## 基本命令

ll /proc/进程id

#!/bin/bash

# 遍历HDFS目录

afsshell --username=lbs-mapsearch --password=lbs-mapsearch lsr afs://aries.afs.baidu.com:9902/app/dt/minos/212835095/70087304/20240519 | grep -v '/$' | grep 'bj' | while read -r line ; do

afsshell --username=lbs-mapsearch --password=lbs-mapsearch get afs://aries.afs.baidu.com:9902$line .

Done

all=`cat service.log\*`

echo 'done'

cat logid.txt | grep 'xxx' | while read -r line ; do

b=`grep aicontrol\_search $all | grep nlu\_res | grep $line`

echo "$line $b"

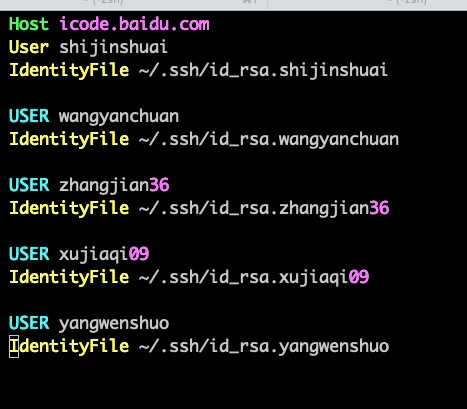
Done

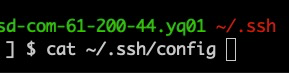
# 获取当前脚本所在的绝对路径

script\_path=$(cd `dirname $0`; pwd)

# 获取当前执行路径

current\_path=$(pwd)





这个user就是iconde的user

这个User其实是icode的user 不是linux的user

nice -19 执行命令加上这个指定这个进程的优先级为最低 这通常用于运行那些你不希望它们干扰系统其他重要进程的任务

Grep log的时候用

ll -h 查看大小

vim -d diff两个文件

> filename 清空文件内容

-gt是大于的意思。 -eq是等于的意思。 -ne是不等于的意思。 -ge是大于等于的意思。 -lt是小于的意思。 -le是小于等于的意



sh表示脚本默认使用sh脚本解释器。

未指定脚本解释器默认为 ./。

具体解释：

使用“./”执行脚本，对应的xxx.sh脚本必须要有执行权限。

使用“sh” 执行脚本，对应的xxx.sh没有执行权限，亦可执行。

当脚本开头使用#！设置使用的shell类型时，使用“./”执行脚本时，则使用“#！”标志的shell执行脚本；若无使用“#！”标记，则使用系统设置的默认shell执行脚本。

nohup ./program >/dev/null 2>log &

lsof |grep deleted

查看被删deleted的文件 这些文件du -sh算不到 但是还是会占用磁盘

memcpy memmove区别和实现

memcpy与memmove的目的都是将N个字节的源内存地址的内容拷贝到目标内存地址中。

但当源内存和目标内存存在重叠时，memcpy会出现错误，而memmove能正确地实施拷贝，但这也增加了一点点开销。

根据名字杀进程

ps -ef | grep -v grep | grep auto\_process | awk '{print $2}' | xargs kill -9

shuf 111.txt

Grep -E 正则匹配

Grep -o 只匹配符合正则的那个字符串

du -sh \* 查看当前目录下所有文件大小

cat fileA fileB fileC | sort | uniq

cat 222.txt | awk -F '//' '{print $3}' | sort | uniq -c | sort -k1nr | head -3

| sort | uniq -c

sort排序是选择第一列排序, uniq这个只有重复的挨着才会去重,所以要先配合sort使用, -c是为了显示有多少个

kill -9 $( ps -ef | grep 进程名关键字 | awk -F {print $2}' | tr '\n' ' ')

Netstat -ant | grep 3306

telnet 10.12.187.136 9001

cat /etc/issue

mkdir -pv ./abc/123 递归创建文件加.

Learn vim

https://github.com/wsdjeg/Learn-Vim\_zh\_cn/blob/master/ch00\_read\_this\_first.md

## Linux

Linux内核都是同一个,那么多不同的版本(centos ubuntu)都是给套的壳.

Sbin是super bin的意思,root才能访问到.

Usr不是用户的意思,而是unix shared resource,共享资源

dev用于存放设备文件

mnt mount系统管理员安装临时文件系统的安装点.

boot 存放用于系统引导时使用的各种文件.

root是超级用户目录,与之相对的是home,存放普通用户文件.

etc 主配置目录,网卡的配置啊,防火墙的配置啊

opt 当前linux把那些额外安装的软件都装在opt目录下.

ASCII就是a 97 A 65那个.

Unicode java中char类型的编码不是太好,能用一个字节存的东西他也用两个存

Utf-8,变长的编码,优化unicode而来,中文三个字节.

GBK,中文两个字节,操作系统通常是这个.而我们java的字符串就是根据系统编码来解析的.

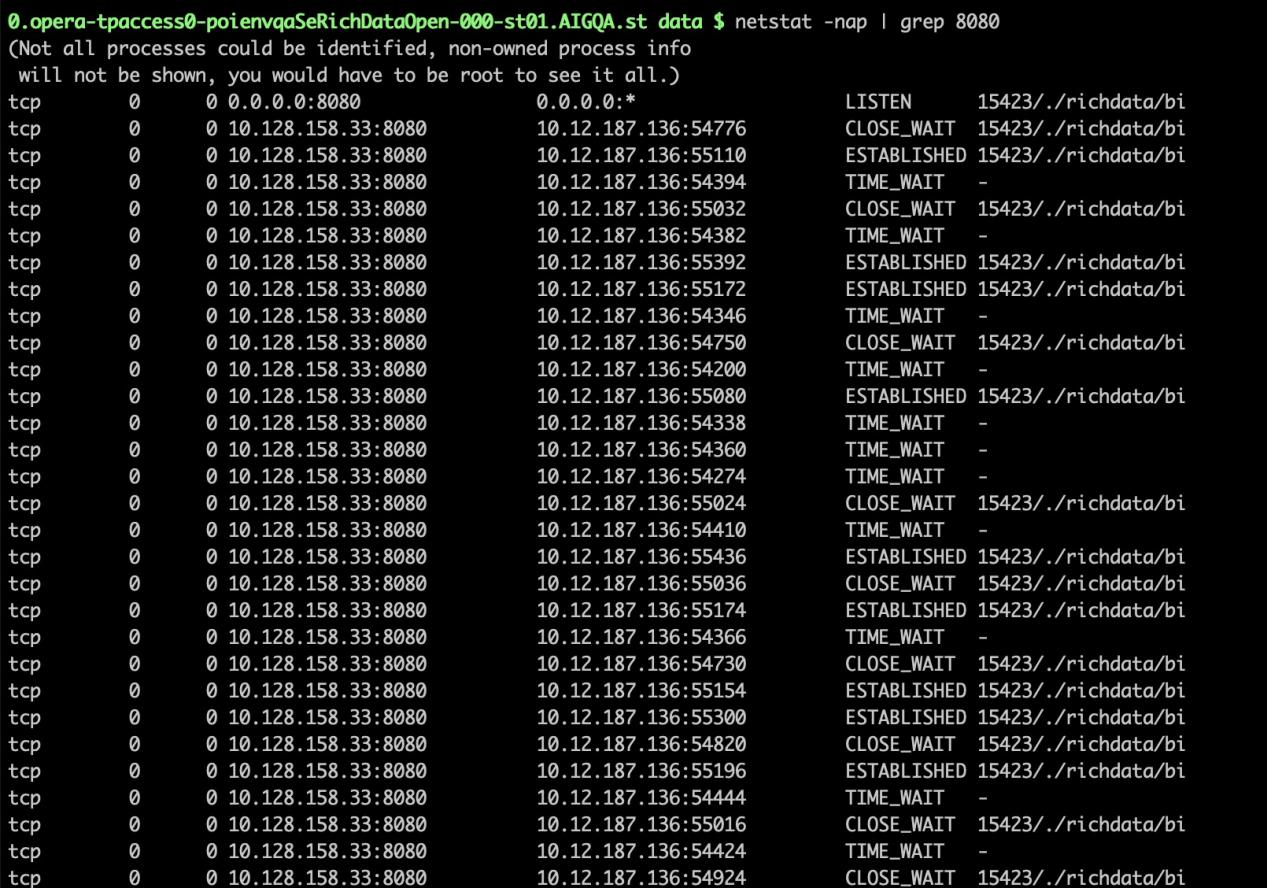
## Shell

使用Source命令 在当前Shell脚本中引入其他脚本的变量

function log\_print() {

echo "$(date '+%Y-%m-%d %H:%M:%S') ${1}" >> $LOG\_FILE

}



A=`[ls](https://www.baidu.com/s?wd=ls&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "/Users/yangwenshuo/Documents\\x/_blank) [-l](https://www.baidu.com/s?wd=-l&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "/Users/yangwenshuo/Documents\\x/_blank)`

这个符号的意思,就是把后面的那个命令的结果当作值赋给前面的那个变量

grep -Po xx.\*? | sort | uniq -c

.\*? 任意字符 任意次数 匹配一次

get\_instance\_by\_service -a group.opera-online-PoiStreamGateway-StreamGateway-all.map-poi.all |cut -d " " -f 3

command1 && command2

&&左边的命令（命令1）返回真(即返回0，成功被执行）后，&&右边的命令（命令2）才能够被执行；换句话说，“如果这个命令执行成功&&那么执行这个命令”。

字符串变量如果是用双引号,里面就可以用转义符号

如果字符串是用单引号包裹的,那么转义符号是没有用的

字符串的拼接,字符串拼接肯定是有变量的拼接,常量之间你还拼接个啥啊

Your\_name=”yws”

Greeting = “hello, “$your\_name””

Greeting = “hello, ${your\_name}”

Tests=”abcd”

Echo ${#tests} 获取一个字符串的长度

Test=”aa bb cc dd”

Echo ${test:1:4}

从第二个字符截取到第4个字符

shell支持数组,只支持一维数组.

数组元素之间用空格隔开, 下标从0开始.

没有长度限制这一说的

Array\_name=(a b c d e f g)

Array\_name[0]=a

Array\_name[1]=b

Array\_name[2]=c

Array\_name[3]=d

这都是定义数组,往数组的元素里面赋值.

读数组的内容,

shell里面只要是读取内容都是需要用到$这个符号的.

${array\_name[0]}

${array\_name[1]}

${array\_name[2]}

Length=${#array\_name[\*]}

获取整个数组的长度.

反正就是获取变量都是要用到$ 获取长度就会用到#

获取到外面传递进来的参数.

./test.sh 1 2 3

那么脚本里面就可以这样

#!/bin/bash

Echo “第一个参数 $0”

Echo “第二个参数 $1”

Echo “第三个参数 $2”

是这样获取传递进来的参数的.

$# 传入到脚本的参数的个数

$$ 当前脚本执行的进程号

获取所有的参数可以用$\*

For i in $\*; do

Echo $i

done

for循环就是这样弄就行

***For xx in xxx; do***

***Xxx***

***done***

a=10  
b=20  
  
val=`expr $a + $b`  
echo "a + b : $val"  
  
val=`expr $a - $b`  
echo "a - b : $val"

shell是不支持原生的加减乘除的,必要用用`expr xxxx`

还有就是+ - 这些符号要和左右两边的变量要有一个空格.

if [ $a == $b ]  
then  
   echo "a 等于 b"  
fi  
if [ $a != $b ]  
then  
   echo "a 不等于 b"  
fi

If [ 1 + 2 -eq 3 ]

Then

Xxx

Fi

注意 if的结尾是个fi 还有就是if[ 这里使用中括号包裹的 ]

If [ xxx ]

Then

Xxx

Else

Xxx

Fi

关系运算符 shell里是不能直接用< > = 这些东西的

-eq 等于

-ne 不等于

-gt 大于 bigger than

-lt 小于 little than

-ge 大于等于 bigger equal

-le 小于等于 little equal

a=10  
b=20  
  
if [ $a -eq $b ]  
then  
   echo "$a -eq $b : a 等于 b"  
else  
   echo "$a -eq $b: a 不等于 b"  
fi

逻辑运算符 也就是与或非

非 !

或 -o

与 -a

if [ $a -lt 100  ***-a***  $b -gt 15 ]  
then  
   echo "$a 小于 100 且 $b 大于 15 : 返回 true"  
else  
   echo "$a 小于 100 且 $b 大于 15 : 返回 false"  
fi

马德好像用||和&&也是可以的.

字符串串运算符

[$a = $b] 这个是用来比较这两个字符串是否一摸一样的

[$a != $b] 比较两个字符串是否不相等

[-z $a] 检测字符串的长度是否为0 为0返回true

[$a] 检测字符串是否为空,如果为空的话结果是true

file="/var/www/runoob/test.sh"

其实这样的一个字符串路径就是一个file,你就可以对文件进行一些系列操作的了

if [ -r $file ]  
then  
   echo "文件可读"  
else  
   echo "文件不可读"  
fi  
if [ -w $file ]  
then  
   echo "文件可写"  
else  
   echo "文件不可写"  
fi  
if [ -x $file ]  
then  
   echo "文件可执行"  
else  
   echo "文件不可执行"  
fi  
if [ -f $file ]  
then  
   echo "文件为普通文件"  
else  
   echo "文件为特殊文件"  
fi  
if [ -d $file ]  
then  
   echo "文件是个目录"  
else  
   echo "文件不是个目录"  
fi  
if [ -s $file ]  
then  
   echo "文件不为空"  
else  
   echo "文件为空"  
fi  
if [ -e $file ]  
then  
   echo "文件存在"  
else  
   echo "文件不存在"  
fi

Echo xxx > xxfile

Echo xxx >> xxfile

> 把内容输入到一个文件

>> 把内容追加到一个文件后面

printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg

printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234

printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543

printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876

printf和echo比不就是有个占位符么.

if condition

then

command1

command2

...

commandN

fi

if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi

也可以写成一行,写成一行的话中间就需要有分号了.

a=10

b=20

if [ $a == $b ]

then

echo "a 等于 b"

elif [ $a -gt $b ]

then

echo "a 大于 b"

elif [ $a -lt $b ]

then

echo "a 小于 b"

else

echo "没有符合的条件"

fi

for var in item1 item2 ... itemN

do

command1

command2

...

commandN

done

while condition

do

Command

done

while true

do

Command

done

文件的引用,也就是一个shell文件引用另一个shell文件.

两种方式

. Filename

Source filename

当你引用了别的变量后,他里面的函数啊,变量啊啥的你就可以直接拿来用了

参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255)

demoFun(){

echo "这是我的第一个 shell 函数!"

}

这样定义一个函数

echo "-----函数开始执行-----"

demoFun 这样执行一个函数

echo "-----函数执行完毕-----"

funWithReturn(){

echo "这个函数会对输入的两个数字进行相加运算..."

echo "输入第一个数字: "

read aNum

echo "输入第二个数字: "

read anotherNum

echo "两个数字分别为 $aNum 和 $anotherNum !"

return $(($aNum+$anotherNum))}

funWithReturn

echo "输入的两个数字之和为 $? "

看到了么这个函数,首先里面的read关键字可以用于从控制台获取到输入的内容.

还有就是我们看到了上面那个函数是有返回值的,但是函数的返回值我们去哪里获取呢,

就是在平淡的用函数名调用了这个函数后,就通过$?获取上一个函数的返回值.

还有就是函数的参数,其实你看我们的函数也没有在定义的时候指定参数列表

其实压根就不用指定

funWithParam(){

echo "第一个参数为 $1 !"

echo "第二个参数为 $2 !"

echo "第十个参数为 $10 !"

echo "第十个参数为 ${10} !"

echo "第十一个参数为 ${11} !"

echo "参数总数有 $# 个!"

echo "作为一个字符串输出所有参数 $\* !"

}

funWithParam 1 2 3 4 5 6 7 8 9 34 73

在调用一个函数并指定参数的时候,就这样操作就行

在函数名后面排列好你的参数,

在函数里面用$1 $2 来获取,但是如果是超过10的话,就用

${10} ${11}来获取

$# 这个就会获取参数的总个数.

## AWK

awk -F '\_' '{print $0$1}' shangquan\_info 俩挨着

awk -F '\_' '{print $0"\t"$1}' shangquan\_info \t分隔 双引号

ARGIND==1{

bid = $1;

bad[bid];

}

ARGIND==2{

bid = $1;

if((bid in bad))

next;

print $0;

awk中 $0就是全部

awk -F"\t" 'ARGIND==1{map[$1]} ARGIND==2&&($1 in map){print $0}' guanstatus20230322 tp\_alive\_low\_to\_qingbaoku\_20230321.pv > tp\_alive\_low\_to\_qingbaoku\_20230321.pv.merge &

awk -F '\t' '{if($2=="LOW")print $1}' poi\_class\_info.20230316 > poi.al

## VIM

