## Tip

$( )中放的是命令，相当于` `，例如todaydate=$(date +%Y%m%d)意思是执行date命令，返回执行结果给变量todaydate，也可以写为todaydate=`date +%Y%m%d`；  
${ }中放的是变量，例如echo ${PATH}取PATH变量的值并打印，也可以不加括号比如$PATH

wget --no-check-certificate

## 统计最多的10个url

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

## 根据名字杀进程

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

tr就是字符串替换的

还有就是注意awk要在后面,我们awk和grep正常后面都是跟的文件,但是如果你是用管道的话, 内容是在|前面, grep和awk在侯敏

## Head

## Sort

## uniq

## Netstat

查看端口使用情况, 本机ip和端口, 对端的ip和端口都可以查到

Netstat -an 是基础的

Netstat -ant 只查tcp协议的端口

-anu 只查udp协议的端口

Netstat -ant | grep 3306 也可以通过管道和grep再过滤

## Grep

Grep -r xxx ./ 就是在当面目录的所有文件中查找

grep -r  从文件夹里查找

grep -v 查找反面

grep -c 查看个数。-cv就是查找有多少不包含的

grep -w 这个只能是字符匹配

grep mybankedet xxx.log | grep -v err\_no = 0    这个-v一般都是这么用

grep fullpath=/ev/mine/submit\_exam\_question logs\_74527.txt | fgrep android | fgrep version\_name=1.2.8 | awk -F "user\_id=" '{print $2}' | awk '{a[$1] += 1}END{for (b in a) print b, a[b]}'   | wc -l

先不看后面的awk,就先看你想获取一个包含指定的多个内容的行的grep语句,就是grep 第一个你要获取的内容 | grep 第二个你要获取的内容 | grep 第三个你要获取的内容 这样用管道来操作你要获取的多重循环的东西.

## Awk

awk是一个报告生成器,他拥有强大的文本格式化的能力.

awk是一门编程语言,它支持条件判断,数组,循环等功能,所以我们可以把awk理解成一个脚本语言解释器.

Awk [options] ‘Pattern{Action}’ file

Awk ‘{print}’ xxx 打印文件的内容

这个就是执行了一个打印的动作 ‘{print}’

Awk ‘{print $5}’ xxx

这个命令呢会把每一行的第5列的那个单词给拿出来.

Awk ‘{print $0}’ xxx

$0是awk内置的一个变量,是会打印一整行的.,那么上面这个语句其实就会把整个一行都给打印了.

Awk ‘{print $1 $2 $3}’ xxx

但是只有$0是一整行,而 $1 $2 $3都是其中的某一列的意思. 这样都是会把整个文件的第123列打印. 而$NF是这个意思,每一行的最后一列都是不同的,如果你想打印最后一列的话就需要用到$NF了

Awk ‘{print $1 $NF “我想打印的东西”}’ xxx

这样的话还会把自己想打印的东西当作一列给打印出来,就相当于给拼接上了.

但是注意awk这样打印出来的列都是连在一起的咯

上面我们说了awk的语法是这样的,

Awk [option] ‘Pattern{Action}’ file

上面看到一个最简单的action ‘{print}’

所以上面压根就没有option和pattern的事还,下面说一下pattern

下面先介绍两个pattern BEGIN和END

awk 'BEGIN{print "aaa"}{print $0}' xxx

Awk ‘END{print “aaa”}{print $0}’ xxx

明白啥意思了吧,指的就是操作别的之前先操作下BEGIN里面的,操作完之后再操作END里面的.

那么BEGIN和END就可以用来打印表头和表尾咯

上面我们说了分隔符,默认的分隔符是空格,如果想要指定分隔符呢

Awk -F “#” ‘{print $1}’ xxx

这样就是指定了分隔符,并且以这个分隔符打印第一列的代码.

也就是上面的-F就是我们遇到的第一个上面的标准语法里面的option

除了上面的那个通过-F指定分隔符,还可以用另一个option来指定分隔符.

Awk -v FS=’#’ ‘{print $1}’ xxx

这里我们通过-v这个option来指定变量,FS=‘#’ 就是指定了分隔符

还有就是输出分隔符,刚才我们用awk ‘{print $1 $2}’ xxx 获取的时候我们其实发现了,这两列是挤在一起了,中间没有任何分隔,这个时候我们就需要用到输出分隔符了.

Awk -v OFS=’ ’ ‘{print $1 $2}’ xxx

后我们发现还是$1和$2挤在一起的,分隔符好像没有生效啊,

我们要加一个, 才会让分隔符生效

Awk -v OFS=’ ’ ‘print $1,$2’ xxx 这样就生效了

如果你想同时指定这两个就是

Awk -v FS=’xx’ -v OFS=’xxx’

这两个写两遍.

NF上面已经说了就是最后一列的东西, 可以用$NF来打印最后一列的内容,也可以

‘{print NF}’直接来打印内容.

NR就是行号的意思

Awk ‘{print NR $0}’ xxxx 这个就会把行号和内容一起打印出来.

RS这个变量就是用来修改换行符的,比如正常肯定是根据回车来换行,如果你用RS的话就会用你指定的东西来当作新的一行.

Awk -v RS=’xxx’

FILENAME 这个也是一个内置变量,是用来打印文件名的.

我发现BEGIN还有其他的作用,如果你不用BEGIN的话,只用一个‘{print}’ xxx

你后面这个xxx必须是一个文件名,不能是随便的测试内容.

awk 'BEGIN{print ARGV[0], ARGV[1], ARGV[2], ARGC}' t1 t2

ARGV[] 是个数组,表示后面的哪些参数的数组,在这里ARGC[0]是awk

ARGC[1] 是t1 2就是t2 ARGC就是数组的长度.

自定义变量 awk -v testVar=”test” ‘BEGIN{print testVar}’

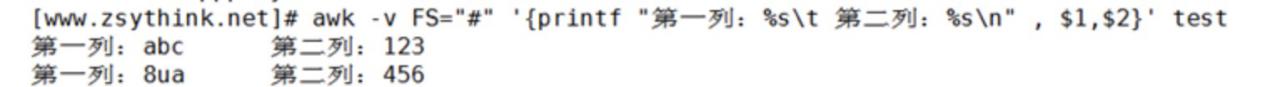
可以通过-v这种来定义,这样的话,你自己定义的变量用起来就和普通的内置变量差不多.

还有一种就是在后面的action里面去定义.

awk 'BEGIN{testv1="1";testv2='2'; print testv1,testv2}' 不过这样的话定义的语句要在中间加上一个分号.

还有就是在这里应该看出来了,所有的action就是一个个的语句啊.

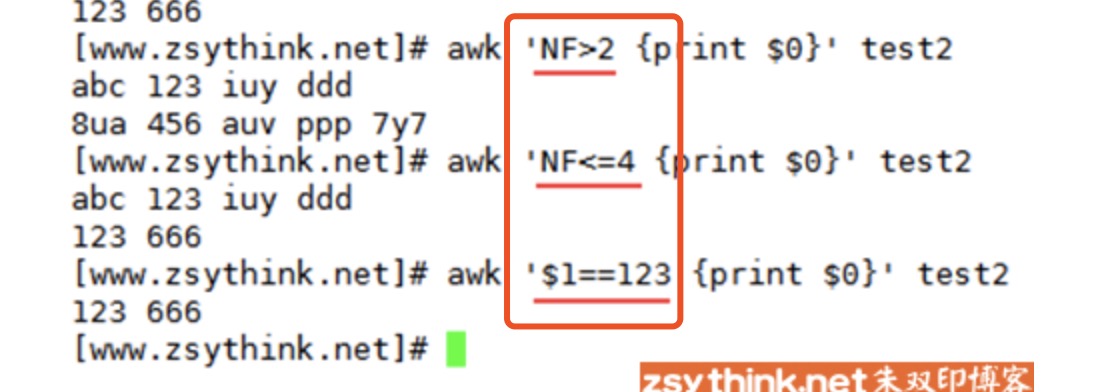
print只能就是啥呼呼的打印,其实还有个printf



加入换行就是/n tab就是/t, 注意中间又个逗号.

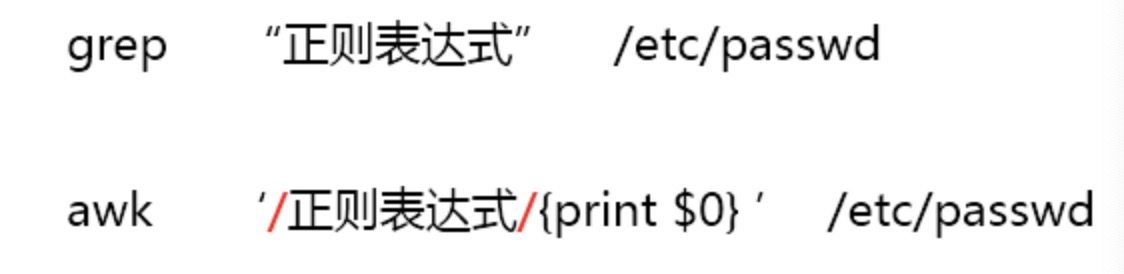
上面我们说了BEGIN和END是两个特殊的模式,那么不特殊的模式是什么样子的呢,

模式这个东西翻译成条件比较好,咋说呢



上面这些就是些条件NF>2的, NF《=4的 $1==123的,只有这些条件满足了我才操作.

上面所说的模式,也就是条件只是这种逻辑运算的条件,还有就是正则的模式.



上面也说了,{}所谓动作就是几行代码而已啊可以用;号来分隔的啊

Awk ‘{ print $1 ; print $2}’ 这样也是可以的

Awk ‘{ if(NR=1) {print $0} }’ test

首先if语句是action里面的,必须要用{包在里面}

然后就是if(){ 这里 } 也必须再写一个大括号包起来

也是有if else else if啥的

Awk ‘{if($3)<500{xxx} else {xxx}}’

Awk ‘{if (xxx){xxx} else if {xxx} else {xxx}}’

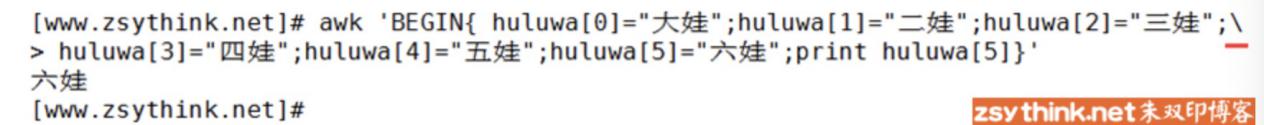
Awk ‘{ if(NR==2){ next } ; print $0}’ test9

next可以直接跳到下一句.

Awk ‘BEGIN{for(i=0;i<6;i++){if(i==3){break};print i }}’

看到了么 当i==3的时候打印i

还有就是普通的语句直接用;分隔就行



是有数组的,可以直接赋值.

数组的元素是可以为空的

比如arrtest是个数组,可以用

If(5 in arrtest) 判断一个元素是否在一个数组中.

也可以!If(5 in arrtest) 来取反.



也不一定非得是下标

Delete arrtest[1] 删除一个元素

Delte arrtest 删除整个数组

遍历数组

For (i in arrtest){print i, arrtest[i]}

打印数组,注意啊,这个i是下标,所以数组内容要根据下标获取下.

当我们直接引用数组中一个不存在的元素,awk就会在数组中创建、这个元素,并且赋值为空

还有就是如果对自已字符串做自加用酸,他居然会被当作0,加完后就编程1了.

grep fullpath=/ev/mine/submit\_exam\_question logs\_74527.txt | fgrep android | fgrep version\_name=1.2.8 | awk -F "user\_id=" '{print $2}' | awk '{a[$1] += 1}END{for (b in a) print b, a[b]}' | wc -l

## Sed

## Curl

## Shell

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

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”。

Echo 后面的内容完全不用加上双引号

shell主要是要有个文本编辑器来写,然后用个解释器来解释就行.

我们现在的shell一般都是bash,这个就是个程序,可以用来解释我的shell文本.

#!/bin/bash

这样就指定了

这个要以 # 开头 后面跟个!

定义变量

TESTVAR=”sdd” 注意就是中间不能有空格

TESTV2=122

使用变量

Your\_name=”1111”

Echo $your\_name

Echo ${your\_name}

反正shell变量的获取使用$来获取到的.

也可以用语句里面获取变量

For file in ‘ls/etc’

For file in $(ls/etc)

上面的语句会把这个目录下的文件名给循环出来

#!/bin/bash

myUrl="http://www.google.com"

readonly myUrl 设置为只读变量,下次在想修改就会报错

myUrl="http://www.runoob.com"

shell的变量类型

局部变量 局部变量在脚本或者命令中定义,仅在当前shell实例中生效,其他shell程序是看不到这个变量的.

环境变量 这个是和linux相关的 所有的程序,包括shell程序所共有的,只是操作系统的东西.

可以用echo $shell 这样来查看一个操作系统的变量.

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各书

$$ 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.