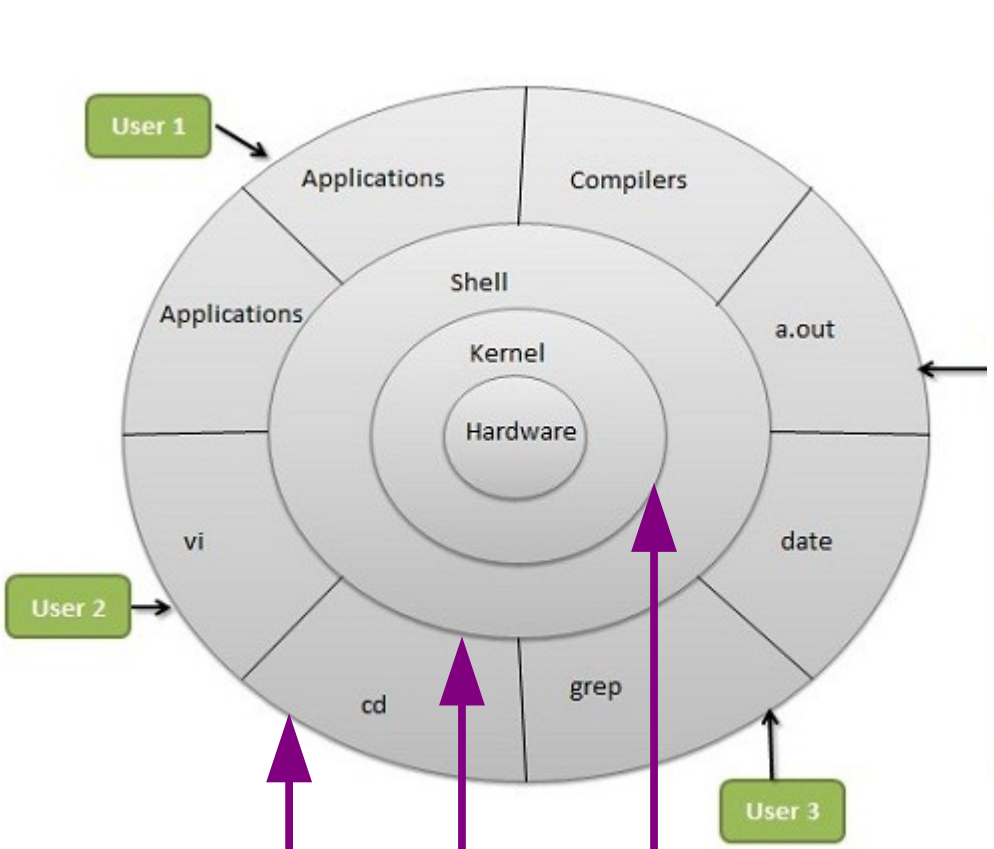
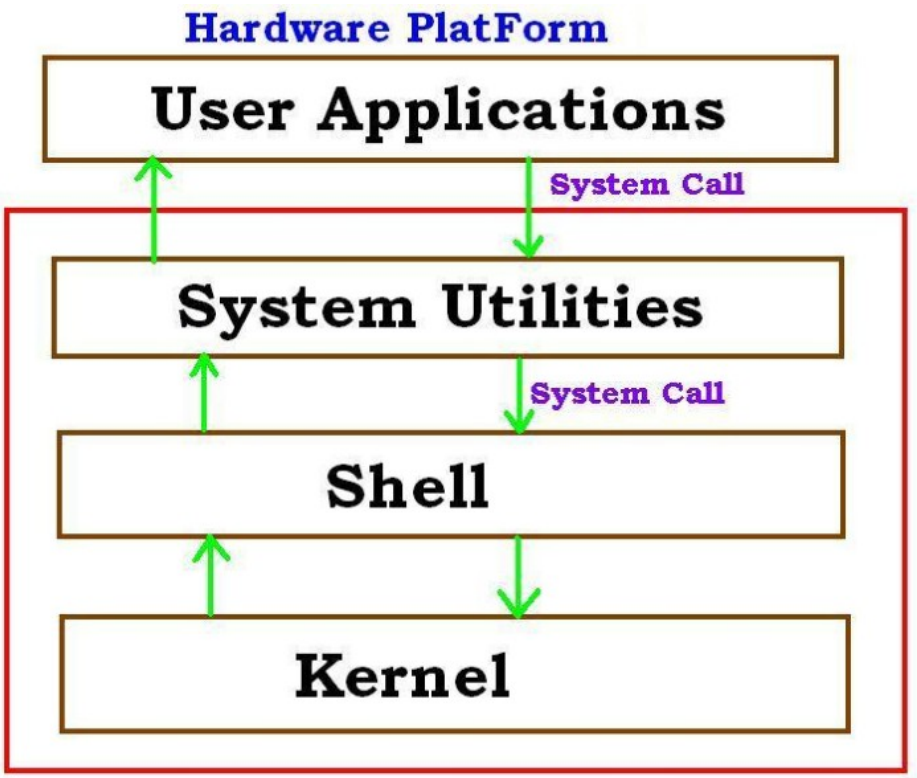

唐多令
林黛玉
粉墜百花洲，香殘燕子樓，一團團逐隊成球。漂泊亦如人命薄，空縷縷，說風流！
草木也知愁，韶華竟白頭，欵今生誰捨誰收？嫁與東風春不管，隨爾去，忍淹留！
原圖：淑芬繪　設計：衣芸
romance.myweb.hinet.net

Linux 学习路线图





第一境界 第二境界 第三境界

可编程过滤器 awk 与流编辑器 sed

李中国
lzg@suda.edu.cn
苏大计算机学院
2014 年 3 月 12 日

awk 语言简介

- 语言发明人
  - Alfred **A**ho（龙书作者；哥伦比亚大学教授）
  - Peter J. **W**einberger
    （原贝尔实验室科学家；现就职于Google）
  - Brian **K**ernighan
    （顶级技术作家；普林斯顿大学教授）
- 语言特点
  - 解释型；标准的Unix过滤器程序(programmable filter)
  - 擅长结构化文本数据处理及报表生成；执行速度快

awk 语言简介

标准输入：

```
Beth     4.00     0
Dan      3.75     0
Kathy    4.00     10
Mark     5.00     20
Mary     5.50     22
Susie    4.25     18
```

过滤器：

```
awk '$3 > 0 { print $1, $2 * $3 }'
```

标准输出：

```
Kathy 40
Mark 100
Mary 121
Susie 76.5
```

awk：为什么叫做可编程过滤器？

- awk 可以读标准输入并写标准输出，因此符合经典过滤器模式的程序定义
- 但是 awk 与 grep,tr,wc,sort,uniq 等功能单一的过滤器程序的最大不同之处，是它的文本过滤功能需要通过**用户自己编程**去实现，因此更加强大、丰富、灵活

```
cat emp.data | awk ' $3 > 0 { print $1, $2 * $3 } '
```

此处为用户编写的实现特定过滤功能的程序

```
cat emp.data | awk ' $3 > 0 && $2 >=4 { print $1 } '
```

用户编写的实现另一过滤功能的程序

awk 可编程过滤器的五种使用方式

数据文件 emp.data:

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

方式一

```
cat  emp.data  |  awk  '$3 > 0 { print $1, $2 * $3 } '
```

方式二

```
awk  '$3 > 0  { print $1, $2 * $3 } '  emp.data
```

方式三

```
awk   -f  programfile   emp.data
```

方式四

```
cat  emp.data  |  awk   -f  programfile
```

awk 可编程过滤器的五种使用方式

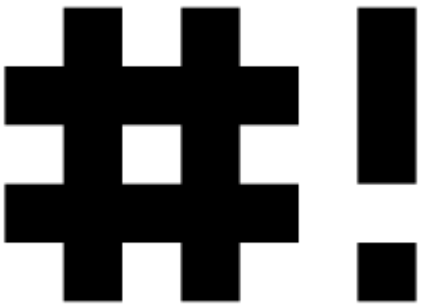
awk 源文件 programfile:

```
#!/usr/bin/awk -f
$3 > 0 { print $1, $2 * $3 }
```

方式五：

chmod +x programfile

cat emp.data | ./programfile



awk 可编程过滤器的五种使用方式

- 课外思考：文件 myprog 的内容如下，

```
#! /bin/cat
Hello World!
```

则执行以下命令序列后，输出结果是什么？

```
chmod +x myprog

./myprog
```

awk 程序的基本结构

- awk 程序 = “模式 + {动作}” 序列

文件 emp.data

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

逐行扫描，如果模式匹配则执行相应动作。

模式　　动作

```
$ awk '$3 == 0 { print $1 }' emp.data
Beth
Dan
$
```

awk 程序的基本结构：缺省情形

文件 emp.data

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

只有动作：默认模式为匹配所有行

```
awk '{print $1}' emp.data
```

```
Beth
Dan
Kathy
Mark
Mary
Susie
```

只有模式：默认动作为打印匹配的行

```
awk '$3 == 0' emp.data
```

```
Beth    4.00    0
Dan     3.75    0
```

awk 程序的特殊模式： BEGIN

```
BEGIN { print "NAME    RATE    HOURS";  print "" }
      { print }
```

```
NAME     RATE    HOURS

Beth     4.00     0
Dan      3.75     0
Kathy    4.00     10
Mark     5.00     20
Mary     5.50     22
Susie    4.25     18
```

awk 程序的特殊模式： END

文件 emp.data

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

```
$3 > 15 { emp = emp + 1 }
END     { print emp, "employees worked more than 15 hours" }
```

```
3 employees worked more than 15 hours
```

awk 程序的特殊变量：NR 与 NF

文件 emp.data

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

```
{ print NR, $0 }
```

```
1 Beth   4.00    0
2 Dan    3.75    0
3 Kathy  4.00    10
4 Mark   5.00    20
5 Mary   5.50    22
6 Susie  4.25    18
```

NR: 记录当前行号
NF: 当前行内字段数
$0: 当前整行的内容

awk 程序的特殊变量： NR 与 NF

文件 emp.data

```
Beth     4.00     0
Dan      3.75     0
Kathy    4.00     10
Mark     5.00     20
Mary     5.50     22
Susie    4.25     18
```

```
    { nc = nc + length($0) + 1
      nw = nw + NF
    }
END { print NR, "lines,", nw, "words,", nc, "characters" }
```

```
6 lines, 18 words, 77 characters
```

awk 控制结构

与 C 语言一样 ,awk 支持 if 语句、 for 循环、 while 循环等控制语句 (具有相同的语法):

```
{ for (i = 1; i <= NF; i = i + 1) if ($i < 0) $i = -$i
  print
}
```

判断一下上述 awk 程序所实现的功能。

awk 数组

以下是一个完整的 awk 程序，注意到其中使用了数组（不用事先声明）；这个程序的功能是什么？

```
    { line[NR] = $0 }  # remember each input line

END { for (i = NR; i > 0; i = i - 1)
          print line[i]
    }
```

流编辑器 sed

- sed = stream editor
- 流编辑器 sed
  - 它编辑的对象通常是Unix 管道中的文本流，故名
  - 诞生于 1973 — 1974 年，发明人是贝尔实验室的 Lee E. McMahon（毕业于哈佛大学）
  - 常常作为过滤器应用于管道之中，实现对文本的自动编辑处理；与 awk 类似，其编辑功能同样是可编程的(programmable)
- sed 是标准的过滤器模式的程序

  ... | sed 's/xxx/yyy/g' | ...

流编辑器 sed：替换命令 s

sed 最基本、最重要的命令 s ：文本替换

文件 input.txt

```
2014-03-03,37.92,38.13,37.49,37.78,29717500,37.78
2014-02-28,37.98,38.46,37.82,38.31,41215000,38.31
2014-02-27,37.45,37.89,37.23,37.86,33903400,37.86
2014-02-26,37.58,37.74,37.19,37.47,41041800,37.47
2014-02-25,37.61,37.85,37.35,37.54,30736500,37.54
2014-02-24,37.69,37.98,37.54,37.69,32085100,37.69
2014-02-21,37.94,38.35,37.86,37.98,38021300,37.98
2014-02-20,37.57,37.87,37.40,37.75,27526100,37.75
2014-02-19,37.22,37.75,37.21,37.51,29750400,37.51
```

```
cat input.txt | sed 's/,/\t/g' > output.xt
```

文件 output.txt

```
2014-03-03	37.92	38.13	37.49	37.78	29717500	37.78
2014-02-28	37.98	38.46	37.82	38.31	41215000	38.31
2014-02-27	37.45	37.89	37.23	37.86	33903400	37.86
2014-02-26	37.58	37.74	37.19	37.47	41041800	37.47
2014-02-25	37.61	37.85	37.35	37.54	30736500	37.54
2014-02-24	37.69	37.98	37.54	37.69	32085100	37.69
2014-02-21	37.94	38.35	37.86	37.98	38021300	37.98
2014-02-20	37.57	37.87	37.40	37.75	27526100	37.75
2014-02-19	37.22	37.75	37.21	37.51	29750400	37.51
```

流编辑器 sed：替换命令 s

标准输入：

```
one two three, one two three
four three two one
one hundred
```

过滤器：

```
sed  's/one/ONE/'
```

标准输出：

```
ONE two three, one two three
four three two ONE
ONE hundred
```

替换命令详解：

```
s         Substitute command
/../../   Delimiter
one       Regular Expression Pattern Search Pattern
ONE       Replacement string
```

流编辑器 sed：替换命令 s

正则表达式及替换文本的分割符 / 可以换成其它符号：

```
> head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
```

```
> head -5 /etc/passwd | sed 's|/bin/sh|/bin/bash|'
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/bash
bin:x:2:2:bin:/bin:/bin/bash
sys:x:3:3:sys:/dev:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
```

试比较：

```
head -5 /etc/passwd | sed 's/\/bin\/sh/\/bin\/bash/'
```

流编辑器 sed：替换命令 s

在**被替换字符串**中，可以用符号 & 表示前面正则表达式匹配的内容；这一特性在进行文本替换时非常有用：

```
> cat telephone.txt
4000001476
4000000049
4000000041
4000002754
4000000944
4000002549
4000000084
4000002745
4000000064
4000002547
4000002200
4000000543
```

```
> cat telephone.txt | sed 's/[0-9]\{3\}/(&)/'
(400)0001476
(400)0000049
(400)0000041
(400)0002754
(400)0000944
(400)0002549
(400)0000084
(400)0002745
(400)0000064
(400)0002547
(400)0002200
(400)0000543
```

流编辑器 sed：替换命令 s

思考：下列命令的输出结果是什么？

```
echo "123 abc" | sed 's/[0-9][0-9]*/& &/'
```

基础正则表达式与扩展正则表达式：上述命令等价于以下命令；二者区别请参考教材第 348-361 页。

```
echo "123 abc" | sed -r 's/[0-9]+/& &/'
```

流编辑器 sed: 替换命令 s

此外，在正则表达式及被替换字符换中，可用 \n 表示第几个匹配的字符串；匹配字符串如需参加编号，则用 \(和 \) 标志：

```
> cat telephone.txt | sed 's/\([0-9]\{3\}\)\([0-9]\{3\}\)/\1-\2-/'
400-000-1476
400-000-0049
400-000-0041
400-000-2754
400-000-0944
400-000-2549
400-000-0084
400-000-2745
400-000-0064
400-000-2547
400-000-2200
400-000-0543
```

流编辑器 sed：替换命令 s

同上例；唯一区别是使用了扩展正则表达式，其书写比基础正则表达式显得更简洁：

```
> cat telephone.txt | sed -r 's/([0-9]{3})([0-9]{3})/\1-\2-/'
400-000-1476
400-000-0049
400-000-0041
400-000-2754
400-000-0944
400-000-2549
400-000-0084
400-000-2745
400-000-0064
400-000-2547
400-000-2200
400-000-0543
```

流编辑器 sed：替换命令 s

思考：以下三个命令分别起什么作用？

```
echo abcd123 | sed 's/\([a-z]*\).*/\1/'
```

```
sed 's/\([a-z][a-z]*\) \([a-z][a-z]*\)/\2 \1/'
```

```
sed 's/^\(.\)\(.\)\(.\)/\3\2\1/'
```

流编辑器 sed：替换命令 s

检测英语文本中相邻的两个单词重复的情况（the the ...）：

```
sed -n '/\([a-z][a-z]*\) \1/p'
```

```
sed -n '/\([a-z]+\) \1/p'
```

把相邻的两个重复单词替换为一个：

```
sed 's/\([a-z]*\) \1/\1/'
```

流编辑器 sed：替换命令 s

替换命令可以使用全局替换标志 g

给第一个单词（或其它不含空格的字符串）加上括号：

```
sed 's/[^ ]*/(&)/' <old >new
```

给所有不含空格的字符串加上括号：

```
sed 's/[^ ][^ ]*/(&)/g' <old >new
```

流编辑器 sed：替换命令 s

以下命令使用了全局替换标记 g，它会不会导致死循环？

```
sed 's/loop/loop the loop/g' <old >new
```

提示：替换命令仅在原始输入文本上进行操作。

流编辑器 sed: 替换命令 s

除标记 g 外，替换命令还可以使用标记 /n ，用以具体描述替换发生的位置。

以下两个命令完成同样功能；是什么？

```
sed 's/\([a-zA-Z]*\) \([a-zA-Z]*\) /\1 /' <old >new
```

```
sed 's/[a-zA-Z]* //2' <old >new
```

以下命令的功能是什么？

```
sed 's/[a-zA-Z]* /DELETED /2g' <old >new
```

流编辑器 sed: 命令 d,p,q

命令 d,p,q 分别表示删除、打印、退出编辑程序

```
sed '/^$/ p'
```

```
sed '11,$ d' <file
```

```
sed -n '1,10 p' <file
```

```
sed -n '/match/ p'
```

Unix 哲学（之二）

- Brian W. Kernighan & Rob Pike 论 Unix 哲学：
  - First, let the machine do the work.
  - Second, let other people do the work.
  - Third, do the job in stages.
  - Fourth, build tools.
- 什么样的软件在 Unix 下可以称得上工具（software tools）？
  - 必须能够与其它程序配合；
  - 必须能够容易地进行自动化（写入无人职守的脚本）；

Unix 哲学（之二）

```c
int isPrime(int n)
{
    if (n <= 1) return 0;
    int i;
    for (i = 2; i * i <= n; i++)
        if (n % i == 0)
            return 0;
    return 1;
}


int nPrimes(int begin, int end)
{
    int total = 0;
    int i;
    for (i = begin; i <= end; i++)
        if (isPrime(i))
            total++;
    return total;
}
```

Unix 哲学（之二）

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int b = atoi(argv[1]);
    int e = atoi(argv[2]);
    printf("%d\n", nPrimes(b, e));
    return 0;
}
```

请同学们思考：

这两个程序哪一个更容易写到脚本里进行自动化运行？

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int b, e;
    scanf("%d",&b);
    scanf("%d",&e);
    printf("%d\n", nPrimes(b, e));
    return 0;
}
```

Unix 哲学（之二）

一个相对不太容易写入脚本进行自动化运行的例子（why?）：

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int b, e;
    printf("Please input two numbers:\n");
    scanf("%d", &b);
    scanf("%d", &e);
    printf("There are %d prime numbers between %d and %d\n",
           nPrimes(b, e), b, e);
    return 0;
}
```

教材阅读章节

- 复习
  - 第 12 章关于 awk 及 sed 的内容
  - 重点阅读 12.2 及 12.4
- 预习
  - 第 6 、 7 、 8 章：目录及文件系统
  - 第 3 、 4 章：磁盘分区介绍