# NSD SHELL DAY01

案例1：Shell基础应用

案例2：简单Shell脚本的设计

案例3：使用Shell变量

案例4：变量的扩展应用

案例5：read

# 1 案例1：Shell基础应用

1.1 问题

本案例要求熟悉Linux Shell环境的特点，主要练习以下操作：

1. 切换用户的Shell环境
2. 练习命令历史、命令别名
3. 重定向标准输入/输出/错误输出
4. 管道操作实践

## 步骤一：切换用户的Shell环境

**什么是shell？**

Shell是在Linux内核与用户之间的解释器程序，通常指的是bash，负责向内核翻译及传达用户/程序指令，如图-1所示。



**shell种类:bash sh csh ksh**

[root@server0 ~]# cat /etc/shells

/bin/sh

/bin/bash ----默认使用

/sbin/nologin

/usr/bin/sh

/usr/bin/bash

/usr/sbin/nologin

/bin/tcsh

/bin/csh

[root@server0 ~]# sh ---------更换使用/bin/sh shell

sh-4.2#

sh-4.2# exit ------退出

exit

shell的使用方式：

1.交互执行指令：人工干预，执行效率底。

2.非交互执行指令：安静地在后台执行，执行效率高，方便写脚本。

若需要临时使用另一种Shell环境，可以直接执行对应的Shell解释器程序，比如只要执行ksh可以切换到ksh命令行环境。

[root@svr5 ~]# yum -y install ksh //若缺少ksh请先安装ksh包

1. [root@svr5 ~]# cat /etc/shells
2. /bin/sh
3. /bin/bash
4. /sbin/nologin
5. /bin/tcsh
6. /bin/csh
7. /bin/ksh //确认当前系统已识别ksh
8. [root@svr5 ~]# ksh //进入ksh环境
9. [root@svr5]~# exit //返回到切换前的bash环境

若希望修改用户的登录Shell，管理员可以直接通过usermod（或useradd）命令设置。比如，以下操作可将用户zhangsan的登录Shell改为/bin/ksh：

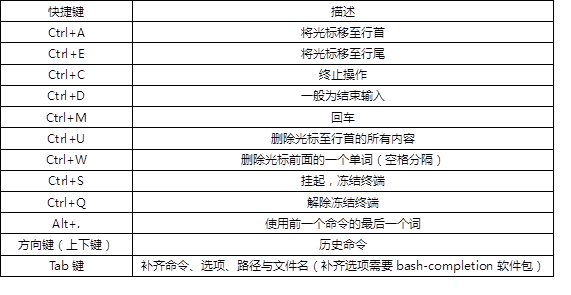
[root@svr5 ~]# usermod -s /bin/ksh zhangsan             //执行修改操作[root@svr5 ~]# grep 'zhangsan' /etc/passwd

zhangsan:x:516:516::/home/zhangsan:/bin/ksh //修改后

## 步骤二：bash特点，快捷键与

Tab键补齐，常见快捷键如表-1所示

支持：tab、管道、历史记录、支持别名、快捷键

**Ctrl + K 向后删除命令串**

## 步骤三：练习命令历史

### 1）检查历史命令的容量。

默认记录1000条，通过全局变量HISTSIZE设置，对所有用户有效：

[root@svr5 ~]# grep HISTSIZE /etc/profile

HISTSIZE=1000

[root@svr5 ~]# history | wc -l #查看已为当前用户记录的历史命令条数：

1000

**修改历史记录长度**

[root@server0 ~]# vim /etc/profile

HISTSIZE=1000 ---修改此参数，设置历史记录条数

[root@server0 ~]# history -c ---清空历史记录的缓存,重新打开终端记录还有

[root@server0 ~]# ls -a

. .bashrc Documents Pictures test.sh

.. .cache Downloads Public Videos

anaconda-ks.cfg .config .esd\_auth .sh\_history .viminfo

.bash\_history .cshrc .ICEauthority .ssh .Xauthority

[root@server0 ~]# > .bash\_history --重定空白向覆盖.bash\_history文件，永久清空历史记录 或者删除 .bash\_history文件

### 2）查看历史命令列表。

列出最近执行的10条历史命令：

[root@server0 ~]# history | tail

43 yum -y install ksh

44 cat /etc/shells

45 ksh

46 cat /etc/passwd

47 usermod -s /bin/ksh nb1

48 su - nb1

49 grep histsize /etc/profile

50 grep HISTSIZE /etc/profile

51 history | wc -l

52 history | tail

### 3）调用指定的历史命令。

重新执行历史命令列表中的第48条操作：

[root@server0 ~]# !48

su - nb1

上一次登录：三 1月 9 18:52:53 CST 2019pts/0 上

[root@server0 ~]# !cat ---重新执行下以cat开头的命令

### 4）清空历史的命令。

[root@svr5 ~]# history -c //清空自己的历史命令

[root@svr5 ~]# > ~/.bash\_history //清空记录文件

[root@svr5 ~]# history //再次检查历史命令列表

42 > ~/.bash\_history

43 history

## 步骤四：练习命令别名

## 1）查看已经定义的命令别名列表。

当前的别名列表：

[root@svr5 ~]# alias

alias cp='cp -i'

alias l.='ls -d .\* --color=tty'

alias ll='ls -l --color=tty'

… …

[root@server0 ~]# alias lh='ls -lh' -- 临时设置别名

[root@server0 ~]# ls -a

. .bashrc Documents Pictures test.sh

别名设置一般存放在用户的.bashrc文件内：

[root@svr5 ~]# grep '^alias' ~/.bashrc

alias rm='rm -i'

alias cp='cp -i'

alias mv='mv -i'

[root@server0 ~]# vim .bashrc ---修改.bashrc 文件 永久修改别名

### 2）自定义新的命令别名

添加一个名为lh的命令别名，实际执行的是“ls -lh”：

[root@svr5 ~]# alias lh='ls -lh' //定义别名命令lh

[root@svr5 ~]# alias lh //确认定义结果

alias lh='ls -lh'

验证别名命令的效果：

[root@svr5 ~]# lh /etc/fstab //使用别名

-rw-r--r-- 1 root root 733 10-09 15:34 /etc/fstab

[root@svr5 ~]# ls -lh /etc/fstab //使用完整的命令

-rw-r--r-- 1 root root 733 10-09 15:34 /etc/fstab

### 3）取消别名

取消单个别名：

[root@svr5 ~]# unalias lh //取消名为lh的命令别名

[root@svr5 ~]# alias lh //查询时已没有lh

-bash: alias: lh: not found

## 步骤四：重定向标准输入/输出/错误输出

标准输入（stdin），描述号为0；

标准输出（stdout），描述号为1；

标准错误（stderr），描述号为2。

### 1）重定向标准输出。

使用 > 将命令执行的正常输出重定向到文件：

> :收集正确的输出 2>收集错误的输出 &>收集所有

[root@svr5 ~]# ls -ld /etc/ //正常应输出到屏幕

drwxr-xr-x. 140 root root 8192 8月 2 04:45 /etc/

[root@svr5 ~]# ls -ld /etc/ > stdout.txt //重定向到文件

[root@svr5 ~]# cat stdout.txt //确认重定向输出的结果

drwxr-xr-x. 140 root root 8192 8月 2 04:45 /etc/

> 操作会覆盖目标文件（先清空、再写入）：

[root@svr5 ~]# echo "I am the king." > stdout.txt //覆盖目标文件

[root@svr5 ~]# cat stdout.txt //确认结果

I am the king.

改用 >> 可实现追加重定向输出：

[root@svr5 ~]# ls -ld /etc/ >> stdout.txt //追加输出

[root@svr5 ~]# cat stdout.txt

I am the king. //原有内容还保留

drwxr-xr-x. 140 root root 8192 8月 2 04:45 /etc/

### 2）重定向标准错误。

对于命令执行出错的信息，使用 > 无法保存，仍然会输出到屏幕。比如，可使用ls命令同时查看两个对象（其中nb.txt并不存在），重定向输出：

[root@svr5 ~]# ls -l nb.txt /etc/fstab > stderr.txt

[root@svr5 ~]# cat stderr.txt //正常信息成功重定向到目标文件

-rw-r--r--. 1 root root 541 1月 5 2017 /etc/fstab

使用 2> 可重定向错误信息，比如，可执行一个错误的命令：

[root@svr5 ~]# ls -l nb.txt /etc/fstab 2> stderr.txt

-rw-r--r--. 1 root root 541 1月 5 2017 /etc/fstab

//正确的信息默认输出至屏幕，错误信息重定向到目标文件

[root@svr5 ~]# cat stderr.txt //从文件中查看出错信息

ls: nb.txt: 没有那个文件或目录

类似的，2>> 可实现追加输出：

[root@svr5 ~]# ls tmpfile 2>> stderr.txt

[root@svr5 ~]# cat stderr.txt

ls: nb.txt: 没有那个文件或目录

ls: tmpfile: 没有那个文件或目录

若希望将正常输出、错误输出重定向同一个文件，可使用 &> :

[root@svr5 ~]# ls -l nb.txt /etc/fstab &> stderr.txt

[root@svr5 ~]# cat stderr.txt

ls: nb.txt: 没有那个文件或目录

-rw-r--r--. 1 root root 541 1月 5 2017 /etc/fstab

### 3）重定向标准输入。

[root@svr5 ~]# mail -s Error root < /etc/passwd

### 4）案例

[root@svr5 ~]# cat /root/test.sh

#!/bin/bash

for i in {1..10}

do

useradd user$i 2>>/root/user.log

echo '123456' | passwd --stdin user$i >/dev/null

done

对于命令执行出错的信息，使用 > 无法保存，仍然会输出到屏幕。比如，可使用ls命令同时查看两个对象（其中123并不存在），重定向输出：

[root@server0 ~]# ls /123 /opt

ls: 无法访问/123: 没有那个文件或目录

/opt:

rh

[root@server0 ~]# ls /123 /opt >1.txt ---收集正确输出

ls: 无法访问/123: 没有那个文件或目录

[root@server0 ~]# cat 1.txt

/opt:

rh

[root@server0 ~]# ls /123 /opt 2>2.txt ---收集错误输出

/opt:

rh

[root@server0 ~]# cat 2.txt

ls: 无法访问/123: 没有那个文件或目录

[root@server0 ~]# ls /123 /opt &>2.txt ---正确错误全部收集

[root@server0 ~]# cat 2.txt

ls: 无法访问/123: 没有那个文件或目录

/opt:

rh

## 步骤五：管道操作实践

借助于管道符“|”，可以将一条命令的标准输出交给另一条命令处理，在一条命令行内可依次使用多个管道。

### 1）统计/etc/目录下资料的数量。

[root@svr5 ~]# ls -l /etc | wc -l

254

### 2）列出Yum库里名称中含cluster的软件包。

[root@svr5 ~]# yum list | grep cluster

cluster-cim.x86\_64 0.12.1-7.el5 RHEL5-Cluster

cluster-snmp.x86\_64 0.12.1-7.el5 RHEL5-Cluster

… …

**管道实践操作**

[root@server0 ~]# cat /etc/passwd | wc  ----wc 统计

49 80 2377

[root@server0 ~]# cat /etc/passwd | wc -l ----wc 统计行数

49

# 2 案例2：简单Shell脚本的设计

2.1 问题

本案例要求编写三个脚本程序，分别实现以下目标：

* 在屏幕上输出一段文字“Hello World”
* 能够为本机快速配好Yum仓库
* 能够为本机快速装配好vsftpd服务

2.2 方案

一个规范的Shell脚本构成包括：

* 脚本声明（需要的解释器、作者信息等）
* 注释信息（步骤、思路、用途、变量含义等）
* 可执行语句（操作代码）

## 步骤一：编写第一个Shell脚本，

输出“Hello World”

### 1）根据手动任务操作编写脚本文件

[root@svr5 ~]# vim /root/first.sh

#!/bin/bash

echo 'Hello World'

exit

[root@svr5 ~]# chmod +x /root/first.sh //添加可执行权限

### 2）执行脚本，测试效果

[root@svr5 ~]# /root/first.sh

Hello World

### Shell脚本的执行方式：

执行脚本

1.赋予X权限 使用路径

2.解释器 脚本名称 (解释器有:bash sh 等等)

3.Source 脚本文件名 =. 脚本文件

Bash 与 source 的区别

Bash 开启新的解释器/进程来执行,,,执行新开进程,执行完毕时候结束进程

Souce 不开启新的解释器/进程执行 ,在当前的进程执行

方法一，作为“命令字”：指定脚本文件的路径，前提是有 x 权限

[root@svr5 ~]# ./first.sh //指定相对路径

[root@svr5 ~]# /root/first.sh //指定绝对路径

方法二，作为“参数”：使用bash、sh、source来加载脚本文件

[root@svr5 ~]# bash first.sh //开启子进程

[root@svr5 ~]# sh first.sh //开启子进程

[root@svr5 ~]# source first.sh //不开启子进程

## 步骤一：创建脚本:自动创建YUM源,且安装vstftpd

[root@server0 ~]# vim yum.sh

---------------------------------------------------

#! /bin/bash

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

echo “[YUM]

name=yum

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

enabled=1

gpgcheck=0” >/etc/yum.repos.d/yum.repo

echo "当前可用yum"

yum repolist | tail -1

yum install -y vsftpd &> /dev/null

echo "vsftpd安装成功"

systemctl start vsftpd

systemctl enable vsftpd &> /dev/null

echo "vsftpd已设置开启,并开机自启"

# 3 案例3：使用Shell变量

3.1 问题

本案例要求熟悉Shell变量的使用，主要练习或验证下列内容：

1. 定义/赋值/查看变量
2. 环境/预定义/位置变量的应用

除了学会建立和引用变量以外，还要认识环境变量PWD、USER、HOME、SHELL，还有预定义变量$0、$$、$?、$#、$\*，以及位置变量$1、$2、$10、……的作用。

## 步骤一：变量的定义/赋值/查看

### 1）新建/赋值变量

新建变量test，赋值“hello world”，通过set命令可以检查变量设置：

[root@svr5 ~]# test=11

### 2）查看变量

通过echo $变量名 可输出变量值：

[root@svr5 ~]# echo $test

11

查看变量时，若变量名称与后面要输出的字符串连在一起，则应该以{}将变量名括起来以便区分：

[root@svr5 ~]# echo $testRMB //无法识别变量名test

[root@svr5 ~]# echo ${test}RMB //区分后可以识别

11RMB

### 3）撤销自定义变量

若要撤销已有的变量，可使用unset命令：

[root@svr5 ~]# unset test //撤销变量test

[root@svr5 ~]# echo $test //查看时已无结果

## 步骤二：使用环境变量

### 1）查看环境变量相关文件

全局文件为/etc/profile，对所有用户有效；用户文件为~/.bash\_profile，仅对指定的用户有效。

查看/etc/profile文件内容：

[root@svr5 ~]# cat /etc/profile

.. ..

HOSTNAME=`/bin/hostname`

HISTSIZE=1000

.. ..

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC

.. ..

### 2）使用环境变量

当前用户的环境变量USER记录了用户名、HOME记录了宿主目录、SHELL记录了登录Shell、HOSTNAME记录主机名、UID是用户的id号：

[root@svr5 ~]# echo $USER $HOME $SHELL $UID

root /root /bin/bash 0

[root@svr5 ~]# echo $HOSTNAME

svr5

环境变量PS1表示Shell环境的一级提示符，即命令行提示符（\u 用户名、\h 主机名、\W 工作目录、\$ 权限标识）：

[root@svr5 src]# echo $PS1 //查看默认的一级提示

[\u@\h \W]\$

[root@svr5 src]#PS1='hehe#' //修改一级提示

hehe# //更改结果

hehe# PS1='[\u@\h \W]\$ ' //恢复原有设置

[root@svr5 src]#

环境变量PS2表示二级提示符，出现在强制换行、at任务编辑等场合：

[root@svr5 ~]# echo $PS2 //查看默认的二级提示

>

[root@svr5 src]# cd \ //强制换行，观察提示符效果

> /root/

[root@svr5 ~]# PS2='=> ' //手动修改二级提示

[root@svr5 ~]# cd \ //再次验证提示符效果

=> ~

[root@svr5 ~]# PS2='> ' //恢复原有设置

### 3）查看系统变量

使用env可查看所有环境变量：

[root@svr5 src]# env

HOSTNAME=svr5.tarena.com

SHELL=/bin/bash

HISTSIZE=1000

SSH\_CLIENT=192.168.4.110 59026 22

OLDPWD=/root

SSH\_TTY=/dev/pts/0

USER=root

.. ..

使用set可查看所有变量（包括env能看到的环境变量）：

[root@svr5 src]# set

BASH=/bin/bash

BASH\_ARGC=()

BASH\_ARGV=()

BASH\_LINENO=()

.. ..

## 步骤三：使用位置变量与预定义变量

### 1）创建一个测试脚本，用来展示。

[root@svr5 ~]# vim location.sh

#!/bin/bash

echo $0 //脚本的名称

echo $1 //第一个参数

echo $2 //第二个参数

echo $\* //所有参数

echo $# //所有的综合

echo $$ //当前进程的进程号

echo $? //上一个程序的返回状态码

[root@svr5 ~]# chmod +x location.sh //添加可执行权限

### 执行脚本location.sh，

细心观察结果（高明的武功需要用心参悟）。

[root@svr5 ~]# ./location.sh one 1 2 abc qq 8 7

## 步骤四：创建账户与修改密码的脚本

### 1）编写脚本。

[root@svr5 ~]# vim /root/user.sh

#!/bin/bash

useradd $1

echo "$2" |passwd --stdin $1

### 2) 执行脚本测试：

[root@svr5 ~]# ./user.sh jerry 123456

更改用户 jerry 的密码 。

passwd： 所有的身份验证令牌已经成功更新。

# 4 案例4：变量的扩展应用

4.1 问题

本案例要求进一步熟悉Shell变量的赋值控制，主要练习或验证下列内容：

三种引号对赋值的影响

使用read命令从键盘读取变量值

使用export发布全局变量

## 步骤一：三种引号对变量赋值的影响

### 1）双引号的应用

使用双引号可以界定一个完整字符串。

[root@svr5 ~]# xx=Tarena IT Group

-bash: IT: command not found //未界定时赋值失败

[root@svr5 ~]# xx="Tarena IT Group" //界定后成功

[root@svr5 ~]# touch aa bb //创建了两个文件

[root@svr5 ~]# touch "aa bb" //创建了一个文件

[root@svr5 ~]# ls //查看结果

### 2）单引号的应用

界定一个完整的字符串，并且可以实现屏蔽特殊符号的功能。

[root@svr5 ~]# test=11

[root@svr5 ~]# echo "$test"

11

[root@svr5 ~]# echo '$test'

$test

### 3）反撇号或$()的应用

使用反撇号或$()时，可以将命令执行的标准输出作为字符串存储，

因此称为命令替换。

[root@svr5 ~]# tar -czf log-`date +%Y%m%d`.tar.gz /var/log

## 步骤二：使用read命令从键盘读取变量值

### 1）read基本用法

执行后从会等待并接受用户输入（无任何提示的情况），并赋值给变量str：

[root@svr5 ~]# read str

What's happen ? //随便输入一些文字，按Enter键提交

[root@svr5 ~]# echo $str //查看赋值结果

What's happen ?

为了不至于使用户不知所措、莫名其妙，推荐的做法是结合-p选项给出友好提示：

[root@svr5 ~]# read -p "请输入一个整数：" i

请输入一个整数：240

[root@svr5 ~]# echo $i

240

### 2）stty终端显示控制

将回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

可参考下列操作创建一个测试脚本：

[root@svr5 ~]# vim user.sh //创建一个测试脚本

#!/bin/bash

read -p "请输入用户名:" username //读取用户名

stty -echo //关闭回显

read -p "请输入密码:" passwd //读取密码

stty echo //恢复回显

echo "" //恢复回显后补一个空行

useradd "$username"

echo "$passwd" | passwd --stdin "$username"

[root@svr5 ~]# chmod +x user.sh //添加执行权限

执行测试脚本user.sh，验证效果：

[root@svr5 ~]# ./user.sh

请输入用户名: root //输入root，回车

请输入密码: //输入1234567（不会显示），回车

## 步骤三：使用export发布全局变量

默认情况下，自定义的变量为局部变量，只在当前Shell环境中有效，而在子Shell环境中无法直接使用。比如已定义的SCHOOL变量，当进入到sh或bash子Shell后，变量SCHOOL将处于未定义的状态：

[root@svr5 ~]# yy="Tarena IT Group"

[root@svr5 ~]# echo $yy

Tarena IT Group

[root@svr5 ~]# bash //开启bash子进程

[root@svr5 ~]# echo $yy //查看SCHOOL变量值无结果

[root@svr5 ~]# exit //返回原有Shell环境

exit

[root@svr5 ~]# echo $yy

若希望定义的变量能被子进程使用，可以使用export命令将其发布为全局变量。使用export发布时，只需指定变量名（可以有多个）即可，也可以通过export

命令直接设置新的全局变量：

[root@svr5 ~]# export yy //发布已定义的变量

[root@svr5 ~]# export XX="1234" //发布新变量

验证刚刚发布的全局变量：

[root@svr5 ~]# bash //进入bash子Shell环境

[root@svr5 ~]# echo $yy //查看全局变量的值 .. ..

Tarena IT Group

[root@svr5 ~]# echo $XX

1234

**SHELL变量**

**让某一个特定字符串代表不固定的内容**

1.**自定义变量:**自定义/赋值/查看变量

2.**系统变量:**环境变量/预定义变量/位置变量的应用

了让脚本的增加灵活度,适应多变的环境,使用变量

变量=容器 ----以不变的名称,存放可以变化的值

**自定义变量:**

变量名=存放的值

定义/赋值变量

• 设置变量时的注意事项

– 变量名只能由字母/数字/下划线组成,区分大小写

– 变量名不能以数字开头,不要使用关键字和特殊字符

– 若指定的变量名已存在,相当于为此变量重新赋值

– 等号两边不要有空格

• 基本格式

– 引用变量值:$变量名

– 查看变量值:echo $变量名、echo ${变量名}

双引号” ”:界定一个完整的字符串

单引号 ' ' :界定一个完整的字符串,并取消特殊字符的意义,原样输出

例如:

[root@server0 ~]# a=10

[root@server0 ~]# echo "$a"

10

[root@server0 ~]# echo '$a'

$a

[root@server0 ~]# echo \$a

$a

反撇号 ` ` 或 $( ) :将命令的输出,直接用于其他命令运行

**系统变量:** 环境变量\位置变量\预定义变量

根据变量的用途不同区分

环境变量: 变量名一般都大写,用来设置用户/系统环境

由系统定义完成,由系统赋值完成

认识环境变量PWD、USER、HOME、HOSTNAME SHELL UID

RANDOM=

USER=当前系统登陆的用户名

HOME=当前用户家目录

PWD=当前目录

HOSTNAME=主机名

SHELL=当前用户的解释器

UID=

PATH=外部命令路径所在地,将创建的脚本文件放入PATH路径下,可实现直接运行

[root@server0 ~]# echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

PS1=

例如:

[root@server0 ~]# echo $PS1

[\u@\h \W]\$

[root@server0 ~]# PS1=haha

haha ---将提示符[root@server0 ~]变成了haha,重启终端恢复

PS2=二级提示符

例如:

[root@server0 ~]# ls \

>

[root@server0 ~]# PS2=你好

[root@server0 ~]# ls \

你好 ----将>变成了’你好’

位置变量: bash内置,存储执行脚本时提供的命令行参数

由系统定义完成,由系统赋值完成

表示为 $n,n为序号 $1、$2、.. .. ${10}、${11}、.. ..

为了降低脚本的使用难度,提高用户的体验度,采用交互式传递参数方法

预定义变量: bash内置,可直接调用的特殊值,不能直接修改

由系统定义完成,由系统赋值完成

预定义变量$0、$$、$?、$#、$\*

$0 脚本名称

$$ 当前进程的进程号(PID)

$# 已加载的位置变量的个数

$\* 所有位置变量的值

$? 程序退出后的状态值, 0表示正常,其他值异常

$! 最后一个程序的进程号

$@ 与$\*号类似.可当做数组使用

**stty**终端显示控制

将回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

例如:

创建用户,输入密码时盲输

--------------------------------------

#! /bin/bash

read -p "请输入用户名:" n

useradd $n

stty -echo

read -p "请输入密码:" p

stty echo

echo

echo $p | passwd --stdin $n

------------------------------------------

**export发布全局变量**

默认情况下，自定义的变量为局部变量，只在当前Shell环境中有效，而在子Shell环境中无法直接使用。比如已定义的SCHOOL变量，当进入到sh或bash子Shell后，变量SCHOOL将处于未定义的状态：

[root@server0 ~]# p=20

[root@server0 ~]# echo $p

20

[root@server0 ~]# bash ----进入bash 的子shell

[root@server0 ~]# echo $p

---bash 进入子shell 后变量p的值无效了

[root@server0 ~]# exit

exit

若希望定义的变量能被子进程使用，可以使用export命令将其发布为全局变量。使用export发布时，只需指定变量名（可以有多个）即可，也可以通过export命令直接设置新的全局变量：

第一种定义方法:通过export命令直接设置新的全局变量

[root@server0 ~]# export p=20

[root@server0 ~]# echo $p

20

[root@server0 ~]# bash

[root@server0 ~]# echo $p

20

[root@server0 ~]#

第二种定义方法:发布已定义的变量

[root@server0 ~]#p=20

[root@server0 ~]# export p                             //发布已定义的变量

脚本格式

1,定义解释器

2,注释

3,代码

## date  ：  设定 和 显示日期

date MMDDhhmm[cc]YY.ss  
　　date [+FORMAT]  
　　%Y:四位年份  
　　%y：两位年份  
　　%m:月  
　　%M：分钟  
　　%d：日  
　　%h：英文简写的月  
　　%H：时  
　　%S：秒  
　　%s：现在距离1970年1月1号0点0分（unix元年）的秒数，timestamp（时间戳）  
　　%D:月/日/年  
　　%F:年-月-日  
　　%T:时：分：秒

[root@server0 ~]# date +%Y%m%d

20190111

[root@server0 ~]#

**read作用:**

1.产生交互

2.自动记录用户在键盘上的输入

3.自动将用户在键盘上的输入,赋值给一个变量储存

为了不至于使用户不知所措、莫名其妙，推荐的做法是结合-p选项给出友好提示：

[root@svr5 ~]# read -p "请输入一个整数：" i

使用read命令存在着潜在危险。脚本很可能会停下来一直等待用户的输入。如果无论是否输入数据脚本都必须继续执行，那么可以使用-t选项指定一个计时器。

-t选项指定read命令等待输入的秒数。当计时满时，read命令返回一个非零退出状态;

#! /bin/bash

read -t 5 -p "please enter your name:" name #5s

-n选项，后接数值1，指示read命令只要接受到一个字符就退出。只要按下一个字符进行回答，read命令立即接受输入并将其传给变量。无需按回车键。

#!/bin/bash

read -n1 -p "Do you want to continue [Y/N]?" answer

case $answer in

Y | y)

echo "fine ,continue";;

N | n)

echo "ok,good bye";;

\*)

echo "error choice";;

esac

exit 0

# NSD SHELL DAY02

1. [**案例1：Shell中的数值运算**](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY02/CASE/01/index.html#case1)
2. [**案例2：条件测试操作**](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY02/CASE/01/index.html#case2)
3. [**案例3：使用if选择结构**](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY02/CASE/01/index.html#case3)

**Shell数值运算**

**条件测试**

**使用If选择结构**

## Shell中的数值运算: expr、$[ ]、let等整数运算工具(不支持小数)

**1）使用expr命令**

乘法操作应采用 \\* 转义，避免被作为Shell通配符；参与运算的整数值与运算操作符之间需要以空格分开，引用变量时必须加$符号。

[root@server0 ~]# expr 1 + 1

2

[root@server0 ~]# expr 1+1

1+1

[root@server0 ~]# expr 2 \\* 2 ---\*本为通配符,需取消

4

[root@server0 ~]# expr 2 '\*' 2

4

[root@server0 ~]# expr 2 % 2

0

[root@server0 ~]# a=100

[root@server0 ~]# expr $a % 10 ----求模(取余)

0

**2）使用$[]或$(())表达式**

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；计算结果替换表达式本身，可结合echo命令输出。

[root@server0 ~]# a=100

[root@server0 ~]# echo $[a+a]

200

[root@server0 ~]# echo $[a\*a]

10000

[root@server0 ~]# echo $[2\*2]

4

[root@server0 ~]# echo $[a%10]

0

[root@server0 ~]# echo $[RANDOM%4]

0

[root@server0 ~]# echo $[RANDOM%4]

1

**3）使用let命令**

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。let运算操作并不显示结果，变量不需要加$,可以结合echo命令来查看：

[root@server0 ~]# x=10

[root@server0 ~]# let y=x+5

[root@server0 ~]# echo $y

15

[root@server0 ~]# let x=x+1

[root@server0 ~]# echo $x

11

[root@svr5 ~]# let X++; echo $X # X++(X=X+1)

[root@svr5 ~]# let X--; echo $X # X- -(X=X-1)

[root@svr5 ~]# let X+=78 ; echo $X # X+=78(X=X+78)

[root@svr5 ~]# let X-=78 ; echo $X # X-=78(X=X-78)

[root@svr5 ~]# let X\*=78 ; echo $X # X\*=78(X=X\*78)

[root@svr5 ~]# let X/=78 ; echo $X # X/=78(X=X/78)

[root@svr5 ~]# let X%=78 ; echo $X # X%=78(X=X%78

[root@server0 ~]# x=1234

[root@server0 ~]# let x++

[root@server0 ~]# echo x

x

[root@server0 ~]# echo $x

1235

[root@server0 ~]# let x--

[root@server0 ~]# echo $x

1234

[root@server0 ~]# let x-=4

[root@server0 ~]# echo $x

1230

[root@server0 ~]# let x+=4

[root@server0 ~]# echo $x

1234

[root@server0 ~]# let x/=2

[root@server0 ~]# echo $x

617

[root@server0 ~]# let x\*=2

[root@server0 ~]# echo $x

1234

步骤二：小数运算工具bc支持小数运算,支持简单判断

1）**bc**交互式运算

先执行bc命令进入交互环境，然后再输入需要计算的表达式。

[root@server0 ~]# bc

bc 1.06.95

Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.

This is free software with ABSOLUTELY NO WARRANTY.

For details type `warranty'.

1.1+1

2.1

2.2/2

1

scale=3 --------定义小数点后位数

2.2/2

1.100

10/3

3.333

Quit

2）**bc**非交互式运算

将需要运算的表达式通过管道操作交给bc运算。注意，小数位的长度可采用scale=N限制，除此以外也受参与运算的数值的小数位影响。利用管道使用bc工具

[root@server0 ~]# echo "1+1" |bc

2

[root@server0 ~]# echo "10/3" |bc

3

[root@server0 ~]# echo "scale=3;10/3" |bc

3.333

[root@server0 ~]# echo "1.1111+1" |bc

2.1111

[root@server0 ~]# echo "scale=3;1.1111+1" |bc --给的数据超过了定义的小数点后位数

2.1111

3) bc用作判断

[root@server0 ~]# echo "1>2" |bc ---错误的输出值为 0

0

[root@server0 ~]# echo "1=2" |bc

(standard\_in) 1: syntax error

[root@server0 ~]# echo "1<2" |bc ---正确的输出值为 1

1

[root@server0 ~]# echo "1<=2" |bc

1

## 2 案例2：条件测试操作

* 字符串匹配
* 比较整数值的大小
* 识别文件/目录的状态
* 多个条件/操作的逻辑组合

**步骤一**：条件测试的基本用法

1）语法格式

使用“test 表达式”或者[ 表达式 ]都可以，表达式两边至少要留一个空格。

条件测试操作本身不显示出任何信息。测试的条件是否成立主要体现在命令执行后的返回状态（即 $?），所以可以在测试后查看变量$?的值来做出判断，或者结合&&、||等逻辑操作显示出结果（或作其他操作） 。

[root@server0 ~]# [ a == b ] ---用 [ 表达式 ]

[root@server0 ~]# echo $?

1

[root@server0 ~]# [ a == a ] --- 为空格

[root@server0 ~]# echo $?

0

[root@server0 ~]#

**步骤二**：字符串测试

1）== 比较两个字符串是否相同

[root@server0 ~]# echo $USER

root

[root@server0 ~]# [ $USER == "root" ] --判断当前用户是否为”root”

[root@server0 ~]# echo $? --输出执行结果.0为正确

0

[root@server0 ~]# [ $USER == "asdf" ]

[root@server0 ~]# echo $? --输出执行结果.1为错误

1. != 比较两个字符串是否不相同

[root@server0 ~]# [ $USER != "asdf" ]

[root@server0 ~]# echo $?

0

[root@server0 ~]# [ $USER != "root" ]

[root@server0 ~]# echo $?

1

3）一行执行多条命令的情况

# A && B //仅当A命令执行成功，才执行B命令

# A || B //仅当A命令执行失败，才执行B命令

# A ; B //执行A命令后执行B命令，两者没有逻辑关系

# A && B || C //思考？

4) -z 检查变量的值是否未设置（空值）

[root@svr5 ~]# var1="nb" ; var2=""

[root@svr5 ~]# [ -z "$var1" ] && echo "空值" || echo "非空值"

非空值

[root@svr5 ~]# [ -z $var2 ] && echo "空值" || echo "非空值"

空值                                      //变量var2已设置，但无任何值，视为空

[root@svr5 ~]# [ ! -z $var1 ]          //测试var1是否为非空

[root@server0 opt]# a=""

[root@server0 opt]# [ -z $a ] --判断是否为空

[root@server0 opt]# echo $? --是空的判断正确

0

[root@server0 opt]# [ ! -z $a ] --判断是否为非空

[root@server0 opt]# echo $? --是空的判断错误

1

还有一个-n可以测试变量是否不为空（相当于! -z）。

**步骤三：整数值比较**

参与比较的必须是整数（可以调用变量），比较非整数值时会出错：

-gt 大于 > greater than

-ge 大于等于 >= greater or equal

-eq 等于 = equal

-ne 不等于 != not eqal

-le 小于等于 <= less or equal

-lt 小于 < less tan

[root@server0 opt]# a=100

[root@server0 opt]# [ $a -gt 100 ] && echo 等于 || echo 错

错

[root@server0 opt]# [ $a -ge 100 ] && echo 等于 || echo 错

等于

[root@server0 opt]# [ $a -ne 100 ] && echo 等于 || echo 错

错

[root@server0 opt]# [ $a -eq 100 ] && echo 等于 || echo 错

等于

[root@server0 opt]# [ $a -le 100 ] && echo 等于 || echo 错

等于

[root@server0 opt]# [ $a -lt 100 ] && echo 等于 || echo 错

每两分钟检测,登录系统的账号,如果超过3个,给管理员发报警邮件

发邮件: echo ‘内容’ | mail -s ‘标题’ root

收邮件: mail -u root

计划任务:创建crontab -e

\*/2 \* \* \* \* /opt/test3.sh

----------脚本一-------------------------------------------------------------------------------------

#! /bin/bash

a=$(who | wc -l)

[ $a -gt 3 ] && echo 警告1 | mail -s '标题1' root || echo 正常

----------脚本二--------------------------------------------------------------------------------------

#! /bin/bash

[ $(who | wc -l) -gt 3 ] && echo 警告2 | mail -s '标题2' root || echo 正常

---------------------------------------------------------------------------------------------------------

[root@server0 opt]# vim test3.sh ----编辑脚本

[root@server0 ~]# chmod +x /opt/test3.sh ----赋予脚本执行权限

[root@server0 ~]# crontab -e -----创建计划任务

no crontab for root - using an empty one

crontab: installing new crontab

您在 /var/spool/mail/root 中有邮件

[root@server0 ~]# crontab -l -----查看创建的计划任务

\*/2 \* \* \* \* /opt/test3.sh

[root@server0 ~]# mail ----查看邮件,root登录直接查看root的邮件

Heirloom Mail version 12.5 7/5/10. Type ? for help.

"/var/spool/mail/root": 2 messages 2 new

>N 1 root Thu Jan 10 14:52 18/607 "标题1"

N 2 root Thu Jan 10 14:52 18/607 "标题2"

&

步骤四：识别文件/目录的状态

1. -e 判断对象是否存在（不管是目录还是文件）

2）-d 判断对象是否为目录（存在且是目录）

3）-f 判断对象是否为文件（存在且是文件）

4）-r 判断对象是否可读:此测试对root用户无效，无论文件是否设置r权限，root都可读：

5）-w 判断对象是否可写:此测试同样对root用户无效无论文件是否设置w权限，root都可写：

6）-x 判断对象是否具有可执行权限

[root@server0 ~]# [ -e /root ]

[root@server0 ~]# echo $?

0

[root@server0 ~]# [ -d /etc/fstab ] && echo "目录" || echo "不是目录"

不是目录

例:判断/opt中是否有文件ABC,没有则新建,有则显示”文件已存在”

[root@server0 ~]#[ -e /opt/ABC ] && echo "文件已经存在" || touch ABC

## 3 案例3：使用if选择结构

if单分支的语法组成一：

if 条件测试 ； ---条件测试满足则执行下面,不成功则结束

then

命令序列

fi

if单分支的语法组成二：

if 条件测试 ; then

命令序列

if双分支的语法组成：

if 条件测试

then

命令序列1

else

命令序列2

fi

if多分支的语法组成：

if 条件测试1 ;then

命令序列1

elif 条件测试2 ;then

命令序列2

else

命令序列n

fi

例:判断opt下是否有mnt目录,没有则创建

------脚本------------------------------------------------------------------------------------

#! /bin/bash

if [ ! -e /opt/mnt ];then

mkdir /opt/mnt

fi

步骤二：检测并判断指定的主机是否可ping通

1）分析任务需求

使用ping命令检测目标主机时，人工可直接判断反馈结果，而脚本却不方便。但是当ping测试成功时，执行状态$?的值为0；而ping测试失败时，$?的值不为0。因此在Shell脚本中可以利用这一点来判断ping目标主机的成败。

为了节省ping测试时间，可以只发送3个测试包（-c 3）、缩短发送测试包的间隔秒数（-i 0.2）、等待反馈的超时秒数（-W 1）。比如，检查可ping通的主机：

-c ping的次数 -i ping的间隔秒数 -W ping 反馈结果的等待秒数

----脚本一-------------------------------------------------------------------------

#! /bin/bash

ip=172.25.0.253

c=10

i=0.2

W=1

ping -c $c -i $i -W $W $ip &> /var/null

if [ $? -eq 0 ];then

echo "$ip 通了"

else

echo "$ip 不通"

fi

------脚本二-------------------------------------------------------------------------

#! /bin/bash

ping -c $1 -i $2 -W $3 $4 &> /var/null

if [ $? -eq 0 ];then

echo "$4 通了"

else

echo "$4 不通"

fi

[root@server0 opt]# bash test7.sh 10 0.2 1 172.25.0.254

172.25.0.254 通了

------脚本三--------------------------------------------------------------------

#! /bin/bash

c=10

i=0.2

W=1

ping -c $c -i $i -W $W $1 &> /var/null

if [ $? -eq 0 ];then

echo "$1 通了" >> /opt/pinglog.txt

else

echo "$1 不通" >>/opt/pinglog.txt

fi

[root@server0 opt]# bash test8.sh 172.25.0.254

随机数: RANDOM

[root@server0 opt]# echo $[RANDOM]

12029

[root@server0 opt]# echo $[RANDOM]

18489

[root@server0 opt]# echo $[RANDOM]

16031

取10以内的随机数(去多少以内的随机数,就取多少数的模)

[root@server0 opt]# echo $[RANDOM%10]

1

[root@server0 opt]# echo $[RANDOM%10]

9

例如:猜数字游戏

-----脚本一----------------------------------------------------------------------------

#! /bin/bash

a=$[RANDOM%10]

read -p "请输入一个数字:" b -----------第一次猜

if [ $b -eq $a ];then

echo "恭喜猜对了"

exit

elif [ $b -gt $a ];then

echo "猜大了"

else

echo "猜小了"

Fi

read -p "请输入一个数字:" b ----------第二次猜

if [ $b -eq $a ];then

echo "恭喜猜对了"

exit

elif [ $b -gt $a ];then

echo "猜大了"

else

echo "猜小了"

fi

read -p "请输入一个数字:" b ----------第三次猜

if [ $b -eq $a ];then

echo "恭喜猜对了"

exit

elif [ $b -gt $a ];then

echo "猜大了"

else

echo "猜小了"

fi

-------------脚本二----------------------------------------------------

#! /bin/bash

a=$[RANDOM%10]

i=1

while :

do

read -p "请输入一个数字:" b

if [ $b -eq $a ];then

echo "恭喜猜对了"

echo "$i"

exit

elif [ $b -gt $a ];then

echo "猜大了"

else

echo "猜小了"

fi

let i++

done

# NSD SHELL DAY03

[案例1：使用for循环结构](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html#case1)

[案例2：使用while循环结构](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html#case2)

[案例3：基于case分支编写脚本](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html#case3)

[案例4：使用Shell函数](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html#case4)

[案例5：中断及退出](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html#case5)

## 1 案例1：使用for循环结构

在Shell脚本应用中，常见的for循环采用遍历式、列表式的执行流程，通过指定变量从值列表中循环赋值，每次复制后执行固定的一组操作。

for循环的语法结构如下所示：

1. for 变量名 in 值列表 ---值列表可多个,例如 a b c
2. do
3. 命令序列
4. done
6. for 变量名 in {1..5}
7. do
8. 命令序列
9. done
10. for 变量名 in `seq 5` ----5个序号
11. do
12. 命令序列
13. done
14. for 变量名 in `ls /etc/\*.conf`
15. do
16. 命令序列
17. Done

-----------脚本-------------

#! /bin/bash

for i in `seq 5`

do

echo $i

done

~

[root@server0 opt]# bash test1.sh

1

2

3

4

5

-----------脚本-------------

#! /bin/bash

for i in `seq 5 10`

do

echo $i

done

~

[root@server0 opt]# bash test1.sh

1

2

3

4

5

-------------脚本-------------

#! /bin/bash

for i in `seq $1`

do

echo $i

done

[root@server0 opt]# bash test1.sh 4

1

2

3

4

--------------------------------

#! /bin/bash

for i in `seq $1`

do

echo nb ----并不一定取用 $1变量值,

done

~

[root@server0 opt]# bash test1.sh 4

nb

nb

nb

nb

-----------------------------------------------

-------测试一个网络地址范围是否通-脚本１----------------------------

#! /bin/bash

read -p "请输入主机范围" a

ip=172.25.0. #设置IP地址段

c=2 #设置测试包数

i=0.2 #设置发包间隔

W=1 #设置反馈超时秒数

for zj in `seq $a`

do

ping -c $c -i $i -W $W $ip$zj #&>/dev/null

[ $? -eq 0 ] && echo "$ip$zj 通了" || echo "$ip$zj 不通"

done

-------测试一个网络地址范围是否通-脚本２---------------------------------

#! /bin/bash

read -p "请输入主机范围" a

ip=172.25.0. #设置IP地址段

c=2 #设置测试包数

i=0.2 　 #设置发包间隔

W=1 #设置反馈超时秒数

for zj in `seq $a`

do

ping -c $c -i $i -W $W $ip$zj &>/dev/null

if [ $? -eq 0 ];then echo "$ip$zj 通了"

else

echo "$ip$zj　不通"

fi

done

~

----测试一个网络地址范围是否通-脚本3---统计成功不成功数量--------------------------

#! /bin/bash

a=0

b=0

for i in `seq $1`

do

ping -c 2 -i 0.2 -W 1 172.25.0.$i &>/dev/null

if [ $? -eq 0 ];then

echo "通了"

let a++

else

echo "不通"

let b++

fi

done

echo "$a 台通了"

echo "$b 台不通"

~

------按名单文件创建用户,设置默认密码----脚本1----------------------

#! /bin/bash

a=123456 #请在此定义默认密码

b=/opt/user #请在此定义账户文件

for i in $(cat $b)

do

useradd $i

echo $a | passwd --stdin $i #设置默认密码

echo "$i 用户创建成功,密码为默认"

done

------按名单文件创建用户,设置默认密码---脚本2-----------------------

#! /bin/bash

a=/opt/pass1 #请在此定义默认密码

b=/opt/user1 #请在此定义账户文件

n=`cat $b | wc -l` #统计账户文件中账户个数

for i in `seq $n` #按照账户统计数循环次数

do

useradd `head -$i $b | tail -1`

echo `head -$i $a | tail -1` | passwd --stdin `head -$i $b | tail -1`

done

## 2 案例2：使用while循环结构

while循环属于条件式的执行流程，会反复判断指定的测试条件，只要条件成立即执行固定的一组操作，直到条件变化为不成立为止。所以while循环的条件一般通过变量来进行控制，在循环体内对变量值做相应改变，以便在适当的时候退出，避免陷入死循环。

------猜数字游戏 并记录回显猜的次数,无输入将退出-----------------------

#! /bin/bash

a=$[RANDOM%100+1]

b=1

while : ----死循环,满足继续循环,知道不满足条件

do

read -p "请输入一个数组(1-100):" n

if [ -z $n ];then

echo "无输入,2s后退出"

sleep 2

exit

elif [ $n -eq $a ];then

echo "猜对了,游戏结束"

echo "总共猜了 $b 次"

exit

elif [ $n -gt $a ];then

echo "大了"

else

echo "小了"

fi

let b++

done

**3 案例3：基于case分支编写脚本**

case分支属于匹配执行的方式，它针对指定的变量预先设置一个可能的取值，判断该变量的实际取值是否与预设的某一个值相匹配，如果匹配上了，就执行相应的一组操作，如果没有任何值能够匹配，就执行预先设置的默认操作。

case分支的语法结构如下所示：

1. case 变量 in
2. 模式1)
3. 命令序列1 ;;
4. 模式2)
5. 命令序列2 ;;
6. .. ..
7. \*)
8. 默认命令序列
9. Esac

------实现简单的功能,-脚本1---------------------

#! /bin/bash

case $1 in

t)

echo "已创建 $2 文件"

touch $2 ;;

vim)

echo "已打开 $2 文件"

vim $2;;

m)

echo "已创建 $2 目录"

mkdir $2;;

\*)

echo "请在命令后面输入t,v,m"

esac

[root@server0 opt]# bash test6.sh t 123qwe

已创建 123qwe 文件

-------实现简单的功能,-脚本2--------------

#! /bin/bash

case $1 in

t|-t|T) #定义t,-t,T效果一样

echo "已创建 $2 文件"

touch $2 ;;

vim|VIM|-vim) #定义 vim,VIM,-vim 效果一样

echo "已打开 $2 文件"

vim $2;;

m)

echo "已创建 $2 目录"

mkdir $2;;

\*)

echo "请在命令后面输入t,v,m"

esac

~

## 4 案例4：使用Shell函数

在Shell脚本中，将一些需重复使用的操作，定义为公共的语句块，即可称为函数。通过使用函数，可以使脚本代码更加简洁，增强易读性，提高Shell脚本的执行效率

**1）函数的定义方法**

格式1：

1. **function** 函数名 **{**
2. 命令序列
3. **..** **..**
4. **}**

格式2：

1. 函数名**()** **{**
2. 命令序列
3. **..** **..**
4. **}**

[root@server0 opt]# abc(){ ---开始定义函数abc

> mkdir $1

> cd $1

> echo 这是个函数

> }

[root@server0 opt]# abc 123 ---使用abc函数,创建123目录,并进入

这是个函数

[root@server0 123]# pwd ---查看当前目录,已进入新建的目录123

/opt/123

[root@server0 123]# type abc ----查看abc详情

abc 是函数

abc ()

{

mkdir $1;

cd $1;

echo 这是个函数

}

[root@server0 123]#

**颜色**

格式: echo -e "\033[字背景颜色;字体颜色m字符串\033[0m"

例如: echo -e "\033[41;36m something here \033[0m"

\033是准备加色 字背景颜色范围:40--49  字颜色:30---39  0m是还原

字颜色:30---39 黑 红 绿 黄 蓝 紫 深绿 白

字背景颜色范围:40--49 黑 深红 绿 黄 蓝 紫 深绿 白色

-----创建函数,定义颜色---------------------------------------------------------------

#! /bin/bash

cecho(){ #先定义函数cecho

echo -e "\033[$1m$2\033[0m" #函数内容是:定义文字颜色,

}

cecho 31 ABCD #调用cecho函数31是$1 ABCD是$2

cecho 32 ABCD

cecho 41 ABCD #背景变色

--------------------------------------------------------------------------------------------

--------自己调用自己的函数-差错炸弹,---------------------------------------------

#! /bin/bash

.(){

.|.&

}

.

如果将点换成常用的命令,则一运行将造成死机

## 5 案例5：中断及退出

通过break、continue、exit在Shell脚本中实现中断与退出的功能。

break可以结束整个循环；continue结束本次循环，进入下一次循环；exit结束整个脚本，案例如下：

---------------------------------------------------

#! /bin/bash

for i in {1..5}

do

echo $i

break

echo 123 #因为break结束整个循环,echo 123将不执行

done

echo abc

[root@server0 opt]# bash test8.sh

1

abc

------------------------------------------------

#! /bin/bash

for i in {1..5}

do

echo $i

[ $i -eq 3 ] && continue #continue只是结束本次循环,本次下面不执行

echo 123

done

echo abc

[root@server0 opt]# bash test8.sh

123

2

123

3 #continue只是结束本次循环,少执行一次echo 123

4

123

5

123

abc

例子:让用户输入数字,每输入一次,都与之前的想家,如果输入的是0,则结束并显示之前的总和

------------------

#! /bin/bash

while :

do

read -p "请输入数字:" n

let a+=$n

[ $n -eq 0 ] && break

done

echo $a

-----------------

# NSD SHELL DAY04

[案例1：字符串截取及切割](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY04/CASE/01/index.html#case1)

[案例2：字符串初值的处理](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY04/CASE/01/index.html#case2)

[案例3：expect预期交互](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY04/CASE/01/index.html#case3)

[案例4：使用正则表达式](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY04/CASE/01/index.html#case4)

使用Shell完成各种Linux运维任务时，一旦涉及到判断、条件测试等相关操作时，往往需要对相关的命令输出进行过滤，提取出符合要求的字符串。

本案例要求熟悉字符串的常见处理操作，

**子串截取的三种用法：**

1. ${变量名:起始位置:长度}
2. expr substr "$变量名" 起始位置 长度
3. echo $ 变量名 | cut -b 起始位置-结束位置

**子串替换的两种用法：**

* 只替换第一个匹配结果：${变量名/old/new}
* 替换全部匹配结果：${变量名//old/new}

**字符串掐头去尾：**

* 从左向右，最短匹配删除：${变量名#\*关键词}
* 从左向右，最长匹配删除：${变量名##\*关键词}
* 从右向左，最短匹配删除：${变量名%关键词\*}
* 从右向左，最长匹配删除：${变量名%%关键词\*}

**定义初始值:**

echo ${a:-} 定义初值

# 1 案例1：字符串截取及切割

## 步骤一:字符截取

### 方法一，使用 ${}表达式

格式：${变量名:起始位置:长度}

***使用${}方式截取字符串时，起始位置是从0开始的。***

[root@server0 opt]# a=13973169942

[root@server0 opt]# echo ${a::1} #截取第一个,起始位置省略,从第一个开始

1

[root@server0 opt]# echo ${a:0:1} #截取第一个,起始位置为零,从第一个开始

1

[root@server0 opt]# echo ${a:0:} #长度不写,无法截取

[root@server0 opt]# echo ${a:0:0} #长度为零,无法截取

[root@server0 opt]# echo ${a:0} #从第一个开始,后全部截取

13973169942

[root@server0 opt]# echo ${a:3} #从第四个开始,后面的全部截取

73169942

### 方法二，使用 expr substr

格式：expr substr "$变量名" 起始位置 长度

[root@server0 opt]# a=13973169942

[root@server0 opt]# expr substr "$a" 1 3

139

[root@server0 opt]# expr substr "$a" 7 4

6994

[root@server0 opt]# expr substr "$a" 8 4

9942

[root@server0 opt]# expr substr "$a" 8 #无法省略长度

expr: 语法错误

### 方式三，使用cut分割工具

格式：echo $变量名 | cut -b 起始位置-结束位置

选项 -b 表示按字节截取字符，其中起始位置、结束位置都可以省略。当省略起始位置时，视为从第1个字符开始（编号也是从1开始，与expr类似），当省略结束位置时，视为截取到最后。

[root@server0 opt]# a=123456789

[root@server0 opt]# echo $a |cut -b 1 2 #起始位置与结束位置之间少-

cut: 2: 没有那个文件或目录

[root@server0 opt]# echo $a |cut -b 1-3 #从第一个到第三个

123

[root@server0 opt]# echo $a |cut -b 2-3 #从第二个到第三个

23

[root@server0 opt]# echo $a |cut -b 1 #只截取第一个

1

[root@server0 opt]# echo $a |cut -b 2 #只截取第二个

2

[root@server0 opt]# echo $a |cut -b 2- #从第二个开始到最后,省略了结束位置

23456789

[root@server0 opt]# echo $a |cut -b -3 #从第一个开始到第三个,省略了起始位置

123

[root@server0 opt]# echo $a |cut -b 1,3 #截取第一个和第三个

13

[root@server0 opt]# echo $a |cut -b 1,5 #截取第一个和第五个

15

[root@server0 opt]# echo $a |cut -b 1-3,9 #截取第一个到第三个和第九个

1239

---随机选取8位-------------------------------

#! /bin/bash

b=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

#第一种方法随机选取8位:$[变量名:起始位置:长度]for i in `seq 8`

do

a=$[RANDOM%62]

x=${b:a:1}

p1=$p1$x

done

echo $p1

#第二种方法随机选取8位:expr subster

for i in `seq 8`

do

c=$[RANDOM%62+1]

y=$(expr substr $b "$c" 1)

p2=$p2$y

done

echo $p2

#echo $b | cut -b $c

---------------------------------

#! /bin/bash

pass(){

p=

b=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

#第一种方法随机选取8位:$[变量名:起始位置:长度]

for i in `seq 8`

do

a=$[RANDOM%62]

c=${b:a:1}

p=$p$c

done

echo $p

}

d=$(cat /opt/4day/user)

for d in $d

do

useradd $d

pass

echo $p | passwd --stdin $d

echo "$d 的密码是:$p" >> /opt/4day/userpasswd

done

## 步骤二：字符串的替换

### 1）只替换第1个子串 格式：${变量名/old/new}

### 2）替换全部子串 格式：${变量名//old/new}

[root@server0 4day]# a=aabbccddeeffgg

[root@server0 4day]# echo ${a/g/h} #将第一个g替换成h

aabbccddeeffhg

[root@server0 4day]# a=aabbccddeeffgg

[root@server0 4day]# echo ${a//g/h} #将所有的g替换成h

aabbccddeeffhh

## 步骤三：字符串的匹配删除

[root@server0 4day]# a=`head -1 /etc/passwd`

[root@server0 4day]# echo $a

root:x:0:0:root:/root:/bin/bash

1）从左向右，最短匹配删除 格式：${变量名#\*关键词}

删除从左侧第1个字符到最近的关键词“：”的部分，\* 作通配符理解：

[root@server0 4day]# echo ${a#\*:} #包括:都被删除

x:0:0:root:/root:/bin/bash

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

2）从左向右，最长匹配删除 格式：${变量名##\*关键词}

删除从左侧第1个字符到最远的关键词“:”的部分：

[root@server0 4day]# echo ${a##\*:}

/bin/bash

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

3）从右向左，最短匹配删除 格式：${变量名%关键词\*}

删除从右侧最后1个字符到往左最近的关键词“:”的部分，\* 做通配符理解：

[root@server0 4day]# echo ${a%:\*}

root:x:0:0:root:/root

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

4）从右向左，最长匹配删除 格式：${变量名%%关键词\*}

删除从右侧最后1个字符到往左最远的关键词“:”的部分：

[root@server0 4day]# echo ${a%%:\*}

root

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

----------批量修改文件名--------------------

#! /bin/bash

a=`ls /opt/4day/\*.txt`

for i in $a

do

mv $i ${i%.\*}.doc

done

[root@server0 4day]# touch a{1..10}.txt

[root@server0 4day]# ls

1.sh 3.sh a1.txt a3.txt a5.txt a7.txt a9.txt userpasswd

2.sh a10.txt a2.txt a4.txt a6.txt a8.txt user

[root@server0 4day]# vim 3.doc

[root@server0 4day]# vim 3.sh

[root@server0 4day]# bash 3.sh

[root@server0 4day]# ls

1.sh 3.sh a1.doc a3.doc a5.doc a7.doc a9.doc userpasswd

2.sh a10.doc a2.doc a4.doc a6.doc a8.doc user

-----------------------------------------------------------------

# 2 案例2：字符串初值的处理

通过${var:-word}判断变量是否存在，决定变量的初始值。

1）只取值，${var:-word}

若变量var已存在且非Null，则返回 $var 的值；否则返回字串“word”，原变量var的值不受影响。

[root@server0 4day]# a=123456

[root@server0 4day]# ${a:-123456}

bash: 123456: 未找到命令...

[root@server0 4day]# a=123 #定义了a=123

[root@server0 4day]# echo ${a:-123456} #定义a的初值为:123456并输出

123 #输出a为123

[root@server0 4day]# a= #定义a= 空值

[root@server0 4day]# echo ${a:-123456} #定义a的初值为:123456并输出

123456 #输出a为初值123456

-----------------

#! /bin/bash

read -p "请输入用户名:" n

useradd $n

read -p "请输入$n 的密码:" p

echo ${p:-123456} |passwd --stdin $n #如没有输入密码,将会把123456设置为用户密码

----------------------------------------------------

echo ${a:} 截取

echo ${a/} 替换

1）只替换第1个子串 格式：${变量名/old/new}

2）替换全部子串 格式：${变量名//old/new}

echo ${a#} 掐头

echo ${a%} 去尾

echo ${a:-} 定义初值

例:用来从键盘读入一个正整数x，求从1到x的和；当用户未输入值（直接回车）时，为了避免执行出错，应为x赋初值1 。

# 3 案例3：expect预期交互

expect可以为交互式过程（比如FTP、SSH等登录过程）自动输送预先准备的文本或指令，而无需人工干预。触发的依据是预期会出现的特征提示文本。

步骤一：准备expect及SSH测试环境

1）安装expect工具

-------脚本使用SSH登录远程主机并创建文件再退出-----------------

[root@svr5 ~]# vim expect\_ssh.sh

#! /bin/bash

expect << EOF #固定格式

spawn ssh [root@172.25.0.10](mailto:root@172.25.0.10) #创建交互进程

expect "password:" { send "redhat\r" } #出现”passwd:”发送密码,并回车

expect "#" { send "touch /tmp111.txt\r" } #发现”#”发送创建命令并回车

expect "#" { send "exit\r" } #发现”#”发送”exit”并回车”\r:” 默认不执行

EOF

注意事项：

expect脚本的最后一行默认不执行

如果不希望ssh时出现yes/no的提示，远程时使用如下选项:

# ssh -o StrictHostKeyChecking=no server0

**数组:**

[root@server0 4day]# w=(11 22)

[root@server0 4day]# echo ${w[0]} # 显示第一个元素 0为下标

11

[root@server0 4day]# echo ${w[1]}

22

[root@server0 4day]# echo ${w[@]} #显示所有元素

11 22

[root@server0 4day]# w[2]=33 #增加第三个元素

[root@server0 4day]# w[3]=33 #增加第四个元素

[root@server0 4day]# echo ${w[@]}

11 22 33 33

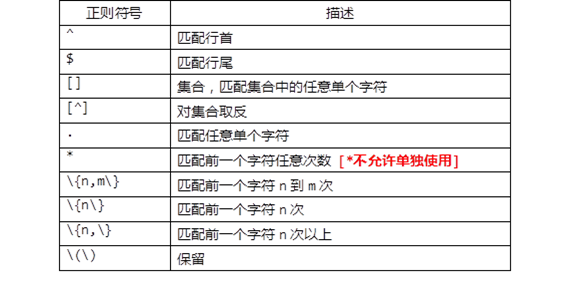
[root@server0 4day]# echo ${w[@]:1:2}#截取数组元素1为起始下标,2为元素个数

22 33

# 4 案例4：使用正则表达式

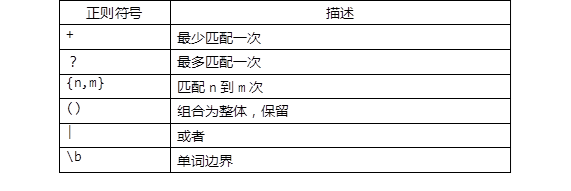
**正则表达式:**使用”一串符号”来描述有共同属性的数据

**基本正则列表:**



^$ 组合,表示空行

**扩展正则列表:**



grep 只能使用基本正则,如要使用扩展正则需要加 -E ,egrep -E 0

egrep 即可使用基本正则,又可以使用扩展正则

[root@server0 4day]# grep -E "^root|^daemon" /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[root@server0 4day]# egrep "^root|^daemon" /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:2:2:daemon:/sbin:/sbin/nologin

**grep -q** #使用 -q 选项表示quiet(静默)的效果与使用 **&> /dev/null的效果类似。**

[root@ver0 4day]# grep -q "^127.0.0.1" /etc/hosts && echo "YES"||echo "NO"

YES

[root@ver0 4day]# grep "^127.0.0.1" /etc/hosts && echo "YES" || echo "NO"

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

YES

egrep -m10 #-m10仅在文件的前10行中过滤，后面的行不再过滤。

[root@server0 4day]# egrep -m10 "/sbin/nologin$" /etc/passwd

**egrep -c** #使用 -c 选项可输出匹配行数，这与通过管道再 wc -l的效果是相同的

[root@server0 4day]# egrep -c "/bin/bash$" /etc/passwd

17

[root@server0 4day]# egrep "/bin/bash$" /etc/passwd |wc -l

17

3）基本元字符 ^、$ —— 匹配行首、行尾

1. 基本元字符 . —— 匹配任意单个字符

\* —— 匹配任意单个字符 匹配前一个字符的任意次数,不允许单独使用

[root@server0 4day]# egrep ".\*" xyz

aaaaaaaaaethe

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

# “.\*” 将此行空行也筛选出来了

[root@server0 4day]# egrep "." xyz

aaaaaaaaaethe #”.” 输出xyz文件内至少包括一个字符的行(\n换行符除外)

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

输出/etc/rc.local文件内的空行（用 –v 选项将条件取反）：

[root@server0 4day]# egrep -v "." xyz #与”^$”效果类似

[root@server0 4day]# egrep "^$" xyz #与 -v “.” 类似

1. 基本元字符 +、?、\* —— 目标出现的次数

[root@server0 4day]# egrep "a?" xyz # ”a?”最多匹配一次,包括零次

aaaaaaaaaethe

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

#此行空行,包括零次

[root@server0 4day]# egrep "a\*" xyz #匹配前面一个字符任意次数,包括零次

aaaaaaaaaethe

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

#此行空行,包括零次

[root@server0 4day]# egrep "a+" xyz #匹配前面一个字符最少一次,可超过

aaaaaaaaaethe

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

输出/etc/passwd文件内“t”开头且以“a”结尾的用户记录，即中间可以是任意字符：

[root@server0 4day]# egrep "^t.\*a$" xyz

thebbbbbbbaaaaaaaa

6）元字符 {} —— 限定出现的次数范围

[root@server0 4day]# cat xya #先查看实验内容

dedef abab ghighi

cdcd ababab

Hello abababab World >xya

aabbccaabbcc

aaabbbcccdddeee

[root@server0 4day]# egrep "a{1}" xya #a连续出现1次

dedef abab ghighi

cdcd ababab

Hello abababab World >xya

aabbccaabbcc

aaabbbcccdddeee

[root@server0 4day]# egrep "a{2}" xya #a连续出现2次

aabbccaabbcc

aaabbbcccdddeee

[root@server0 4day]# egrep "a{3}" xya #a连续出现3次

aaabbbcccdddeee

[root@server0 4day]#

[root@server0 4day]# egrep "a{2,4}" xya #a连续出现2-4次

aabbccaabbcc

aaabbbcccdddeee

[root@server0 4day]#

7）元字符 [] —— 匹配范围内的单个字符

[root@server0 4day]# egrep "[Hc]" xya #只要含有[]内的行都会显示

cdcd ababab

Hello abababab World >xya

aabbccaabbcc

aaabbbcccdddeee

[root@server0 4day]# egrep "[A-Z]" xya #使用[A-Z]匹配连续范围,全大写

Hello abababab World >xya

**[^] 对集合内取反**, "[^a-z]"除了小写,其余全匹配,包括空格

[root@server0 4day]# egrep "[^a-z]" xya #

dedef abab ghighi #有空隔

cdcd ababab #有空隔

Hello abababab World >xya

**8）单词边界匹配“\b**”

[root@server0 4day]# egrep "\bab" xya #匹配含有ab开头的单词的行

dedef abab ghighi

cdcd ababab

Hello abababab World >xya

[root@server0 4day]# egrep "f\b" xya #匹配含有f结尾的单词的行

dedef abab ghighi

9）多个条件的组合

[root@server0 4day]# egrep -i "d|h" xya #查找含有d与h的行,且不区分大小写

dedef abab ghighi

cdcd ababab

Hello abababab World >xya

aaabbbcccdddeee

# NSD SHELL DAY05

案例1：sed基本用法

案例2：使用sed修改系统配置

案例3：sed多行文本处理

案例4：sed综合脚本应用

sed(流式编辑器)工具概述:

* 非交互式,基于模式匹配过滤及修改文本
* 逐行处理,并将结果输出到屏幕
* 可实现对文本的输出,删除,替换,复制,剪切,导入,导出等各种操作
* 与vim文本处理工具的区别,vim交互式,sed非交互式

# sed文本处理工具的用法：

用法1：前置命令 | sed [选项] '条件指令' #管道

用法2：sed [选项] '条件指令' 文件.. ..

* 条件指令含”[地址符]处理动作”
* 地址符含”[地址1,[地址2]]”
* 条件可以是行号或者/正则/
* 没有条件时，默认为所有条件
* 指令可以是增、删、改、查等指,必须有
* 默认sed会将所有输出的内容都打印出来，可以使用-n屏蔽默认输出
* 选项中可以使用-r选项，让sed支持扩展正则

## sed命令的常用选项如下：

-n（屏蔽默认输出，默认sed会输出读取文档的全部内容）

[root@client50 ~]# sed -n '1p' 1.txt

> bb

-r（让sed支持扩展正则）,默认支持基本正则

-i（sed直接修改源文件，默认sed只是通过内存临时修改文件，源文件无影响）

[root@client50 ~]#sed -i '1d' 1.txt

## sed 命令的 指令 选项

**p 打印**

]# sed -n '1p' 1.txt

]# sed -n '2,4p' /opt/abcdef #显示第2到4行,加了-n

]# sed -n '2p;4p' /opt/abcdef #显示第2和第4行

]# sed -n '2,+2p' /opt/abcdef #从第2行开始,再打印后面2行,

]# sed -n '1~2p' /opt/abcdef #从第一行开始,每跨2行打印,奇数

]# sed -n '$=' /opt/abcdef #显示最后一行的行数

]# sed -n '$p' /opt/4day/xyz #显示最后一行的内容

]# sed -n '=' /opt/4day/xyz #显示行号

**d 删除**

# sed '1d' 1.txt #删除第一行

**s 替换 （s/旧内容/新内容/选项）**

]# sed 's/the/xyz/' /opt/4day/xyz #替换每行第一个the为xyz

]# sed 's/the/xyz/2' /opt/4day/xyz #替换每行第二个the为xyz

]# sed 's/the/xyz/g' /opt/4day/xyz #替换每行所有的the为xyz

]# sed '1s/the/xyz/' /opt/4day/xyz #替换第1行的第一个the为xyz

]# sed '2s/a/x/' /opt/4day/xyz #第2行第一个a替换成x

]# sed '2,5s/a/x/' /opt/4day/xyz #第二行到第五行第一个a替换成x

]# sed 's/^the/xyz/' /opt/4day/xyz #将以the开头的替换为xyz

]# sed 's/^the/xyz/' /opt/4day/xyz #将以the开头的替换为xyz

i： 在指定的行之前插入文本

a：在指定的行之后追加文本

c：替换指定的行

sed ‘2a xxx’ #在第二行下面一行插入xxx

sed ‘2i xxx’ #在第二行上面一行插入xxx

sed ‘3c xxx’ #替换第三行

sed ‘/root/c xxx’ 替换包含root的行为xxx

### 命令指令: P 打印出查询信息

[root@server0 ~]# cat /opt/abcdef

aa

bb

cc

dd

ee

ff

[root@server0 ~]# sed '2,4p' /opt/abcdef #显示第2行到第4行,未加-n

aa

bb

bb

cc

cc

dd

dd

ee

ff

[root@server0 ~]# sed -n '2,4p' /opt/abcdef #显示第2到4行,加了-n

bb

cc

dd

[root@server0 ~]# sed -n '2p;4p' /opt/abcdef #显示第2和第4行

bb

dd

[root@server0 ~]# sed -n '2,+2p' /opt/abcdef #从第2行开始,再打印后面2行,

bb

cc

dd

[root@server0 ~]# sed -n '1~2p' /opt/abcdef #从第一行开始,每跨2行打印,奇数

aa

cc

ee

[root@server0 ~]# sed -n '$=' /opt/abcdef #显示最后一行的行数

7

[root@server0 ~]# sed -n '$p' /opt/4day/xyz #显示最后一行的内容

[root@server0 ~]# sed -n '=' /opt/4day/xyz #显示行号

### 命令指令: d 删除指定信息

[root@server0 ~]# sed '1d' /opt/abcdef #删除第一行

bb

cc

dd

ee

ff

[root@server0 ~]# sed -n '1d' /opt/abcdef #带了-n 实际操作成功但是看不到效果

[root@server0 ~]# sed '1d;3d' /opt/abcdef 删除第1行和第3行

bb

dd

ee

ff

[root@server0 ~]# sed '1d;+3d' /opt/abcdef #错误操作

sed：-e 表达式 #1，字符 5：无法将 +N 或 ~N 作为第一个地址

[root@server0 ~]# sed '1,+3d' /opt/abcdef #删除第1行,以及第1行后面的三行

ee

ff

[root@server0 ~]# sed '1~3d' /opt/abcdef #删除第一行,每隔三行删除一个

bb

cc

ee

Ff

[root@server0 ~]# sed '1,2d' nssw.txt #删除1到2行

dsfa

dfAsd

sdf

### 命令指令:s替换基本功能（s/旧内容/新内容/选项）：

[root@server0 ~]# sed 's/the/xyz/' /opt/4day/xyz

#替换每一行的第一个the为xyz

aaaaaaaaaexyz

xyzbbbbbbbaaaaaaaa

aaaaaaa xyz bbbbbb

dddddddddcccccccccaaaaaaaaa

a

[root@server0 ~]# sed '1s/the/xyz/' /opt/4day/xyz

#替换第1行的第一个the为xyz

aaaaaaaaaexyz

thebbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

[root@server0 ~]# sed -n '1s/the/xyz/p' /opt/4day/xyz #替换第1行的the为xyz

aaaaaaaaaexyz #替换第1行的the为xyz,并且只显示修改了的行

[root@server0 ~]# sed 's/a/x/' /opt/4day/xyz #每行第一个a替换成x

xaaaaaaaaethe

thebbbbbbbxaaaaaaa

xaaaaaa the bbbbbb

dddddddddcccccccccxaaaaaaaa

x

[root@server0 ~]# sed '2s/a/x/' /opt/4day/xyz #第二行第一个a替换成x

aaaaaaaaaethe

thebbbbbbbxaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

[root@seer0 ~]# sed '2,5s/a/x/' /opt/4day/xyz #第二行到第五行第一个a替换成x

aaaaaaaaaethe

thebbbbbbbxaaaaaaa

xaaaaaa the bbbbbb

dddddddddcccccccccxaaaaaaaa

x

[root@server0 ~]# sed 's/a/x/2' /opt/4day/xyz #所有行的第二个a替换为x

axaaaaaaaethe

thebbbbbbbaxaaaaaa

axaaaaa the bbbbbb

dddddddddcccccccccaxaaaaaaa

a

s/peer0.org1.example.com/peer1.org1.example.com/

[root@server0 ~]# sed '3s/a/x/2' /opt/4day/xyz #第三行的第二个a替换为x

aaaaaaaaaethe

thebbbbbbbaaaaaaaa

axaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

[root@server0 ~]# sed '2s/a/x/g' /opt/4day/xyz #第二行的所有a替换为x

aaaaaaaaaethe

thebbbbbbbxxxxxxxx

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

[root@server0 ~]# sed 's/^the/xyz/' /opt/4day/xyz #将以the开头的替换为xyz

aaaaaaaaaethe

xyzbbbbbbbaaaaaaaa

aaaaaaa the bbbbbb

dddddddddcccccccccaaaaaaaaa

a

## sed 使用基本正则 ‘ ’内加/

sed -n ‘/the/p’ xyz #查找含”the”的行

sed -n ‘/^the/p’ xyz #查找含”the”开头的行

sed -n '1p' 1.txt

sed -n '2,4p' /opt/abcdef #显示第2到4行,加了-n

sed -n '2p;4p' /opt/abcdef #显示第2和第4行

sed -n '2,+2p' /opt/abcdef #从第2行开始,再打印后面2行,

sed -n '1~2p' /opt/abcdef #从第一行开始,每跨2行打印,奇数

sed -n '$=' /opt/abcdef #显示最后一行的行数

sed -n '$p' /opt/4day/xyz #显示最后一行的内容

sed -n '=' /opt/4day/xyz #显示行号

sed ‘/the/d’ xyz #删除含”the”的行

sed ‘/^the/d’ xyz #删除含”the”开头的行

sed ‘/^the/!d’ xyz #含”the”开头的行不删除,别的全部删除

sed ‘/^$/d;/^#/d’ xyz #删除空行与#开头的(注释)

Sed ‘s/a/x’ xyz #替换所有行中的第一个a

sed ‘2s/a/x’ xyz #替换第2行中的第一个a

sed -n '1s/the/xyz/p' xyz #替换第1行的the为xyz,并且只显示修改了的行

sed '2,5s/a/x/' /opt/4day/xyz #第二行到第五行第一个a替换成x

sed 's/a/x/2' /opt/4day/xyz #所有行的第二个a替换为x

sed '2s/a/x/g' /opt/4day/xyz #第二行的所有a替换为x

sed 's/[0-9]//g' nssw.txt #删除所有的数字

sed -r 's/^ +//g' nssw.txt #删除所有行行首空格^ +之间有空格

sed 's/^the/xyz/' /opt/4day/xyz #将所有行,以the开头的替换为xyz

sed '4,7s/^/#/' a.txt #将第4~7行注释掉（行首加#号）文本中,每行的 前面都有默认有个^

sed 's/^#an/an/' a.txt #解除以#an开头的行的注释（去除行首的#号）

sed 's#/bin/bash#/sbin/sh#' a.txt #将/bin/bash替换为/sbin/sh

sed 's/([A-Z])/(\1)/g' nssw.txt #将所有大写添加括号

sed 's/.$//;s/.//2' nssw.txt #每行的第二个和最后一个删除

sed -i 's/\(^[[:space:]]\)/#\1/g' /tmp/rc.local #将空格开头的前面添加一个#

# THIS FILE I #修改前

# # THIS FILE I #修改后

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt #第一个最后一个对调

## sed支持扩展正则 加 -r 选项

Sed -rn ‘/the|bbb/p’ xyz #查找含有”the”或者”bbb”的行 | 为扩展正则

[root@server0 ~]# cat nssw.txt

Hello the world

ni hao ma beijing

[root@server0 ~]# sed 's/.$//;s/.//2' nssw.txt #每行的最后一个和第二个删除

Hllo the worl

n hao ma beijing

**sed正则表达式匹配,各种括号的转义和不转义**

[ ] 需要匹配的时候，需要转义(这个是叛徒)

echo "[ ]"|sed 's/\[.\*\]/aaa/g'

( ) 需要匹配的时候，不要转义

echo "( )"|sed 's/( )/c/g'

{ } 需要匹配的时候，不要转义

echo "{ }"|sed 's/{ }/c/g'

当需要匹配数字，字母等使用中括号时候[]不要转义

但使用{}大括号作为特殊字符时候，{}需要转义。

echo "333"|sed 's/[0-9]\{3\}/ccc/g'

当需要适配符，需要使用\1来替换正则表达式的对应参数时：()需要转义

不能写(regrexxxx)，要写\(regrexxxx\)

echo "{1234567}"|sed 's/{\([0-9]\*\)}/\1/g'

与此对应的还有 +和\*

在做为特别字符时候+必须转义为

\+才有效，

而\*则不需要。

举个例子：

echo "ccc"| sed 's/c\*/aaa/g'#正确

echo "ccc"| sed 's/c\\*/aaa/g'#错误

echo "ccc"| sed 's/c+/aaa/g'#错误

echo "ccc"| sed 's/c\+/aaa/g'#正确

## sed复制粘贴

2）将文件中每行的第一个、倒数第1个字符互换

每行文本拆分为“第1个字符”、“中间的所有字符”、“倒数第1个字符”三个部分，然后通过替换操作重排顺序为“3-2-1”：

[root@svr5 ~]# sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt

^(.) 复制开头为\1 (.\*)复制身体为\2 (.)$复制结尾为\3 重新组合:\3\2\1

[root@server0 ~]# sed -r 's/([A-Z])/(\1)/' nssw.txt #将所有的大写添加()

(H)ello the world

ni hao a (B)eijing

dsfa

df(A)sd

## sed (i a c)多行文本处理

sed [选项] '条件指令' 文件..

sed工具的多行文本处理操作指令：

i： 在指定的行之前插入文本

a：在指定的行之后追加文本

c：替换指定的行

[root@server0 ~]# sed 'i x' nssw.txt #不加行号,默认在所有行之前插入

x

Hello the wor44ld

x

ni hao a Bei333jing

x

dssdf12fa

[root@server0 ~]# sed '/^H/i x' nssw.txt #在H开头的行之前插入

x

Hello the wor44ld

ni hao a Bei333jing

[root@server0 ~]# sed '/^H/a x' nssw.txt #在H开头的行之后插入

Hello the wor44ld

x

ni hao a Bei333jing

[root@server0 ~]# sed '2c x' nssw.txt #将整个第2行替换为x

Hello the wor44ld

x

dssdf12fa

## sed读取(r)与保存文件(w)

sed [选项] '条件指令' 文件..

指令:

r 读取文件

w 保存到文件

[root@server0 ~]# sed '1r b.txt' a.txt #在a.txt文件的第1行插入b.txt里面的内容

aaa

111

222

333

bbb

ccc

[root@server0 ~]# cat a.txt #原文件内容未变化

aaa

bbb

ccc

[root@server0 ~]# cat b.txt #原文件内容未变化

111

222

333

[root@server0 ~]# sed '2w c.txt' a.txt #复制a.txt里面的第二行写入到c.txt

aaa

bbb

ccc

[root@server0 ~]# cat c.txt #查看c.txt(事先未创建)

bbb

[root@server0 ~]# sed '2w c.txt' b.txt #再复制b.txt里面的第二行写入到c.txt

111

222

333

[root@server0 ~]# cat c.txt #再查看c.txt 之前的内容被覆盖

222

## sed复制剪切

基本动作:

H:模式空间--[追加]-->保持空间 追加复制

h:模式空间--[覆盖]-->保持空间 覆盖复制

G:保持空间--[追加]-->模式空间 追加粘贴

g:保持空间--[覆盖]-->模式空间 覆盖粘贴

[root@server0 ~]# sed '1H;$G' a.txt #追加复制第一行到剪切板,在末尾追加粘贴

aaa

bbb

ccc

#追加粘贴到这末尾 空行:剪切板中默认第一行是空行

aaa #复制粘贴到这里

[root@server0 ~]# sed '1h;$G' a.txt #覆盖复制第一行,在末尾追加粘贴

aaa

bbb

ccc

aaa #追加粘贴到这末尾 空行:剪切板中的空行被覆盖

Sed ‘1h;2g’ a.txt

Sed ‘1h;2G’ a.txt

# NSD SHELL DAY06

案例1：使用awk提取文本

案例2：awk处理条件

案例3：awk综合脚本应用

案例4：awk流程控制

案例5：awk扩展应用

awk编程语言工具概述:

由三个人研发出来(Aho Weiberger Kernighan)

grep 模糊查找, awk 精确查找,

# 步骤一：awk文本过滤的基本用法

1）基本操作方法

格式一: 前置命令 |　awk [选项] '[条件]{指令}' 文件

格式二：awk [选项] '[条件]{指令}' 文件

选项 -F 可指定分隔符,不指定，默认为空格（包括tab）

print：是最常用的编辑指令，若有多条编辑指令，可用分号分隔。

[root@server0 ~]# cat test.txt

hello the world

welcome to beijing

[root@server0 ~]# awk '{print}' test.txt 　　#未加-F选项以默认的空格为分隔符

hello the world

welcome to beijing

[root@server0 ~]# awk '{print $1}' test.txt 　　#$1表示第一列，不是位置变量

hello

welcome

[root@server0 ~]# awk '{print $3}' test.txt

world

beijing

[root@server0 ~]# head -1 /etc/passwd

root:x:0:0:root:/root:/bin/bash

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $1};{print $7}'

root

/bin/bash

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $1 888 $7}'

root888/bin/bash

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $1,$7}'

root /bin/bash

　以上都是以：号为分隔符

**步骤二：利用awk提取本机的网络流量、根分区剩余容量、获取远程失败的IP地址**

[root@server0 ~]# df -h

文件系统 容量 已用 可用 已用% 挂载点

/dev/vda1 10G 3.1G 7.0G 31% /

devtmpfs 906M 0 906M 0% /dev

tmpfs 921M 80K 921M 1% /dev/shm

tmpfs 921M 17M 904M 2% /run

tmpfs 921M 0 921M 0% /sys/fs/cgroup

[root@server0 ~]# df -h | awk '{print $1,$4}'

文件系统 可用

/dev/vda1 7.0G

devtmpfs 906M

tmpfs 921M

tmpfs 904M

tmpfs 921M

**awk 常用内置变量:**

$0 文本当前行的全部内容

$1 文本的第1列

$2 文件的第2列

$3 文件的第3列，依此类推

NR 文件当前行的行号

NF 文件当前行的列数（有几列）

**常量需要双引号**

[root@server0 ~]# head -1 /etc/passwd #测试内容

root:x:0:0:root:/root:/bin/bash

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $0}'

root:x:0:0:root:/root:/bin/bash

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $1}'

root

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $2}'

x

[root@server0 ~]# head -3 /etc/passwd | awk -F: '{print NR}'

1

2

3

[root@server0 ~]# head -1 /etc/passwd | awk -F[:/] '{print NF}'

10

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print "NR"}' #常量需要双引号

NR

[root@server0 ~]# head -1 /etc/passwd | awk -F: '{print $1,"的解释器"$7}'

root 的解释器/bin/bash

[root@server0 ~]# df -h

文件系统 容量 已用 可用 已用% 挂载点

/dev/vda1 10G 3.1G 7.0G 31% /

devtmpfs 906M 0 906M 0% /dev

tmpfs 921M 80K 921M 1% /dev/shm

tmpfs 921M 17M 904M 2% /run

tmpfs 921M 0 921M 0% /sys/fs/cgroup

题目:显示 /分区的剩余容量是:\*\*

[root@server0 ~]# df -h | awk '/vda1/{print $6,"分区的剩余容量是:"$4}' 查找

/ 分区的剩余容量是:7.0G

[root@server0 ~]# df -h | awk '/ \ /$/{print $6,"分区的剩余容量是:"$4}'

/ 分区的剩余容量是:7.0G

[root@server0 ~]# ifconfig eth0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.25.0.11 netmask 255.255.255.0 broadcast 172.25.0.255

inet6 fe80::5054:ff:fe00:b prefixlen 64 scopeid 0x20<link>

ether 52:54:00:00:00:0b txqueuelen 1000 (Ethernet)

RX packets 14611 bytes 1031558 (1007.3 KiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 3567 bytes 395826 (386.5 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@server0 ~]# ifconfig eth0 |grep "RX p"|awk -F[\(\)] '{print "当前主机接受 的网络数据是:",$2}'

当前主机接受的网络数据是: 1016.2 KiB

[root@server0 ~]# ifconfig eth0 |awk -F[\(\)] '/RX p/{print "当前主机接受的网络 数据是:",$2}'

当前主机接受的网络数据是: 765.0 KiB

3）根据/var/log/secure日志文件，过滤远程连接密码失败的IP地址

?

?

**步骤三：格式化输出/etc/passwd文件**

**1）awk处理的时机**

awk会逐行处理文本，支持在处理第一行之前做一些准备工作，以及在处理完最后一行之后做一些总结性质的工作。在命令格式上分别体现如下：

1. awk [选项] '[条件]{指令}' 文件
2. awk [选项] ' BEGIN{指令} [条件]{指令} END{指令}' 文件

BEGIN{指令}行前处理，读取文件内容前执行，指令执行1次

条件{指令} 逐行处理，读取文件过程中执行，指令执行n次

END{指令} 行后处理，读取文件结束后执行，指令执行1次

例子:统计系统中使用bash作为登录Shell的用户总个数

a.预处理时赋值变量x=0

b.然后逐行读入/etc/passwd文件，如果发现登录Shell是/bin/bash则x加1

c.全部处理完毕后，输出x的值即可。相关操作及结果如下：

[root@server0 ~]# awk 'BEGIN{x=0}/bash$/{x++}END{print x}' /etc/passwd

17

1. **格式化输出/etc/passwd文件**

**\t:插入TAB,常量需要 ”\t”**

要求: 格式化输出passwd文件内容时，要求第一行为列表标题，中间打印用户的名称、UID、家目录信息，最后一行提示一共已处理文本的总行数，如图-1所示。

[root@room9pc01 ~]# head -5 /etc/passwd |awk -F: 'BEGIN{print "User\tUID\tHome"}{print $1"\t"$6"\t"$7}END{print "Total",NR,"lines."}'

User UID Home

root /root /bin/bash

bin /bin /sbin/nologin

daemon /sbin /sbin/nologin

adm /var/adm /sbin/nologin

lp /var/spool/lpd /sbin/nologin

Total 5 lines.

[root@room9pc01 ~]#

**步骤一：认识awk处理条件的设置**

格式二：awk [选项] '[条件]{指令}' 文件

1）使用正则设置条件

[root@server0 ~]# awk -F: '/root/' /etc/passwd #默认含有{print}

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

[root@server0 ~]# awk -F: '/root/{print}' /etc/passwd #{print}可省略

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

**输出root或adm账户的用户名和UID信息：**

[root@server0 ~]# awk -F: '/^root|^adm/' /etc/passwd #方法一

root:x:0:0:root:/root:/bin/bash

adm:x:3:4:adm:/var/adm:/sbin/nologin

[root@server0 ~]# awk -F: '/^(root|adm)/' /etc/passwd #方法二

root:x:0:0:root:/root:/bin/bash

adm:x:3:4:adm:/var/adm:/sbin/nologin

**输出账户名称包含root的基本信息（第1列包含root）：**

[root@server0 ~]# awk -F: '$1~/root/' /etc/passwd

root:x:0:0:root:/root:/bin/bash

**输出其中登录Shell不以nologin结尾（对第7个字段做!~反向匹配）的用户名、登录Shell信息：**

[root@server0 ~]# awk -F: '$7!~/nologin$/' /etc/passwd

[root@server0 ~]# cat a.txt

abc 123 xyz

xyz abc

123

456

aaa

[root@server0 ~]# awk '$1~/abc/' a.txt #查找第1列为abc的行 ~匹配

abc 123 xyz

[root@server0 ~]# awk '$2~/abc/' a.txt #查找第2列为abc的行

xyz abc

[root@server0 ~]# awk '$2!~/abc/' a.txt #查找除了第2列为abc其他的所有行

abc 123 xyz

123

456

aaa

2）使用数值/字符串比较设置条件

比较符号：==(等于) !=（不等于） >（大于）

>=（大于等于） <（小于） <=（小于等于） -- ++

[root@server0 ~]# awk -F: 'NR==3{print}' /etc/passwd #输出行号等于3的

daemon:x:2:2:daemon:/sbin:/sbin/nologin

输出行中第三列大于等于1000的第一列与第三列

[root@server0 ~]# awk -F: '$3>=1000{print $1,$3}' /etc/passwd

[root@server0 ~]# awk -F: '$3<10{print $1,$3}' /etc/passwd #输出第三列小于10的

root 0

bin 1

[root@server0 ~]# awk -F: '$1=="root"' /etc/passwd #输出第一列等于root的行

root:x:0:0:root:/root:/bin/bash

第三列的值大于10和小于20的

[root@server0 ~]# awk -F: '$3>10&&$3<20' /etc/passwd operator:x:11:0:operator:/root:/sbin/nologin

games:x:12:100:games:/usr/games:/sbin/nologin

ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin

第三列值大于1000或者小于20的

[root@server0 ~]# awk -F: '$3>1000 && $3<20' /etc/passwd #错误

[root@server0 ~]# awk -F: '$3>1000 || $3<20' /etc/passwd

找到使用bash作登录Shell的本地用户,列出这些用户的shadow密码记录，

[root@server0 ~]# bash awk.sh

root ---> $6$UiGI4Tc2$htsXYn5cJnOqv3P1VLcUSgfjDu2pL5yiJBuua6foZAHdwqeuLHfYUfS/vBn27Wjvoel8EJgtdsMjyquqvKAmf1

student ---> $6$8oIjLCsc$/n1iQXYh1E6.uOEuJKgioqAtmqm2TQmkJGF2RwyteIr1tIfrPdiRYgWe6Sjen5/eMij2uHM/a1tue/QRlo3X80

----------脚本1-------------------------------------------------

#! /bin/bash

a=` awk -F: '/bash$/{print $1}' /etc/passwd`

for i in $a

do

p=`awk -F: '$1=="'$i'"{print $2}' /etc/shadow`

echo "$i ---> $p"

Done

-----------脚本2---------------------------------------------

#! /bin/bash

a=` awk -F: '/bash$/{print $1}' /etc/passwd`

for i in $a

do

p=`grep "^$i" /etc/shadow |awk -F: '{print $2}'`

echo "$i ---> $p"

done

------------------------------------------------------------------

## 4 案例4：awk流程控制

**步骤一：awk过滤中的if分支结构**

if分支结构（单分支、双分支、多分支）

统计/etc/passwd文件中UID小于或等于1000的用户个数：

[root@server0 ~]# awk -F: '{if($3<=1000){i++}}END{print i}' /etc/passwd

38

统计/etc/passwd文件中UID大于1000的用户个数：

[root@server0 ~]# awk -F: '{if($3>1000){i++}}END{print i}' /etc/passwd

16

统计/etc/passwd文件中登录Shell是“/bin/bash”的用户个数：

[root@server0 ~]# awk -F: '{if($7~/bash$/){i++}}END{print i}' /etc/passwd

17

分别统计/etc/passwd文件中UID小于或等于1000、UID大于1000的用户个数：

[root@server0 ~]# awk -F: '{if($3<=1000){x++}else{y++}}END{print x,y}' /etc/passwd

38 16

分别统计/etc/passwd文件中登录Shell是“/bin/bash”、 登录Shell不是“/bin/bash”的用户个数：

[root@server0 ~]# awk -F: '{if($7~/bash$/){x++}else{y++}}END{print x,y}' /etc/passwd

17 37

**步骤二：awk数组**

1. 数组的语法格式

数组是一个可以存储多个值的变量，具体使用的格式如下：

定义数组的格式：数组名[下标]=元素值

调用数组的格式：数组名[下标]

遍历数组的用法：for(变量 in 数组名){print 数组名[变量]}。

下标可以是:数字,小数,字母,IP都行 元素就只能是数字,小数,不能是字母

[root@server0 ~]# awk 'BEGIN{a[0]=11;a[1]=99;print a[1],a[0]}'

99 11

[root@server0 ~]# awk 'BEGIN{a++;print a}'

1

[root@server0 ~]# awk 'BEGIN{a[0]++;print a[0]}'

1

[root@server0 ~]# awk 'BEGIN{a[0]=0;a[1]=11;a[2]=22;for(i in a){print i,a[i]}}'

0 0

1 11

2 22

[root@server0 ~]# awk 'BEGIN{a["hehe"]=11;print a["hehe"]}'

11

统计ssh远程失败的IP地址,并统计次数

[root@server0 ~]# awk '/Failed/{a[$11]++}END{for(i in a){print a[i],i}}' /var/log/secure

6 172.25.0.250

# NSD SHELL DAY07

案例1：编写一键部署软件脚本

案例2：启动脚本

案例3：编写监控脚本

案例4：编写安全检测脚本

案例5：编写进度显示脚本

1 案例1：编写一键部署软件脚本

1.1 问题

本案例要求编写脚本实现一键部署Nginx软件（Web服务器）：

* 一键源码安装Nginx软件
* 脚本自动安装相关软件的依赖包
* 脚本自动判断yum是否可用

-------脚本-----------------------------------------------------------------

#! /bin/bash

a=`yum repolist |awk '/^repolist/{print $2}' |sed 's/,//'`

if [ $a -eq 0 ];then

echo "此yum源不可用"

exit

fi

echo "yum有$a 个包可用"

read -p "是否继续安装 gcc openssl-devel pcre-devel编译工具y or n:" b

if [ $b == y ];then

echo "3"

sleep 1s

echo "2"

sleep 1s

echo "1"

else

exit

fi

echo "开始安装 gcc openssl-devel pcre-devel编译工具"

sleep 1s

yum -y install gcc openssl-devel pcre-devel #开始源码编译安装,安装开发工具

tar -xf nginx-1.12.2.tar.gz

cd nginx-1.12.2

./configure #配置,指定安装目录.功能模块等选项

make #编译,生成可执行的二进制程序文件

make install #安装,将编译好的文件复制到安装目录

------------------------------------------------------------------------------------------

2 案例2：启动脚本

2.1 问题

本案例要求编写Ngin启动脚本，要求如下：

* 脚本支持start、stop、restart、status
* 脚本支持报错提示
* 脚本具有判断是否已经开启或关闭的功能

脚本通过位置变量$1读取用户的操作指令，判断是start、stop、restart还是status。

netstat命令可以查看系统中启动的端口信息，该命令常用选项如下：

-n以数字格式显示端口号

-t显示TCP连接的端口

-u显示UDP连接的端口

-l显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p显示监听端口的服务名称是什么（也就是程序名称）

------------------------------脚本-------------------------------------------------------------

#! /bin/bash

case $1 in

start)

/usr/local/nginx/sbin/nginx;;

stop)

/usr/local/nginx/sbin/nginx -s stop;;

restart)

/usr/local/nginx/sbin/nginx -s stop

/usr/local/nginx/sbin/nginx;;

starus)

netstat -ntulp | grep -q nginx #查看系统中启动的端口信息

[ $? -eq 0 ] && echo "服务已经启动" || echo ""服务未启动 ;;

\*)

echo "Error";;

esac

3 案例3：编写监控脚本

3.1 问题

本案例要求编写脚本，实现计算机各个性能数据监控的功能，具体监控项目要求如下：

* CPU负载
* 网卡流量
* 内存剩余容量
* 磁盘剩余容量
* 计算机账户数量
* 当前登录账户数量
* 计算机当前开启的进程数量
* 本机已安装的软件包数量

---------------脚本---------------------------------

#! /bin/bash

echo "CPU负载情况:"

cpu=`uptime|awk '{print $NF}' `

echo "本机CPU最近15分钟的负载是:$cpu"

echo

echo "网卡流量:"

js=`ifconfig eth0|awk -F[\(\)] '/RX p/{print $2}'`

fs=`ifconfig eth0|awk -F[\(\)] '/TX p/{print $2}'`

echo "入站网卡流量为:$js"

echo "出站网卡流浪为:$fs"

echo

echo "内存信息:"

nc=`free | awk '/Mem/{print $4}'`

echo "内存剩余:$nc"

echo

echo "磁盘空间情况:"

df -h |awk 'NR!=1{print $1"分区可用""\t"$4}'

echo

echo "账户数量:"

zh=`cat /etc/passwd | wc -l`

echo "本机共有账户:$zh"

echo

echo "进程数"

jcs=`ps aux | wc -l`

echo "当前开启的进程数为:$jcs"

echo

echo "软件安装:"

rjs=`rpm -qa |wc -l`

echo "本机共安装软件:$rjs 个"

## 4 案例4：编写安全检测脚本

### 4.1 问题

本案例要求编写脚本，防止远程ssh暴力破解密码，具体监控项目要求如下：

* 检测ssh登录日志，如果远程登陆账号名错误3次，则屏蔽远程主机的IP
* 检测ssh登录日志，如果远程登陆密码错误3次，则屏蔽远程主机的IP

-----------脚本1----------------------------------------------------

#! /bin/bash

echo "用户名输错的记录"

awk '/Invalid u/{ip[$10]++}END{for(i in ip){print ip[i],i}}' /var/log/secure

echo

echo "密码输错的记录"

awk '/Failed p/{ip[$11]++}END{for(i in ip){print ip[i],i}}' /var/log/secure

-----------脚本2----------------------------------------------------

#!/bin/bash

awk '/Failed password/{print $11}' /var/log/secure | awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}' | awk '$1>3{print $2}'

awk '/Invalid user/{print $10}' /var/log/secure | awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}' | awk '$1>3{print $2}'

## 5 案例5：编写进度显示脚本

### 5.1 问题

本案例要求编写脚本，实现带进程显示的复制脚本，具体要求如下：

* 默认Linux的cp命令不具有进度显示
* 我们需要自己编写脚本实现进度显示
* 可以使用进度条的方式，或者显示百分比的方式

--------脚本1---------------------

#! /bin/bash

jindu(){

while :

do

echo -ne '\033[43m \033[0m'

sleep 0.3

done

}

jindu &

sleep 5

echo

kill $!

-----------脚本2-----------------------

#! /bin/bash

jindu(){

while :

do

echo -ne '\033[33m# \033[0m'

sleep 0.3

done

}

jindu &

sleep 5

echo

kill $!

# 新增：

## set

linux shell 脚本编写好要经过漫长的调试阶段，可以使用sh -x 执行。但是这种情况在远程调用脚本的时候，就有诸多不便。又想知道脚本内部执行的变量的值或执行结果，这个时候可以使用在脚本内部用 set -x ,set去追踪一段代码的显示情况，执行后在整个脚本有效

set -x 开启

set +x关闭

set -o 查看