姓名：张治卓

学号:5171111910078

上机实验二：正则表达式练习

一、grep练习题

1. 有一个形如 `/etc/passwd` 的用户资料文件 `user` ，其格式为：

   用户名:密码:用户ID:组ID:用户描述:主目录:登录shell

   - 分别用grep/sed/awk找出所有登录shell为bash的用户；
     `grep 'bash user | sed 's/:.*//'`
     `sed -n '/bash$/p' user | sed 's/:.*//'`
     `awk -F : '$7~"bash$" {print $1}' user`
   - 找出UID或者GID为4位以上数字的用户；
     `grep -E '[[:digit:]]{5,}' user | sed 's/:.*//'`
   - 找出主目录为/var/spoo...l的用户，这里.表示不确定o有多少次重复；
     `grep -E '/var/spoo+' user | sed 's/:.*//'`
   - 已知有个用户名形如mimi...M，这里.表示mi至少一次以上的重复。
     `grep -E 'mi(mi)+M' user | sed 's/:.*//'`

2. 有个非常混乱的文本文件 `regex.txt` ，需要进行一些正则表达式分析：

   - 用 `grep` 查找其中的大写字母；
     `grep -E '[A-Z]' regex.txt`
   - 用 `grep` 查找其中的数字；
     `grep -E '[0-9]' regex.txt`
   - 查找包含”the”的行，忽略大小写，统计其总共出现的次数；
     `grep -oi 'the' regex.txt | grep -ci 'the'`
   - 统计不包含”the”的行数；
     不区分大小写 `grep -cvi 'the' regex.txt`
     区分大小写 `grep -cv 'the' regex.txt`
   - 查找包含”test”或者”tast”的行；
     `grep -E 't[ae]st' regex.txt`
   - 查找不包含”#”的行；
     `grep -vE '#' regex.txt`
   - 查找”oog”但前面的字符不能是”g”或者”o”；
     `grep -E 'oog' regex.txt`
   - 查找以”the”开头的行；
     `grep -E '^(the)' regex.txt`
   - 查找以”d”结束的行；
     `grep -E 'd @@footer regex.txt`
   - 过滤掉空行；
     `grep -vE '^ @@footer regex.txt`
   - 过滤掉注释行，也就是以”#”开始的行；
     `grep -vE '^#' regex.txt`
   - 查找存在至少两个连续字符”e”的行；
     `grep -E 'ee' regex.txt`
   - 查找字母”g”后面跟着2-5个”o”且后面还有一个”g”的行；
     `grep -E 'go{2,5}g' regex.txt`
   - 输出 `regex.txt` 的内容，并打印行号，并删除2-5行；
     `sed -n '2,5d;=;p' regex.txt`
   - 同上，但删除第5行到最后一行；
     `sed -n '5,$d;=;p' regex.txt`
   - 原位删除第一行；
     `sed -n '1d;p' regex.txt`
   - 在第二行前插入两行”test”，在第三行后添加”;;”行；
     `sed -e '2i\test\ntest' -e '3a\;;' regex.txt`
   - 将2-5行的内容更换为”nothing but 2nd-5th line”；
     `sed '2,5cnothing but 2nd-5th line' regex.txt`
   - 只输出5-8行
     `sed -n '5,8p' regex.txt`

gene

二、sed练习题

1. 编写sed脚本，将文件中每行的exon坐标信息如 `chr1:28427874-28425431` 转换为TAB分隔的信息：

   chr1 28427874 28425431

   `sed -i filename -e 's/:/\t/' -e 's/-/\t/'`

2. 将转录组文件 `Mus_musculus.GRCm38.75_chr1.gtf` 中的转录本名称的提示信息去除，仅保留转录本的ID信息。
   `sed -i Mus_musculus.GRCm38.75_chr1.gtf -e 's/.*transcript_id/transcript_id/g' -e 's/gene_name.*//g' -e 's/exon_number.*//g'`
   运行结果（部分）

```
#!genome-build GRCm38.p2
#!genome-version GRCm38
#!genome-date 2012-01
#!genome-build-accession NCBI:GCA_000001635.4
#!genebuild-last-updated 2013-09
1       pseudogene      gene    3054233 3054733 .       +       .       gene_id
"ENSMUSG00000090025";
transcript_id "ENSMUST00000160944";
transcript_id "ENSMUST00000160944";
1       snRNA   gene    3102016 3102125 .       +       .       gene_id "ENSMUSG00000064842";
transcript_id "ENSMUST00000082908";
transcript_id "ENSMUST00000082908";
1       protein_coding  gene    3205901 3671498 .       -       .       gene_id
"ENSMUSG00000051951";
transcript_id "ENSMUST00000162897";
transcript_id "ENSMUST00000162897";
transcript_id "ENSMUST00000162897";
transcript_id "ENSMUST00000159265";
transcript_id "ENSMUST00000159265";
transcript_id "ENSMUST00000159265";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
transcript_id "ENSMUST00000070533";
1       antisense       gene    3466587 3513553 .       +       .       gene_id
"ENSMUSG00000089699";
transcript_id "ENSMUST00000161581";
transcript_id "ENSMUST00000161581";
transcript_id "ENSMUST00000161581";
1       snRNA   gene    3783876 3783933 .       -       .       gene_id "ENSMUSG00000088333";
transcript_id "ENSMUST00000157708";
transcript_id "ENSMUST00000157708";
```

   其中为了区分不同基因下的转录本，因此保留了基因的信息，而对转录本信息只留下了ID信息

3. 已知命令 `ip addr` 可输出本机多张网卡的IP地址等信息，你能否从中抽取MAC地址、IP地址和子网掩码。
   `ip addr | sed -ne 's/inet //p' -ne 's/link\/[a-z]* //p' | sed -ne 's/ scope host.*//p' -ne 's/ brd.*//p'`
   其中IP地址和子网掩码的表示方法按照ip addr输出格式的 `IP address/掩码位数` 输出，MAC地址也可直接看到。
   运行结果

```
zhangzz@zhangzhizhuo:~/下载/lab2$ ip addr | sed -ne 's/inet //p' -ne 's/link\/[
a-z]* //p' | sed -ne 's/ scope host.*//p' -ne 's/ brd.*//p'
    00:00:00:00:00:00
    127.0.0.1/8
    00:0c:29:e5:11:88
    192.168.8.136/24
```

三、awk练习题

1. 针对FASTA格式的序列文件 `improper.fa`

   - 输出每条序列的长度；
   - 输出每条序列中A/C/G/T四种碱基出现的频率；
   - 计算每条序列中A/C/G/T四种碱基的比例；
   - 计算所有序列中碱基的平均比例；
   - 这些序列中包含一个错误，你能找出来么？
     `awk -f Question3.1.awk improper.fa`
     代码为

gene

```
#Question 1
BEGIN{
    FS=""
}
/^>/{
        print $0
}
!/^>/{
    s_length=NF
    for(i=1;i<=NF;i++){
        if($i=="A") A++
        else if($i=="T") T++
        else if($i=="C") C++
        else if($i=="G") G++
        else break;
    }
    A_percentage=A/s_length
    C_percentage=C/s_length
    G_percentage=G/s_length
    T_percentage=T/s_length
    A_total+=A
    C_total+=C
    G_total+=G
    T_total+=T
    total+=s_length
    printf("Sequnce Length:%d\n",s_length)
    printf("A:%d\t",A)
    printf("C:%d\t",C)
    printf("G:%d\t",G)
    printf("T:%d\n",T)
    printf("A percentage:%f\n",A_percentage)
    printf("C percentage:%f\n",C_percentage)
    printf("G percentage:%f\n",G_percentage)
    printf("T percentage:%f\n",T_percentage)
    printf("\n")
        A=0
        C=0
        G=0
        T=0
}
END{
    printf("Overall A percentage: %2.2f\n",A_total/total*100)
    printf("Overall C percentage: %2.2f\n",C_total/total*100)
    printf("Overall G percentage: %2.2f\n",G_total/total*100)
    printf("Overall T percentage: %2.2f\n",T_total/total*100)
}
```

错误是在bad sequence里面第35个基因A在输出时为乱码，并且占两个长度，经过搜索这个A实际上是α的大写。
修改前运行结果为

```
zhangzz@zhangzhizhuo:~/下载/lab2$ awk -f Question3.1.awk improper.fa
>good-sequence
Sequnce Length:46
A:12    C:14    G:11    T:9
A percentage:0.260870
C percentage:0.304348
G percentage:0.239130
T percentage:0.195652

>bad-sequence
Sequnce Length:47
A:10    C:9     G:10    T:5
A percentage:0.212766
C percentage:0.191489
G percentage:0.212766
T percentage:0.106383

Overall A percentage: 23.66
Overall C percentage: 24.73
Overall G percentage: 22.58
Overall T percentage: 15.05
```

gene

修改后运行结果为

```
zhangzz@zhangzhizhuo:~/下载/lab2$ awk -f Question3.1.awk improper.fa
>good-sequence
Sequnce Length:46
A:12    C:14    G:11    T:9
A percentage:0.260870
C percentage:0.304348
G percentage:0.239130
T percentage:0.195652

>bad-sequence
Sequnce Length:46
A:14    C:11    G:12    T:9
A percentage:0.304348
C percentage:0.239130
G percentage:0.260870
T percentage:0.195652

Overall A percentage: 28.26
Overall C percentage: 27.17
Overall G percentage: 25.00
Overall T percentage: 19.57
```

2. 针对文件 `twoseqs.fa` 中的两条DNA序列，编写awk脚本

- 计算每个序列的3-mer的发生频率，所谓3-mer指的是序列中长度为3的子串，如在序列ACTGGACT中ACT的发生率为2。
- 根据3-mer的频率按照下列公式计算两序列的相似性得分：

$$score = \exp\left(-\sum_i (occ_{1i} - occ_{2i})^2\right)$$

```
awk '/>seq/{print $0;row=NR}{if(NR==row+1)T1=$0}{if(NR==row+2)T2=$0}{if(NR==row+3){T3=$0;printf("%s%s%s\n",T1,T2,T3)}}' twoseqs.fa | awk -f Question3.2.awk
```

代码为

gene

```
#Question3.2
BEGIN{
    FS=""
}
/^>/{
    print "\n" $0
}
!/^>/{
    for(i=1;i<=NF-2;i++){
        mer1=$i$(i+1)$(i+2)
        if(mer1 in mer_name){}
            else{
                mer_name[i]=mer1
            }
        for(j=1;j<=NF-2;j++){
            mer2=$j$(j+1)$(j+2)
            if(mer1==mer2){
                count+=1
            }
        }
        if(NR==2){
            if(mer1 in mer_array){}
            else{
                mer_array[mer1]=count
                printf("%s的发生频率为%-3d",mer1,mer_array[mer1])
            }
        }
        if(NR==4){
            if((mer1,mer1) in mer_array){}
            else{
                mer_array[mer1,mer1]=count
                printf("%s的发生频率为%-3d",mer1,mer_array[mer1,mer1])
            }
        }
        count=0
    }
}
END{
    for(i=1;i<=NF-2;i++){
        if(mer_name[i] in mer_done){}
        else{
            sum+=(mer_array[mer_name[i]]-mer_array[mer_name[i],mer_name[i]])^2
            mer_done[mer_name[i]]="TRUE"
        }
    }
    score=exp(-sum)
    print "\n发生频率列表为（第一行为3-mer，第二、三行分别为seq1和seq2"
    for(c in mer_done){
        printf("%-4s",c)
    }
    print "\n"
    for(c in mer_done){
        printf("%-4d",mer_array[c])
    }
    print "\n"
    for(c in mer_done){
        printf("%-4d",mer_array[c,c])
    }
    print "\n"
    printf("相似性得分为%f\n",score)
}
```

运行结果为

gene

```
zhangzz@zhangzhizhuo:~/下载/lab2$ awk '/>seq/{print $0;row=NR}{if(NR==row+1)T1=$0}{if(NR==
row+2)T2=$0}{if(NR==row+3){T3=$0;printf("%s%s%s\n",T1,T2,T3)}}' twoseqs.fa | awk -f Questi
on3.2.awk

>seq1
AAT的发生频率为2  ATC的发生频率为3  TCC的发生频率为2  CCA的发生频率为5  CAC的发生频率为4
ACA的发生频率为3  CAG的发生频率为4  AGC的发生频率为2  GCT的发生频率为3  CTG的发生频率为3
TGA的发生频率为3  GAC的发生频率为1  ACT的发生频率为3  TGG的发生频率为2  GGC的发生频率为4
GCC的发生频率为2  CCC的发生频率为3  AGG的发生频率为2  CTC的发生频率为4  TCT的发生频率为2
TCA的发生频率为5  GGA的发生频率为1  GAT的发生频率为2  ACC的发生频率为2  CCG的发生频率为1
CGT的发生频率为1  GTG的发生频率为2  TGT的发生频率为5  GTT的发生频率为2  TTT的发生频率为1
TTC的发生频率为2  CAT的发生频率为3  ATG的发生频率为3  GTA的发生频率为2  TAT的发生频率为1
ATT的发生频率为2  TTG的发生频率为3  GTC的发生频率为1  CAA的发生频率为3  AAA的发生频率为2
AAC的发生频率为2  AAG的发生频率为1  AGT的发生频率为1  TAC的发生频率为1  CTT的发生频率为1
TCG的发生频率为1  CGC的发生频率为1  GCG的发生频率为1  CGG的发生频率为1  GGG的发生频率为2
GCA的发生频率为1  GAA的发生频率为1
>seq2
AAT的发生频率为2  ATC的发生频率为3  TCC的发生频率为2  CCA的发生频率为5  CAC的发生频率为4
ACA的发生频率为3  CAG的发生频率为4  AGC的发生频率为2  GCT的发生频率为3  CTG的发生频率为3
TGA的发生频率为3  GAC的发生频率为1  ACT的发生频率为3  TGG的发生频率为2  GGC的发生频率为4
GCC的发生频率为2  CCC的发生频率为3  AGG的发生频率为2  CTC的发生频率为4  TCT的发生频率为2
TCA的发生频率为5  GGA的发生频率为1  GAT的发生频率为2  ACC的发生频率为2  CCG的发生频率为1
CGC的发生频率为2  GCG的发生频率为2  CGT的发生频率为1  GTG的发生频率为1  TGT的发生频率为4
GTT的发生频率为2  TTT的发生频率为1  TTC的发生频率为2  CAT的发生频率为3  ATG的发生频率为3
GTA的发生频率为2  TAT的发生频率为1  ATT的发生频率为2  TTG的发生频率为3  GTC的发生频率为1
CAA的发生频率为3  AAA的发生频率为2  AAC的发生频率为2  AAG的发生频率为1  AGT的发生频率为1
TAC的发生频率为1  CTT的发生频率为1  TCG的发生频率为1  CGG的发生频率为1  GGG的发生频率为2
GCA的发生频率为1  GAA的发生频率为1
发生频率列表为（第一行为3-mer，第二、三行分别为seq1和seq2
ACC CTC TCA ATT TCC CTG CAA CAC TCG TTT AAT CCA AGC CAG CCC TGA TAT AGG ACT CTT CCG TGG TC
T CAT CGC GTA AGT GTC CGG GTG GAA TGT GAC GCA GCC CGT GCG GTT ATC GGA GAT GGC ATG TTC AAA
AAC GGG GCT TTG AAG ACA TAC

2   4   5   2   2   3   3   4   1   1   2   5   2   4   3   3   1   2   3   1   1   2   2
  3   1   2   1   1   1   2   1   5   1   1   2   1   1   2   3   1   2   4   3   2   2
2   2   3   3   1   3   1

2   4   5   2   2   3   3   4   1   1   2   5   2   4   3   3   1   2   3   1   1   2   2
  3   2   2   1   1   1   1   1   4   1   1   2   1   2   2   3   1   2   4   3   2   2
2   2   3   3   1   3   1

相似性得分为0.018316
```

其中后面的列表因为屏幕限制无法展开，如果展开应该如下所示

```
ACC CTC TCA ATT TCC CTG CAA CAC TCG TTT AAT CCA AGC CAG CCC TGA TAT AGG ACT CTT CCG TGG TCT CAT CGC GTA AGT GTC CGG

2   4   5   2   2   3   3   4   1   1   2   5   2   4   3   3   1   2   3   1   1   2   2   3   1   2   1   1   1

2   4   5   2   2   3   3   4   1   1   2   5   2   4   3   3   1   2   3   1   1   2   2   3   2   2   1   1   1
```

3. 从PDB数据库中下载一个PDB格式的大分子结构文件，计算其中所有C_α原子两两之间的欧氏距离。

```
awk -f Question3.3.awk 3hyd.pdb
```

代码为

```
#Question3.3
BEGIN{
    num=0
    count=0
}
/^ATOM.*CA/{
    num++
    atom_number[num]=$2
    pos_x[num]=$7
    pos_y[num]=$8
    pos_z[num]=$9
}
END{
    for(i=1;i<=num;i++){
        for(j=i+1;j<=num;j++){
            if(i!=j){
                distance=sqrt((pos_x[i]-pos_x[j])^2+(pos_y[i]-pos_y[j])^2+(pos_z[i]-pos_z[j])^2)
                printf("原子序号第%d号和第%d号之间的Ca距离为%.2f\n",atom_number[i],atom_number[j],distance)
                count++
            }
        }
    }
    printf("\n\n总共有%d对Ca原子的距离\n",count)
}
```

gene

运行结果为

```
zhangzz@zhangzhizhuo:~/下载/lab2$ awk -f Question3.3.awk 3hyd.pdb
原子序号第2号和第23号之间的Ca距离为3.78
原子序号第2号和第39号之间的Ca距离为6.84
原子序号第2号和第40号之间的Ca距离为6.83
原子序号第2号和第65号之间的Ca距离为9.87
原子序号第2号和第75号之间的Ca距离为13.55
原子序号第2号和第94号之间的Ca距离为16.46
原子序号第2号和第115号之间的Ca距离为20.07
原子序号第23号和第39号之间的Ca距离为3.79
原子序号第23号和第40号之间的Ca距离为3.78
原子序号第23号和第65号之间的Ca距离为6.18
原子序号第23号和第75号之间的Ca距离为9.93
原子序号第23号和第94号之间的Ca距离为12.71
原子序号第23号和第115号之间的Ca距离为16.35
原子序号第39号和第40号之间的Ca距离为0.01
原子序号第39号和第65号之间的Ca距离为3.81
原子序号第39号和第75号之间的Ca距离为7.01
原子序号第39号和第94号之间的Ca距离为10.40
原子序号第39号和第115号之间的Ca距离为13.73
原子序号第40号和第65号之间的Ca距离为3.81
原子序号第40号和第75号之间的Ca距离为7.01
原子序号第40号和第94号之间的Ca距离为10.40
原子序号第40号和第115号之间的Ca距离为13.73
原子序号第65号和第75号之间的Ca距离为3.79
原子序号第65号和第94号之间的Ca距离为6.70
原子序号第65号和第115号之间的Ca距离为10.22
原子序号第75号和第94号之间的Ca距离为3.82
原子序号第75号和第115号之间的Ca距离为6.74
原子序号第94号和第115号之间的Ca距离为3.79


总共有28对Ca原子的距离
```

4. 有一个文件 `competition.txt` ，记录了红、蓝、绿三队队员的比赛得分情况。编写一个脚本，

   - 计算每个队员的平均得分；
   - 计算每场比赛的平均得分；
   - 计算每支队伍的平均得分；

     `awk -f Question3.4.awk competition.txt`

     代码为

```
#Question3.4
BEGIN{
    FS=","
    printf("每一个队员的平均分为：\n")
}
/Name/{
    for(i=3;i<=NF;i++){
        match_name[i-2]=$i
    }
}
!/Name/{
    row=NR
    col=NF
    score_player=0
    for(i=3;i<=NF;i++){
        if($i==-1){
            col--
            times[i-2]++
            $i=0
        }
        else{
            score_match[i-2]+=$i
            score_player+=$i
            if($2=="Red"){score_team["Red"]+=$i}
            else if($2=="Blue"){score_team["Blue"]+=$i}
            else{score_team["Green"]+=$i}
        }
    }
    printf("%-10s%7.2f\n",$1,score_player/(col-2))
}
END{
    printf("每一场比赛的平均分为：\n")
    for(i=1;i<=3;i++){
        printf("%-13s%7.2f\n",match_name[i],score_match[i]/(row-times[i]-1))
    }
    printf("每一个队伍的平均分为：\n")
    for(c in score_team){
        printf("%-10s%7.2f\n",c,score_team[c]/(NF-2))
    }
}
```

运行结果为

```
zhangzz@zhangzhizhuo:~/下载/lab2$ awk -f Question3.4.awk competition.txt
每一个队员的平均分为：
Tom         14.67
Joe         13.00
Maria       15.00
Fred        13.33
Carlos      19.50
Phuong      15.67
Enrique     13.00
Nancy       15.00
每一场比赛的平均分为：
First Match     5.00
Second Match   15.75
Third Match    22.12
每一个队伍的平均分为：
Blue        28.33
Green       41.67
Red         42.67
```

其中由于比赛平均分的字符串长度超过了10，所以单独设置为 `%-13s`

注意：

- 文件的首行是文件头；
- 如果某个队员的得分为负值，说明该队员缺席了该比赛，统计的时候应该忽略；
- 输出结果时，字符串左对齐占据10个字符长度；数值按右对齐占据7个字符，2位小数位；
- 脚本应该有更广泛的适用性，不应局限于本文件

@@footer