正则表达式

作者：张亚

归档：学习笔记

2021/04/13

快捷键：

Ctrl + 1	标题 1
Ctrl + 2	标题 2
Ctrl + 3	标题 3
Ctrl + 4	实例
Ctrl + 5	程序代码
Ctrl + 6	正文

格式说明：

蓝色字体：注释

黄色背景：重要

绿色背景：注意

老男孩教育教学核心思想 6 重：重目标、重思路、重方法、重实践、重习惯、重总结

学无止境，老男孩教育成就你人生的起点!

联系方式：

网站运维 QQ 交流群：

Linux 385168604
Python 29215534
架构师 390642196
大数据 421358633

官方网站:

http://www.oldboyedu.com

目 录

第 1 章 第一章什么是正则表达式

（1）正则表达式就是为了处理大量的字符串而定义的一套规则和方法。

（2）通过定义的这些特殊符号的辅助，系统管理员就可以快速过滤，替换或输出需要的字符串。Linux 正则表达式一般以行为单位处理的。

简单说：

为处理大量字符串而定义的一套规则和方法

以行为单位处理

第 2 章 第二章 为何使用正则表达式

Linux 运维工作 大量过滤日至工作。化繁为简

简单，高效

高级工具：三剑客 都支持

第 3 章 第三章学习正则表达式容易混淆的两个注意事项

a.正则表达式应用非常广泛，存在于各种语言中，php perl python grep sed awk 支持。ls* 通配符。

b.但现在学的是 Linux 中的正则表达式，最常应用正则表达式的命令是 grep（egrep）、sed、awk --》正则表达式通常只有 awk、sed、grep、egrep 能使用

c.正则表达式和通配符有本质区别

1）通配符例子 ls file *.log filef.log grep "e*" e ee eeee ef

第 4 章 第四章正则表达式使用注意事项

（1）Linux 正则表达式以行为单位处理字符串的

（2）便于区别过滤出来的字符串 一定配合 grep/egrep 命令学习

alias egrep='egrep --color=auto'

alias grep='grep --color=auto'

（3）注意字符集，LC_ALL=C：无论何时，做任何事都要注意字符集。

第 5 章 第五章 正则表达式的分类

POSIX 规范将正则表达式分为了两种

（1）基本正则表达式（BRE，basic regular expression）

（2）扩展正则表达式（ERE，extend regular expression）

BRE 和 ERE 的区别仅仅是元字符的不同

BRE 只承认的元字符有^ $. [ ] * 其他字符识别为普通字符 ； \(\)

ERE 则添加了 () {} ? +| 等

只有在用反斜杠“\”进行转义的情况下，字符 () {} 才会被 BRE 被当作元字符处理，而 ERE 中，任何元符号前面加上反斜杠反而会使其被当作普通字符来处理。

第 6 章 第六章 如何区分通配符和正则表达式

1.不需要思考的判断方式：在三剑客 awk\sed\grep egrep 都是正则，其他都是通配符

2.区分通配符和正则表达式最简单的方法：

a）表达式是文件目录名--》通配符

b）表达式是文件内容--》正则表达式

3.通配符和正则表达式都有“ *” "?" "、" "[]"，但是通配符的这些符号都能自身代表任意字符，而正则表达式的这些符号只能代表这些符号前面的字符。

第 7 章 第七章 基本正则表达式

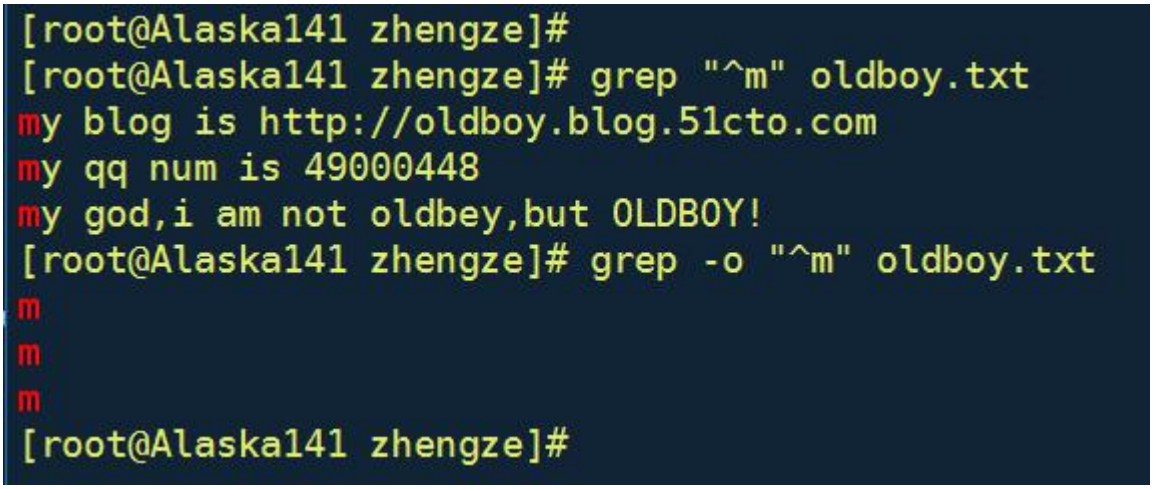
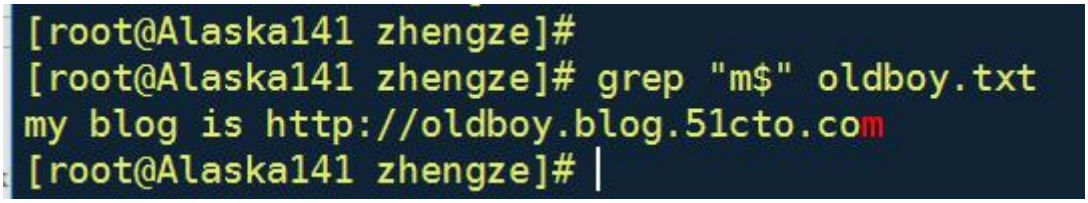
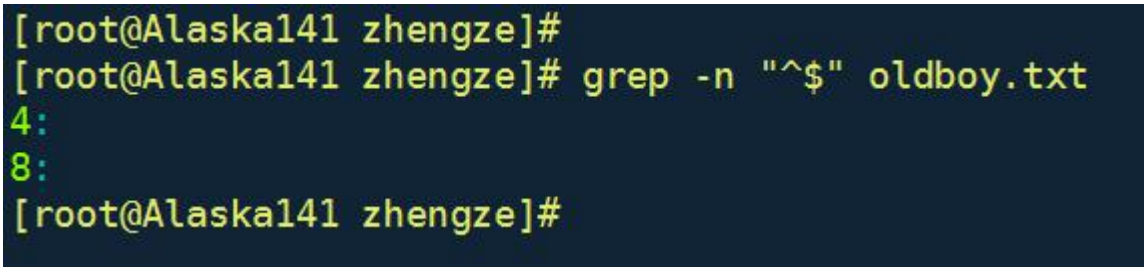
测试文本：

```
[root@Alaska141 zhengze]# cat oldboy.txt
I am oldboy teacher!
I teach linux.

I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448

not 4900000448
my god,i am not oldbey,but OLDBOY!
```

举例实验

字符	描述
^尖角号	^word 搜索以 word 开头的内容 
$	word$搜索以 word 结尾的内容 
^$	表示空行 

.

代表且只能代表任意一个字符

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "." oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448
not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "oldb.y" oldboy.txt
I am oldboy teacher!
my blog is http://oldboy.blog.51cto.com
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

匹配任意字符结尾

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep ".$" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448
not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

只匹配 . 结尾需要使用转义字符\

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "\.$" oldboy.txt
I teach linux.
[root@Alaska141 zhengze]#
```

<table>
<tr><td>\转义字符</td><td>转义字符，让有特殊含义的字符脱掉马甲，现出原形，如\.只表示小数点.
\n 匹配一个换行符
\b 单词边界,\bcool\b 匹配 cool
\r 匹配回车
\t 匹配一个横向制表符 tab</td></tr>
<tr><td>*</td><td>重复之前的字符 0 个或多个</td></tr>
<tr><td>.*</td><td>匹配所有字符。^.*以任意多个字符开头.*$以任意多个字符结尾
.*可以匹配 0 或多个，可以匹配空行，而 . 匹配 1 个，不能匹配空行
<pre>[root@Alaska141 zhengze]# grep ".*" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!

my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448

not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#</pre>
一次匹配任意一个字符,所以全部.*匹配</td></tr>
<tr><td>[abc]
[0-9]
[\.,/]</td><td>匹配字符集合内的任意字符 a 或 b 或 c；[a-z]匹配所有小写字母
#匹配所有小写字母 abc 的字符
<pre>[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "[abc]" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#</pre>
#匹配所有小写字母
<pre>[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "[a-z]" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448
not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#</pre>
#匹配 0 个或任意个字符</td></tr>
</table>

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "0*" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!

my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448

not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

#尽可能匹配多的 0

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep -o "0*" oldboy.txt
000
00000
[root@Alaska141 zhengze]#
```

#所有行最后一个 o 前面全标红

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep ".*o" oldboy.txt
I am oldboy teacher!
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

[^abc]

匹配不包含^后的任意字符 a 或 b 或 c， 是对[abc],且与^含义不同

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "[^abc]" oldboy.txt
I am oldboy teacher!
I teach linux.
I like badminton ball ,billiar ball and chinese chess!
my blog is http://oldboy.blog.51cto.com
our site is http://www.etiantian.org
my qq num is 49000448
not 4900000448
my god,i am not oldbey,but OLDBOY!
[root@Alaska141 zhengze]#
```

a\{1,3\}

重复前面 a 字符 n 到 m 次,如果用 egrep 或 sed -r 可去掉斜线

#匹配前 3 个 0,又再次匹配 2 个 0

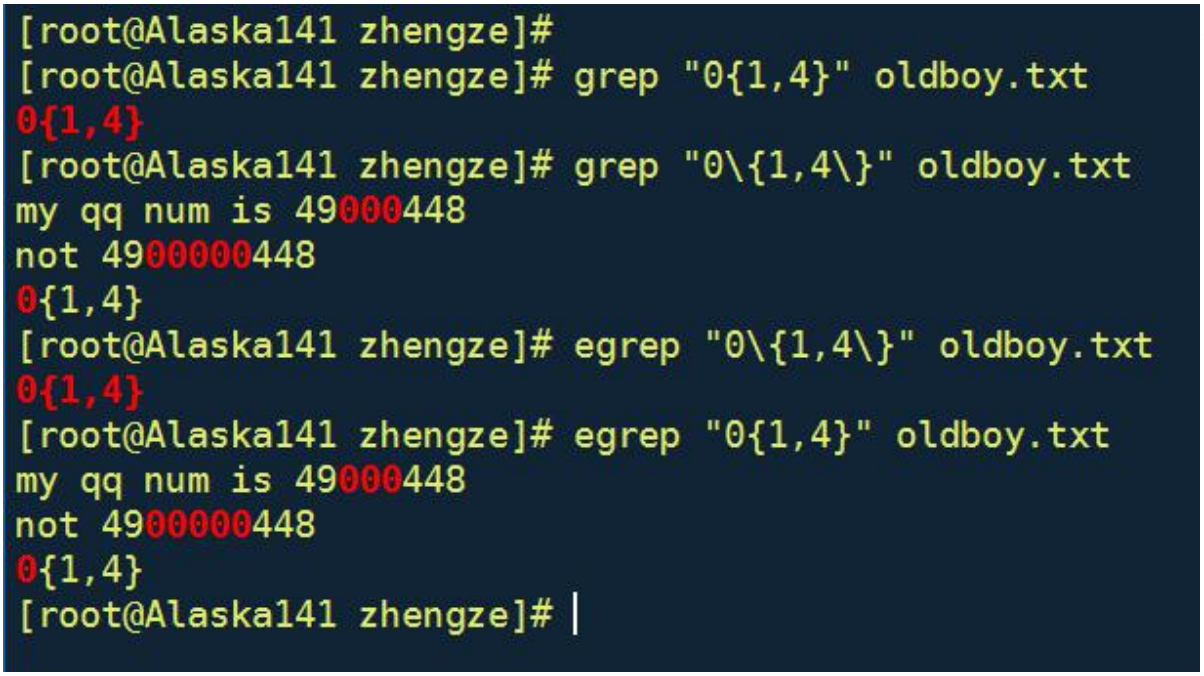
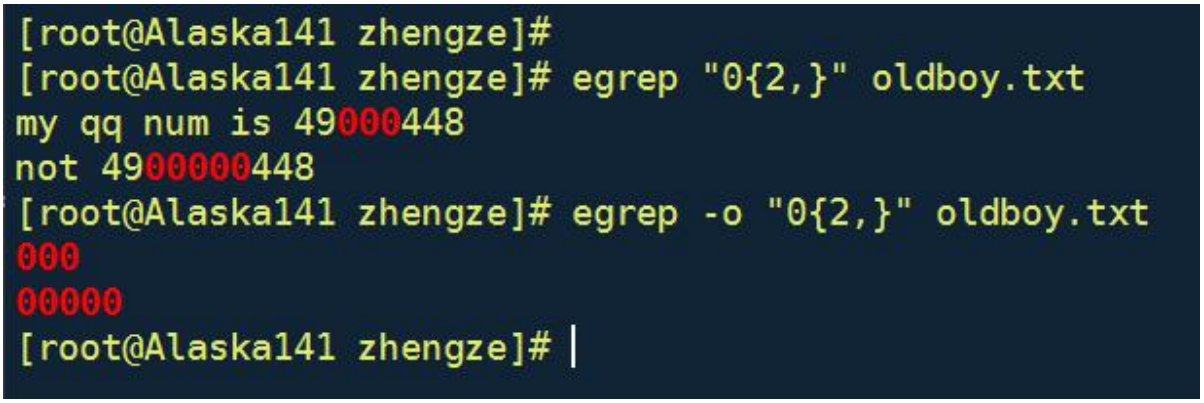
```
[root@Alaska141 zhengze]# grep "0\{1,3\}" oldboy.txt
my qq num is 49000448
not 4900000448
[root@Alaska141 zhengze]# grep -o "0\{1,3\}" oldboy.txt
000
000
00
```

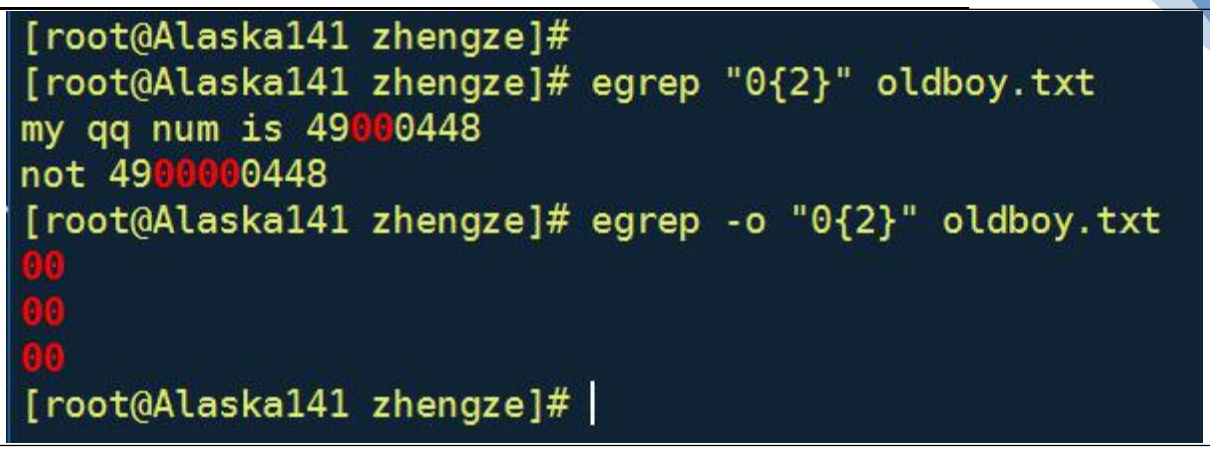
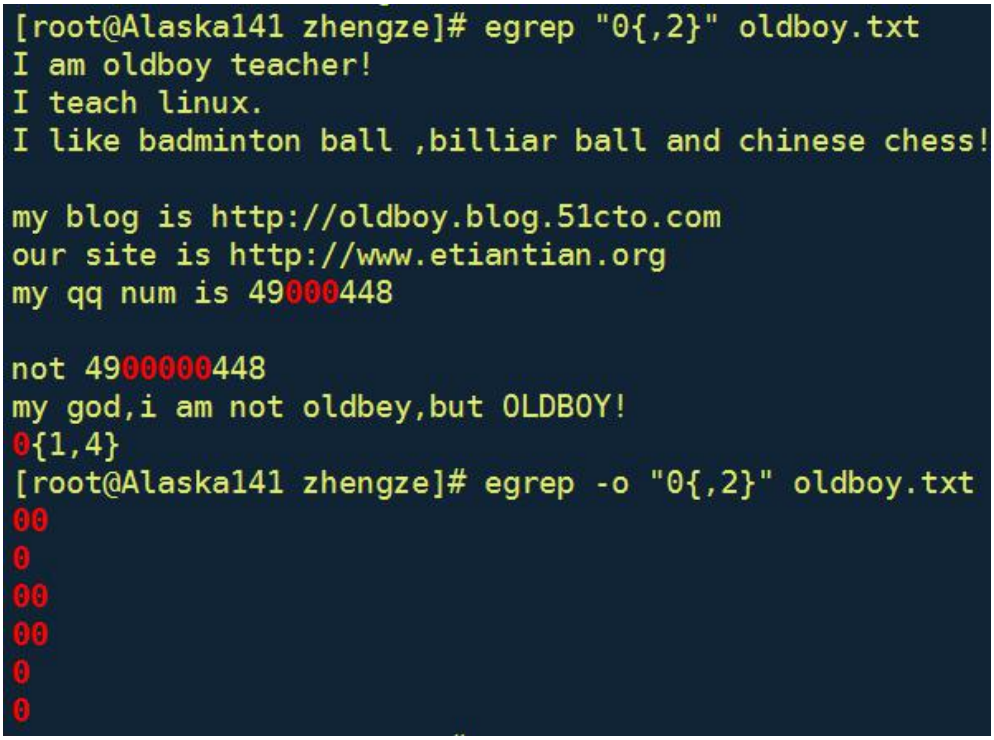
因为前面匹配的 0 会变成最后的 0,所以会一直匹配到不是 0 的字符,也就是匹配所有 0

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# grep "0\{2,4\}" oldboy.txt
my qq num is 49000448
not 4900000448
[root@Alaska141 zhengze]# grep -o "0\{2,4\}" oldboy.txt
000
0000
[root@Alaska141 zhengze]# grep "0\{1,4\}" oldboy.txt
my qq num is 49000448
not 4900000448
[root@Alaska141 zhengze]# grep -o "0\{1,4\}" oldboy.txt
000
0000
0
[root@Alaska141 zhengze]#
```

#因为{2,4}最少匹配 2 次,最多匹配 4 次,所以第一次匹配 3 次满足,第二次匹配 4 次后还剩 1 个 0,但是不满足最少匹配 2 个 0,所以就不会被匹配了.

#如果是{1,4}那么第一次匹配 4 个 0 满足,还剩 1 个 0,也满足最少 1 个 0 的要求,所以 5

	个 0 都会被匹配出来.  #grep 默认不支持{}符号,只会把{}当作字符{}来匹配,所以要使用\使{}变成有意义的符号, #而 egrep 默认是识别{}的作用的,此时再使用\{\}反而会使{}失去效果,变成至匹配{}字符串了.
a\{n,\}	重复前面 a 字符至少 n 次,如果用 egrep 或 sed -r 可去掉斜线 n>=2 
a\{n\}	重复前面 a 字符 n 次,如果用 egrep 或 sed -r 可去掉斜线

	
a\{,m\}	最多 m 次,前一个重复的字符.如果有 egrep 和 sed -r 可以去掉斜线, ###0{,2}因为最多匹配是 2 次,但是最少匹配是 0 次,所以全部都会输出 

第 8 章 第八章 拓展正则表达式 ERE

特殊字符	含义与例子
+	重复前一个或一次以上 (*是 0 个或多个) [a-z]实际至匹配一个字符 #0+ 匹配数字 0 一次或一次以上 [root@Alaska141 zhengze]# [root@Alaska141 zhengze]# egrep "0+" oldboy.txt my qq num is 49000448 not 4900000448 0{1,4} [root@Alaska141 zhengze]# egrep -o "0+" oldboy.txt 000 00000 0 [root@Alaska141 zhengze]# #[0-9]匹配 0 到 9 一次或一次以上 [root@Alaska141 zhengze]# egrep "[0-9]+" oldboy.txt my blog is http://oldboy.blog.51cto.com my qq num is 49000448 not 4900000448 0{1,4} [root@Alaska141 zhengze]# egrep -o "[0-9]+" oldboy.t 51 49000448 4900000448 0 1 4

#o+重复 1 次或 1 次以上的 o

```
[root@Alaska141 zhengze]# egrep "o+" a.log
good
god
goood
[root@Alaska141 zhengze]# egrep -o "o+" a.log
oo
o
ooo
[root@Alaska141 zhengze]#
```

#o*表示重复 0 次或 0 次以上的 0,所以不匹配的行也会被显示出来

```
[root@Alaska141 zhengze]# egrep  "o*" a.log
good
gd
god
goood
[root@Alaska141 zhengze]# egrep  -o "o*" a.log
oo
o
ooo
[root@Alaska141 zhengze]#
```

#go+d 表示匹配 1 次或 1 次以上的

```
[root@Alaska141 zhengze]# egrep  "go+d" a.log
good
god
goood
[root@Alaska141 zhengze]# egrep  "go*d" a.log
good
gd
god
goood
[root@Alaska141 zhengze]#
```

|

```
[root@Alaska141 zhengze]#
[root@Alaska141 zhengze]# dumpe2fs /dev/sda1|egrep -i "^inode size|^block s
ize|^inode count|^block count"
dumpe2fs 1.41.12 (17-May-2010)
Inode count:              51200
Block count:              204800
Block size:               1024
Inode size:               128
[root@Alaska141 zhengze]#
```

#( )与|搭配使用 dumpe2fs /dev/sda1|egrep -i "^(inode|block) (size|count)"

```
[root@Alaska141 zhengze]# dumpe2fs /dev/sda1|egrep -i "^(inode|block) (size
|count)"
dumpe2fs 1.41.12 (17-May-2010)
Inode count:              51200
Block count:              204800
Block size:               1024
Inode size:               128
```

( )

分组过滤被括起来的东西表示一个整体(一个字符),后向引用.

#o+表示匹配一个或一个以上 o

```
[root@Alaska141 zhengze]# cat kuohao.txt
good
glad
gd
god
goood
[root@Alaska141 zhengze]# egrep "g(la|o)d" kuohao.txt
glad
god
[root@Alaska141 zhengze]# egrep "g(la|o+)d" kuohao.txt
good
glad
god
goood
```

#( )与|搭配使用 dumpe2fs /dev/sda1|egrep -i "^(inode|block) (size|count)"

```
[root@Alaska141 zhengze]# dumpe2fs /dev/sda1|egrep -i "^(inode|block) (size
|count)"
dumpe2fs 1.41.12 (17-May-2010)
Inode count:              51200
Block count:              204800
Block size:               1024
Inode size:               128
```

正则表达式小结:

基础正则:BRE

^ $. * .* [abc] [^abc] \{n,m\}

拓展正则:ERE

+ | ? () {} a{n,m} a{n} a{n,} \.$ ===>脱去马甲 贬为平民

注意:AWK 默认支持拓展正则

元字符

T 元字符(mate character)是一种 Perl 风格的正则表达式,只有一部份文本处理工具支持它,并不是所有的文本处理工具都支持.

正则表达式	描述	示例
\b	单词边界	\bcool\b 匹配 cool,不匹配 coolant
\B	非单词边界	cool\B 匹配 coolant,不匹配 cool
\d	单个数字字符	b\db 匹配吧 b2b,不匹配 bcb
\D	单个非数字字符	b\Db 匹配 bcb,不匹配 b2b
\w	单个单词字符(字母,数字与_)	\w 匹配 1 或 a,不匹配&
\n	换行符	\n 匹配一个新行
\s	单个空白字符	x\sx 匹配 x x,不匹配 xx
\S	单个非空白字符	x\S\x 匹配 xkx ,不匹配 xx
\r	回车	\r 匹配回车
\t	横向制表符	\t 匹配一个横向制表符
\v	垂直制表符	\v 匹配一个垂直制表符
\f	换页符	\f 匹配一个换页符

第 9 章

1.egrep/grep 了解

2.egrep/grep -o 参数看正则到底匹配了什么

3.根据多练就好.配合 grep,egrep,sed -r ,awk 更为强大

4.sedline 是个好东西,每天看一点

第 10 章 参考资料

man grep info grep man 7 glob

linux 正则表达式语法

http://aresxin.blog.51cto.com/4734097/1602624

正则表达式 30 分钟入门教程

http://deerchao.net/tutorials/regex/regex.htm#mission

第 11 章 问题小结

2.1 如何取得文件的数字权限

```
1.sed       正则
2.awk       分隔符
3.grep
4.cut

1.sed 方法
[root@Alaska141 nettest]# stat sed.txt |sed -n '4p'
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[root@Alaska141 nettest]# stat sed.txt |sed -n '4p'|sed 's#^.*(0##g'
644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
[root@Alaska141 nettest]# stat sed.txt |sed -n '4p'|sed 's#^.*(0##g'|sed 's#/.*$##g'
644
###优化显示行 4p
[root@Alaska141 nettest]# stat sed.txt |sed -n '4s#^.*(0##gp'|sed 's#/.*$##g'
644
###放大招:正则
[root@Alaska141 nettest]# stat sed.txt|sed -rn '4s#^.*\(0(.*)/-.*$#\1#gp'
644
```

2.awk 方法：

```
[root@Alaska141 ~]# stat 333.txt
  File: `333.txt'
  Size: 461             Blocks: 8          IO Block: 4096   regular file
Device: 803h/2051dInode: 280024    Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2017-01-16 11:24:28.082998897 +0800
Modify: 2017-01-13 18:31:22.819988920 +0800
Change: 2017-01-13 18:31:22.819988920 +0800

[root@Alaska141 ~]# stat 333.txt |awk -F "[(0/]" 'NR==4{print $3}'
644
###优化后
[root@Alaska141 ~]# stat 333.txt |awk -F "[0/]" 'NR==4{print $2}'
644
###第三种
[root@Alaska141 ~]# stat 333.txt |awk -F "0|/" 'NR==4{print $2}'
644
```

3.cut 方法：

```
stat 333.txt |sed -n '4p' |cut -c11-13
stat 333.txt |sed -n '4p' |cut -d "0" |cut -d "/"
```

4.stat 方法：

```
stat -c %a 333.txt
```

2.2 将文件中的字符权限转换成数字表示法:

```
[root@Alaska141 nettest]# ls -l |awk '{print $1}'|tr 'rwx-' '4210'
total
0420400400
d421401401
[root@Alaska141 nettest]# ls -l /etc/hosts|cut -c2-10|tr 'rwx-' '4210'|awk -F "" '{print $1+$2+$3 $4+$5+$6 $7+$8+$9}'
644
[root@Alaska141 nettest]# ls -l /etc/hosts|cut -c2-10|tr 'rwx-' '4210'|awk -F ""
'{s1=$1+$2+$3;s2=$4+$5+$6;s3=$7+$8+$9;print s1,s2,s3}'
644
###纯 AWK
[root@Alaska141 nettest]# ls -l /etc/hosts|awk -F "" '{gsub("r","4");gsub("w","2");gsub("x","1");gsub("-","0");print $2+$3+$4
$5+$6+$7 $8+$9+$10}'
644
###egrep
[root@Alaska141 nettest]# stat /etc/hosts|egrep -i UID|egrep -o "[0-9]{3}"
064
```