

案例:编写脚本,弹出提示信息提示输入用户名和密码,输入完成后创建用户,输入密码时不显示密码

[root@desktop0 opt]# vim useradd2.sh

#!/bin/bash

read -p "请输入用户名:" u

useradd ${u} &> /dev/null

stty -echo

read -p "请输入密码:" p

stty echo

echo ${p} | passwd --stdin ${u} &> /dev/null

[root@desktop0 opt]# bash useradd2.sh

请输入用户名:xyzz

请输入密码:

[root@desktop0 opt]# id xyzz

uid=1004(xyzz) gid=1004(xyzz) 组=1004(xyzz)

**2.8 变量的作用范围**

局部变量

新定义的变量默认只在当前shell环境中有效

无法在子shell环境中使用

全局变量

在当前shell及子shell环境中均有效

使用export可将局部变量声明为全局变量

格式

**export 局部变量名[=变量值]...** #为局部变量添加全局性

**export -n 全局变量名......** #取消指定变量的全局属性

unset 变量名 #取消变量,取消后变量全局消失

/etc/profile 53行,系统定义的全局变量

**三 整数运算**

**3.1 四则运算**

加法: nmu1 + num2

减法: nmu1 - num2

乘法: nmu1 \* num2

除法: nmu1 / num2

取余(求模): nmu1 % num2

**3.2 expr运算工具 支持常量和变量**

计算指定的表达式,并输出结果

**格式: expr 整数1 运算符 整数2 .....**. **注意空格**

乘法操作应采用\\*转义,避免被作为shell通配符

**运算符两侧必须有空格**

**不需与echo结合即可输出**

类型 运算符 示例

加法 + expr 43 + 21 expr $X + $Y

减法 - expr 43 - 21 expr $X - $Y

乘法 \\* expr 43 \\* 21 expr $X \\* $Y

除法 / expr 43 / 21 expr $X / $Y

取余数 % expr 43 % 21 expr $X % $Y

[root@svr5 ~]# X=1234 #定义变量X

[root@svr5 ~]# expr $X + 78 #加法

1312

[root@svr5 ~]# expr $X - 78 #减法

1156

[root@svr5 ~]# expr $X \\* 78 #乘法，操作符应添加\转义

96252

[root@svr5 ~]# expr $X / 78 #除法，仅保留整除结果

15

[root@svr5 ~]# expr $X % 78   #求模

64

**3.3 $[] 算式替换**

使用$[]或$(())表达式

**格式: $[整数1 运算符 整数2 ......]**

乘法操作\*无需转义,运算符两侧可以无空格

引用变量可省略$符号

计算结构替换表达式本身,**结合echo命令输出**

[root@desktop0 opt]# echo $[11+2]

13

[root@desktop0 opt]# echo $[11-2]

9

[root@desktop0 opt]# echo $[11\*2]

22

[root@desktop0 opt]# echo $[11/2]

5

[root@desktop0 opt]# echo $[11%2]

1

[root@desktop0 opt]# a=10

[root@desktop0 opt]# echo $[a+a]

20

**3.4 变量的自增减等操作**

使用$[]替换,或者let命令来完成

**let 命令执行完成后,使用echo命令输出**

**简写表达式 完整表达式**

**i++ i=i+1**

**i-- i=i-1**

**i+=2 i=i+2**

**i-=2 i=i-2**

**i\*=2 i=i\*2**

**i/=2 i=i/2**

**i%=2 i=i%2**

[root@desktop0 opt]# i=43

[root@desktop0 opt]# echo $[i+=2]

45

[root@desktop0 opt]# echo $[i-=8]

**#$[]和let的输出方式不同**

37

[root@desktop0 opt]# **let i++**;echo $i

**#$[]和let的输出方式不同**

38

[root@desktop0 opt]# **let i-=7**;echo $i

**#let可以改变变量的值**

24

**expr或$[]方式只进行运算，并不会改变变量的值；而let命令可以直接对变量值做运算再保存新的值。另外，$[]和let运算操作并不显示结果，但是可以结合echo命令来查看.**

**运算方式 乘法是否需要转义 是否改变变量值 输出方式 变量运算方式**

**expr 是 否 直接 加$**

**$[] 否 否 echo 直接运算**

**let 否 是 echo 直接运算**