**+++正则表达式**用来在文件中匹配符合条件的字符串，正则是**包含匹配**。grep、awk、sed等命令可以支持正则表达式。

正则表达式是包含匹配，即文件中的语句中包含了查找的字符串，那么就显示整行语句。

**通配符**用来匹配符合条件的文件名，通配符是**完全匹配**。ls、find、cp这些命令不支持正则表达式，所以只能使用shell自己的通配符来进行匹配了。

通配符必须是完全匹配，如find . -name “abc？” 查找到4个字符的文件名其中前三个字符是abc，列出的abcd、abce等文件。

## 通配符

| **元字符** | **作用** |
| --- | --- |
| \* | 匹配0个或任意多个字符，也就是可以匹配任何内容 |
| ？ | 匹配任意一个字符 |
| [] | 匹配[ ]中任意一个字符 |
| [-] | 匹配括号中任意一个字符，-代表一个范围，例如：[a-z]代表匹配一个小写字母 |
| [^] | 逻辑非，表示匹配不是中括号内的一个字符，例如[^0-9]代表匹配一个不是数字的字符 |

## 基础正则表达式

| **元字符** | **作用** |
| --- | --- |
| \* | **前一个字符**匹配0次或任意多次，匹配0次前一个字符则表示匹配任意字符，包括空白行 |
| . | 匹配除了换行符以外任意一个字符，“.\*”匹配所有内容 |
| ^ | 用于指定匹配字符串的头部，也称**行首定位符**；匹配行首。例如：^hello会匹配以hello开头的行，grep -n “^$” test.txt匹配空白行并显示行号 |
| $ | 用于指定匹配字符串的尾部，也称**行尾定位符**；匹配行尾。例如：hello$会匹配以hello结尾的行 |
| [] | 匹配中括号中指定的任意一个字符，只匹配一个字符，要匹配[则要转义\[ |
| [^] | 匹配除中括号的字符以外的任意一个字符 |
| \ | 转义符用于取消特殊符号的含义，匹配包含以.结尾的行grep “\.$” test.txt |
| \{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，但注意添加两边的定界符，以精确匹配 |
| \{n,\} | 表示其前面的字符出现不小于n次。例如：[0-9]\{2,\}匹配2位以上的数字 |
| \{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 |

注1：“?”、“ ()”是扩展正则中的元字符   
注2：正则表达式中使用以上的元字符，如[]、{}、\*、. 等所以在匹配字符串中包含有这些元字符时必须使用反斜杠(“\”)转义，但像<>这个非元字符符号就不需要转义。

## 扩展正则表达式

| **元字符** | **作用** |
| --- | --- |
| | | 管道符，表示“或”，即匹配其中任何一个，”book|desk”将匹配”book”或”desk” |
| () | 小括号，可以将正则字符和元字符或表达式进行组合，”(book|desk)s”将匹配”books”或”desks” |
| ? | 问号，匹配0个或1个前导表达式，如”a?”匹配其他字符串或a |
| \< | 反斜杠+小于号，**词首定位符**， “\< abc”表示所有包含以”abc”开头的单词的行 |
| \> | 反斜杠+大于号，**词尾定位符**， “\>abc”表示所有包含以”abc”结尾的单词的行 |
| - | 减号，用于指明字符范围， “[a-c]”将匹配包含a、b和c中任意一个字符的字符串 |
| + | 加号，匹配一个或多个前导表达式，相当于 expr{1,} |

注：grep -E pattern file才能在pattern匹配扩展正则表达式

### 正则中“？”的四种用法

#### 元字符？

因为 ? 在正则中有特殊的含义，所以如果想匹配？本身，则需要转义，\?

#### 有无量词

问号可以表示重复前面内容的零次或一次，也就是要么不出现，要么出现一次。

#### 非贪婪匹配

**贪婪匹配**   
在满足匹配时，匹配尽可能长的字符串，默认情况下采用贪婪匹配

string pattern1 = @"a.\*c"; // greedy match

Regex regex = new Regex(pattern1);

regex.Match("abcabc"); // return "abcabc"

* 1
* 2
* 3

**非贪婪匹配**   
在满足匹配时，匹配尽可能短的字符串，使用?表示非贪婪匹配

string pattern1 = @"a.\*?c"; // non-greedy match

Regex regex = new Regex(pattern1);

regex.Match("abcabc"); // return "abc"

* 1
* 2
* 3
* 4

#### 几个常用的非贪婪匹配Pattern

* \*? 重复任意次，但尽可能少重复
* +? 重复一次货更多次，但尽可能少重复
* ?? 重复0次或1次，但尽可能少重复
* {n, m}? 重复n到m次，但尽可能少重复
* {n,}? 重复ｎ次以上，但尽可能少重复

#### 不捕捉模式

如何关闭圆括号的捕获能力？而只是用它来做分组，方法是在左括号的后边加上:?，这里第一个圆括弧只是用来分组，而不会占用捕获变量，所以$1的内容只能是steak或者burger，而永远不可能是bronto。

while(<>){

if(/(?:bronto)(steak|burger)/){

print "Fred wants a $1\n" ;

}

}

* 1
* 2
* 3
* 4
* 5

## 字符集和字符类

字符集和字符类具有集合的概念，用于表示一个范围。   
系统会对一些特殊的字符集预先进行定义，称其为字符类。   
字符集表达的是一种统配关系。

// 字符集举例[a-zA-Z]：表示匹配任意一个字母[0-9\\*\+]：表示匹配数字、星号、加号中任意一个[^A-Z]：表示匹配任意一个除大写字母之外的字符

* 1
* 2
* 3
* 4

### 字符类

| **字符类** | **说明** |
| --- | --- |
| [[:alpha:]] | 小写及大写字母，等价于[a-zA-Z] |
| [[:alnum:]] | 小写和大写字母及数字，等价于[a-zA-Z0-9] |
| [[:cntrl:]] | 控制字符，如制表符（Tab）、换行符或反斜线 |
| [[:digit:]] | 数字0~9，等价于[0-9] |
| [[:graph:]] | ASCII码从33~126之间的可打印字符 |
| [[:lower:]] | 小写字母，等价于[a-z] |
| [[:punct:]] | 标点符号，包括问号、句号、逗号、冒号、分号、单引号、英文的重音符或中文的间隔号(`)、！、@、#、%、^、&、\*、左右括号、左右中括号、左右大括号、正斜杠(/)、-、\_ |
| [[:upper:]] | 大写字母，等价于[A-Z] |
| [[:space:]] | 空白字符，包括空格、水平制表符、垂直制表符、换行、换页、回车 |
| [[:xdigit:]] | 十六进制字符，等价于[a-fA-F0-9] |

注：中括号仅表示匹配指定字符集中的一个字符。

## 重复

重复有多种表示方式，不同的表示方式由不同的符号表示：

* 符号\*
* 符号？
* 符号+
* 大括号{}，又分{n}、{n,}、{n,m}

符号+的含义：表示匹配1个或多个前导表达式。注：其他符号前面已介绍。

// 表示可以匹配任意一个或多个字母组成的字符串[[:alpha:]]+

* 1
* 2

## 样例

// 子表达式：在正则表达式中可以使用小括号将字母、数字、元字符等进行组合，组合后再与其他组合联合使用，组成更长的表达式// 匹配good、very good、very very good..#grep "(very)\* much" test.txt// 匹配 wy、wz、xy、xz#grep "([wx])([yz])" test.txt

// 字符串的开始和结尾匹配：使用行首定位符(^)和行尾定位符($)// 匹配以world结尾的字符串#grep "world$" test.txt// 匹配以hello开头的字符串#grep "^hello" test.txt// 匹配 hello world字符串#grep "^hello world$" test.txt

// 分支：使用|表示一个分支，有或的意义// 匹配字符串goodness或goodful#grep "^good(ness|ful)$" test.txt

// 匹配IP地址#grep "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}" test.txt // 匹配日期格式 2014-12-22#grep "[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}" test.txt

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22

# 字符截取命令

## cut字段提取命令

cut [options] 文件名 options:   
-f 列号：提取第几列   
-d 分隔符：按照指定分隔符分隔列，默认制表符是分隔符

// 截取第一列

cut -f 1 test.txt// 截取第一列和第二列

cut -f 1,2 test.txt// 截取第一列到第三列

cut -f 1-3 test.txt// 在/etc/passwd中提取普通用户的信息，并删除该用户#userdel $(grep "/bin/bash" /etc/passwd |grep -v "root" |cut -f 1 -d ":")

// cut无法以空格做分隔符#df -h|cut -f 1,2 -d " "

Filesystem

/dev/xvda1

devtmpfs

tmpfs

tmpfs

tmpfs

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

## printf命令

prinf ‘输出类型 输出格式’ 输出内容   
输出类型：   
- %ns:输出字符串。n是数字指代输出几个字符   
-%ni:输出整数。n是数字指代输出几个数字   
-%m.nf:输出浮点数。m和n是数字，指代输出的整数位数和小数位数。如%8.2f%共输出8位数，其中2位是小数，6位是整数   
输出格式：   
-\a：输出警告声音   
-\b : 输出退格键，也就是Backspace键   
-\f : 清除屏幕   
-\n : 换行   
-\r ：回车，也就是Enter键   
-\t : 水平输出退格键，也就是Tab键   
-\v : 垂直输出退格键，也就是Tab键

#printf 1 2 3 4 51 //print 1#printf %s 1 2 3 4 512345#printf %s %s %s 1 2 3 4 5

%s%s12345#printf '%s %s %s' 1 2 3 4 5 // 所有内容都必须指定格式1 2 34 5 6

//printf打印文件内容#printf '%s\t%s\t%s\t%s\n' $(cat stu.txt)

ID Name gender Mark1 furong F 852 fengj F 603 cangls F 70

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

## awk命令

#awk '条件1{动作1}条件2{动作2}...' 文件名

条件(Pattern)

一般使用关系表达式作为条件

x>10判断变量x是否大于10

动作(Action)

格式化输出

流程控制语句

#cut -f 2,4 stu.txt

Name Mark

furong 85

fengj 60

cangls 70

// 输出stu.txt第2个字段和第4个字段// awk执行流程，先读取一行，按照条件做动作，然后读取第二行....#awk '{printf $2 "\t" $4 "\n"}' stu.txt // \t、\n必须添加双引号#awk '{print $2 "\t" $4}' stu.txt // 不用添加换行

//#df -h |awk '{print $5}'

Use%5%0%0%3%0%

// print 和printf的区别，BEGIN的用法，END的用法#awk 'BEGIN{printf "Test" "\n"}{print $2 "\t" $4}' stu.txt

Test

Name Mark

furong 85

fengj 60

cangls 70

# awk 'BEGIN{print "Test" }{print $2 "\t" $4}' stu.txt

Test

Name Mark

furong 85

fengj 60

cangls 70

# awk '{print $2 "\t" $4}END{print "Test"}' stu.txt

Name Mark

furong 85

fengj 60

cangls 70

Test

// FS指定分隔符# cat /etc/passwd|grep "/bin/bash" |awk 'BEGIN{FS=":"}{print $1 "\t" $3}'

root 0

test 1000

// 使用条件# cat stu.txt|grep -v Name |awk '$4>70{print $2}'

furong

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59

## sed命令

sed是一种几乎包括在所有UNIX平台（包括Linux）的轻量级流编辑器。sed主要是用来将数据进行选取、替换、删除、新增的命令。

sed [options] ‘[动作]’ 文件名   
options：   
-n ： 一般sed命令会把所有数据都输出到屏幕，如果加入此选项则只会把经过sed命令处理的行输出到屏幕   
-e： 允许对输入数据应用多条sed命令编辑   
-i ： 用sed的修改结果直接修改读取数据的的文件，而不是由屏幕输出

Action：   
-a ： 追加，在当前行后添加一行或多行   
-c ： 行替换，用c后面的字符串替换原数据行   
-i ： 插入，在当前行前插入一行或多行。   
-d ： 删除，删除指定的行   
-p ： 打印，输出指定的行   
-s ： 字符串替换，用一个字符串替换另一个字符串。格式为“行范围s/旧字串/新字串/g”，替换指定范围内所有的旧字符串，不加g则只替换第一个旧字符串。（和vim中的替换格式类似）

// -n 参数# cat stu.txt|grep -v Name |awk '$4>70{print $2}'

ID Name gender Mark1 furong F 851 furong F 852 fengj F 603 cangls F 70# sed -n '2p' stu.txt 1 furong F 85

// 删除第2行到第4行# sed '2,4d' stu.txt

ID Name gender Mark

// -a参数# sed '2a piaoliang jiushi renxing' stu.txt

ID Name gender Mark1 furong F 85

piaoliang jiushi renxing2 fengj F 603 cangls F 70

// 替换# sed -e 's/furong//g;s/fengj//g' stu.txt

ID Name gender Mark1 F 852 F 603 cangls F 70

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

# 常用命令

## sort

sort [options] 文件名   
options：   
-f ： 忽略大小写   
-n： 以数值型进行排序，默认使用字符串型排序   
-r ： 反向排序   
-t ：指定分隔符，默认分隔符是制表符   
-k n[,m]：按照指定的字段范围排序。从第n个字段开始，m字段结束（默认到行尾）

# sort -t ":" -k 3,3 /etc/passwd# sort -n -t ":" -k 3,3 /etc/passwd

* 1
* 2

## wc

wc [options] 文件名   
options：   
-l ： 只统计行数   
-w ： 只统计单词数   
-m： 只统计字符数

默认统计行数、单词数和字符数。字符数包括空格。

# 参考

[正则表达式中各种字符的含义](http://www.cnblogs.com/afarmer/archive/2011/08/29/2158860.html" \t "/tmp/wps-root/x/_blank)

个人分类： [Linux](https://blog.csdn.net/renwotao2009/article/category/2317857" \t "/tmp/wps-root/x/_blank)