Shell正则表达式

# 正则表达式概念

正则表达式用来在文件中匹配符合条件的字符串，正则包含匹配。支持正则表达式的命令有grep,egrep,awk,sed。与正则表达式不同的是，通配符用来匹配条件的文件名，通配符是完全匹配，ls,find,cp这些命令不支持正则表达式。

## 基础正则表达式

基础正则表达式：\* . ^ $ [-] [^] \ \{n\} \{n,\} \{n,m\}

|  |
| --- |
| .：匹配任意一个非换行字符  \*：匹配前一个字符或者分组出现0次或者多次  ^：匹配开头的字符  $：匹配结尾的字符  [ ]：匹配方括号中出现的任一字符  [-]：[0-9]匹配一个数字，[a-z]匹配所有小写字母  [ ^]：不匹配方括号中出现的任一字符  \{n\} \{n,\} \{n,m\}：匹配重复前一个字符或者分组的次数  \b：匹配单词的边界\bhello\b（匹配hello）  \w：匹配字母数字和下划线（\W相反）  \<word\>：定界匹配单词  \：元字符的使用需要转义，例如\\,\[，\{，\.，\\*，\^等  转义空白字符：\n换行，\r回车，\t制表符，\f换页，\s任何空白字符，\S任何非空白字符 |

注意，支持的命令有：grep,egrep,vi,sed,awk。

## 扩展正则表达式

扩展正则表达式：grep -E 等效于egrep :+ ? | ()

|  |
| --- |
| +：匹配前一个字符或者分组出现1次或者多次  ？：匹配前一个字符或者分组出现0次或者1次  |：或者的意思(并集)  ()：表示一个整体的分组 |

支持的命令有：egrep,awk

使用：egrep -v '^$|^#' nginx.conf

## POSIX规范



## GUN规范

在linux中的GUN中还会支持转义元字符。这些转义元字符包括\b(边界字符)，用来匹配单词的开始和结束。

## 文件通配符

|  |
| --- |
| \*：0个或者读个任意字符  ？:任意一个字符  {}：拥有匹配所有括号内包含的以逗号隔开的字符。{a,b,c}  [^!]：表示取反，[^A]代表不是A |

# 正则表达式常用命令

## Sed命令

Linux中sed命令主要用于文本的处理，可处理文本的行和列。

### 基本语法

sed命令行格式为：　sed [-nefri ] ‘command’ 输入文本

常用选项：

*  -n∶使用安静（silent）模式，显示sed处理过的行。
*  -e∶直接在指令列模式上进行 sed 的动作编辑，默认选项。
*  -f∶直接将 sed 的动作写在一个档案内， -f filename 则可以执行 filename 内的sed 动作；
*  -r∶sed 的动作支持扩展正则表示法。（预设是基础正则表示法语法）
*  -i∶直接修改读取的档案内容，而不是输出。（修改文件内容）

command：地址匹配操作指令

操作地址：

* num：指定行号
* num1,num2：指定num1-num2之间所有行($表示最后一行的)
* num,+n：指定num行及它后面n行
* num~step：指定开始操作行，和操作步长。
* /regexp/：正则表达式匹配的行。
* \cregexpc：匹配扩展的正则表达式行。

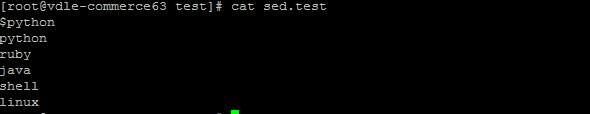
常用指令：

* a ∶追加， a 的后面可以接字串，而这些字串会在新的一行出现（目前的下一行）～
* c ∶取代， c 的后面可以接字串，这些字串可以取代 n1，n2 之间的行！
* d ∶删除行，可以删除指定n1,n2之间的行。
* i ∶插入， i 的后面可以接字串，而这些字串会在新的一行出现（目前的上一行）；
* p ∶列印，亦即将某个选择的资料印出。通常 p 会与参数 sed -n 一起运作
* y：按字符转换。
* r：读入文件内容
* s ∶替换，可以直接进行取代的工作！通常这个 s 的动作可以搭配正则表示法！例如 1，20s/old/new/g 就是啦！

替换的格式：[addr]s/pattern/replace/flag。常用flag:

* n：替换第n次匹配
* g：默认只替换第一个，设置g进行全局替换。
* p：打印模式空间的内容
* w file：将模式空间的内容写入到文件
* &：用正则表达式匹配的内容进行匹配
* /n：匹配第n个子串
* \：转义

常见应用(sed.test)



***删除某行***

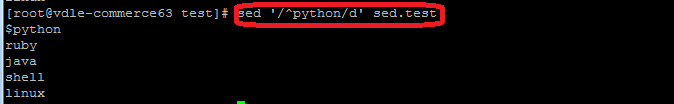
　 sed ‘1d’ sed.test #删除第一行

　 sed ‘$d’ sed.test #删除最后一行

　 sed ‘1，2d’ sed.test #删除第一行到第二行

sed ‘2，$d’ sed.test #删除第二行到最后一行

sed '/^python/d' sed.test#删除匹配python开头的行



***显示某行***

sed -n '1p' sed.test #显示第一行

sed -n '$p' sed.test #显示最后一行

sed -n '1,2p' sed.test #显示第一行到第二行

sed -n '2,$p' sed.test #显示第二行到最后一行

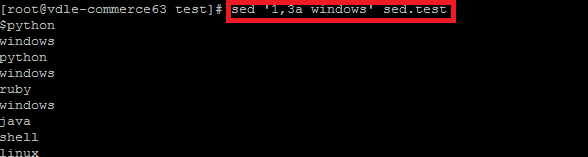
sed -n '/ruby/p' sed.test #查询包括关键字ruby所在所有行

sed -n '/\$/p' sed.test #查询包括关键字$的行，用反斜线\转义

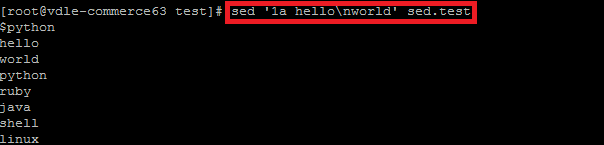


***追加一行或者多行（使用i表示行前添加，a表示行后添加）***

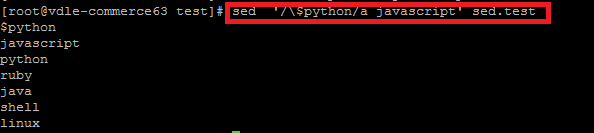
sed '1,3a hello\nworld' sed.test #在第一行到第三行后面追加windows



sed '1a hello\nworld' sed.test #第一行后添加多行

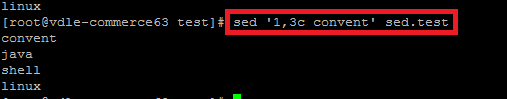


sed '/\$python/a javascript' sed.test #匹配到得行追加



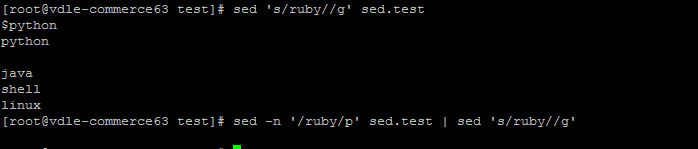
***替换行或者行内字符串***

sed '1,3c convent' sed.test #把第一行到第三行整体换成convent



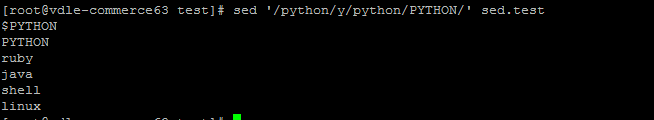
###替换行中某部分字符：sed ‘s/要替换的字符串/新的字符串/g’

###删除行中某部分字符：sed ‘s/要替换的字符串/ /g’



***转换字符指令***

[addr]y/source-chars/dest-chars/



### sed脚本指令

sed.sh文件的书写：

/匹配要操作的行(正则表达式)/{

#要执行的动作

}

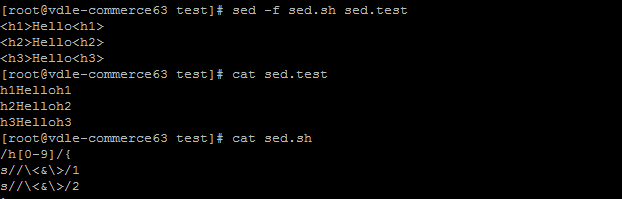
例如：

#删除所有的空行

/.\*/{

d

}



### 常见应用

|  |
| --- |
| --删除行首空格  sed 's/^ \*//g' filename  --在行前行后添加新行  sed 's/pattern/&＼n/g' filename  sed 's/pattern/＼n&/g' filename  --变量替换使用“”  sed -e "s/$var1/$var2/g" filename  --在第一行前插入文本  sed -i '1 i＼插入字符串' filename  --在最后一行插入  sed -i '$ a＼ 插入字符串' filename  --在匹配行前插入  sed -i '/pattern/ i "插入字符串"' filename  --在匹配行后插入  sed -i '/pattern/ a "插入字符串"' filename  --删除文本中的空行和空格组成的行和#号注释的行  grep -v ^# filename | sed /^［［:space:］］\*$/d | sed /^$/d  --文件的匹配替换字符字符串，字符中包含s/可以使用s@替代  sed -i "s@BASEDIR=/usr/local@BASEDIR=/usr/local/zabbix@g"  --目录下所有文件匹配替换字符串,/modules目录记得备份  sed -i 's/zhangsan/list/g' 'grep zhangsan -rl /modules' |

## Awk命令

awk的工作流程：逐行扫描文件，从第一行到最后一行，寻找到匹配特定模式的行，并在这些行上进行用户想要的操作。awk基本结构有模式匹配和处理过程(处理动作)组成。awk有两个特殊的模式：BEGIN和END，它们被放置在没有读取任何数据之前以及在所有数据读取完成以后执行。

BEGIN模式

动作指令

/模式/{动作}。模式匹配要操作的行，逐行读取匹配的行，并应用的动作指令。

模式

END模式

动作指令

### 基本语法格式

awk [选项] –f program\_file [--] file …

常用选项：

* -F fs：指定以fs作为行的分隔符（默认分隔符是空格或制表符）
* -v var=val:在处理过程以前，设置一个变量var的值为val。
* -f program-file:从脚本文件中读取awk指令。

### 操作指令

***记录与字段***

awk执行时，其浏览标记为$1，$2...$n，这种方法称为域标记。使用$1，$3表示参照第1和第3域，注意这里使用逗号分隔域，使用$0表示使用所有域

***字段分隔符***

Awk默认的字段分隔符是空格或者tab键。可以通过-F 来改变指定分隔符。

***数据类型***

Awk仅有两个主要的数据类型：字符串和数字。即便如此，Awk的字符串和数字还可以相互转换。字符串能够被解释为数字并把它的值转换为数字值。如果字符串不包含数字，它就被转换为0.它们都可以在你代码里的ACTIONS部分使用 = 操作符给变量赋值。我们可以在任意时刻、任意地方声明和使用变量，也可以使用未初始化的变量，此时它们的默认值是空字符串：“”。

最后，Awk有数组类型，并且它们是动态的一维关联数组。它们的语法是这样的：var[key] = value 。Awk可以[模拟多维数组](http://www.staff.science.uu.nl/~oostr102/docs/nawk/nawk_87.html#SEC90)，但无论怎样，这是一个大的技巧（big hack）。

定义数组

* Tarray[1]=“cheng mo”
* Tarray[2]=“800927”
* Tarray[“first”]=“cheng ”
* Tarray[“last”]=”mo”
* Tarray[“birth”]=”800927”

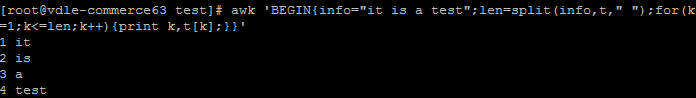
获取数组长度：length(数组名)

输出数组：

###用for in语句输出的东西是无序的

awk 'BEGIN{info="it is a test";split(info,t," ");for(k in t){print k,t[k];}}'

awk 'BEGIN{info="it is a test";len=split(info,t," ");for(k=1;k<=len;k++){print k,t[k];}}'



判断键值存在：元素 in 数组

awk 'BEGIN{t["a"]="a";t["b"]="b";if( "c" in t){print "ok";};for(k in t){print k,t[k];}}' //正确的判断方式，"c" in tB

删除键值：//delete array[key]可以删除，对应数组key的，序列值。



***内置变量***

* ARGC:命令行参数个数
* ARGV：命令行参数排列
* ENVIRON:支持队列中系统环境变量的使用
* FILENAME：awk浏览的文件名
* FNR:浏览文件的记录数
* FS：设置输入域分隔符，等价于命令行 -F选项
* NF:浏览记录的域的个数，可以表示当前行总列数
* NR：已读的记录数，可以表示总行数
* OFS:输出域分隔符
* ORS:输出记录分隔符
* RS：控制记录分隔符

***表达式和操作符***

表达式由变量、常量、函数、正则表达式、操作符组成。操作符的种类有：

* + - \* / % ^ (幂运算) ++ --
* += -= \*= /=
* < > = <= == !==
* ~：匹配 !~：不匹配
* && ||
* 特殊操作符：$(字段引用)，空格(字符串连接),?:(条件表达式)，in(判断数组中是否存在某键值。

### 程序结构

If语句（选择结构）

|  |
| --- |
| if(表达式){  语句1  }else if(表达式){  语句2  }else{  语句3  } |

循环语句(类似于c语法)

|  |
| --- |
| while(表达式){  语句  }  for(变量 in 数组){  语句  }  for(变量;条件;表达式){  语句  }  do{  语句  }while(条件) |

跳转语句

* break：跳出循环
* continue：跳出本次循环

### 常用函数

***算术函数***

* rand( )：返回任意数字 n，其中 0 <= n < 1。

***字符串函数***

* gsub( Ere, Repl, [ In ] ) :除了正则表达式所有具体值被替代这点，它和 sub 函数完全一样地执行，。
* sub( Ere, Repl, [ In ] )：Ln来匹配Ere,将匹配的内容替换成Repl。ln参数缺省值是$0（整个记录）。sub 函数返回替换的数量。
* index( String1, String2 )：String1 参数指定的字符串是否包含String2，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。
* length [(String)]：返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
* substr( String, M, [ N ] )：N 参数:指定返回字符数量,不指定表示到字符串尾部；String：指定字符串；N:指定开始截取的位置，从一开始数起.
* match( String, Ere )：在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH 特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为 -1（负一）。
* split( String, A, [Ere] ) ：String表示要分割字符串，A表示切割后返回的数组。Ere 指定的正则表达式，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere 参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数组中的元素用字符串值来创建。
* tolower( String )：返回 String 参数指定的字符串，字符串中每个大写字符将更改为小写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。
* toupper( String )：返回 String 参数指定的字符串，字符串中每个小写字符将更改为大写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。
* sprintf(Format, Expr, Expr, . . . )：根据 Format 参数指定的 printf 子例程格式字符串来格式化 Expr 参数指定的表达式并返回最后生成的字符串。

例如：awk 'BEGIN{info="this is a test2010test!";print substr(info,4,10);}'

***格式化字符串输出(printf)***

* %d：十进制有符号整数
* %u：十进制无符号整数
* %f ：浮点数
* %s ：字符串
* %c ：单个字符
* %p：指针的值
* %e ：指数形式的浮点数
* %x %X ：无符号以十六进制表示的整数
* %o：无符号以八进制表示的整数
* %g：自动选择合适的表示法

awk 'BEGIN{n1=124.113;n2=-1.224;n3=1.2345; printf("%.2f,%.2u,%.2g,%X,%o\n",n1,n2,n3,n1,n1);}'

***时间函数***

* mktime( YYYY MM DD HH MM SS[ DST]) 生成时间格式
* strftime([format [, timestamp]]) 格式化时间输出，将时间戳转为时间字符串
* systime() 得到时间戳,返回从1970年1月1日开始到当前时间(不计闰年)的整秒数

例如：awk 'BEGIN{tstamp=mktime("2001 01 01 12 12 12");print strftime("%c",tstamp);}'

***其他函数***

* getline [ Variable ] < Expression：从 Expression 参数指定的文件读取输入的下一个记录，并将 Variable 参数指定的变量设置为该记录的值。只要流保留打开且 Expression 参数对同一个字符串求值，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。
* getline [ Variable ]：将 Variable 参数指定的变量设置为从当前输入文件读取的下一个输入记录。如果未指定 Variable 参数，则 $0 记录变量设置为该记录的值，还将设置 NF、NR 和 FNR 特殊变量。
* close( Expression )：用同一个带字符串值的 Expression 参数来关闭由 print 或 printf 语句打开的或调用 getline 函数打开的文件或管道。如果文件或管道成功关闭，则返回 0；其它情况下返回非零值。如果打算写一个文件，并稍后在同一个程序中读取文件，则 close 语句是必需的。

例如：awk 'BEGIN{while(getline < "/etc/passwd"){print $0;};close("/etc/passwd");}'

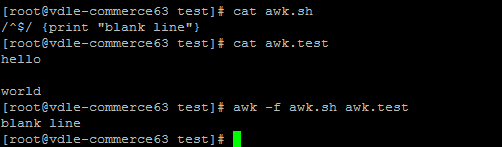
awk '/.\*root.\*/{print $0}' /etc/passwd



### 使用awk脚本

脚本编写：/模式/{ 动作 }

脚本的调用：awk –f xxx.sh 操作的文件



### 常见应用

|  |
| --- |
| --文本行数  cat /etc/passwd | awk -F ":" '{print NR}' | tail -1  --文本各行的列数  cat /etc/passwd | awk -F ":" '{print NF}'  --打印固定域，最后一列  cat /etc/passwd | awk -F ":" '{print $NF}'  --匹配某行，并对域进行字符函数应用  cat /etc/passwd | awk -F ":" '{print substr($NF,1,5)}'  cat /etc/passwd | awk -F ":" '{print length($0)}' |