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 |