3.1正则表达式

正则表达式使用被称为元字符（metacharacters）的字符组，元字符解析超出字符本省的含义。正则表达式的元字符集如下：

|  |  |
| --- | --- |
| . | 只与除了newline以外的单个字符一致（dot）。例如:”1..e”正则表达式包含１，并且２个字符后面显示ｅ的行 |
| ? | 与前面的０个或者一个正则表达式一致，一般匹配单个字符 |
| \* | 表示前导字符串或者正则表达式中反复出现０次以上的字符 |
| ＋ | 与前面的多个正则表达式一致，功能类似于\*,但是必须匹配多个 |
| {N} | 匹配N次 |
| {N,} | 匹配Ｎ次或是Ｎ次以上 |
| {N,M} | 匹配至少Ｎ次，不能超过M次 |
| - | 表示除了开始和最后一个以外的目录范围，或者是表示目录的末尾位置 |
| ^ | 表示行首的空格字符串，表示目录范围内不存在的字符 |
| $ | 表示行尾的空格字符，linux$表示所有以linux字符串结束的所有行｀ |
| ^$ | 与空行匹配 |
| [...] | 方括号在单一正则表达式中绑定字符集，  [xyz]:匹配xyz之中的一个字符  [c-n]:匹配从c~n的任意一个字符  [B-Pk-y]:匹配从B-P或k-y的任意一个字符  [a-z0-9]:匹配小写字母或数字中的任意一个字符  [^b-d]:匹配除了b-d以外的所有字符 |
| \ | 将特殊字符解析为字符原语。即元字符前添加反斜杠字符就解析为字符本义。 |
| \b | 表示词末尾的空格字符串 |
| \B | 表示除词末尾以外的空格字符串 |
| \< | 表示词首的空格字符串，\<linux:包含以linux字符串开始的词的行 |
| \> | 表示词尾的空格字符串，linux\>:包含以linux字符串结束的词的行。 |

使用／\<use\>的查找模式表示匹配use单个字符的模式,use前后都是空格，而/use\>这种模式表示查找到的以use结尾的字符，即use后面是空格，而前面可以是任何的字符，如because,而使用/use查找的是任何的包含use的字符。例如可以搜索~~:/\<b\*e\>~~/\<b.\*\>表示以b开头以e结尾的字符串。字符串之间可以包含空格。例如，bill is a wander

第四章　grep模式检索

Grep用于检索与出入文件的已有模式列表匹配的行，之后再通过标准输出复制和显示检索行，并且，若使用对齐选项，可以对其之后再显示。，在不超出内存上限的范围内，grep的检索范围对输入行没有限制，而且可以匹配１行中的所有字符。Grep使用换行符分隔匹配对象，所以无法在文本中匹配换行符。

|  |  |
| --- | --- |
| -b | 显示时在搜索行添加块编号　grep -b /etc/passwd |
| -c | 不显示匹配的行，只显示匹配的行数 |
| -h | 不显示文件名 |
| -i | 检索时忽略模式中使用的字符串大小写字母，即检索并显示所有大小写字母 |
| -l | 显示的文件名至少包括一个匹配模式匹配行，显示时每一个文件名用newline隔开。例如：grep -l root /etc/passwd /etc/hosts /etc/services .即搜索这些文件中存在匹配这一个模式的文件，并显示其文件名。 |
| -n | 显示匹配行时，与文件行号一起显示。 |
| -s | 正常执行，不显示错误消息 |
| -v | 只显示与模式不匹配的行 |
| -w | 用一个\<和\>内的词检索表达式 |

Grep示例

|  |  |
| --- | --- |
| grep ‘\.bak$’ file | 显示行的末尾以.bak结束的行，此时单引号保护$符号不被解析器解析 |
| Grep [A-Z] file | 现实的行至少包含一个大写字母 |
| Grep -w [tT]est file | 显示的行包含单词test和Test |
| Grep “$name” file | 显示所有拥有name变量值的行，使用变量时必须要使用双引号”” |
| Grep ‘$5’ file | 显示包含$5字符的行，必须要添加单引号，单引号内的$被识别为其本身。 |
| Ps-aux|grep‘^\*root’ | 显示首行有空格，并且包含root的所有行。 |

Sed 实用工具

Linux的文本处理实用工具主要有sed(stream editor)和awk .

Sed:非交互式模式文本文件编辑器

Awk:带有Ｃ语言格式语法的模式处理语言，以段为单位。

两者虽然略有差异，但是都使用正则表达式，默认使用标准的ＩＯ。Sed是以行为单位的非交互式编辑器，对指定的行进行逐行处理，然后将结果输出传送到标准输出或文件。Sed也可以在vim编辑器中使用。Sed决定对指定的地址范围内的哪些种输入进行处理，此时可以在地址范围内使用行号或正则表达式模式。如5d表示删除第五行，/windows/d表示删除包含“windows”的所有行。

|  |  |  |
| --- | --- | --- |
| [地址范围]/p | Print | 显示指定的地址范围 |
| [地址范围]/d | Delete | 删除指定的地址范围 |
| s/pattern1/pattern2 | substitute | 将行中首次出现的pattern1替换为pattern2 |
| [地址范围]/s/pattern1/pattern2 | Substitute | 将地址范围内的行中首次出现的pattern1替换为pattern2 |
| [地址范围]/y/pattern1/pattern2 | Transform | 在地址范围内将任何位于pattern1的字符变更为pattern2位置上的字符（与tr相同） |
| G | Global | 在所有的行启动与输入内容匹配的模式。 |

Ｓｅｄ示例

|  |  |
| --- | --- |
| ８d | 删除输入的第八行 |
| /^$/d | 删除所有的空行 |
| 1,/^$/d | 从第一行开始删除，直到发现空行 |
| /jones/p | 只显示包含“jones”的行 |
| s/windows/Linux/ | 在输入的各行用Linux替换第一次出现的Windows |
| s/windows/Linux/g | 在输入的各行用Linux替换每次出现的Windows |
| s/\*$// | 删除所有行尾出现的空格 |
| s/00\*/0/g | 以０替换所有出现的连续０ |
| /GUI/d | 删除包含GUI的所有行 |
| s/GUI//g | 在输入的各行删除GUI三个字符 |

Sed流编辑器在一个文件或者是一个输入中每次只能处理一行并显示到显示器当中。通过sed处理并不会修改源文件，若是想要保存修改后的文件，应该将其重定向到一个新文件当中。

Sed使用的正则表达式的元字符集及其含义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| ^ | 行首 | /^linux/ | 所有以linux开始的行 |
| $ | 行尾 | /linux$/ |  |
| . | 匹配一个字符但是newline除外 | /l...x/ |  |
| \* | 匹配０或０个以上字符串 | /\*linux/ | 无内容或者是以开始且带有linux字符的行 |
| [] | 匹配括号内任意一个字符 | /[Ll]inux/ | 包含linux或是Linux的行 |
| [^] | 不匹配括号内任意一个字符 | /[^A-KM-Z]inux/ | 显示的行中inux之前不是从A-K或M-Z的字符 |
| \(..\) | 保存匹配的字符 | s/\(love\)able/\1er/ | 使用\1保存匹配的模式方便以后浏览，并标记为１号标签。示例中loveable保存为替换over的字符串 |
| & | 保存可以记忆为替换字符串的检索字符串 | s/linux/\*\*&\*\*/ | 由于＆是检索字符串，所以将linux字符串置于\*字符中，linux字符串变为\*\*linux\*\* |
| \< | 词首 | /\<linux/ | 匹配包含以linux开始的词的行 |
| \> | 词尾 | /linux\>/ | 匹配包含以linux结束的词的行 |
| x\{m\} | 将x字符的重复次数设置为m | /o\{5\}/ | O重复５次 |
| x\{m,\} | Ｘ字符至少重复m次 | /o\{5,\}/ | O至少重复５次 |
| x{m,n\} | X至少重复ｍ词，至多n次 | /o\{5,10\}/ | O重复５－１０次之间。 |

6 awk编程

awk是操作数据并创建报告的语言，Linux系统使用的awk版本以ＧＵＮ版本的gawk作为符号连接。

Awk可以在命令行使用简单的运算符，也可以用于高级程序。Awk可以操作数据,是shell脚本不可或缺的工具，也用在管理低级的数据库当中。若没有指定分隔符，则awk默认将空格作为分隔符。

|  |
| --- |
| Ａｗｋ格式 |
| awk ‘pattern’ filename |
| Awk ‘{action}’ filename |
| Awk ‘pattern {action}’ filename |

进行awk编程时，先输入ａｗｋ指令，然后输入带有单引号的模式或动作，最后输入文件。若没有指定输入的文件，则使用键盘进行标准的输入。然后ａｗｋ将已经输入行的数据用空格或ＴＡＢ隔开，分隔为以$1开始的各字段变量进行识别。在使用空格隔开的字段当中执行以下的命令：

awk ‘{print $1}’ files

显示从左开始第一个出现的字段（$1）.再如下面的命令：

awk ‘linux {print $1,$2}’ files

显示以ｌｉｎｕｘ开始的行，并且显示第二个字段。在输出的结果当中字段１和字段２之间会存在空格符，这是因为在命令当中添加了逗号，使用逗号映射输出字段分隔符(OFS)默认向该OFS分配空格，因此，可以使用逗号创建分配给OFS变量的空格字符。

df | awk ‘$4 > 10000’

Ａｗｋ命令的动作（action）部分使用｛｝,若没有指定动作而匹配模式，ａｗｋ就执行默认的动作，在屏幕上显示匹配行，使用print函数显示内容时无需格式化。如果想要执行更高级的格式化操作，就需要使用printf,fprintf函数。Print函数可以使用参数变量计算出的值或者是字符串常数，字符串需要使用双引号(“”)。逗号用于分隔参数，不使用逗号的话参数就连在一起。例如:

date | awk ‘{print “today is ”$1”day” “\n现在的时间是:”$4}’

Print 函数的转义字符

|  |  |
| --- | --- |
| 转义字符 | 含义 |
| \b | 退格(backspace) |
| \f | 换页符(from feed) |
| \n | Newline 下一行 |
| \r | 回车 |
| \t | Ｔａｂ |
| \047 | 八进制数４７ |
| \c | Ｃ代表字符 |

6.2.7 awk -f选项

若文件中已生成ａｗｋ动作和命令，则使用-f选项。Awk命令保存在特定的文件当中。使用-f选项在该文件中使用输入命令处理其他文件。

格式：awk -f [awk 命令文件]　[awk 命令适用文本文件]

6.2.8 记录和字段

6.2.8.1记录

awk不能够查看输入的数据，但是可以查看数据的格式和结构。默认每一行称为“记录”（record）,由newline分隔。

1. 记录分隔符

默认设置的输出和输入分隔符(行分隔符)是回车符（newline）,保存在内置的awk变量ORS和RS中。虽然ORS和RS的值可以改变，但是最好不要更改。

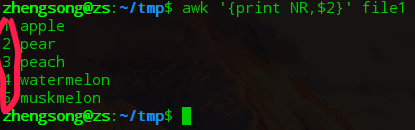
1. $0变量

Awk中的所有记录都被浏览为$0。(可以修改$0，以替换和分配对$0进行时，就成为NF值和字段号。)

awk ‘{print $０}’　awkfile == cat awkfile == awk ‘{print}’ awkfile

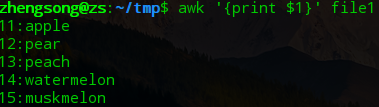
1. NR变量。

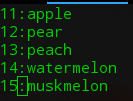
每一个记录号保存在awk的内置变量NR。记录保存以后，NR值就加１。

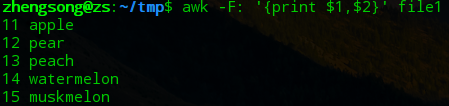


6.2.8.3 字段分隔符

Awk内置变量FS拥有输入字段分隔符(FS)的值。FS默认值为空格和Tab，利用该值分隔输入字段。可以在BEGIN语句或者命令行更改FS的值，那么我们就在命令行给FS值分配新的值。在命令行需要使用-F选项更改FS的值，此时，-F后面输入的字符是新字段分隔符。







－F 选项可以在 [ ] 内可以使用正则表达式，例如

awk -F’[ :\t]’ ‘／tmp/print{$1,$2}’ file

上面将空格、冒号、Tab字符识别为字符段分隔符。使用单引号防止识别为shell的元字符。检索tmp的所有行，并显示字段１和字段２．

6.2.9 模式和动作]

6.2.9.1模式

Awk管理模式awk在输入行的动作，该模式由正则表达式、真和假的状态结果或者是二者结合而成。默认动作显示的每行表达式状态值为真（true），读取模式表达式时暗示为if语句。



6.2.9.2　动作

Awk中的动作是使用花括号({})的语句，以分号(;)分隔。模式在动作之前，动作可以创建为简单的语句或者是高级语句组。语句可以使用分号(;)或者是newline分隔。例如｛print $1｝。模式和动作可以连接起来。但是记住，动作使用花括号。模式管理的动作范围从第一个花括号｛到第一个后花括号｝。若模式后面需要输入动作，必须在同一行输入第一个前花括号｛和模式。

格式：　pattern { action statement ;action statement;etc.}

或者pattern {

Action statement

Action statement

}

例如：awk ‘/zhengsong/ {print “hello,”$1,”sir”}’ file

6.3 awk和正则表达式

Awk的正则表达式是使用斜杠（／）且由字符组成的模式。Awk支持使用正则表达式元字符（与egrep相同），该字符可以更改正则表达式。以正则表达式匹配输入行的字符串，结果状态为真。并执行与表达式相关的动作。若没有指定的动作时则检索和正则表达式相匹配的行，则显示全部的记录行。

|  |  |
| --- | --- |
| Awk 支持的元字符 | |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的结尾 |
| . | 匹配一个字符 |
| \* | 匹配０以及０个以上的字符 |
| + | 匹配一个及多个字符 |
| - | 匹配０个或１个字符 |
| [ABC] | Ａ、Ｂ、Ｃ字符集中只匹配一个字符 |
| [^ABC] | Ａ、Ｂ、Ｃ中无匹配字符 |
| [A-Z][a-z] | Ａ-Z范围和a-z范围内存在匹配的字符 |
| [A|B] | 匹配Ａ或Ｂ字符 |
| (AB)+ | 匹配多个ＡＢ字符集（如：ＡＢ、ＡＢＡＢ、ＡＢＡＢＡＢＡＢ...） |
| \\* | 匹配\*号字符 |
| & | 在检索字符串中使用检索出的字符串代替。 |

例如：awk　‘/^[A-Z][a-z]+ ’ file

以上的awk命令显示的行以大写字母开始，从第二个字符开始包含多个小写字母，后面是空格。

Ｍａｔｃｈ运算符

标记波浪线（～）的match运算符检查某一个记录或者字段是否匹配表达式。

例如：awk ‘$2 ~ /[Kk]ill’ file

检索文件中的２号字段是否匹配大写Kill或者是小写的kill，并显示检索结果。

例如：awk ‘$2 !~ /g$/’ file

检索文件中的２号字段不是以g结尾的记录，并显示检索结果。

|  |  |
| --- | --- |
| POSIX 字符类 | |
| [:alnum:] | 由[A-Za-z0-9]字母字符和数字组成的字符串 |
| [:alpha:] | [A-Za-z]字母字符 |
| [:blank:] | [\x09]Space和Tab |
| [:cntrl:] | Control控制字符 |
| [:digit:] | [0-9]数字 |
| [:graph:] | [!-~]非空格的字符(Space，控制字符除外) |
| [:lower:] | [a-z]小写字母 |
| [:print:] | 类似于[-~][:graph:],但是包含space |
| [:punct:] | [!-/:-@[-{-~]语句符号字符 |
| [:space:] | [\t\v\f]所有空格字符(newline换行，space,Tab) |
| [:upper:] | [A-Z]大写字母 |
| [:xdigit:] | 能用于[0-9a-fA-F]16进制中的数字 |

例如：awk ‘/[[:lower:]]+g[[:space:]]+[[digit]]’ file

以上命令检索file中存在多个小写字母后面跟着一个g然后后面存在多个空格，多个空格以后是数字的行，并将其打印出来．

|  |  |  |
| --- | --- | --- |
| 比较表达式 | | |
| < | 小于 | Awk ‘$2 < 10{print $1,$2}’ file |
| <= | 小于等于 |  |
| == | 等于 |  |
| != | 不等于 |  |
| > | 大于 |  |
| >= | 大于等于 |  |
| ~ | 匹配正则表达式 | Awk ‘$2~/zhengsong/{print $1,$2}’ file |
| !~ | 与正则表达式不匹配 | Awk ‘$2!~/zhengsong/ {print $1,$2}’ file |

|  |  |
| --- | --- |
| Awk内置变量 | |
| ARGC | 命令行参数的个数 |
| ARGIND | 在命令行处理的当前文件的ARGV索引 |
| ARGV | 命令行参数的数组 |
| CONVFMT | 数字转换格式，默认值为%.6g |
| ENVIRON | 该数组包含传给shell的当前环境变量值 |
| ERRNO | 用getline函数读取或者使用终止函数时包含描述重定向的系统错误的字符串 |
| FIELDWIDTHS | 若分隔的记录是固定的字段宽，则为空格(而非FS)分隔开的字段宽目录 |
| FILENAME | 当前输入文件的名称 |
| FNR | 当前文件的全部记录数 |
| FS | 输入字段分隔符，默认值为空格 |
| IGNORECASE | 在正则表达式和字符处理的时候不使用case sensitivity，不区分大小写 |
| NF | 当前记录的全部字段数 |
| NR | 记录数 |
| OFMT | 数字输出格式 |
| OFS | 输出字段分隔符 |
| ORS | 输出记录分隔符 |
| RLENGTH | 匹配match函数的字符串长度 |
| RS | 输入记录分隔符 |
| RSTART | 匹配match函数的字符串选项 |
| RT | 记录结束符，设置为与字符或者是RS指定正则表达式相匹配的输入文本 |
| SUBSEP | 子脚本分隔符 |

例如：awk -F: ‘{IGNORECASE=1}; $1 == “zhengsong” {print $1,$2}’ file

6.6.3 BEGIN模式

Awk处理输入文件的行之前执行BEGIN模式，并将其置于动作块前．若是awk没有完成BEGIN动作块，则BEGIN块不读取输入的内容，所以输入的文件不存在也可以测试．BEGIN动作常常用于更改内置的变量(OFS,RS,FS等)，分配用户自定义变量的初始值，并且常常作为输出的一部分用于打印头文件或者是标题．

例如：awk ‘BEGIN{FS=”:”;OFS=”\t”;ORS=”\n\n”}{print $1,$2,$3}’ file

上述示例处理输入的文件之前，设置冒号为字段的分隔符(FS)，设置Tab为输出字段的分隔符(OFS);将２个newline设置为记录的分隔符(OFS).若是动作块存在两个以上的语句，则可以使用(;)或者是行分隔符(shell提示符的反斜杠＼用于创建newline).

6.6.4　END模式

END模式不匹配任何的输入行，但是执行END模式相关的动作．先处理所有的输入行，然后处理END模式．

例如：awk ‘/zhengsong/{count++} END{print “zhengsong was fount” count “times.”}’ file

在输入的文件file创建用户的自定义变量count，目的是计算包含zhengsong模式的所有输入行数．该count变量对包含zhengsong模式的输入行进行检索，若成功则加１，读取输入行时执行END模块，目的在于显示包含count最后结果值的”zhengsong was found n times”字符串．

在awk命令使用END块时必须要输入参数的文件名，只有使用BEGIN块时，不输入参数文件名也能启动．