**正则表达式基础**

**实验介绍**

虽然我们这一节的标题是正则表达式，但实际这一节实验只是介绍grep，sed，awk这三个命令，而正则表达式作为这三个命令的一种使用方式（命令输出中可以包含正则表达式）。正则表达式本身的内容很多，要把它说明清楚需要单独一门课程来实现，不过我们这一节中涉及到的相关内容通常也能够满足很多情况下的需求了。

想要更深入地学习使用正则表达式，在这里 [正则表达式基础](https://www.shiyanlou.com/courses/90)。

**一、正则表达式**

什么是正则表达式呢？

**正则表达式**，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（英语：Regular Expression，在代码中常简写为 regex、regexp 或 RE），计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、 替换那些符合某个模式的文本。

许多程序设计语言都支持利用正则表达式进行字符串操作。例如，在 Perl 中就内建了一个功能强大的正则表达式引擎。正则表达式这个概念最初是由 UNIX 中的工具软件（例如sed和grep）普及开的。正则表达式通常缩写成“regex”，单数有 regexp、regex，复数有 regexps、regexes、regexen。

简单的说形式和功能上正则表达式和我们前面讲的通配符很像，不过它们之间又有很大差别，特别在于一些特殊的匹配字符的含义上，希望初学者注意不要将两者弄混淆。

**1. 举例**

假设我们有这样一个文本文件，包含"shiyanlou"，和"shilouyan"这两个字符串，同样一个表达式：

shi\*

如果这作为一个正则表达式，它将只能匹配 shi，而如果不是作为正则表达式\*作为一个通配符，则将同时匹配这两个字符串。这是为什么呢？因为在正则表达式中\*表 示匹配前面的子表达式（这里就是它前面一个字符）零次或多次，比如它可以匹配"sh","shii","shish","shiishi"等等，而作为通 配符表示匹配通配符后面任意多个任意字符，所以它可以匹配"shiyanlou"，和"shilouyan"两个字符。

体验完了，下面就来开始正式学习正则表达式吧。

**2.基本语法：**

一个正则表达式通常被称为一个模式（**pattern**），为用来描述或者匹配一系列符合某个句法规则的字符串。

选择

|竖直分隔符表示选择，例如"boy|girl"可以匹配"boy"或者"girl"

数量限定

数量限定除了我们举例用的\*,还有+加号,?问号,.点号，如果在一个模式中不加数量限定符则表示出现一次且仅出现一次：

* +表示前面的字符必须出现至少一次(1次或多次)，例如，"goo+gle",可以匹配"gooogle","goooogle"等；
* ?表示前面的字符最多出现一次(0次或1次)，例如，"colou?r",可以匹配"color"或者"colour";
* \*星号代表前面的字符可以不出现，也可以出现一次或者多次（0次、或1次、或多次），例如，“0\*42”可以匹配42、042、0042、00042等。

范围和优先级

()圆括号可以用来定义模式字符串的范围和优先级，这可以简单的理解为是否将括号内的模式串作为一个整体。例 如，"gr(a|e)y"等价于"gray|grey"，（这里体现了优先级，竖直分隔符用于选择a或者e而不是gra和ey），" (grand)?father"匹配father和grandfather（这里体验了范围，?将圆括号内容作为一个整体匹配）。

语法（部分）

正则表达式有多种不同的风格，下面列举一些常用的作为 PCRE 子集的适用于perl和python编程语言及grep或egrep的正则表达式匹配规则：(**由于markdown表格解析的问题，下面的竖直分隔符用全角字符代替，实际使用时请换回半角字符**)

PCRE（Perl Compatible Regular Expressions中文含义：perl语言兼容正则表达式）是一个用 C 语言编写的正则表达式函数库，由菲利普.海泽(Philip Hazel)编写。PCRE是一个轻量级的函数库，比Boost 之类的正则表达式库小得多。PCRE 十分易用，同时功能也很强大，性能超过了 POSIX 正则表达式库和一些经典的正则表达式库。

| **字符** | **描述** |
| --- | --- |
| \ | **将下一个字符标记为一个特殊字符、或一个原义字符。**例如，“n”匹配字符“n”。“\n”匹配一个换行符。序列“\\”匹配“\”而“\(”则匹配“(”。 |
| ^ | **匹配输入字符串的开始位置。** |
| $ | **匹配输入字符串的结束位置。** |
| {n} | n是一个非负整数。**匹配确定的n次**。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。 |
| {n,} | n是一个非负整数。**至少匹配n次**。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*”。 |
| {n,m} | m和n均为非负整数，其中n<=m。**最少匹配n次且最多匹配m次。**例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。 |
| \* | **匹配前面的子表达式零次或多次**。例如，zo\*能匹配“z”、“zo”以及“zoo”。\*等价于{0,}。 |
| + | **匹配前面的子表达式一次或多次**。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。 |
| ? | **匹配前面的子表达式零次或一次**。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。 |
| ? | 当该字符紧跟在任何一个其他限制符（\*,+,?，{n}，{n,}，{n,m}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的 字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o”。 |
| . | **匹配除“\n”之外的任何单个字符**。要匹配包括“\n”在内的任何字符，请使用像“(.｜\n)”的模式。 |
| (pattern) | **匹配pattern并获取这一匹配的子字符串**。该子字符串用于向后引用。要匹配圆括号字符，请使用“\(”或“\)”。 |
| x｜y | **匹配x或y**。例如，“z｜food”能匹配“z”或“food”。“(z｜f)ood”则匹配“zood”或“food”。 |
| [xyz] | 字符集合（character class）。**匹配所包含的任意一个字符**。例如，“[abc]”可以匹配“plain”中的“a”。其中特殊字符仅有反斜线\保持特殊含义，用于转义字符。其它特殊字符如星号、加号、各种括号等均作为普通字符。脱字符^如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。**连字符 - 如果出现在字符串中间表示字符范围描述；如果如果出现在首位则仅作为普通字符。** |
| [^xyz] | 排除型（negate）字符集合。**匹配未列出的任意字符。**例如，“[^abc]”可以匹配“plain”中的“plin”。 |
| [a-z] | 字符范围。**匹配指定范围内的任意字符。**例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。 |
| [^a-z] | 排除型的字符范围。**匹配任何不在指定范围内的任意字符**。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。 |

优先级

优先级为从上到下从左到右，依次降低：

| **运算符** | **说明** |
| --- | --- |
| \ | 转义符 |
| (), (?:), (?=), [] | 括号和中括号 |
| \*、+、?、{n}、{n,}、{n,m} | 限定符 |
| ^、$、\任何元字符 | 定位点和序列 |
| ｜ | 选择 |

更多正则表达式的内容可以参考以下链接：

* [正则表达式wiki](http://zh.wikipedia.org/wiki/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F)
* [几种正则表达式引擎的语法差异](http://www.greenend.org.uk/rjk/tech/regexp.html)
* [各语言各平台对正则表达式的支持](http://en.wikipedia.org/wiki/Comparison_of_regular_expression_engines)

regex的思导图：

**二、grep模式匹配命令**

上面空谈了那么多正则表达式的内容也并没有提及具体该如何使用它，实在枯燥，如果说正则表达式是一门武功话，那它也只能算得上一些口诀招式罢了，要把它真正练起来还得需要一些兵器在手才行，这里我们要介绍的grep命令以及后面要讲的sed,awk这些就该算作是这样的兵器了。

**1.基本操作**

grep命令用于打印输出文本中匹配的模式串，它使用正则表达式作为模式匹配的条件。grep支持三种正则表达式引擎，分别用三个参数指定：

| **参数** | **说明** |
| --- | --- |
| -E | POSIX扩展正则表达式，ERE |
| -G | POSIX基本正则表达式，BRE |
| -P | Perl正则表达式，PCRE |

不过在你没学过perl语言的大多数情况下你将只会使用到ERE和BRE,所以我们接下来的内容都不会讨论到PCRE中特有的一些正则表达式语法（它们之间大部分内容是存在交集的，所以你不用担心会遗漏多少重要内容）

在通过grep命令使用正则表达式之前，先介绍一下它的常用参数：

| **参数** | **说明** |
| --- | --- |
| -b | 将二进制文件作为文本来进行匹配 |
| -c | 统计以模式匹配的数目 |
| -i | 忽略大小写 |
| -n | 显示匹配文本所在行的行号 |
| -v | 反选，输出不匹配行的内容 |
| -r | 递归匹配查找 |
| -A n | n为正整数，表示after的意思，除了列出匹配行之外，还列出后面的n行 |
| -B n | n为正整数，表示before的意思，除了列出匹配行之外，还列出前面的n行 |
| --color=auto | 将输出中的匹配项设置为自动颜色显示 |

注：在大多数发行版中是默认设置了grep的颜色的，你可以通过参数指定或修改GREP\_COLOR环境变量。

**2.使用正则表达式**

使用基本正则表达式，BRE

* 位置

查找/etc/group文件中以"shiyanlou"为开头的行

$ grep 'shiyanlou' /etc/group

$ grep '^shiyanlou' /etc/group

* 数量

# 将匹配以'z'开头以'o'结尾的所有字符串

$ echo 'zero\nzo\nzoo' | grep 'z.\*o'

# 将匹配以'z'开头以'o'结尾，中间包含一个任意字符的字符串

$ echo 'zero\nzo\nzoo' | grep 'z.o'

# 将匹配以'z'开头,以任意多个'o'结尾的字符串

$ echo 'zero\nzo\nzoo' | grep 'zo\*'

注意：其中\n为换行符

* 选择

# grep默认是区分大小写的，这里将匹配所有的小写字母

$ echo '1234\nabcd' | grep '[a-z]'

# 将匹配所有的数字

$ echo '1234\nabcd' | grep '[0-9]'

# 将匹配所有的数字

$ echo '1234\nabcd' | grep '[[:digit:]]'

# 将匹配所有的小写字母

$ echo '1234\nabcd' | grep '[[:lower:]]'

# 将匹配所有的大写字母

$ echo '1234\nabcd' | grep '[[:upper:]]'

# 将匹配所有的字母和数字，包括0-9,a-z,A-Z

$ echo '1234\nabcd' | grep '[[:alnum:]]'

# 将匹配所有的字母

$ echo '1234\nabcd' | grep '[[:alpha:]]'

下面包含完整的特殊符号及说明：

| **特殊符号** | **说明** |
| --- | --- |
| [:alnum:] | 代表英文大小写字节及数字，亦即 0-9, A-Z, a-z |
| [:alpha:] | 代表任何英文大小写字节，亦即 A-Z, a-z |
| [:blank:] | 代表空白键与 [Tab] 按键两者 |
| [:cntrl:] | 代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等 |
| [:digit:] | 代表数字而已，亦即 0-9 |
| [:graph:] | 除了空白字节 (空白键与 [Tab] 按键) 外的其他所有按键 |
| [:lower:] | 代表小写字节，亦即 a-z |
| [:print:] | 代表任何可以被列印出来的字节 |
| [:punct:] | 代表标点符号 (punctuation symbol)，亦即：" ' ? ! ; : # $... |
| [:upper:] | 代表大写字节，亦即 A-Z |
| [:space:] | 任何会产生空白的字节，包括空白键, [Tab], CR 等等 |
| [:xdigit:] | 代表 16 进位的数字类型，因此包括： 0-9, A-F, a-f 的数字与字节 |

**注意**：之所以要使用特殊符号，是因为上面的[a-z]不是在所有情况下都管用，这还与主机当前的语系有关，即设置在LANG环境变量的值，zh\_CN.UTF-8的话[a-z]，即为所有小写字母，其它语系可能是大小写交替的如，"a A b B...z Z"，[a-z]中就可能包含大写字母。所以在使用[a-z]时请确保当前语系的影响，使用[:lower:]则不会有这个问题。

# 排除字符

$ echo 'geek|good' | grep '[^o]'

**注意:**当^放到中括号内为排除字符，否则表示行首。

使用扩展正则表达式，ERE

要通过grep使用扩展正则表达式需要加上-E参数，或使用egrep。

* 数量

# 只匹配"zo"

$ echo 'zero\nzo\nzoo' | grep -E 'zo{1}'

# 匹配以"zo"开头的所有单词

$ echo 'zero\nzo\nzoo' | grep -E 'zo{1,}'

**注意：**推荐掌握{n,m}即可，+,?,\*，这几个不太直观，且容易弄混淆。

* 选择

# 匹配"www.shiyanlou.com"和"www.google.com"

$ echo 'www.shiyanlou.com\nwww.baidu.com\nwww.google.com' | grep -E 'www\.(shiyanlou|google)\.com'

# 或者匹配不包含"baidu"的内容

$ echo 'www.shiyanlou.com\nwww.baidu.com\nwww.google.com' | grep -Ev 'www\.baidu\.com'

**注意：**因为.号有特殊含义，所以需要转义。

关于正则表达式和grep命令的内容就介绍这么多，下面会介绍两个更强大的工具sed和awk，但同样也正是因为这两个工具的强大，我们的内容无法包含它们的全部，这里将只对基本内容作介绍。

**三、sed 流编辑器**

sed工具在 man 手册里面的全名为"sed - stream editor for filtering and transforming text "，意即，用于过滤和转换文本的流编辑器。

在 Linux/UNIX 的世界里敢称为编辑器的工具，大都非等闲之辈，比如前面的"vi/vim(编辑器之神)","emacs(神的编辑器)","gedit"这些个编辑器。sed与上述的最大不同之处大于它是一个非交互式的编辑器，下面我们就开始介绍sed这个编辑器。

**sed常用参数介绍**

sed 命令基本格式：

sed [参数]... [执行命令] [输入文件]...

# 形如：

$ sed -i '1s/sad/happy/' test # 表示将test文件中第一行的"sad"替换为"happy"

| **参数** | **说明** |
| --- | --- |
| -n | 安静模式，只打印受影响的行，默认打印输入数据的全部内容 |
| -e | 用于在脚本中添加多个执行命令一次执行，在命令行中执行多个命令通常不需要加该参数 |
| -f filename | 指定执行filename文件中的命令 |
| -r | 使用扩展正则表达式，默认为标准正则表达式 |
| -i | 将直接修改输入文件内容，而不是打印到标准输出设备 |

**sed编辑器的执行命令（这里”执行“解释为名词）**

sed执行命令格式：

[n1][,n2]command

[n1][~step]command

# 其中一些命令可以在后面加上作用范围，形如：

$ sed -i 's/sad/happy/g' test # g表示全局范围

$ sed -i 's/sad/happy/4' test # 4表示指定行中的第四个匹配字符串

其中n1,n2表示输入内容的行号，它们之间为,逗号则表示从n1到n2行，如果为～波浪号则表示从n1开始以step为步进的所有行；command为执行动作，下面为一些常用动作指令：

| **命令** | **说明** |
| --- | --- |
| s | 行内替换 |
| c | 整行替换 |
| a | 插入到指定行的后面 |
| i | 插入到指定行的前面 |
| p | 打印指定行，通常与-n参数配合使用 |
| d | 删除指定行 |

**sed操作举例**

我们先找一个用于练习的文本文件：

$ cp /etc/passwd ~

打印指定行

# 打印2-5行

$ nl passwd | sed -n '2,5p'

# 打印奇数行

$ nl passwd | sed -n '1~2p'

行内替换

# 将输入文本中"shiyanlou" 全局替换为"hehe",并只打印替换的那一行，注意这里不能省略最后的"p"命令

$ sed -n 's/shiyanlou/hehe/gp' passwd

**注意：** 行内替换可以结合正则表达式使用。

行间替换

$ nl passwd | grep "shiyanlou"

# 删除第21行

$ sed -n '21c\www.shiyanlou.com' passwd

关于sed命令就介绍这么多，你如果希望了解更多sed的高级用法，你可以参看如下链接：

* [sed简明教程](http://coolshell.cn/articles/9104.html)
* [sed单行脚本快速参考](http://sed.sourceforge.net/sed1line_zh-CN.html)
* [sed完全手册](http://www.gnu.org/software/sed/manual/sed.html)

**四、awk文本处理语言**

看到上面的标题，你可能会感到惊异，难道我们这里要学习的是一门“语言”么，确切的说，我们是要在这里学习awk文本处理语言，只是我们并不会在这里学习到比较完整的关于awk的内容，还是因为前面的原因，它太强大了，它的应用无处不在，我们无法在这里以简短的文字描述面面俱到，如果你有目标成为一个linux系统管理员，确实想学好awk，你一不用担心，实验楼会在之后陆续上线linux系统管理员的学习路径，里面会有单独的关于正则表达式,awk,sed等相关课程，敬请期待吧。下面的内容，我们就作为一个关于awk的入门体验章节吧，其中会介绍一些awk的常用操作。

**1.awk介绍**

AWK是一种优良的文本处理工具，Linux及Unix环境中现有的功能最强大的数据处理引擎之一.其名称得自于它的创 始人Alfred Aho（阿尔佛雷德·艾侯）、Peter Jay Weinberger（彼得·温伯格）和Brian Wilson Kernighan（布莱恩·柯林汉)姓氏的首个字母.AWK程序设计语言，三位创建者已将它正式定义为“样式扫描和处理语言”。它允许您创建简短的程 序，这些程序读取输入文件、为数据排序、处理数据、对输入执行计算以及生成报表，还有无数其他的功能。最简单地说，AWK是一种用于处理文本的编程语言工 具。

在大多数linux发行版上面，实际我们使用的是gawk（GNU awk，awk的GNU版本），在我们的环境中ubuntu上，默认提供的是mawk，不过我们通常可以直接使用awk命令（awk语言的解释器），因为 系统已经为我们创建好了awk指向mawk的符号链接。

$ ll /usr/bin/awk

nawk： 在 20 世纪 80 年代中期，对 awk语言进行了更新，并不同程度地使用一种称为 nawk(new awk) 的增强版本对其进行了替换。许多系统中仍然存在着旧的awk 解释器，但通常将其安装为 oawk (old awk) 命令，而 nawk 解释器则安装为主要的 awk 命令，也可以使用 nawk 命令。Dr. Kernighan 仍然在对 nawk 进行维护，与 gawk 一样，它也是开放源代码的，并且可以免费获得; gawk： 是 GNU Project 的awk解释器的开放源代码实现。尽管早期的 GAWK 发行版是旧的 AWK 的替代程序，但不断地对其进行了更新，以包含 NAWK 的特性; mawk 也是awk编程语言的一种解释器，mawk遵循 POSIX 1003.2 （草案 11.3）定义的 AWK 语言，包含了一些没有在AWK 手册中提到的特色，同时 mawk 提供一小部分扩展,另外据说mawk是实现最快的awk

**2.awk的一些基础概念**

awk所有的操作都是基于pattern(模式)—action(动作)对来完成的，如下面的形式：

$ pattern {action}

你可以看到就如同很多编程语言一样，它将所有的动作操作用一对{}花括号包围起来。其中 pattern通常是是表示用于匹配输入的文本的“关系式”或“正则表达式”，action则是表示匹配后将执行的动作。在一个完整awk操作中，这两者 可以只有其中一个，如果没有pattern则默认匹配输入的全部文本，如果没有action则默认为打印匹配内容到屏幕。

awk处理文本的方式，是将文本分割成一些“字段”，然后再对这些字段进行处理，默认情况下，awk以空格作为一个字段的分割符，不过这不是固定了，你可以任意指定分隔符，下面将告诉你如何做到这一点。

**3.awk命令基本格式**

awk [-F fs] [-v var=value] [-f prog-file | 'program text'] [file...]

其中-F参数用于预先指定前面提到的字段分隔符（还有其他指定字段的方式） ，-v用于预先为awk程序指定变量，-f参数用于指定awk命令要执行的程序文件，或者在不加-f参数的情况下直接将程序语句放在这里，最后为awk需要处理的文本输入，且可以同时输入多个文本文件。现在我们还是直接来具体体验一下吧。

**4.awk操作体验**

先用vim新建一个文本文档

$ vim test

包含如下内容：

I like linux

www.shiyanlou.com

* 使用awk将文本内容打印到终端

# "quote>" 不用输入

$ awk '{

> print

> }' test

# 或者写到一行

$ awk '{print}' test

说明:在这个操作中我是省略了patter，所以awk会默认匹配输入文本的全部内容，然后在"{}"花括号中执行动作，即print打印所有匹配项，这里是全部文本内容

* 将test的第一行的每个字段单独显示为一行

$ awk '{

> if(NR==1){

> print $1 "\n" $2 "\n" $3

> } else {

> print}

> }' test

# 或者

$ awk '{

> if(NR==1){

> OFS="\n"

> print $1, $2, $3

> } else {

> print}

> }' test

说明:你首先应该注意的是，这里我使用了awk语言的分支选择语句if,它的使用和很多高级语言如C/C++语言基本一致，如果你有这些语言的基础，这里将很好理解。另一个你需要注意的是NR与OFS，这两个是awk内建的变量，NR表示当前读入的记录数，你可以简单的理解为当前处理的行数，OFS表示输出时的字段分隔符，默认为" "空格，如上图所见，我们将字段分隔符设置为\n换行符，所以第一行原本以空格为字段分隔的内容就分别输出到单独一行了。然后是$N其中N为相应的字段号，这也是awk的内建变量，它表示引用相应的字段，因为我们这里第一行只有三个字段，所以只引用到了$3。除此之外另一个这里没有出现的$0，它表示引用当前记录（当前行）的全部内容。

* 将test的第二行的以点为分段的字段换成以空格为分隔

$ awk -F'.' '{

> if(NR==2){

> print $1 "\t" $2 "\t" $3

> }}' test

# 或者

$ awk '

> BEGIN{

> FS="."

> OFS="\t" # 如果写为一行，两个动作语句之间应该以";"号分开

> }{

> if(NR==2){

> print $1, $2, $3

> }}' test

说明：这里的-F参数，前面已经介绍过，它是用来预先指定待处理记录的字段分隔符。我们需要注意的是除了指定OFS我们还可以在print 语句中直接打印特殊符号如这里的\t，**print打印的非变量内容都需要用""一对引号包围起来**。上面另一个版本，展示了实现预先指定变量分隔符的另一种方式，即使用BEGIN，就这个表达式指示了，其后的动作将在所有动作之前执行，这里是FS赋值了新的"."点号代替默认的" "空格

**注意**:首先说明一点，我们在学习和使用awk的时候应该尽可能将其作为一门程序语言来理解，这样将会使你学习起来更容易，所以初学阶段在练习awk时应该尽量按照我那样的方式分多行按照一般程序语言的换行和缩进来输入，而不是全部写到一行（当然这在你熟练了之后是没有任何问题的）。

**6.awk常用的内置变量**

| **变量名** | **说明** |
| --- | --- |
| FILENAME | 当前输入文件名，若有多个文件，则只表示第一个。如果输入是来自标准输入，则为空字符串 |
| $0 | 当前记录的内容 |
| $N | N表示字段号，最大值为NF变量的值 |
| FS | 字段分隔符，由正则表达式表示，默认为" "空格 |
| RS | 输入记录分隔符，默认为"\n"，即一行为一个记录 |
| NF | 当前记录字段数 |
| NR | 已经读入的记录数 |
| FNR | 当前输入文件的记录数，请注意它与NR的区别 |
| OFS | 输出字段分隔符，默认为" "空格 |
| ORS | 输出记录分隔符，默认为"\n" |

关于awk的内容本课程将只会包含这些内容，如果你想了解更多，请期待后续课程，或者参看一下链接内容：

* [awk程序设计语言](http://awk.readthedocs.org/en/latest/chapter-one.html)
* [awk简明教程](http://coolshell.cn/articles/9070.html)
* [awk用户指南](http://www.gnu.org/software/gawk/manual/gawk.html)