软件开发平台与应用
ROS编程技术
---1.6 Shell

主讲教师: 罗云翔

智能软件研究中心
Intelligent Software Research Center

awk sed附加课件

为什么要学awk、sed、正则表达式?

- 文本处理的例子:
  - 全文搜索“git”:

    word就能实现，grep也能实现。
  - 搜索“g_t”,g和t中间有一个任意字符:

    word不够用了!

    grep、sed、awk都可以，需要“正则表达式”支持。

    比如: $ grep ″g.t″ test.txt //用句点匹配单字符
  - 搜索并替换全部“g_t”为“git”:

    grep不行，只能搜索，不能处理;

    sed和awk都可以。

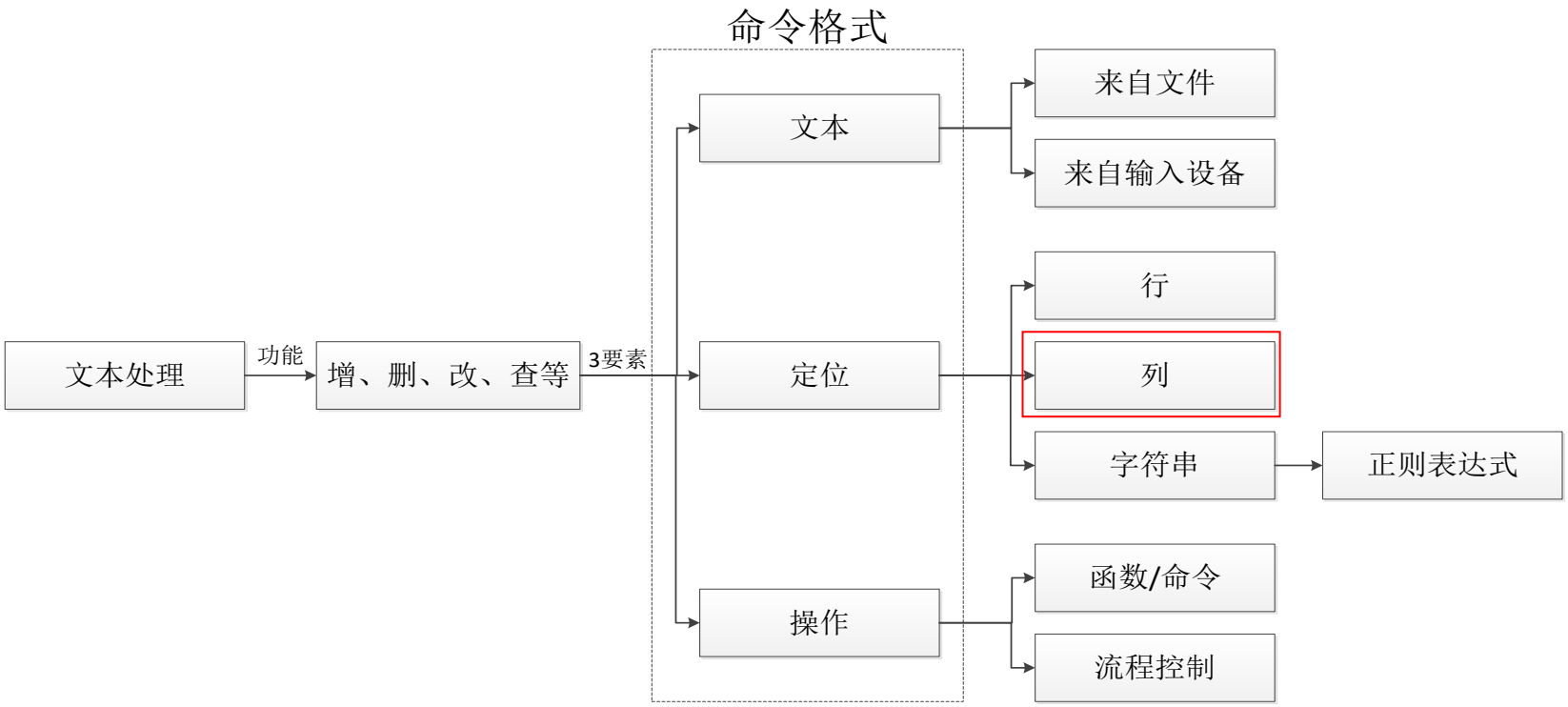
本节重点

- 本节的三个重点:
  - 正则表达式
  - awk
  - sed

- 三者是什么区别联系呢?

既然它们都是与文本处理相关，就从“文本处理”做什么说起。

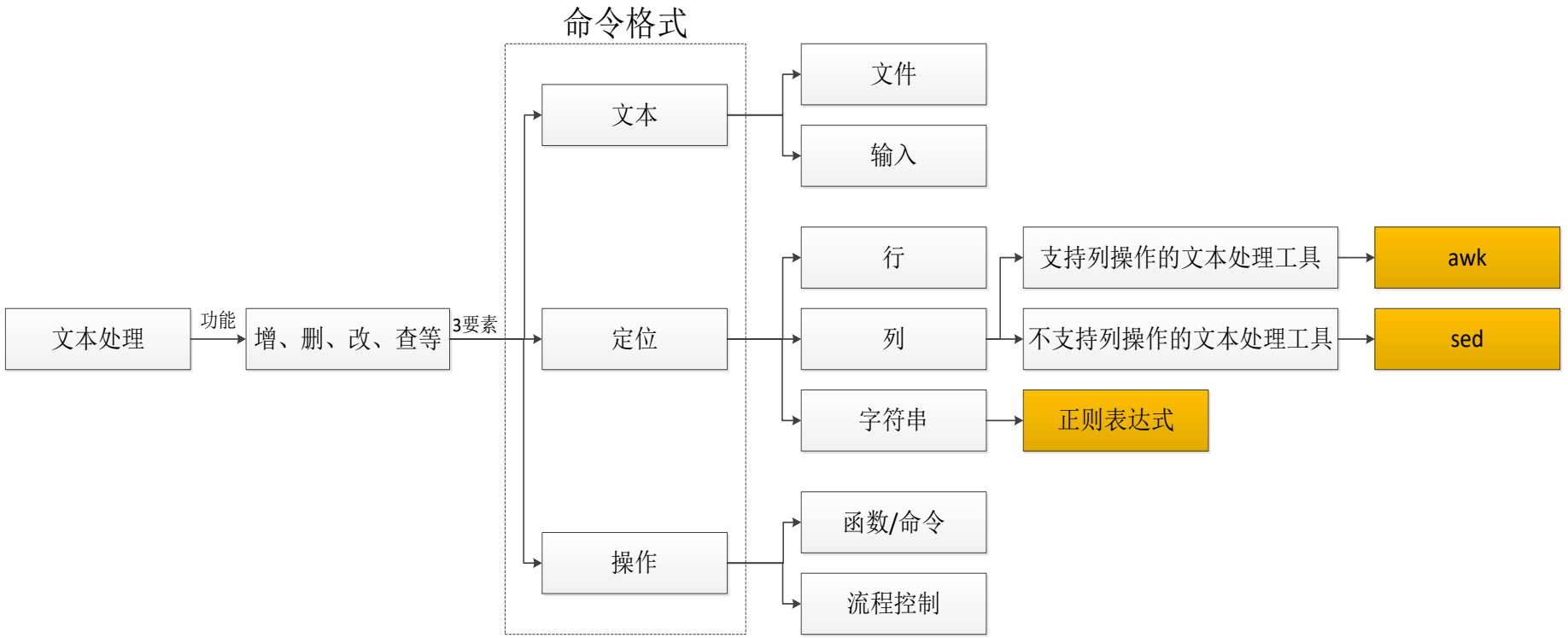
知识体系-1 文本处理做什么



在文本中“定位”时，awk和sed对“列操作”的支持有明显差异

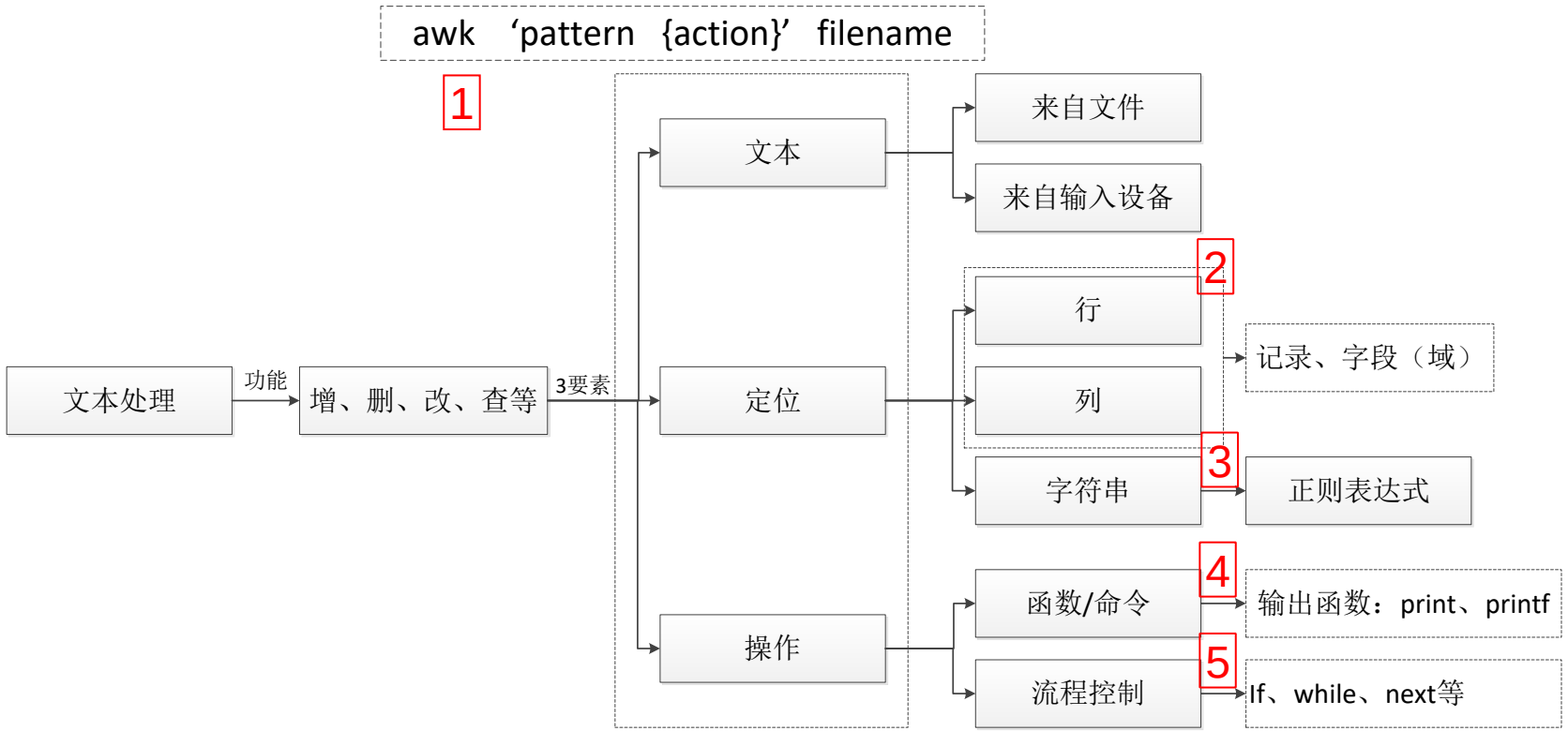
知识体系-2 awk、sed、正则表达式的关系

- 关系：
  - awk、sed的联系：都是文本处理工具
  - awk、sed的区别：awk支持列操作；sed不支持列操作
  - 正则表达式：是awk、sed定位到要处理字符串的模式



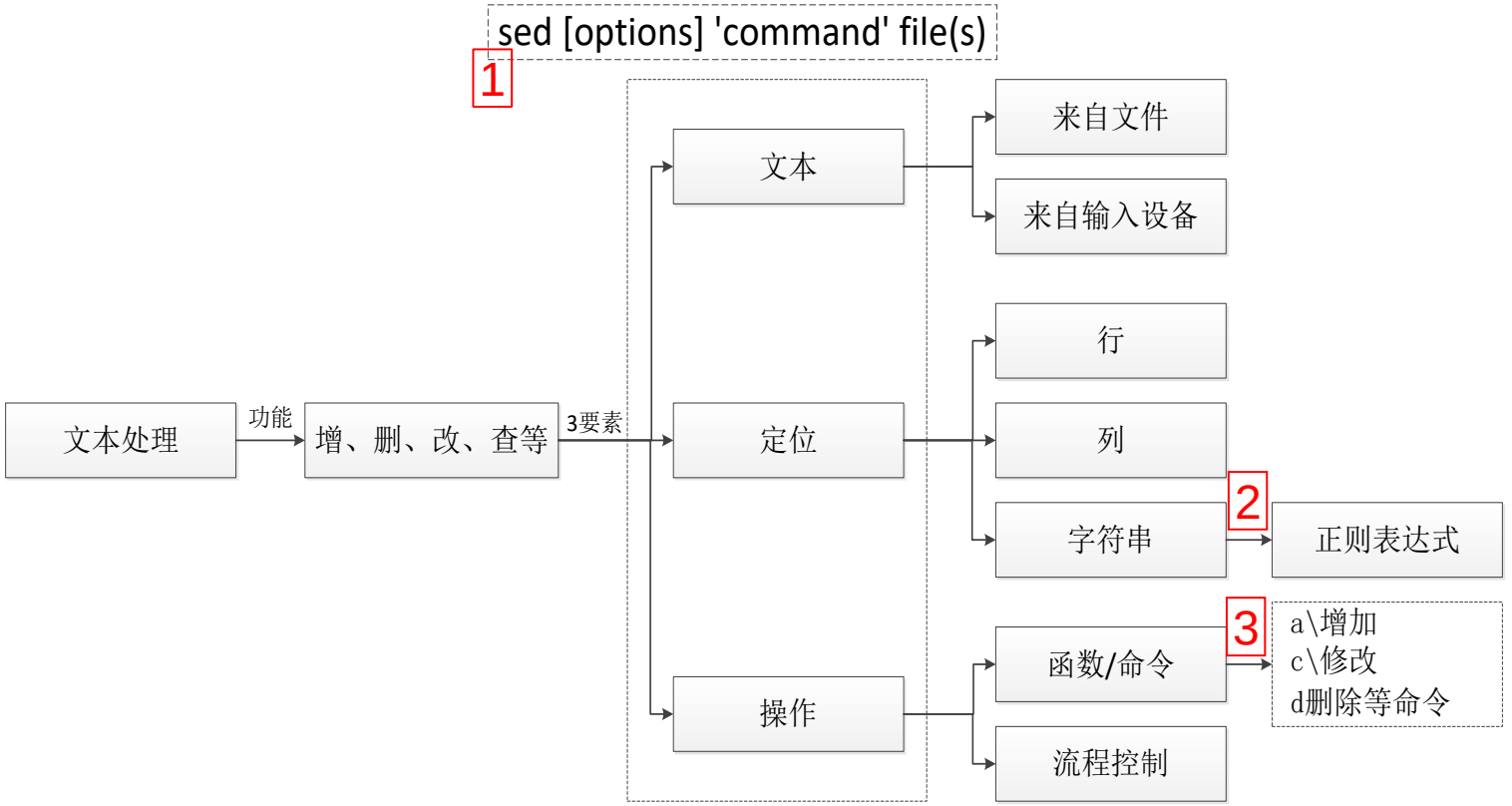
知识体系-3 awk的知识体系

- awk讲什么：



知识体系-4 sed的知识体系

- sed讲什么：



什么是awk/gawk？

- awk是一种程序设计语言，主要用来处理数据和产生报表。
  它对输入数据（文件、标准输入或命令的输出）逐行进行扫描，匹配制定的模式(pattern)，并执行制定的操作(action)。

- gawk是GNU下开发的awk，经过不断改进和更新，现已包含awk的所有功能。
  awk/gawk的主要功能就是处理文本文件的数据，它是通过自动将变量分配给每行的每个数据元素实现这一功能。

awk/gawk编程基础知识——记录、字段(域）和规则

- 记录

  记录是单个的、连续长度的输入数据，是awk的操作对象。记录由记录分隔符限定，记录分隔符是一个字符串，并且定义为RS变量。在缺省情况下，RS的值设置为换行符，所以awk的缺省行为是将整行输入作为记录。

- 字段(域)

  将每个记录进一步分解为称作字段的单独的块。字段受字段分隔符FS限定。缺省的字段分隔符是任意数量的空白字符，包括制表符和空格字符。所以在缺省情况下，将输入行进一步分解为单独的单词（由空白字符分隔的任何字符组）

- 规则

  规则是一些模式，后面跟着由换行分隔的操作。当awk执行一条规则时，它在输入记录中搜索给定模式的匹配项，然后对这些记录执行给定的操作（可以在规则中省略模式或操作）：/pattern/ { action }

记录、字段（域）

	$1	$2	$3	$4	$5	
NR=1	Tom	Jones	4424	5/12/66	543354	NF=5
NR=2	Mary	Adams	5436	11/4/63	28765	NF=5
NR=3	Sally	Chang	1654	7/22/54	650000	NF=5
NR=4	Billy	Black	1683	9/23/44	336500	NF=5

$0: 整个记录(整个行)

$1: 记录中的第一个域

$2: 记录中的第二个域

….

NF: 记录中域的个数

NR: 输入数据中的记录号

awk/gawk的基本格式

- 语法格式
  awk/gawk ‘pattern {action}’ filename
- awk/gawk扫描filename中的每一行，对符合模式pattern的行执行操作action。
- awk/gawk程序是由多个pattern与action所组成，action写在{}中，一个pattern后面跟一个Action。
  Pattern（1） {Action（1）}
  Pattern（2） {Action（2）}
  Pattern（3） {Action（3）}
- 特例:
  ① gawk ‘pattern’ filename
  显示所有符合模式pattern的行
  ② gawk ‘{action}’ filename
  对所有行执行操作action

Pattern

- 输入的记录符合Pattern，其相对应的Action才会被执行。
- gawk中有如下 几种Pattern:

（1）/regular expression/:正则表达式，是一个用斜线包围的Pattern

（2）expression:当该表达式的值不为 0 或一个不是空的字串，则可视为符合

（3）其它Pattern：

对以逗号分开的（如pat1,pat2）,指定记录的范围。

BEGIN和END是特别的Pattern，gawk在开始执行或要结束时分别执行对应BEGIN或END的Action。

空的Pattern表示每个输入记录皆视为符合Pattern。

正则表达式

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。

给定一个正则表达式和另一个字符串，可以达到如下的目的：

给定的字符串是否符合正则表达式的过滤逻辑（称作“匹配”）

可以通过正则表达式，从字符串中获取我们想要的特定部分”

正则表达式的特点是：

灵活性、逻辑性和功能性非常的强

可以迅速地用极简单的方式达到字符串的复杂控制

使用正则表达式应用举例

验证用户名不能包含数字和特殊字符:

```
var fname = *****//用户名
for(i=0;i<fname.length;i++){
        var ftext = fname.substring(i,i+1);
        if(ftext < 9 || ftext > 0)
        {
                alert("用户名非法");
                return false
        } else{
                alert("用户名有效！ ");
                return true
        }
 }
```

图1

正则验证：

```
var fname = *****//用户名;
var patternString=/^[a-zA-Z]*$/
var boolValue=
    patternString.test(fname)
if(boolValue==false)
{
    alert("用户名非法");
}else{
    alert("用户名有效！ ");
}
```

图2

正则表达式

- 字符
- 字符串
- 元字符
  - 位置匹配字符
  - 频率匹配字符
  - 字符串匹配字符

正则表达式——位置匹配字符

- \b 单词边界
- ^ 字符串的开始
- $ 字符串结束
- (?=pattern) 正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串
- (?! pattern) 正向否定预查，，在任何不匹配pattern的字符串开始处匹配查找字符串

正则表达式——位置匹配字符举例

- \b匹配一个单词边界，是指单词和空格间的位置。

  例1：“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。

- (?=pattern) 正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。

  例2：“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。

- (?!pattern)正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。

  例3：“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。

正则表达式——频率匹配字符

- *　　0到无数次
- +　　1到无数次
- ?　　0或者1 次
- {n}　　重复N次
- {n,}　　重复至少N次
- {n,m}　　n到m次
- []　　字符组，字符范围
- ()　　捕获组（子表达式）

正则表达式——频率匹配字符举例

- 例1：email地址的正则表达式可以写成

  /^[a-zA-Z0-9_]+@[a-zA-Z0-9_]+.[a-zA-Z0-9_]+$/

  - 给定首先这个email的正则表达式中的3个中括号都表示是字符范围，
  - +号表示出现1次或者多次

  ps：中括号[]里面的a-zA-Z0-9_可以用\w替代，表示的是数字、字母和下划线。

- 例2：匹配腾讯QQ号：[1-9][0-9]{4,}

  这里的{4,}就是指重复出现至少4次。

正则表达式——字符串匹配字符

- . 除换行以外的其他任意字符
- \s 空白字符——[\f\n\r\t\v]
- \S 除空白字符以外的任意字符—— [^ \f\n\r\t\v]
- \w 字母、数字、下划线——[A-Za-z0-9_]
- \W 除了字母、数字、下划线以外的任意字符—— [^A- Za-z0-9_]
- \d 数字 0-9——[0-9]
- \D 除了数字之外的任意字符——[^0-9]
- ……

正则表达式——字符串匹配字符举例

- 例1：email地址的正则表达式可以写成
  /^[a-zA-Z0-9_]+@[a-zA-Z0-9_]+.[a-zA-Z0-9_]+$/
  前面讲过的eamil地址的正则表达式就可以写成/^\w+@\w+.\w+$/

- 例2：匹配身份证

\d{15}|\d{18},中国的身份证为15位或18位，这里的\d就是数字的含义。

Action

- 对所读取的记录进行某种特定的操作，由一条或多条语句或命令组成，语句、命令之间用分号进行分隔。
- Action中的算式有算术、比较、布尔、条件。
- Action中的语句可以是流程控制语句：

（1）if条件判断语句

（2）while循环语句

（3）do-while循环语句

（4）for循环语句

（5）break语句

（6）continue语句

（7）next、next file 、exit语句

Action流程控制语句—if表达式

- if 表达式

  if 表达式的语法如下:

```
if (expression){
        commands
}
else{
        commands
 }
```

- 例如:

```
# a simple if loop
(if ($1 == 0){
        print "This cell has a value of zero"
 }
else {
        printf "The value is %dn",  $ 1
} )
```

Action流程控制语句—while循环

- while 循环
  while 循环的语法如下:
  while (expression){
  commands
  }
- 例如:
  awk‘{i = 1; while (i<NF) {print NF,$i,i++}}’ test

Action流程控制语句—next、exit

- next 和exit

  next 指令用来告诉gawk 处理文件中的下一个记录，而不管现在正在做什么。语法如下：

  ```
  {
      command1
      command 2
      command 3
      next
      command 4
  }
  ```

- 程序只要执行到next指令，就跳到下一个记录从头执行命令。command4指令永远不会被执行。
- 程序遇到exit指令后，就转到程序的末尾去执行END，如果有END的话。

自动内部变量

- gawk的每次执行，都建立了一些缺省的变量，也叫做内部变量，这些变量有固定的名字和固定的含义，它们在程序运行期间可以随时被引用。其具体定义如下：

FS:	输入记录字段间的分隔符(默认是空格和制表符)
RS:	输入记录的分隔符(默认是NEWLINE)
OFS:	输出记录字段间的分隔符(默认是空格)
ORS:	输出记录的分隔符(默认是NEWLINE)
NR:	当前行数
NF:	当前记录字段数
ARGC:	命令行参数个数
ARGV:	命令行参数数组

自动内部变量--OFS

```
jl@ubuntu:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
```

图1

```
jl@ubuntu:~$ awk -F: -v 'OFS=**' '{print $1,$5}' /etc/passwd
root**root
daemon**daemon
bin**bin
sys**sys
sync**sync
games**games
man**man
lp**lp
```

图2

OFS是输出记录字段间的分隔符。

例子：

① awk –F: -v ‘OFS=**’ ‘{print $1,$5}’ /etc/passwd

图2中给出一个以**为OFS的例子。

在/etc/passwd文件匹配以冒号为记录字段分隔符，**为OFS输出记录字段分隔符。图2显示打印出了第一个字段和第五个字段。

自动内部变量—ARGC、ARGV

① awk ‘BEGIN{print ARGV[0]}’

```
jl@ubuntu:~$ awk 'BEGIN {print ARGV[0]}'
awk
```

②

gawk常用输出函数

1、print函数:

用于不需要复杂格式的简单输出,
print 格式: print item1, item2, ...

例如:

$ ps -e | gawk '/ 05 / {print "tty05: " $4, $1}'

(查看5号终端上的用户现在在干什么及其PID)

```
lic@lic-PC:~$ ps -e |gawk '/05/{print "tty05:" $4,$1}'
tty05:ksoftirqd/0 3
tty05:ksoftirqd/1 13
tty05:ksoftirqd/2 18
tty05:ksoftirqd/3 23
tty05:cupsd 1053
tty05:NetworkManager 1057
tty05:ssh-agent 2058
tty05:wineserver 7225
lic@lic-PC:~$
```

图1

2、printf函数:
高级格式化输出函数. 用法与C语言中相同.
格式: printf (format, item1, item2, ...)
实例: ps | gawk '{if(NR!=1)printf "PID:%5d\t\t%s\n",$1,$4}'

```
lic@lic-PC:~$ ps
  PID TTY          TIME CMD
 7438 pts/0    00:00:00 bash
 7532 pts/0    00:00:00 ps
lic@lic-PC:~$ ps | gawk '{if(NR!=1)printf  "PID:%5d\t\t%s\n",$1,$4}'
PID:  7438               bash
PID:  7533               ps
PID:  7534               gawk
lic@lic-PC:~$
```

图2

输入

1、从标准输入设备(键盘)输入

格式:　　gawk ‘pattern {action}’

由于未指定输入数据来源, 缺省情况下从标准输入设备(键盘)读取数据. 键盘上每输入一行, gawk就处理一行, 直到结束进程。

例:

```
lic@lic-PC:~/temp$ gawk '/abc/ {printf "%s\t%d-----gawk print\n" , $0, NF}'
sda
sada
abc asd
abc asd 2-----gawk print
abcdad
abcdad  1-----gawk print
^C
lic@lic-PC:~/temp$
```

图1

2、从其它命令输入

格式:　command | gawk ‘pattern’

command | gawk ‘{action}’

command | gawk ‘pattern {action}’

例:

```
lic@lic-PC:~/temp$ who
lic      tty7         2014-04-11 08:28 (:0)
lic      pts/0        2014-04-11 10:09 (:0)
lic@lic-PC:~/temp$ who | gawk '/tty7/{print $1}'
lic
lic@lic-PC:~/temp$
```

图2

gawk命令文件

格式:

gawk -f awk_file data_file

当需要对输入数据中的一行执行多项操作时, 常把这些操作命令放在一个命令文件中, 而不是在命令行上发出.

awk运行时, 对输入数据中的每一行执行命令文件中的所有操作后, 再对下一行数据进行同样的处理过程, 依此类推, 直到输入数据的最后一行.

例:

```
lic@lic-PC:~/temp$ cat my_awk
/Sally/{print "***found Sally!**\n"  $1,"",$2,$3}
lic@lic-PC:~/temp$ gawk -f my_awk employees
***found Sally!**
Sally  Chang 1654
lic@lic-PC:~/temp$
```

什么是Sed?

sed(stream editor)是一种在线编辑器，它一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。文件内容并没有改变，除非你使用重定向存储输出。

sed特点：

1、可以编辑一个或多个文件

2、简化对文件的反复操作

3、由于一次处理一行，读非常大的文件不会出问题，如果全部读取可能会内存溢出或处理速度非常慢。

Sed命令

指令形式：　　sed [options] 'command' file(s)

　　sed 's/test/mytest/' sed.txt

脚本文件形式：　sed [options] -f scriptfile file(s)

　　sed -f cmd.sed test.txt

常用选项：

- -e command, --expression=command
  允许执行多条命令编辑。
- -n, --quiet, --silent
  取消默认输出。
- -f, --filer=script-file
  引导sed脚本文件名
- -i
  直接修改读取的档案内容，而不是由屏幕输出
- --help　display this help and exit
  显示帮助信息并退出
- --version　output version information and exit
  显示sed版本信息并退出

Sed实例——多命令编辑e命令

sed -e '1,2d' -e 's/test/check/' sed.txt

-e选项允许在同一行里执行多条命令。

第一条命令删除1至2行，第二条命令用check替换test。命令的执行顺序对结果可能有影响。如果两个命令都是替换命令，那么第一个替换命令将影响第二个替换命令的结果。

Sed实例——多命令编辑e命令

还可以使用“;”来执行多条命令，结果一样

1 sed -e '1,2d;s/test/check/' sed.txt

sed --expression='s/test/check/' --expression='/love/d' sed.txt

一个比-e更好的命令是--expression。它能给sed表达式赋值。

同一匹配模式使用多个命令

2 sed –n ‘/test/ {=;l}’ sed.txt

使用{}可以执行多个命令，如果不加{},效果是不一样的

sed –n ‘/test/ =;l’ sed.txt

Sed实例——Sed脚本

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

cmd.sed

```
lic@lic-PC:~/temp$ cat cmd.sed
#n
1,$ =
#s/test/mytest/g;p
/test/ {s//mytest/g;p}
```

图2

sed -f cmd.sed test.txt

```
lic@lic-PC:~/temp$ sed -f cmd.sed sed.txt
1
2
3
4
mytest11mytest
5
6
7
8
mytest   xxxx
9
10
11
12
lic@lic-PC:~/temp$
```

① ② 图3

元字符集

	元字符		
1	^	锚定行的开始	/^sed/匹配所有以sed开头的行
1	$	锚定行的结束	/sed$/匹配所有以sed结尾的行
2	.	匹配一个非换行符的字符	s.d/匹配s后接一个任意字符，然后是d，像 sed/sad/std等都能匹配
2	*	匹配零或多个字符	/ *sed/匹配所有模板一个或多个空格后紧跟sed的行
3	[]	匹配一个指定范围内的字符	/[Ss]ed/匹配sed和Sed
3	[^]	匹配一个不在指定范围内的字符	/[^A-RT-Z]ed/匹配不包含A-R和T-Z的一个字母开头，紧跟ed的行
4	\(..\)	保存匹配的字符	s/\(love\)able/\1rs，loveable被替换成lovers
5	&	保存搜索字符用来替换其他字符	s/love/**&**/，love被替换成**love**
6	\<	锚定单词的开头	/\<love/匹配包含以love开头的单词的行
6	\>	锚定单词的结束	/love\>/匹配包含以love结尾的单词的行
7	x\{m\}	重复字符x，m次	/o\{5\}/匹配包含5个o的行
7	x\{m,\}	重复字符x,至少m次	/o\{5,\}/匹配至少有5个o的行
7	x\{m,n\}	重复字符x，至少m次，不多于n次	/o\{5,10\}/匹配5--10个o的行

Sed的基本命令

- a\　　在当前行后面加入一行文本。
- c\　　用新的文本改变本行的文本。
- d　　从模板块（Pattern space）位置删除行。
- i\　　在当前行上面插入文本。
- l　　列出非打印字符。
- p　　打印模板块的行。
- q　　退出Sed。
- r file　从file中读行。
- w file　写并追加模板块到file末尾。
- !　　表示后面的命令对所有没有被选定的行发生作用。
- y　　表示把一个字符转换为另外的字符
- =　　打印当前行号码。
- s/re/string　　用string字符串替换正则表达式re匹配的字符串。

Sed实例——p(显示),=(打印行号)

sed -n ‘2p’ sed.txt

sed -n ‘1,3p’ sed.txt

sed -n ‘/hello/p’ sed.txt

sed -n ‘1,$p’ sed.txt

sed -n ‘/hello/=’ sed.txt

sed -n ‘1,$=’ sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

```
① lic@lic-PC:~/temp$ sed -n '2p' sed.txt
123
lic@lic-PC:~/temp$ sed -n '1,3p' sed.txt
② hello sed
123
abc
lic@lic-PC:~/temp$ sed -n '/hello/p' sed.txt
③ hello sed
lic@lic-PC:~/temp$ sed -n '1,$p' sed.txt
④ hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图2

```
① lic@lic-PC:~/temp$ sed -n '/hello/=' sed.txt
1
lic@lic-PC:~/temp$ sed -n '1,$=' sed.txt
② 1
2
3
4
5
6
7
8
9
10
11
12
lic@lic-PC:~/temp$
```

图3

Sed实例——匹配元字符

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed -n '/^$/=' sed.txt

sed -n '/A\{4,\}/ p' sed.txt

sed -n '/.test/p' sed.txt

sed -n '/ *test/p' sed.txt

```
lic@lic-PC:~/temp$ sed -n '/^$/=' sed.txt
6
lic@lic-PC:~/temp$ sed -n '/A\{4,\}/p' sed.txt
AAAAAAAAA
lic@lic-PC:~/temp$ sed -n '/.test/p' sed.txt
test11test
lic@lic-PC:~/temp$ sed -n '/ *test/p' sed.txt
test11test
test   xxxx
lic@lic-PC:~/temp$
```

① ② ③ ④

图2

Sed实例——a\(附加文本),i\(插入文本)

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed '3a\test' sed.txt

```
lic@lic-PC:~/temp$ sed '3a\test' sed.txt
hello sed
123
abc
test
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图2

sed /test/'i\insert' sed.txt

```
lic@lic-PC:~/temp$ sed /test/'i\insert' sed.txt
hello sed
123
abc
insert
test11test
456     xxxx

DDDvvvv
insert
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图3

Sed实例——c\(修改文本), d(删除文本)

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed /test/'c\modify' sed.txt

```
lic@lic-PC:~/temp$ sed /test/'c\modify' sed.txt
hello sed
123
abc
modify
456     xxxx

DDDvvvv
modify
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

① ②

图2

sed ‘/test/d’ sed.txt

```
lic@lic-PC:~/temp$ sed '/test/d' sed.txt
hello sed
123
abc
456     xxxx

DDDvvvv
loveable
10
AAA
AAAAAAAAA
```

①

图3

sed '2,$d' sed.txt

```
lic@lic-PC:~/temp$ sed '2,$d' sed.txt
hello sed
lic@lic-PC:~/temp$
```

②

Sed实例——q(退出),l(列出非打印字符)

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed 'q' sed.txt

```
lic@lic-PC:~/temp$ sed '3q' sed.txt
hello sed
123
abc
lic@lic-PC:~/temp$
```

图2

sed -n 'l' sed.txt

```
lic@lic-PC:~/temp$ sed -n 'l' sed.txt
hello sed$
123$
abc$
test11test$
456\txxxx$
$
DDDvvvv$
test   xxxx$
loveable$
10$
AAA$
AAAAAAAAA$
lic@lic-PC:~/temp$
```

图3

Sed实例——y(替换对应的字符),!(取反命令)

sed y/abcd/ABCD/' sed.txt

第一个字符串(abcd)必须和第二个字符串(ABCD)长度相等

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

```
lic@lic-PC:~/temp$ sed 'y/abcd/ABCD/' sed.txt
hello seD
123
ABC
test11test
456     xxxx

DDDvvvv
test   xxxx
loveABle
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图 2

sed -n '/test/!p' sed.txt

```
lic@lic-PC:~/temp$ sed -n '/test/!p' sed.txt
hello sed
123
abc
456     xxxx

DDDvvvv
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图3

Sed实例——w命令

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456      xxxx

DDDvvvv
test    xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed '/test/w write1.txt' sed.txt

```
lic@lic-PC:~/temp$ sed '/test/w write1.txt' sed.txt
hello sed
123
abc
test11test
456      xxxx

DDDvvvv
test    xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$ cat write1.txt
test11test
test    xxxx
```

图2

sed '/test/p' sed.txt >write2.txt

```
lic@lic-PC:~/temp$ sed '/test/p' sed.txt >write2.txt
lic@lic-PC:~/temp$ cat write2.txt
hello sed
123
abc
test11test
test11test
456      xxxx

DDDvvvv
test    xxxx
test    xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图3

w命令和重定向>的作用是不同的：

- **w是将匹配的行写入文件中**
- **>是将打印到屏幕上的内容写到文件中**

Sed实例——r命令

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

read.txt:

```
lic@lic-PC:~/temp$ cat read.txt
readfile1
readfile2
lic@lic-PC:~/temp$
```

图2

sed '/test/r read.txt' sed.txt

将read.txt里的内容被读进来，显示在与test匹配的行后面，

如果匹配多行，则file的内容将显示在所有匹配行的下面。

```
lic@lic-PC:~/temp$ sed '/test/r read.txt' sed.txt
hello sed
123
abc
test11test
readfile1
readfile2
456     xxxx

DDDvvvv
test   xxxx
readfile1
readfile2
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

1

2

图3

Sed实例——替换命令s

在整行范围内把test替换为mytest。如果没有g标记，则只有每行第一个匹配的test被替换成mytest.

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed 's/test/mytest/' sed.txt

```
lic@lic-PC:~/temp$ sed 's/test/mytest/' sed.txt
hello sed
123
abc
mytest11test
456     xxxx

DDDvvvv
mytest   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

1

2

图2

sed ‘s/test/mytest/g’ sed.txt

```
lic@lic-PC:~/temp$ sed 's/test/mytest/g' sed.txt
hello sed
123
abc
mytest11mytest
456     xxxx

DDDvvvv
mytest   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

1

2

图3

Sed实例——替换命令s

如果在本身需要替换的文本当中包含了斜杠，可以把斜杠换成冒号，星号都是可以的。

比如：

sed ‘s/test/mytest/g’sed.txt

可以换成

sed ‘s:test:mytest:g’ sed.txt

或者换成

sed ‘s*test*mytest*g’ sed.txt

Sed实例——替换命令s

&符号表示替换字符串中被找到的部份。

sed.txt

```
lic@lic-PC:~/temp$ cat sed.txt
hello sed
123
abc
test11test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

图1

sed 's/^test1/&qq&qq/' sed.txt

```
lic@lic-PC:~/temp$ sed 's/^test1/&qq&qq/' sed.txt
hello sed
123
abc
test1qqtest1qq1test
456     xxxx

DDDvvvv
test   xxxx
loveable
10
AAA
AAAAAAAAA
lic@lic-PC:~/temp$
```

1 2

图2

love被标记为1，abl被标记为2，所有loveable会被替换成lovers ablity，最多存9个

sed -n 's:\(love\)\(abl\)e:\1rs \2ity:p' sed.txt

```
lic@lic-PC:~/temp$ sed -n 's:\(love\)\(abl\)e:\1rs \2ity:p' sed.txt
lovers ablity
lic@lic-PC:~/temp$
```

图3