Shell程序设计 – 正则表达式

目录

目录

正则表达式概念

- 正则表达式 – Regular Expression，**它是一种文本模式，包括普通字符（例如，a到z之间的字母）和特殊字符（称为“元字符”）。正则表达式使用单个字符串来描述、匹配某个句法规则的字符串。**尽管正则表达式是繁琐的，但是它是强大的，Unix/Linux中很多命令：像grep, sed, awk等命令可以支持正则表达式。

- 或许你已经使用过某些正则表达式概念。例如，你很可能使用过？和*通配符来查找硬盘上的文件。？通配符匹配文件名中的0个或1个字符，而*通配符匹配零个或多个字符。我们来看下面的一个简单的例子：

```
[root@test ~]# ls -l test*.sh
-rwxr--r-- 1 root root 192 Feb 15 20:10 test1.sh
-rw-r--r-- 1 root root 425 Feb 16 15:03 test22.sh
-rw-r--r-- 1 root root 388 Feb 15 20:22 test2.sh
-rw-r--r-- 1 root root 323 Feb 15 21:11 test3.sh
-rw-r--r-- 1 root root 574 Feb 16 09:22 test4.sh
-rw-r--r-- 1 root root 530 Feb 16 09:44 test5.sh
-rwxr--r-- 1 root root 214 Feb 16 17:29 test.sh
[root@test ~]#
```

```
[root@test ~]# ls -l test?.sh
-rwxr--r-- 1 root root 192 Feb 15 20:10 test1.sh
-rw-r--r-- 1 root root 388 Feb 15 20:22 test2.sh
-rw-r--r-- 1 root root 323 Feb 15 21:11 test3.sh
-rw-r--r-- 1 root root 574 Feb 16 09:22 test4.sh
-rw-r--r-- 1 root root 530 Feb 16 09:44 test5.sh
[root@test ~]#
```

问题：为什么图二没有列出test22.sh文件呢?

正则表达式概念

- 通过上面的简单例子，你应该对*和？有了基本的了解和认识，*和?是常用的通配符，用来匹配符合条件的文件名。而正则表达式通配符模式的工作原理与之类似。正则表达式模式含有文本或特殊字符，它使用不同的特殊字符就可以定义特定的数据过滤模式。

- 让我们来一起看下面使用grep命令过滤出了上例中那些shell脚本名。

```
[root@test ~]# ls -l test*.sh > mytest.txt
[root@test ~]# cat mytest.txt
-rwxr--r-- 1 root root 192 Feb 15 20:10 test1.sh
-rw-r--r-- 1 root root 425 Feb 16 15:03 test22.sh
-rw-r--r-- 1 root root 388 Feb 15 20:22 test2.sh
-rw-r--r-- 1 root root 323 Feb 15 21:11 test3.sh
-rw-r--r-- 1 root root 574 Feb 16 09:22 test4.sh
-rw-r--r-- 1 root root 530 Feb 16 09:44 test5.sh
-rwxr--r-- 1 root root 214 Feb 16 17:29 test.sh
[root@test ~]#
```

```
[root@test ~]# grep -E [a-z]+[0-9]+.sh$ mytest.txt
-rwxr--r-- 1 root root 192 Feb 15 20:10 test1.sh
-rw-r--r-- 1 root root 425 Feb 16 15:03 test22.sh
-rw-r--r-- 1 root root 388 Feb 15 20:22 test2.sh
-rw-r--r-- 1 root root 323 Feb 15 21:11 test3.sh
-rw-r--r-- 1 root root 574 Feb 16 09:22 test4.sh
-rw-r--r-- 1 root root 530 Feb 16 09:44 test5.sh
[root@test ~]#
```

- 那么问题来了：grep使用了什么技术从文件中过滤出那些shell脚本名？**正则表达式！**

目录

正则表达式基础

- **正则表达式的定义：使用字符串来描述、匹配一系列符合某个规则的字符串。**
- **正则表达式组成：**
  - **普通字符：包括大小写字母、数字、标点符号以及一些其它符号。**
  - **元字符：指在正则表达式中具有特殊意义的专用字符。**
- **支持的工具：find，grep, sed和awk等。**
- **常见的元字符见下表：**

元字符	作用
*	前一个字符串匹配0次或者任意多次。
.	匹配除了换行符外任意一个字符。
^	匹配行首。例如：^hello会匹配以hello开头的行。

正则表达式基础

元字符	作用
$	匹配行尾。例如：hello$会匹配以hello结尾的行。要匹配$字符本身，需要使用转移\$。
[]	匹配中括号中指定的任意一个字符，只匹配一个字符。例如：[aeiou]匹配任意一个元音字母，[0-9]匹配任意一位数字，[a-z][0-9]匹配小写字母和一位数字构成的两位字符。
[^]	匹配除中括号字符以外的任意一个字符。例如：[^0-9]匹配任意一位非数字字符，[^a-z]表示任意一位非小写字母。
\	转义符。用于将特殊符号的含义去掉。
\{n\}	表示其前面的字符恰好出现n次。例如：[0-9]\{4\}匹配4位数字，[1][3-8][0-9]\{9\}匹配早期的手机号码。
\{n,\}	表示其前面的字符出现不少于n次。例如：[0-9]\{2,\}表示两位及以上的数字。
\{n,m\}	表示其前面的字符至少出现n次，最多出现m次。例如：[a-z]\{6,8\}匹配6到8位的小写字母。

扩展正则表达式

限定符	说明
.*	任意长度的任意字符。
?	匹配前面子表达式0次或者1次，即：可有可无。
+	与星号相似，表示其前面字符出现一次或多次，但必须出现一次。
{n,m}	匹配前面的子表达式n到m次。
{n}	匹配前面的子表达式n次。
{n,}	匹配前面的子表达式不少于n次。
{, n}	匹配前面的子表达式最多n次。
|	用逻辑或方式指定要用的模式。
()	字符串分组，将括号中的字符串作为一个整体。

扩展正则表达式

位置限定符	说明
^	行首锚定，用于模式的最左侧。
$	行尾锚定，用于模式的最右侧。
^PATTERN^	用于模式匹配整行。
^$	空行
^[[:space:]]*$	空白行
\<	词首锚定，用于单词模式的左侧。
\>	词尾锚定，用于单词的右侧。
\<PATTERN\>	匹配整个单词。

目录

正则表达式的应用 – 示例

- 表示一个任意字符

```
[root@test ~]# echo abc |grep "a.c"
abc
[root@test ~]# echo abc |grep "a\.c"
[root@test ~]# echo $?
1
[root@test ~]# echo abc a.c |grep "a\.c"
abc a.c
[root@test ~]# echo $?
0
[root@test ~]# echo abc asc |grep "a.c"
abc asc
[root@test ~]# echo $?
0
[root@test ~]#
```

- []匹配括号中的一个字符

```
[root@test ~]# ls |grep "[0-9].txt"
0.txt
1.txt
2.txt
3.txt
4.txt
5.txt
6.txt
7.txt
8.txt
9.txt
centos8.txt
[root@test ~]#
```

```
[root@test ~]# ls |grep "[a-f].txt"
a.txt
b.txt
c.txt
d.txt
e.txt
f.txt
[root@test ~]#
```

正则表达式的应用 – 示例

- [^]表示否定括号出现字符类中的字符，取反。

```
[root@test ~]# ls |grep "[^ABCD].txt"
0.txt
1.txt
2.txt
3.txt
4.txt
5.txt
6.txt
7.txt
8.txt
9.txt
a.txt
b.txt
centos8.txt
c.txt
d.txt
e.txt
f.txt
mytest.txt
nodelist-xcat-recoveros.txt
[root@test ~]#
```

- 匹配大小写字母和数字。

```
[root@test ~]# ls |grep -E "([a-z,A-Z]|[0-9]).txt"
0.txt
1.txt
2.txt
3.txt
4.txt
5.txt
6.txt
7.txt
8.txt
9.txt
a.txt
A.txt
b.txt
B.txt
centos8.txt
c.txt
d.txt
e.txt
f.txt
mytest.txt
nodelist-xcat-recoveros.txt
[root@test ~]#
```

正则表达式的应用 – 示例

- *匹配前面子表达式0次或多次

```
[root@test ~]# echo google ggle |grep "go*gle"
google ggle
[root@test ~]# echo $?
0
[root@test ~]# echo google ggle gggggle |grep "go*gle"
google ggle gggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- {n,m}匹配前面的子表达式n到m次

```
[root@test ~]# echo goooogle goole gggggle |grep -E "go{2,5}gle"
goooogle goole gggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- {n, }匹配前面的子表达式不少于n次

```
[root@test ~]# echo goooogle goole gggggle |grep -E "go{3,}gle"
goooogle goole gggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- {，n}匹配前面的子表达式最多n次

```
[root@test ~]# echo goooogle goole gggggle |grep -E "go{,3}gle"
goooogle goole gggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

正则表达式的应用 – 示例

- *任意长度的任意字符

```
[root@test ~]# echo gggggggggggggggggggle |grep ".*gle"
gggggggggggggggggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- ？匹配前面表达式0次或1次

```
[root@test ~]# echo gooooooogle goole ggggggggggle |grep -E "go?gle"
gooooooogle goole ggggggggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- +与星号相似，表示其前面字符出现一次或多次，但必须出现一次。

```
[root@test ~]# echo goooooooogle goole gggggle |grep -E "go+gle"
gooooooogle goole gggggle
[root@test ~]# echo $?
0
[root@test ~]#
```

- |逻辑或

```
[root@test ~]# echo 1xx 1yy 2xx |grep -E "1|2"
1xx 1yy 2xx
[root@test ~]# echo 1xx 1yy 2xx |grep -E "1|2|yy"
1xx 1yy 2xx
[root@test ~]#
```

正则表达式的应用 – 示例

- （）字符串分组，将括号中的字符串作为一个整体

```
[root@test ~]# echo 1xx 1abc 2abc |grep -E "(1|2)abc"
1xx 1abc 2abc
[root@test ~]# echo $?
0
[root@test ~]#
```

- 提取IP地址

方式一：

```
[root@test ~]# ifconfig ens34 |grep "netmask" |grep -o -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" |head -1
10.10.10.10
[root@test ~]#
```

方式二：

```
[root@test ~]# ifconfig ens34 |grep "netmask" |grep -o -E "([0-9]{1,3}.){3}[0-9]{1,3}" |head -1
10.10.10.10
[root@test ~]#
```

正则表达式的应用 – 示例

- 解析邮件地址

```
[root@test ~]# echo tester123@gmail.com |grep -E "^([a-zA-Z0-9_\-\.\+]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$"
tester123@gmail.com
[root@test ~]# echo tester123@edu.com.cn |grep -E "^([a-zA-Z0-9_\-\.\+]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$"
tester123@edu.com.cn
[root@test ~]#
```

- 解析身份证号（15位、18位数）

```
[root@test ~]# echo 610121200911240128 |grep -E "(^[0-9]{15}$)|(^[0-9]{18}$)|(^[0-9]{17}([0-9]|X|x)$)"
610121200911240128
[root@test ~]# echo 610121091124121 |grep -E "(^[0-9]{15}$)|(^[0-9]{18}$)|(^[0-9]{17}([0-9]|X|x)$)"
610121091124121
[root@test ~]# echo 610121200911240128X |grep -E "(^[0-9]{15}$)|(^[0-9]{18}$)|(^[0-9]{17}([0-9]|X|x)$)"
610121200911240128X
[root@test ~]#
```

谢 谢