# 前言

一个语言的构成要素是语法模式+自身提供的内容

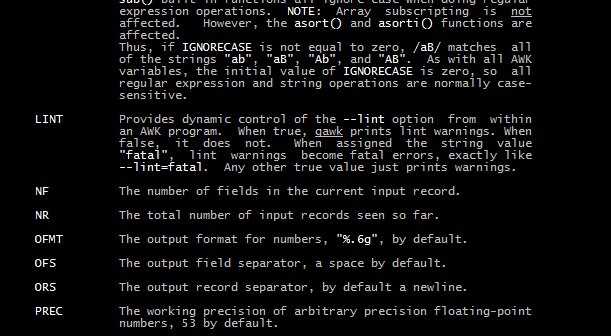
# awk语法

**awk(关键字:分析&处理)** 一行一行的分析处理 awk '条件类型1{动作1}条件类型2{动作2}' filename,

例如：awk ‘BEGIN{} condition{} END{}’

**awk则比较倾向于一行当中分成数个"字段"(区域)来处理, 默认的分隔符是空格键或tab键**

**分隔符由FS=来赋值**



**实例：**

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

打印命令：print $i

如果是域变量，就是$i，如果是自定义变量，就是直接使用名字即可

## awk特殊变量

NR 累积行号

FNR 当前文件行号

NF 当前字段数

FS 字段分离

## awk数组

数组可以直接赋值a[i][j]=$j

注意：数组下标是从1开始，与c数组不一样。

如果是关联数组，可以for(k in Arr)print k,Arr[k]

**数组排序**

**统计单词，排序输出**

**scrpt='{**

**for(i=1;i<=NF;i++)**

**count[$i]++**

**}**

**END{**

**for(i in count)**

**print i,count[i]**

**}**

**'**

**cat words.txt|awk "$scrpt"|sort -r -n -k2**

**这里涉及两个技术**

1. **将scrpt写成字符串以便重定向，字符串必须加上双引号表示这是一个单独的参数**
2. **使用sort排序，指定关键字，关键字比较方式，逆序与否**

## awk打印

print

printf

## awk for循环

for（i=0;i!=9;i++）…

{

}

## ~ 表示正则匹配 ！表示取反 在{}之前可以使用表达式NR==10表示打印第10行

awk '$2 !~ /th/ {print $2,$4}' log.txt

---------------------------------------------

Are like

a

There orange,apple,mongo

$ awk '!/th/ {print $2,$4}' log.txt

---------------------------------------------

Are like

a

There orange,apple,mongo

## awk脚本

$ cat cal.awk

#!/bin/awk -f

#运行前

BEGIN {

math = 0

english = 0

computer = 0

printf "NAME NO. MATH ENGLISH COMPUTER TOTAL\n"

printf "---------------------------------------------\n"

}

#运行中

{

math+=$3

english+=$4

computer+=$5

printf "%-6s %-6s %4d %8d %8d %8d\n", $1, $2, $3,$4,$5, $3+$4+$5

}

#运行后

END {

printf "---------------------------------------------\n"

printf " TOTAL:%10d %8d %8d \n", math, english, computer

printf "AVERAGE:%10.2f %8.2f %8.2f\n", math/NR, english/NR, computer/NR

}

# sed语法：由行号构成命令执行条件，具体使用man sed即可查看相应的命令形式

Sed commands can be given with no addresses, in which case the command will be executed for all input lines; with one address, in which case the command will only be exe‐

cuted for input lines which match that address; or with two addresses, in which case the command will be executed for all input lines which match the inclusive range of

lines starting from the first address and continuing to the second address. Three things to note about address ranges: the syntax is addr1,addr2 (i.e., the addresses are

separated by a comma); the line which addr1 matched will always be accepted, even if addr2 selects an earlier line; and if addr2 is a regexp, it will not be tested against

the line that addr1 matched.

If no **-e**, **--expression**, **-f**, or **--file** option is given, then the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of

input files; if no input files are specified, then the standard input is read.

**sed(关键字: 编辑)** 以行为单位的文本编辑工具 sed可以直接修改档案, 不过一般不推荐这么做, 可以分析 standard input  
基本工作方式: sed [-nef] '[动作]' [输入文本]  
-n : 安静模式, 一般sed用法中, 来自stdin的数据一般会被列出到屏幕上, 如果使用-n参数后, 只有经过sed处理的那一行被列出来.  
-e : 多重编辑, 比如你同时又想删除某行, 又想改变其他行, 那么可以用 sed -e '1,5d' -e 's/abc/xxx/g' filename  
-f : 首先将 sed的动作写在一个档案内, 然后通过 sed -f scriptfile 就可以直接执行 scriptfile 内的sed动作 (没有实验成功, 不推荐使用)  
-i : 直接编辑, 这回就是真的改变文件中的内容了, 别的都只是改变显示. (不推荐使用)  
动作:  
a 新增, a 后面可以接字符串, 而这个字符串会在新的一行出现. (下一行)  
c 取代, c 后面的字符串, 这些字符串可以取代 n1,n2之间的行  
d 删除, 后面不接任何东西  
i 插入, 后面的字符串, 会在上一行出现  
p 打印, 将选择的资料列出, 通常和 sed -n 一起运作 sed -n '3p' 只打印第3行  
s 取代, 类似vi中的取代, 1,20s/old/new/g

[line-address]q 退出, 匹配到某行退出, 提高效率

# grep

主要参数 []  
-c : 只输出匹配的行  
-I : 不区分大小写  
-h : 查询多文件时不显示文件名  
-l : 查询多文件时, 只输出包含匹配字符的文件名  
-n : 显示匹配的行号及行  
-v : 显示不包含匹配文本的所有行(我经常用除去grep本身)  
基本工作方式: grep 要匹配的内容 文件名, 例如:  
grep 'test' d\* 显示所有以d开头的文件中包含test的行  
grep 'test' aa bb cc 显示在 aa bb cc 文件中包含test的行  
grep '[a-z]\{5}\' aa 显示所有包含字符串至少有5个连续小写字母的串

[line-address]r 匹配到的行读取某文件 例如: sed '1r qqq' abc , 注意, 写入的文本是写在了第1行的后边, 也就是第2行

[line-address]w file, 匹配到的行写入某文件  例如: sed -n '/m/w qqq' abc , 从abc中读取带m的行写到qqq文件中, 注意, 这个写入带有覆盖性.

举例:  
sed '1d' abc 删除 abc 档案里的第一行, 注意, 这时会显示除了第一行之外的所有行, 因为第一行已经被删除了(实际文件并没有被删除,而只是显示的时候被删除了)  
sed -n '1d' abc 什么内容也不显示, 因为经过sed处理的行, 是个删除操作, 所以不现实.  
sed '2,$d' abc 删除abc中从第二行到最后一行所有的内容, 注意, $符号正则表达式中表示行末尾, 但是这里并没有说那行末尾, 就会指最后一行末尾, ^开头, 如果没有指定哪行开头, 那么就是第一行开头  
sed '$d' abc 只删除了最后一行, 因为并没有指定是那行末尾, 就认为是最后一行末尾  
sed '/test/d' abc 文件中所有带 test 的行, 全部删除  
sed '/test/a RRRRRRR' abc 将 RRRRRRR 追加到所有的带 test 行的下一行 也有可能通过行 sed '1,5c RRRRRRR' abc  
sed '/test/c RRRRRRR' abc 将 RRRRRRR 替换所有带 test 的行, 当然, 这里也可以是通过行来进行替换, 比如 sed '1,5c RRRRRRR' abc

# 正则表达式断言

1. **前向界定（肯定顺序环视）(?=exp)**

表示右边要匹配exp表达式

1.      strTest = "aaa111aaa , bbb222&, 333ccc"

2.      strRe = r"\d+(?=[a-z]+)"

3.      reObj = re.compile(strRe)

4.      print reObj.findall(strTest)

输出：（正则的意思是找出连续的数字并且最后一个数字跟着至少一个a-z里面的字符序列）

1. ['111', '333']

**2、前向否定界定（否定顺序环视）(?!exp)**  
      表示右边不匹配exp表达式

1.      strTest = "aaa111aaa , bbb222&, 333ccc"

2.      strRe = r"\d+(?![a-z]+)"

3.      reObj = re.compile(strRe)

4.      **print** reObj.findall(strTest)

      输出：（正则的意思是找出连续的数字，并且最后一个数字后面不能跟任何一个a-z里面的字符序列）

1.      ['11', '222', '33']

**3、反向界定（肯定逆序环视）(?<=exp) 文本必须定长**  
      表示左边要匹配exp表达式

1.      strTest = "aaa111aaa , bbb222&, 333ccc"

2.      strRe = r"(?<=[a-z])\d+"

3.      reObj = re.compile(strRe)

4.      **print** reObj.findall(strTest)

      输出：（正则的意思是找出连续的数字，并且第一个数字的前面要是a-z中的一个字符）

1.       ['111', '222']

**4、反向否定界定（否定逆序环视）(? 文本必须定长**

# Python3的字符串处理

不需使用apply，而只使用getattr来实现一个给予对象的调用。实例strproc如下：

strproc.py

import sys

import getopt

''' call like this: strproc -f|--func \_\_split\_\_ -a|--args ‘,’ ‘f,l,k’ '''

args=sys.argv[1:]

opt,val = getopt.gnu\_getopt(args, 'f:a:',['func=','args='])

func=None

args=[]

for i in opt:

if i[0] in ('-f','--func'):

func=i[1]

elif i[0] in ('-a','--args'):

args.extend(i[1].split(','))

res = getattr(val,func)(\*args)

print(res)

一些常用的调用方式：

python3 strproc.py -f \_\_len\_\_ kasdf5dfas

python3 strproc.py -f split kasdf5dfas -a 5

python3 strproc.py -f \_\_eq\_\_ 19 -a 19