正则表达式sed，awk

文本过滤器

- head
- tail
- grep
- sort
- uniq
- tr

使用grep

- grep是全局正则表达式打印的缩写
- grep nobody /etc/passwd显示 /etc/passwd 文件中带有“nobody”字样的一行，区分大小写
- grep -i noBOdy /etc/passwd ,意义同上，但不区分大小写
- grep -v nobody /etc/passwd ,显示/etc/passwd 中不含有 nobody 字样的行
- grep –n nobody /etc/passwd

grep命令

- grep select * ,列出当前目录下所有包含select 字样的文件名及含有select字样的行
- grep -h select * ,和上一条相似，但不显示文件名称.
- grep -l select * ,和第一条相似，但只显
- 示文件名.
- ls |grep nfs 只显示包含nfs 字样的文件

egrep

- egrep 是扩充功能的grep
- egrep ‘2|5’ 1.txt ,列出含有2或者5的行
- egrep '33(2|5)' 1.txt ,列出包含33,而且后边跟的是2或5的行

tr

- 如何显示一个单词在一个文件中出现的次数？wc？
- tr：把一组字符转化为另一组字符
- 例tr ‘set1’ ‘set2’：把set1中的所有字符转化成set2中的字符
- 1.用空格替换文件中的单词分割符

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd
- 2.把所有大写字母转化为小写字母

sort

tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’

- 3.压缩输出的空格：把多个空格压缩为一个空格

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’ |tr –s ‘ ‘

- 4.用sort命令对文件内容进行排序

  Sort是对每个输入行进行排序的，所以使用tr在每行显示一个单词——把所有空格转化为换行符：

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’ |tr –s ‘ ‘ | tr ‘ ’ ‘\n’

  -s:把多个空格压缩成一个空格

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’ | tr –s ‘ ’ | tr ‘ ’ ‘\n’ | sort

uniq

- 5.uniq对重复的行只保留一行

  uniq –c(count)打印一个字符出现的次数。

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’ | tr –s | tr ‘ ’ ‘\n’ | sort | uniq –c

  以上默认按照字母的顺序排序，再次使用sort对数字排序

  tr ‘!?”:;\[\]{}(),.\t\n’ ‘ ’ < /etc/passwd | tr ‘A-Z’ ‘a-z’ | tr –s | tr ‘ ’ ‘\n’ | sort | uniq –c | sort –rn | head

sort

- 默认sort排序第一列，如何让sort排序第二列：
- sort –k start,end files
- sort –rn –k 2,2 abc.txt
  第一个2表示第二列，第二2表示在第二列结束
- -n　　：使用纯数字排序（否则就会以文字型态来排序）
  -r　　：反向排序
  -u　　：相同出现的一行，只列出一次！

sed、awk语法

- cmd ‘script’ file
- cmd是awk和sed，script是能够被awk和sed理解的命令，files是cmd进行操作的一些文件
- 当一个awk和sed命令运行时，如下：

  1.从一个输入文件中读取一行

  2.做一个这一行的拷贝

  3.对这一行执行脚本

  4.到下一行，重复步骤1

script结构

- /pattern/action
- pattern是一个正则表达式，action是awk或sed在遇到pattern才去执行的操作
- 正则表达式是由一系列字符串的简洁的符号组成的，有普通字符和元字符组成

元字符

- . -- 用于匹配任意一个字符, 除了换行符
- *-- 用来匹配它前面字符的任意多次
- ^ -- 匹配行首
- $ -- 用来匹配行尾
- [char]-匹配中括号字符集中的某一个字符
- [^char]—匹配没有在char中的字符
- \ -- 用来转义某个特殊含义的字符
- "\<the\>" 完整匹配单词"the", 不会匹配"them", "there", "other",

有用的正则表达式

- /^$/ 空白行
- /^.*$/ 一整行
- /*/ 一个或多个空格
- /[a-zA-Z][a-zA-Z]*:\/\/[a-zA-Z0-9][a-zA-Z0-9\.]*.*/ 有效的URL

sed

- sed是一种对它的输入的每一行进行一系列操作的流式编辑器，sed被用来作为过滤器
- 语句结构:sed ‘script’ files
- script的形式如下：

  /pattern/action

  当pattern被忽略，action对输入的每一行进行操作

sed

- 命令p：打印
- sed p /etc/passwd
- sed '/222/p' filename
- 将所有的行打印，遇到有222字样的，多打印一行。
- sed –n ‘/0\.[0-9][0-9]$/p’ fruit_prices.txt
- -n:只打印符合条件的

sed

- 命令d：删除
  删除第5行，显示其他行
  sed '5d' filename
  删除1-3行，显示其他行
  sed '1,3d' filename
- sed '4,$d' filename
- 从第四行到最后行都被删除，剩下的打印。
- $代表文件的最后一行。逗号被称为范围运算符(range operator)

sed

- 取代命令:s
- sed 's/222/333/g' filename
- 将所有的222取代为333
- sed -n 's/222/333/p' filename
- s 代表置换，-n选项和命令末端的p标记告诉只打印发生置换的行。

sed

- 转换: y命令
- sed
- '1,3y/abcdefghijklmnopqrstuvwxyz/ABCDEFG
- HIJKLMNOPQRSTUVWXYZ/' filename
- 将文件的前三行转为大写。
- 将文件的所有行转为大写。
- sed
- '1,$y/abcdefghijklmnopqrstuvwxyz/ABCDEFG
- HIJKLMNOPQRSTUVWXYZ/' filename

sed

- #!/bin/bash
- for i in `ls *.txt`
- do
- j=` echo $i |
- sed'1,$y/abcdefghijklmnopqrstuvwxyz/ABCD
- EFGHIJKLMNOPQRSTUVWXYZ/' `
- mv $i $j
- done

sed

- /pattern1/s/pattern2/pattern3/ :1~3都是正则表达式，匹配规则pattern1的每一行中的pattern2被替换为pattern3
- s命令提供了&操作，使得在pattern3中可以重复使用匹配字符串pattern2
- 练习:使用id命令只打印uid号
- a：新增， a 的后面可以接字符串，而这些字符串会在新的一行出现(目前的下一行)

  # nl /etc/passwd | sed '2a drink tea'
- d ： 删除

  #nl /etc/passwd | sed '2,5d'

  删除档内第 10 行资料 , 则指令为 10d

  删除含有 "man" 字串的资料行时 , 则指令为 /man/d

  删除档内第 10 行到含 "man" 字串的资料行 , 则指令为 10,/man/d
- c ： 取代c 的后面可以接字符串，这些字符串可以取代 n1,n2 之间的行

  #nl /etc/passwd | sed '2,5c No 2-5 number'
- nl /etc/passwd | sed -n '5,7p'
- -i 编辑原文件(此选项慎用,如果使用则原文件就会被修改,无法恢复)。

- 插入的用法:
- 在文件file的第2行之前插入“xxx”

  sed '2i xxx' file
- 在file的第2行之后插入“xxx”

  sed '2a xxx' file
- 在file的第2行之后插入“xxx”

  sed '2a xxx' file
- 在file的第2行和第3行之前插入“xxx”
- sed '2,3i xxx' file
- 在file的第2行至最后一行之前插入“xxx”

  sed '2,$i xxx' file
- 总结：在指定行之前插入使用的是“i”，而在指定行之后插入是使用“a”

awk过滤文本

- awk是一种程序语言，用来处理数据和产生报告。
- awk 以她的三位作者的姓的第一个字母命名。
- awk对输入的每一行进行操作
- awk –version 显示版本
- awk的语法
  awk ‘script’ file
  file是一个或多个文件，script如下形式：
  /pattern/{actions}
  pattern是一个正则表达式，actions是命令
  pattern如果省略了如下形式：
  awk ‘{script}’ file

awk用法

- 打印文件内容
  awk ‘{print;}’ /etc/passwd
- 打印文件第三列
  awk '{print $3;}' filename
- 每一个字段就是一列，默认字段的分隔符是制表符和空格。用字段操作符来访问一个字段的值，第一个字段是$1.
- 打印文件中的第一列和第三列
  awk ‘{print $1$3;}’在输出字段中没有分割，为了在每个字段之间打印空格，如下操作：
  awk ‘{print $1,$3}’ 或
  awk ‘{print $1 “\t” $3}’
  last | awk '{print $1 "\t" $3}'

例子

- cat filename

fruit	price/lbs	quantity
Banana	$0.89	100
Peach	$0.79	65
Kiwi	$1.50	22
Pinea	$1.29	35
Apple	$0.99	78

awk用法

- 输入默认为右对齐，如果输出为左对齐则如下操作：

  awk ‘{printf “%s %-8s\n”,$3,$1;}’ filename
- 在输出的每一行后加一个标记(*)

  awk ‘{print $1,$2,$3,“*”;}’ filename

  或awk ‘{print $0,“*”;}’ filename

  注意两者的区别？

awk ‘/ *$[1-9][0-9]*\.[0-9][0-9]*/{print $0,“*”;} / *\$0\.[0-9][0-9]*/{print;}’ filename

awk中的比较操作符

- 比较操作符比较数字和字符串的值，如下：
- < <= > >= == !=
- value ~ /pattern/如果value匹配样式则为真
- value !~ /pattern/如果value匹配样式则为假
- 语法格式：
  表达式{actions;}

cat /etc/passwd | awk '{FS=":"} $3 < 10 {print $1 "\t " $3}‘

cat /etc/passwd | awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t " $3}'

- awk ‘
  $3 <=75 {printf “%s\t%s\n”,$0,“REORDER”;} $3 >75 {print $0;}’
  filename

混合表达式

- *(表达式1)* && *(表达式2)*
- *(表达式1)* || *(表达式2)*

  awk ‘

  ($2 ~ /^$[1-9][0-9]*\.[0-9][0-9]$/) && ($3 < 75) {printf “%s\t%s\t%s\n”,$0,“*”,“REORDER”;}’ filename

列出价格大于1美元且数量小于75的水果

next命令

- awk ‘
  $3 <=75 {printf “%s\t%s\n”,$0,“REORDER”;}
  $3 >75 {print $0;}’ filename
- 思考上面的脚本对于下面这行如何处理？
  Kiwi　$1.50　22
- 为了使脚本执行效率更高需要next命令：
  awk ‘
  $3 <=75 {printf “%s\t%s\n”,$0,“REORDER”;next;} $3 >75 {print $0;}’ filename

awk例子

- $ls -l | awk ‘$1 !~ /total/ {printf “%-32s %s\n”,$8,$5}’

awk中的数值操作符

- + - * / %
- 如果*num1*和*num*2的值为字符串，则awk使用值0，如果使用一个还没有定义的变量，awk会创建这个变量并且给它赋值0
- 统计文件中空行数：

```
if [ -f  $1 ];then
        echo $1
        awk ‘/^*$/{x=x+1;print x;}’ $1
else
        echo “ERROR:$1 not a file ” >&2
fi
```

BEGIN和END

- 上面的例子中

  awk ‘/^$/{x=x+1;print x;}’ $1

  x的值每次增加就打印一次，如果打印总数？使用BEGIN和END样式匹配
- awk ‘

  BEGIN {actions}

  /pattern/ {actions}

  END {actions}’ filename

awk在读入任何行之前先执行BEGIN的actions

awk在退出之前执行END的actions

例子

```
if [ -f  $1 ];then
        echo –n “$1”
        awk ‘/^*$/{x=x+1;next;}
                END {printf “ %s\n”,x}’ $1
   else
        echo “ERROR:$1 not a file ” >&2
fi
```

awk的内部变量

- FILENAME当前输入文件的名称
- NR当前输入文件的总行数
- NF当前行($0)记录中的字段数
- OFS输出字段的分隔符
- FS输入字段的分隔符
- 列出每一行的账号，并且列出目前处理的行数，该行有多少字段

  last |tail| awk '{print $1 "\t lines: " NR "\t columes: " NF}'

NR

- 统计空行在文件中的百分比
- If [-f $1];then

```
    awk ‘
    /^$/{file=FILENAME;x=x+1;next;}
    END {printf “%s %s
    %3.1f\n”,file,x,(100*(x/NR))}’ $1
else
    echo “ERROR”
fi
```

FS

- FS默认是空格和制表符
- 改变FS的默认值
- awk ‘BEGIN {FS=“:”;} {print $1,$6;}’ /etc/passwd 或

  awk –F: ‘{print $1,$6;}’ /etc/passwd

awk中的流程控制

- *if (expr1) {*

  action1

  }else if (expr2) {

  action2

  } else {

  action3

  }

if例子

- awk ‘{printf “%s\t”$0;
  if ($2 ~ /\$[1-9][0-9]*\.[0-9][0-9]/) {printf “ * ”;
  if($3<=75){
  printf “REORDER\n”;
  } else {
  printf “\n”;
  }
  }else {
  if ($3< 75) {
  printf “ REORDER\n”;
  } else {
  printf “\n”;
  }
  }
  }’ filename

while

- while (expr) {
  actions
  }
- awk ‘{x=NF;
  while (x > 0){
  printf(“%16s ”,$x);
  x=x-1;
  }
  print “”;
  }’ filename

do语句

- do {
  actions
  }while (expr)
- do语句至少执行一次
- awk ‘{
  x=NF;
  do {
  printf(“%16s ”,$x);
  x=x-1;
  } while(x>0);
  print “”;
  ’ filename

for语句

- for循环用于反复读取记录中的字段并输出它们
- awk ‘

```
        for (x=1;x<=NF;x=x+1) {
                printf “%s ”,$x;
        }
        printf “\n”;
}’ filename
```

cut命令

- cut 主要的用途在于将一行里面的数据进行分解，最常使用在分析一些数据或文字数据的时候！这是因为有时候我们会以某些字符当作分割的参数，然后来将数据加以切割，以取得我们所需要的数据。
- 命令格式：

  cut -d "分隔字符" [-cf] fields

  -d ： 后面接的是用来分隔的字符，默认是空格符
  -c ： 后面接的是第几个字符
  -f ： 后面接的是第几个区块（列）？

  cat /etc/passwd | cut -d ":" -f 1

  last | cut -d “ ” -f1 //以空格符为分隔，并列出第一列

  last | cut -c1-20 //将 last 之后的数据，每一行的 1-20 个字符取出来

tee命令

- 将数据输出重定向到文件的时候，屏幕上就不会出现任何的数据！那么如果我们需要将数据同时显示在屏幕上跟档案中呢？这个时候就需要 tee 这个指令。
- last | tee last.list | cut -d " " -f1

split命令

- 将大的文件分割成小的文件
  -b size 来将一个分割的文件限制其大小
  -l line以行数来分割
- split -l 5 /etc/passwd test　<==会产生 testaa, testab, testac... 等等的小文件

一些常用工具

- type打印一个命令的绝对路径
- sleep暂停给定的秒数，sleep *n*
- for i in $(seq1 10)

  do

  ```
      echo –e “\a”
      sleep 2
  ```

  done

find命令

- find / -name –print
- find / -name “*apple* -print
- find /home –name “[!abc]”
- find /home –name “?[1-9]”
- find / -type d(f b c l p)
- find / -mtime -5 查找最后修改小于5天的文件
  -mtime文件修改时间//vi、echo修改文件会改变此文件时间
  -atime文件访问时间//用cat、vi查看会改变此文件时间
  -ctime文件改变时间// vi 、echo修改文件、chmod、chown会改变此文件时间
- stat fileanme查看以上3个时间

find

- find / -size +2000
- find / -name “*apple*” –type f –size +50 –mtime -3 –print
- find / \(-size +50 –o –mtime -3 \) –print
- -exec动作允许用户对每一个匹配成功的文件来指定命令运行
- find / -name apple –exec chmod a+r {} \;
- find / -name apple –exec rm -f {} \;

xargs

- xargs从标准输入中接受单词列表的命令，并把这些单词作为参数传递给指定的命令
- ls |xargs rm//把ls列出的内容再通过rm删除
- 可以每次删除一部分文件
- ls |xargs –n 2 rm
- ls | xargs –n 2 echo “==>”
- 有时目录里文件太多了不能使用rm *删除这时

  ls | grep ‘^abc’ |xargs –n 20 rm

bc命令

- bc命令:进行算术运算，但不限于整数运算
- $bs
  scale=4 //表示计算结果保留4位小数
  2+3
  2-3
  2*3
  8/3
  quit
- bc还可以用在shell变量赋值中
  ABC=`echo “scale=4;8/3” | bc`
  echo $ABC