**切换用户的shell环境**

默认环境为bash

bash是一个具体的解释器,可以直接使用的,是一个产品,是默认的解释器

shall是linux内核和人类之间的一个解释器,是一个概念

cat /etc/shells //确认系统都有哪些shall环境

ksh //切换到ksh环境

exit //返回默认环境

usermod -s 解释器目录 用户名

usermod -s /bin/ksh zhangsan //更改用户shall

rpm -qa | grep completion

有这个包才能tab补齐,但是命令和文件名有没有这个包都可以tab补齐

**命令历史**

gerp HISTSIZE | /etc/profile

HISTSIZE=1000 //默认为1000条

history | wc -l //查看当前用户已输入的命令条数

history | tail //列出最后输入的10条命令

!1024 //执行第1024条命令

!cat //执行上一次的cat命令

history -c //清空当前用户的历史命令

> ~/.bash\_history //清空历史命令文件

**别名**

alias //查看当前别名列表

别名的设置在~/.bashrc文件中

grep 'alias' ~/bashrc

alias lh = 'ls -lh' //添加别名

unalias lh //取消别名

**重定向**

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,标准号为2

重定向标准输出

>>追加重定向

>覆盖重定向

2>重定向标准错误

&>正确错误结果全部重定向

ls -ld /etc/ > stdout.txt //将显示结果重定向到文件

mail -s Error root < ErrorList.txt //将txt中文字重定向给mail并发送

/dev/null //黑洞文件,重定向给它将不会做任何记录

**管道**

将上一个命令的输出结果输出给下一个命令使用

ls -l /etc | wc -l //统计etc下有多少文件

yum list | grep cluster //查找yum下有多少包含cluster的软件包

**Shell脚本的设计**

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）

shell脚本运行前要添加执行权限

chmod +x

**Shell脚本的执行方式**

作为"命令字"执行,必须是以相对路径或者绝对路径方式执行,并且有x权限

./first.sh

root/first.sh

作为"参数执行":使用bash,sh,source加载脚本文件

bash first.sh //开启子进程

sh first.sh //开启子进程

source first.sh //不开启子进程

**Shell变量**

test=11 //给test赋值为11

echo $test //调用$test的值

如果变量后面要和输出的字符串连接在一起,用大括号{}将变量括起来

echo $testRMB

//不会报错,会输出空,因为$testRMB被看做一个变量,且此变量未赋值

echo ${test}RMB

//输出11RMB

unset test

//撤销test变量

全局变量存放于/etc/profile中,对所有用户有效

用户变量存放于~/bash\_profile中,仅对此用户有效

USER //记录用户名

LOGNAME //记录登录名

HOME //记录宿主目录

SHELL //记录登录Shell

HOSTNAME //记录主机名

TERM //记录终端类型

环境变量PS1表示Shell环境的一级提示符,是Linux终端用户的一个环境变量，用来定义命令行提示符的参数。

PS1的常用参数以及含义:

\d ：代表日期，格式为weekday month date，例如："Mon Aug 1"

\H ：完整的主机名称

\h ：仅取主机名中的第一个名字

\t ：显示时间为24小时格式，如：HH：MM：SS

\T ：显示时间为12小时格式

\A ：显示时间为24小时格式：HH：MM

\u ：当前用户的账号名称

\v ：BASH的版本信息

\w ：完整的工作目录名称

\W ：利用basename取得工作目录名称，只显示最后一个目录名

\# ：下达的第几个命令

\$ ：提示字符，如果是root用户，提示符为 # ，普通用户则为 $

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

使用env可查看所有环境变量

使用set可查看所有变量(包括env能看到的环境变量)

**位置变量与预定义变量**

echo $0 //脚本的名称

echo $1 //第一个参数

echo $2 //第二个参数

echo $\* //所有位置变量的值//显示你传的每一个参数

echo $# //已加载的位置的变量个数//给Shell传了多少个值

echo $$ //当前进程的进程号

echo $? //程序退出后的状态,0表示正常,其他值表示异常

Echo $! //最后一个后台执行的程序PID

双引号界定一个启用特殊符号的字符串

单引号界定一个屏蔽特殊符号的字符串

test=11

echo "$test" //输出11

echo '$test' //输出$test

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

tar -czf log-`date +%Y%m%d`.tar.gz /var/log

//生成以当前时间为名的tar包

**read,接收用户输入并赋值给变量**

read -p //显示提示信息

read -p "请输入一个整数：" num //引号和变量之间要有空格

echo $num

**stty终端显示控制**

stty -echo //关闭回显

stty echo //开启回显

此命令可以用在让用户输入密码时

stty -echo

read -p "请输入密码：" pwd

stty echo

这样将不会将密码以明文的形式显示在终端中

**export发布全局变量**

默认情况下,自定义的变量为局部变量,只对当前Shell有效,而在其他Shell中无法直接使用,比如定义num变量,当切换到sh或者bash的子Shell中,变量num处于未定义状态

使用export发布全局变量,供子进程使用

**整数运算**

1)使用expr命令

其中乘法运算需要用\\*来转义,避免被作为Shell的通配符

参加运算的整数和运算符号之间需要有空格,变量需要使用$符引用

x=2

expr $x \\* 2 //结果为4

2)使用$[ ]或者$(())表达式

乘法\*无需转义,运算符号两侧可以没有空格,引用变量可以不写$符

x=2

echo $[x\*2] //结果为4

3)let命令

expr和$[ ]只进行计算,不会改变变量的值,let可以直接对变量值作运算在保存新的值

let不会回显结果,需要配合echo输出

x=2

let y=x\*2

echo $y //结果为4

# X++ //X = X + 1

# X-- //X = X - 1

# X += 78 //X = X + 78

# X -= 78 //X = X - 78

# X \*= 78 //X = X \* 78

# X /= 78 //X = X / 78

# X %= 78 //X = X % 79

**小数计算**

1)bc交互式计算

输入bc进入交互式界面

2)bc非交互式计算,scale=N设置小数点长度

echo “scale4;12.34+5.678” | bc //结果为18.018

**条件测试**

A && B //且

A || B //或

A;B //先执行A,后执行B,两者没有逻辑关系

Shell中的且或非从左向右依次执行,没有优先级

-z检查变量是否为空值

var1="nb" ; var2=""

[ -z "$var1" ] && echo "空值" || echo "非空值"

**整数值比较**

-gt //大于

-ge //大于或等于

-ne //不等于

-eq //等于

-le //小于或等于

-lt //小于

who | wc -l //统计当前已登录人数

**识别文件/目录状态**

-e //文件/目录是否存在

-d //是否为目录

-f //是否为文件

-r //是否可读

-w //是否可写

-x //是否可执行

root对任何文件都有可读权限,即使没有r权限一样可读

**if语句**

单分支

if [ 判断式 ] ; then

执行语句

fi

双分支

if [ 判断式 ] ; then

执行语句

else

执行语句

fi

多分支

if [ 判断式 ] ; then

执行语句

elif [ 判断式 ] ; then

执行语句

else

执行语句

fi

测试是否能ping通

-c //发送测试包的个数

-i //每个测试包的间隔秒

-W //超时秒数

ping -c 3 -i 0.2 -W 3 $1 &> /dev/null

if [ $? -eq 0 ] ; then

echo "通了"

else

echo "不通"

fi

**for循环**  
语法  
for 变量名 in 值列表  
do  
命令语句  
done  
  
for语句是用变量的值去和值列表里进行比对,值列表有多少个则循环多少次  
  
for FileName in ls /etc/\*.conf   
for i in {1..5}   
for i in seq 5   
  
seq 5 10 //生成5-10之间的整数  
seq 1 2 10 //生成10以内的奇数  
seq 最小值 递增值 最大值  
  
**while循环**  
语法  
while [条件]  
do  
命令语句  
done  
  
**自建死循环**  
while :  
do  
命令语句  
done  
  
**生成随机数**  
i=$[RANDOM%10+1] //生成10以内的随机整数  
方法

RANDOM%10生成0-9的随机整数,最大值为9,+1获得10以内的随机数  
RANDOM%最小数+(最大数-最小数) //某个范围内的随机整数  
  
**case分支**  
语法  
case 变量值 in  
模式1)  
命令语句;;  
模式2)  
命令语句;;  
\*)  
命令语句;;  
esac  
  
\*)捕获前面没有匹配的选项,并执行后面的语句  
  
**Shall函数**  
方法  
1)  
function 函数名{  
命令语句  
}  
  
2)  
函数名(){  
命令语句  
}  
  
函数在调用前必须先声明  
  
break结束整个循环  
continue结束当前循环,进入下一次循环  
exit退出整个脚本  
  
求一个累加的数  
while read -p "请输入待累加的数字,以0结尾" x  
do  
[$x -eq 0 ] && break //匹配到0的时候退出整个循环  
sum=$[sum+x]  
done  
echo "$sum"  
  
求20以内能被6整除的数的和  
i=0  
while [$i -le 20]  
do  
let i++  
[$[i % 6 ] -ne 0 ] && continue

//匹配到不能被6整除的数的时候,跳过此次循环

let i+i  
done  
echo $i

**字符串截取**  
${变量名:起始位置:长度}  
phone="13788768897"  
echo ${phone:0:6} //位置1从0开始计算,输出结果137887  
echo ${phone::6} //输出结果137887

expr substr "$变量名" 起始位置 长度  
expr substr "$phone" 1 6 //位置1从1开始计算,输出结果137887

echo $变量名 | cut -b 起始位置-结束位置  
echo $phone | cut -b 1-6 //输出结果137887  
echo $phone | cut -b 8- //输出结果8897  
echo $phone | cut -b 9 //输出结果8  
  
**字符串替换**  
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}

x=qq.com.cn  
 echo ${x#\*.} //输出结果com.cn

从左向右，最长匹配删除：${变量名##\*关键词}

echo ${x##\*.} //输出结果cn

从右向左，最短匹配删除：${变量名%关键词\*}

echo ${x%.\*} //输出结果qq.com

从右向左，最长匹配删除：${变量名%%关键词\*}

echo ${x%%.\*} //输出结果qq  
  
**字符串的初始值(默认值)**  
${变量名:-赋的值}  
${var:-word}

//判断var已经存在且非空,返回var原有的值,如果为空返回word  
  
**expect预期交互**  
控制ssh登录  
yum  -y  install  expect //安装expect  
  
expect << EOF  
spawn ssh root@172.25.0.10 //创建交互式进程  
expect "password:" { send "123456\r" } //自动发送密码  
expect "# { send "pwd > /tmp/$user.txt \r" } //发送命令  
expect "#" { send "exit\r" }  
EOF  
  
探测到显示password时,执行大括号中的命令  
expect的最后一行你命令不会被执行

ssh -o StrictHostKeyChecking=no [root@172.242.5.110](mailto:root@172.242.5.110)

//取消确认登录秘钥

**数组**  
声明数组方法1)  
var=(11 22 33 44)  
  
声明数组方法2)  
var[0]=11  
var[1]=22  
var[2]=33  
var[3]=44  
  
输出数组方法1)  
echo ${var[0]} //输出数组中第1个下标的数  
echo ${#var[@]} //输出数组中元素的个数  
echo ${var[\*]} //输出数组中所有的元素  
echo ${var[@]:1:5} //输出数组中从第2个元素开始,连续输出5个元素

**正则表达式**  
^ //匹配行首  
$ //匹配行尾  
[] //匹配集合中的任意单个字符  
[^] //对集合取反  
. //匹配任意单个字符  
\* //匹配前一个字符出现的任意次数,不能单独使用  
\{n,m\} //匹配前一个字符出现n到m次  
\{n\} //匹配前一个字符出现n次  
\{n,\} //匹配前一个字符出现n次以上  
\(\) //保留  
  
**拓展正则表达式**

+ //最少匹配一次

? //最多匹配一次

{n,m} //匹配n到m次

() //组合为整体,保留

| //或者

\b //单词边界

grep不支持扩展正则表达式,需要使用egrep或者grep -E

grep -q //静默显示

egrep -c //统计行数,和wc -l一样

egrep -m10 //显示10行

grep  
 -i //取反查找  
 -v //忽略大小写

**sed基本用法**

语法

sed 选项 '条件指令' 文件名

常用选项

-n //屏蔽屏幕输出

-r //让sed支持扩展正则

-i //sed直接修改源文件,默认sed只会修改内存中的临时文件,源文件无影响

sed '1p' test.txt //显示第一行

但是默认输出会将整个文档都输出来,而且第一行会被输出两次

sed工具p指令

sed -n '1p' test.txt //屏蔽默认输出,只会输出第一行

sed -n '3,6p' test.txt //输出3-6行

sed -n '3p;6p' test.txt //输出第3和第6行

sed -n '3,+10p' test.txt //输出第3行及后面的10行

sed -n '1~2p' test.txt //输出奇数行

sed -n '2~2p' test.txt //输出偶数行

sed -n '/root/p' test.txt //输出带有root的行

sed -n '/bash$/p' test.txt //输出bash结尾的行

sed -n '$=' test.txt //输出test的总行数

sed工具d指令

sed '1-4d' test.txt //删除1-4行

sed '/xml/d' test.txt //删除所有包含xml的行

sed '/xml/!d' test.txt //删除不包含xml的行，!符号表示取反

sed '/^install/d' test.tx //删除以install开头的行

sed '$d' test.txt //删除文件的最后一行

sed '/^$/d' test.txt //删除所有空行

sed工具s指令(s/旧内容/新内容/选项)

sed 's/2017/xxxx/' test.txt //将2017换成xxxx

sed 's/2017/xxxx/g' test.txt //将所有的2017换成xxxx

sed 's/2017/xxxx/2' test.txt //将每行中的第二个2017换成xxxx

sed 's/2017//2' test.txt //删除每行中的第二个2017

sed -n 's/2017/xxxx/p' test.txt //将2017换成xxxx并打印

sed '4,7s/^/#/' test.txt //将第4~7行注释掉(行首加#号)

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

sed 's/.$//' test.txt //删除每行的最后一个字符

sed -r 's/^(.)(.\*)(.)$/\3\2\1/'

#-r 允许sed使用拓展正则

#s/原始文字/替换文字

#s/^..\*.$/替换文文字

//以任意字符开头任意字符结束中间有多个任意字符表示任意行

#s/^(.)(.\*)(.)$/\3\2\1/

//用括号括起来表示此位置保留,不进行替换,替换位置\数字表示替

换后第n个括号内的数据所在的位置

//此句话的意思就是把任意行的第一个和最后一个字调换位置

sed 's/[0-9]//' test.txt //删除所有数字

sed 's/^( )+//' test.txt //删除行首空格

sed 's/([A-Z])/\1/g' test.txt //为所有的大写字母添加中括号

sed -i //在指定的行之前一行添加文本

sed -a //在指定的行之后一行添加文本

sed -c //替换指定的行

sed '2a xx' test.txt //在第二行后一行,追加xx

sed '2i xx' test.txt //在第二行前一行,追加xx

sed '2c xx' test.txt //将第二行替换成xx

sed '$a xx' test.txt //在最后一行后面追加xx

**awk工具**

语法

awk[选项] '[条件]{指令}'

$1为第1列,$2为第2列

awk是最常用的编辑指令,若有多条指令,用分号分隔每条命令

awk '{print $1,$3}' test.txt //打印第3和第5列

df | awk '{print $4}' //打印硬盘剩余空间

awk默认以空格为分隔符,用-F设置其他字符为分隔符,也可以同时设置多个不同的分隔符

awk -F[:/] '{print $1,$10}' /etc/passwd //以:和/作为分隔符显示

NR文件当前的行数,NF文件当前的列数

awk -F: '{print NR,NF}' /etc/passwd //显示passwd的行号和列号

awk -F: 'NR==3{print}' /etc/passwd //输出第3行(NR等于3)

awk -F: 'NR==’$num’{print}' /etc/passwd

//awk中用’’单引号引起来

awk打印常量用""引起来,常量中使用变量"' '"双引号中套单引号

awk -F: '{print $1,"的解释器是:" ,$7}' /etc/passwd

root的解释器是: /bin/bash

过滤网卡流量

RX为接收流量,TX为发送流量

ifconfig eth0|awk '/RX p/{print $5}'

awk '/需要匹配的行中的关键字/{指令}'

#df -h

文件系统 容量 已用 可用 已用% 挂载点

/dev/sda2 19G 7.2G 11G 40% /

df -h | awk '/\/$/{print $4}'

#找到/根目录结尾的行,打印第4列

/var/log/secure //存放ssh登录信息

Failed password //密码错误

Invalid user //非法用户

awk '/Failed password/{print $11}' /var/log/secure

//过滤密码输入错误的ip

**awk预处理**

语法

awk [选项] 'BEGIN{指令} {指令} END{指令}' 文件名

BEGIN{}读取文件前执行一次,{}读取文件的过程中逐行N次,END{}读取文件结束后执行一次

若只有BEGIN或END与处理时,可以不读取文件

BEGIN和END的{}内调用变量不用加$

awk 'BEGIN{A=24;print A\*2}'

awk 'BEGIN{print x+1}' //输出1,变量未初始化默认值为0

awk 'BEGIN{x=0}/bash$/{x++} END{print x}' /etc/passwd

//统计passwd中以bash结尾(用户的bash在行尾)的个数

awk -F: 'BEGIN{print "USER\tUID\tHOME"} {print $1 "\t"$3 "\t"$6} END{print "TOTAL "NR" LINES"}'

/etc/passwd

awk -F: '/^(root|adm)/' /etc/passwd

//输出以root或者adm开头的行

awk -F: '$1~/root/' /etc/passwd

//匹配第1列包含root的行

awk -F: '$1=="root"' /etc/passwd

//匹配第1列为root的行(严格匹配)

awk -F: '$3>=1000{print}' /etc/passwd

//输出第3列大于等于1000的行

awk -F: '$3>1000 || $3<10' /etc/passwd

//输出第3列大于1000或者小于10的行

seq 200| awk '$1%3==0{i++} END{print i}'

//计算200以内能被3整除的个数

//因为要最后输出总个数,所以输出要写在END{}里,否则,执行一次取摩计算将输出一次

awk '/^(127|192)/' /etc/hosts

//输出以127或者192开头的记录

seq 100| awk '$1%7==0||$1~/7/'

//计算100以内能被7整除或者包含7的数

**awk控制流程**

awk '{if($3>1000){i++} END{print i}}' /etc/passwd

//统计passwd中UID($3)小于或等于1000的用户个数

awk '{if($3>1000){i++}else{j++} END{print i,j}}' /etc/passwd

**awk循环**

awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}' /var/log/secure

**awk数组**

awk 'BEGIN{a[0]=0;a[1]=88;print a[0],a[1]}'

//定义一个数组,并输出他们每个下标的值

数组的下标可以使用字符串,但是需要用到双引号

awk 'BEGIN{a["aa"]=0;a["bb"]=88;print a["aa"],a["bb"]}'

awk '{ip[$1]++} END{for(i in ip) {print i,ip[i]}}' /var/log/httpd/access\_log | sort -nr -k 2

//统计IP访问地址及访问量

//sort -r降序,-n以数值排序,sort默认一位和一位进行排序,可能会出现10比2小的情况

//-k 2根据第二列的数值大小进行排序

uptime //查看cpu负载

第一列右边三个数字,第一个为cpu一分钟内的负载量,第二个为五分钟的负载量,第三个为十五分钟的负载量

如果为单核cpu,负载到1则为满载,四核cpu,负载1则为25%负载

free //查看内存信息

df -h //查看硬盘信息

cat /etc/passwd | wc -l //查看计算机用户数量

who | wc -l //查看ssh登录数量

rpm -qa |wc-l //查看已安装软件的数量

进度条脚本

#!/bin/bash

jindu(){ //定义函数

while :

do

echo "#"

sleep 0.3 //每输出一个#暂停0.3秒

done

}

jindu & //后台执行函数

cp -r $1 $2 //复制命令,执行时脚本名+原路径+复制路径

kill $! //$!最后一个后台执行的程序PID