47 用Shell做统计练习

更新时间：2019-08-16 10:06:25

> 学习知识要善于思考，思考，再思考。
>
> —— 爱因斯

内容简介

1. 前言
2. 成果展示
3. 解题步骤和答案
4. 可能的优化

1. 前言

上一课 带你玩转Linux和Shell编程 | 第五部分第八课：Shell实现图片展示网页 中，我们做了一个有趣的练习。

那个练习用一个 Shell 脚本来生成一个 HTML 文件，这个 HTML 文件是一个展示图片缩略图的网页，点击每个缩略图会链接到原始图片。

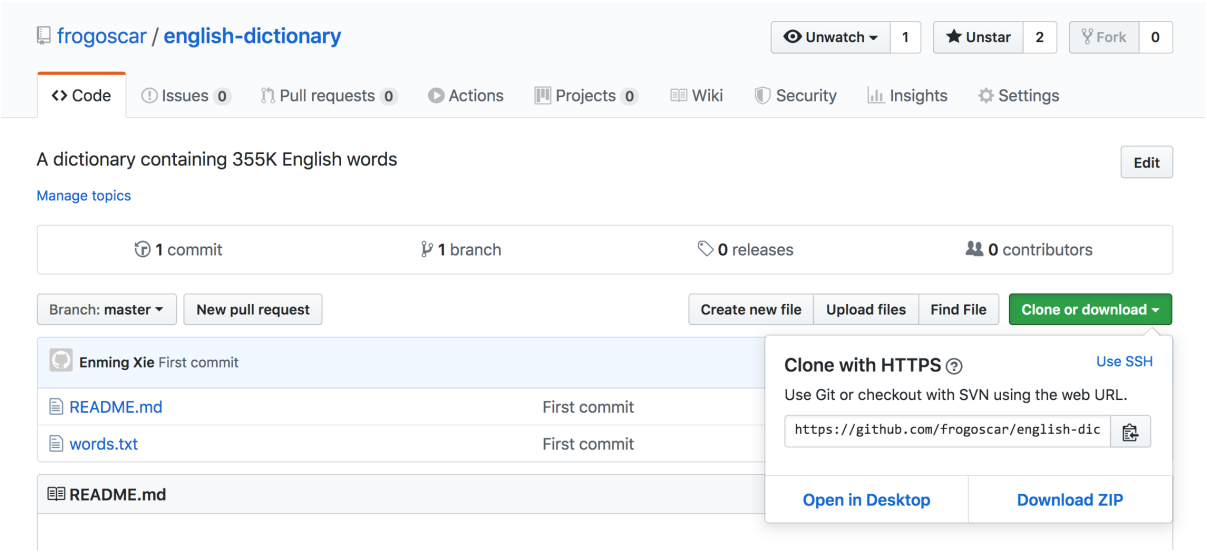
这一课我们继续做一个进阶的 Shell 脚本练习。这个练习要实现的是对一个英语字典做统计。

通过这个练习，你将巩固 Shell 和 Linux 的知识点。

为了完成它，我们需要用到一个文本文件：words.txt。这是一个包含 354935 个英文单词的字典，请从我的 Github 上下载 （下面也会给出百度云盘下载链接）：

https://github.com/frogoscar/english-dictionary

你可以用 `git clone` 到你本地目录，或者直接下载 zip 压缩包。然后提取里面的 words.txt 文件即可。



对于不用 Github 的朋友，我也把字典文件上传到百度云盘了：https://pan.baidu.com/s/1pK8bixt 。

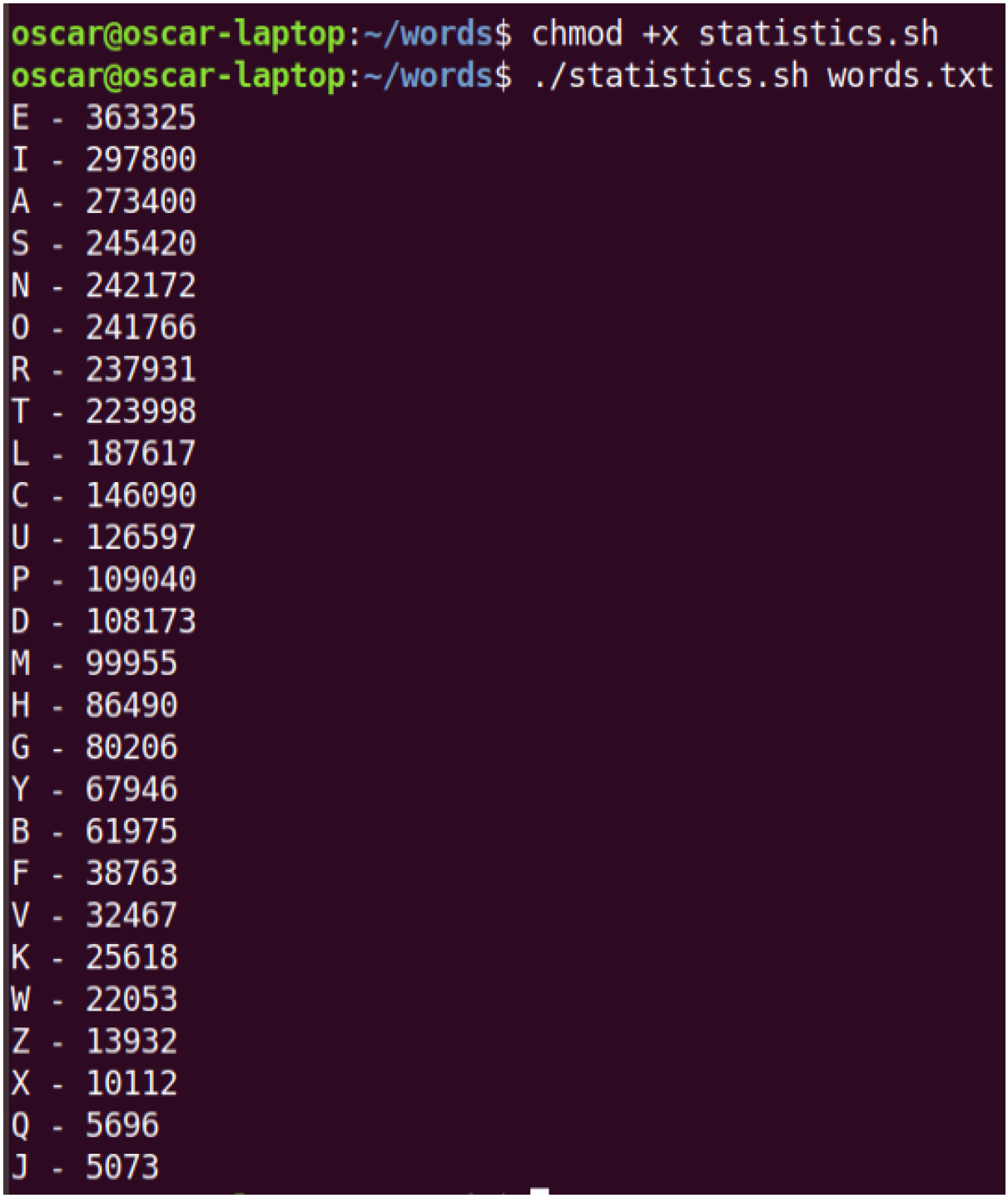
> 当然了，你也可以在网上找到其它完整的英文字典的文本文档，不一定要用我提供的。

2. 成果展示

我们要用到的字典文本文档里的内容类似如下：

我们要写一个 Shell 脚本，来显示这个庞大的字典中 26 个英文字母（从 a 到 z ）出现的次数，而且以次数最多到最少的顺序排列。

成果是像下面这样的：



可以看到，字母 e 出现的次数最多，是 363325 次。字母 j 出现的次数最少，是 5073 次。

> 下面给出我的解题步骤和答案，希望大家最好先不看答案，尝试着自己解决问题，然后再来看答案。
> 你的解法也许比我还要好。相信你可以的，加油！

3. 解题步骤和答案

首先，我们创建一个文件夹，可以起名叫 words（随便你怎么取名）。然后把 words.txt 这个字典文件放进去。

然后，我们在文件夹中创建一个文件，就是我们的 Shell 脚本，叫 statistics.sh 好了，因为 statistics 是英语“统计”的意思：

```
vim statistics.sh
```

根据上面的成果那张截图，我们可以看到要实现的是：

“在终端打印出结果，按照字母出现的次数来排列，由最多到最少。在次数左边，依次是该次数对应的字母、空格、短横杠、空格，而且每个字母是大写的（在字典文件中字母都是小写，因此需要小写到大写的转换）”。

因此，我们首先需要统计每个字母出现的次数。

怎么做呢？我们想到了 grep 命令，它可以帮助我们在文件中查找所需的字母。

我们首先用命令行来测试，之后再着手编写我们的 statistics.sh 这个文件。

首先，在命令行中输入以下命令：

```
grep -io a words.txt
```

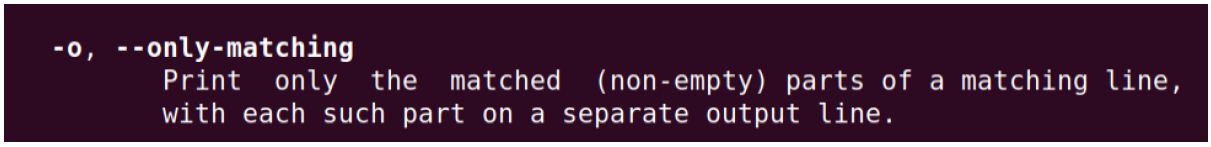
回车运行后可以看到输出了许多行，每一行包含一个 a。

因为 grep 就是用于在文件中查找关键字，并且显示关键字所在的行。

这里我们用了 -i 和 -o 两个参数。-i 参数我们之前学过，是 ignore-case 的简写，表示“忽略大小写”。

而 -o 这个参数我们之前没学过，不过可以用 `man grep` 来看看：

```
man grep
```

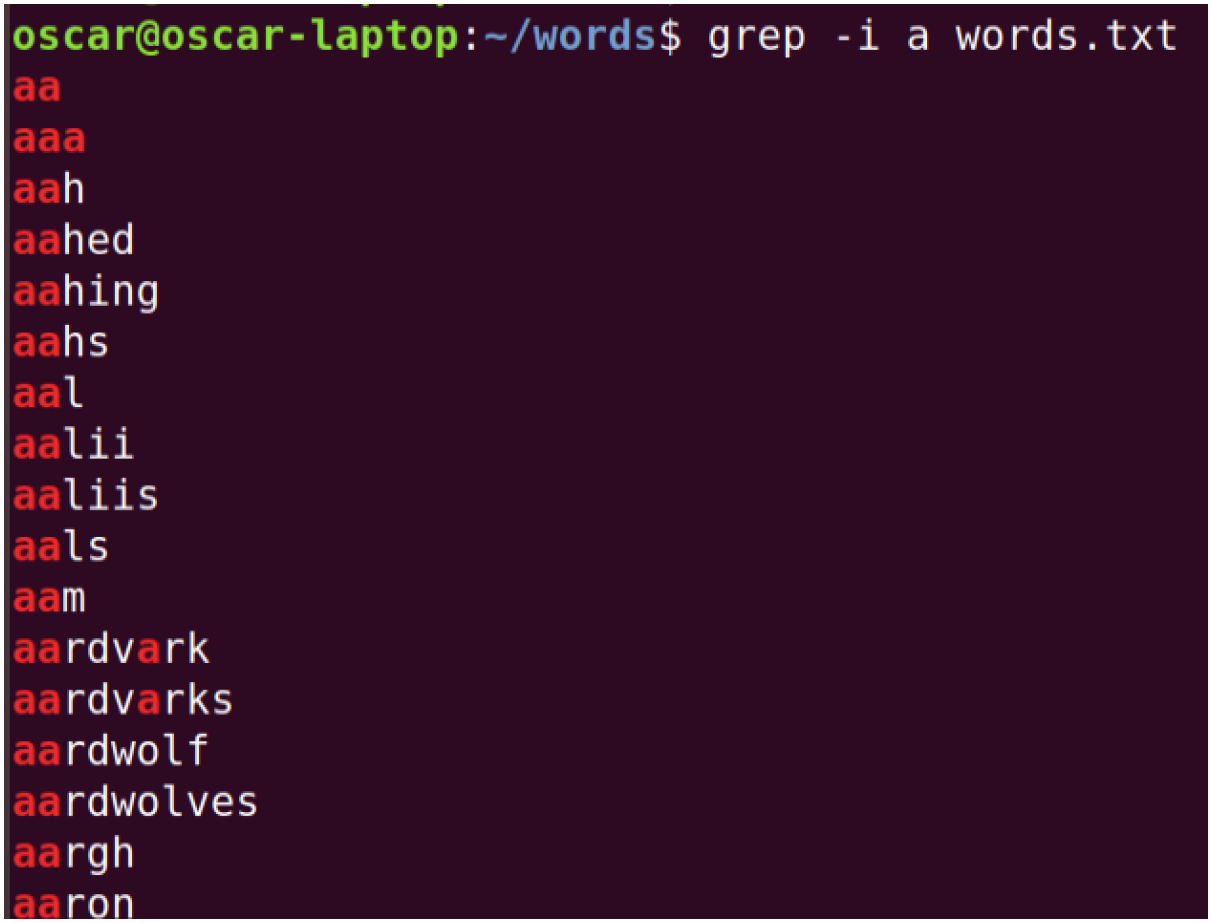


可以看到 -o 参数中的 o 是英语 only-matching 的简写，表示“只匹配”。其描述 “Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.” 可以翻译为“只显示匹配行中不为空的那个匹配的部分，每个这样的部分被单独显示在一行上”。

如果不加 -o 参数而直接用：

```
grep -i a words.txt
```

那么输出是这样的：

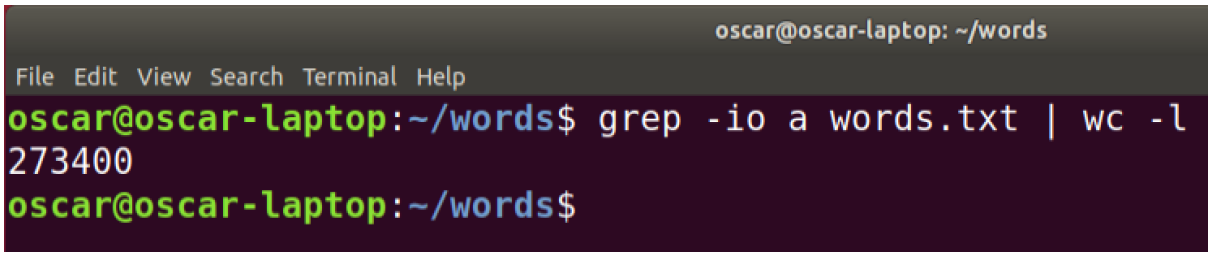


理解了吗？不加 -o 参数，那么 grep 就会输出每一个包含 a 的行。而每一行 （字典文件中一行有一个单词）也许包含不止一个 a。因此为了统计所有的 a，我们须要加上 -o 参数。

既然我们已经用 `grep -io a words.txt` 命令来输出了所有字母 a 的出现（逐行显示），那么我们可以用 `wc -l` 命令来统计行数，就可知道 a 的出现次数了。

接下来我们就用管道来把 grep 命令的结果赋给 wc 命令：

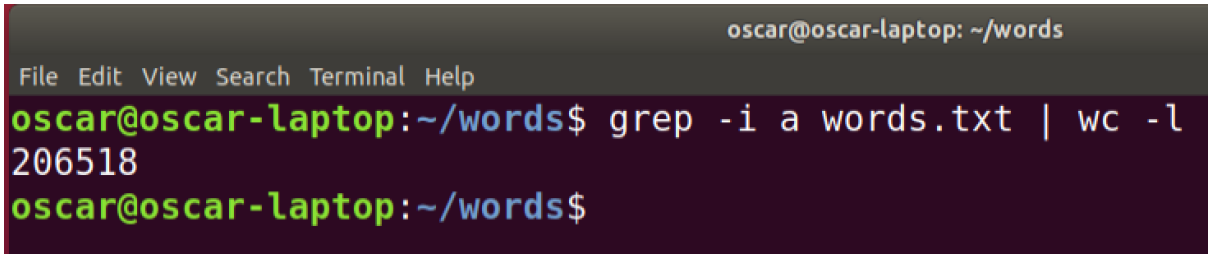
```
grep -io a words.txt | wc -l
```



可以看到输出是 273400，表示 words.txt 文件中字母 a 出现了 273400 次。

我们也可以不加 -o 参数来测试一下：

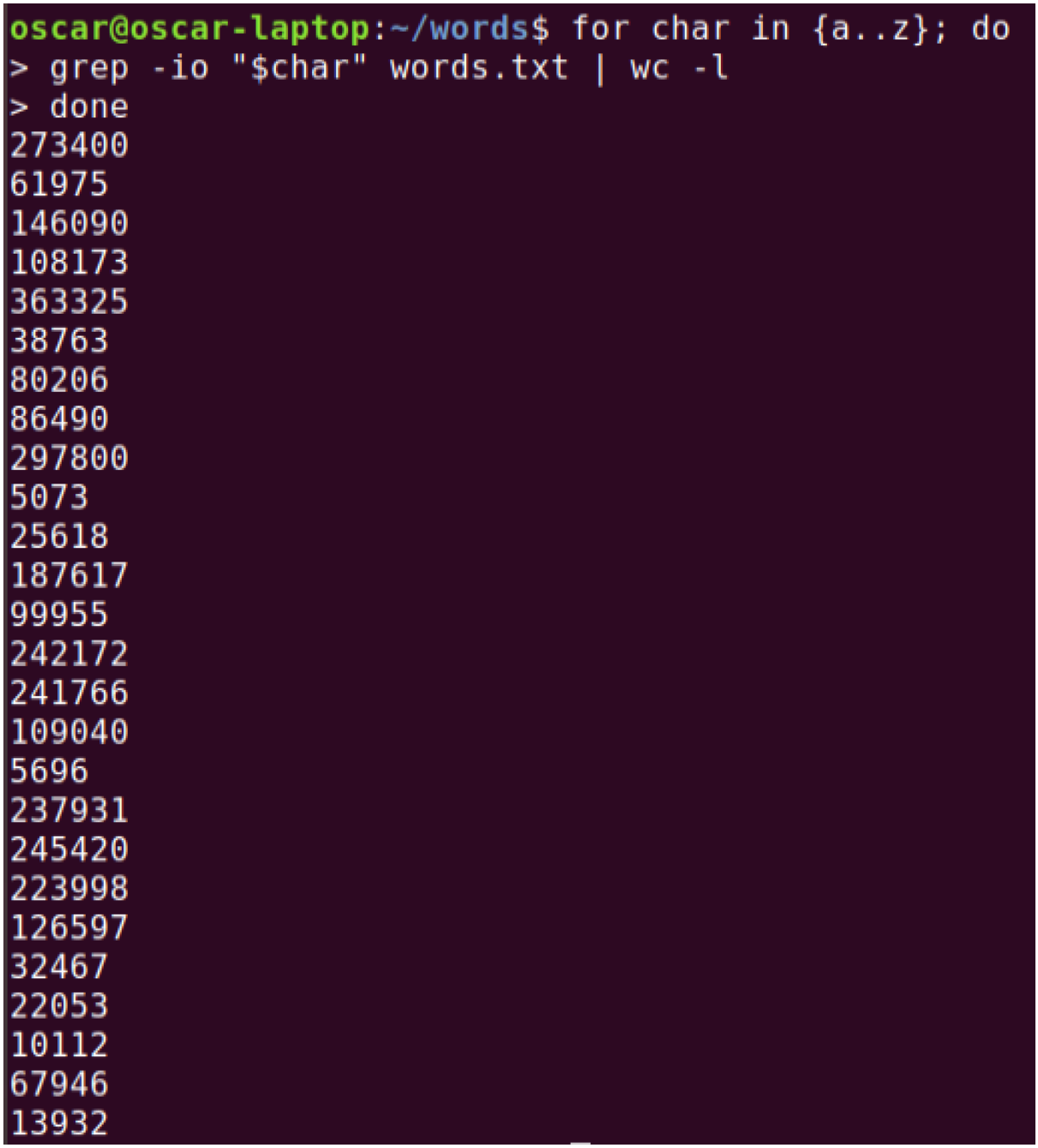
```
grep -i a words.txt | wc -l
```



可以看到输出是 206518，比 273400 少了很多，因为不加 -o 参数的话只统计了 a 出现的那些行（相当于统计了包含 a 的单词的数目），而不是统计 a 的真正出现次数。

我们现在已经知道如何统计字母 a 的次数了，那么举一反三，统计其它 25 个字母也不在话下。我们可以用一个循环语句来实现：

```
for char in {a..z}; do
    grep -io "$char" words.txt | wc -l
done
```



可以看到我们在终端输入 for 循环语句后，依次打印出了 a、 b、 c，一直到 z 这 26 个字母在 words.txt 文件中出现的次数。

虽然现在我们只是开了个头，但是已经可以来写我们的 Shell 脚本了。

我们首先写一些基础的部分：

```
#!/bin/bash

# Verification of parameter
# 确认参数
if [ -z $1 ]
then
   echo "Please enter the file of dictionary !"
   exit
fi

# Verification of file existence
# 确认文件存在
if [ ! -e $1 ]
then
   echo "Please make sure that the file of dictionary exists !"
   exit
fi
```

上面两段代码分别用于确认参数和确认文件存在，如果不满足 if 条件，那么用 echo 显示提示信息，然后用 exit 命令退出 Shell。

接着，我们来定义一个函数，就叫 statistics 好了，我们继续在 statistics.sh 这个文件中加入以下代码：

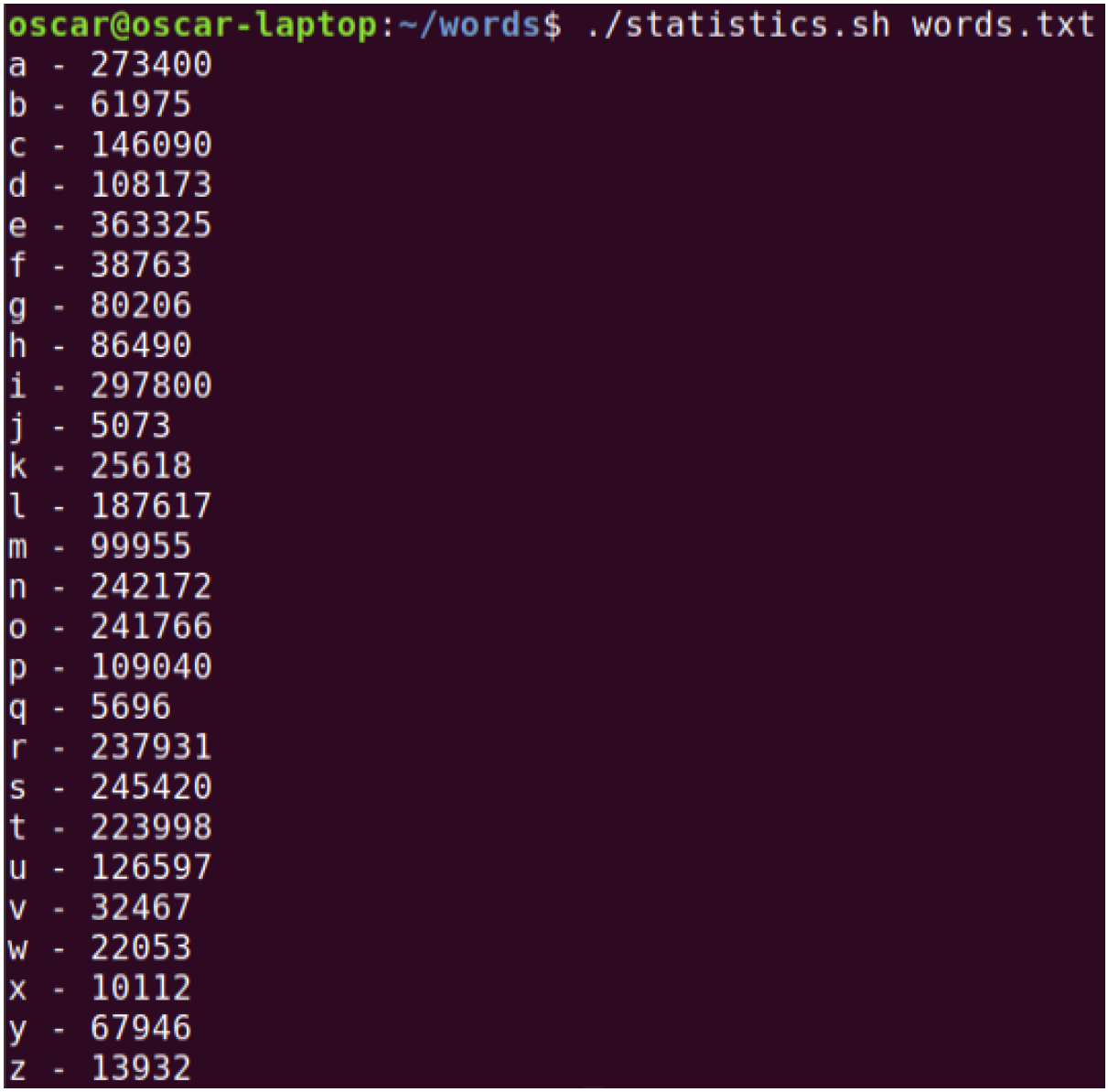
```
# Definition of function
# 函数定义
statistics () {
 for char in {a..z}
 do
  echo "$char - `grep -io "$char" $1 | wc -l`"
 done
}

# Use of function
# 函数使用
statistics $1
```

- `for char in {a..z}` 不难理解，用于遍历 a 到 z 这 26 个英语字母。
- `echo "$char - `grep -io "$char" $1 | wc -l`"` 这句首先用 echo 命令输出 char 变量的值 （依次取值 a 到 z ），然后输出一个空格，输出短横杠，再输出一个空格，然后输出 `grep -io "$char" $1 | wc -l` 这句命令的运行结果，也就是 char 变量对应的字母的出现次数 。

我们运行这个脚本（别忘了先用 `chmod +x statistics.sh` 为脚本加上可执行权限）：

```
./statistics.sh words.txt
```



可以看到，我们的脚本文件如我们所愿从 a 到 z 输出了这 26 个字母，格式也是我们需要的：

```
字母 - 出现次数
```

但是，目前我们的字母没有大写，而且还不是按出现次数由多到少排序的，因此我们还要继续探索。

为了使 echo 命令的输出中的小写字母被转成大写，我们可以用 tr 命令。tr 是 translate 的缩写，表示“翻译，转化”。

我们的函数改为如下：

```
# Definition of function
# 函数定义
statistics () {
  for char in {a..z}
  do
    echo "$char - `grep -io "$char" $1 | wc -l`" | tr /a-z/ /A-Z/
  done
}

# Use of function
# 函数使用
statistics $1
```

`tr /a-z/ /A-Z/` 表示把所有 a 到 z 的小写字母转为对应的大写字母 A-Z。

```
oscar@oscar-laptop:~/words$ ./statistics.sh words.txt
A - 273400
B - 61975
C - 146090
D - 108173
E - 363325
F - 38763
G - 80206
H - 86490
I - 297800
J - 5073
K - 25618
L - 187617
M - 99955
N - 242172
O - 241766
P - 109040
Q - 5696
R - 237931
S - 245420
T - 223998
U - 126597
V - 32467
W - 22053
X - 10112
Y - 67946
Z - 13932
```

这下我们的字母已经都变成大写了，我们还剩最后一点没做：对这 26 行输出根据字母出现次数排序。

为了实现这个，我们需要用到 sort 命令，sort 命令用于对文件的行进行排序。

我们还需要一个中转的文件，用于暂时储存我们的 echo 命令循环输出的这 26 行数据。

因此我们可以用输出重定向来把 `echo "$char - `grep -io "$char" $1 | wc -l`" | tr /a-z/ /A-Z/` 的结果依次写入一个文件，比如取名为 tmp.txt。

然后再用 sort 命令对这个文件的行进行排序，把排序结果显示到终端。

我们的函数改为如下：

```
# Definition of function
# 函数定义
statistics () {
  for char in {a..z}
  do
    echo "$char - `grep -io "$char" $1 | wc -l`" | tr /a-z/ /A-Z/ >> tmp.txt
  done
  sort -rn -k 2 -t - tmp.txt
  rm tmp.txt
}

# Use of function
# 函数使用
statistics $1
```

我们在 `echo "$char - `grep -io "$char" $1 | wc -l`" | tr /a-z/ /A-Z/` 之后加了 `>> tmp.txt`，以把输出重定向到文件 tmp.txt 末尾。

然后用 sort 命令对 tmp.txt 文件中的行进行排序。

我们用了 sort 命令的 -r，-n，-k 和 -t 四个参数。

其中 -r 和 -n 参数我们比较熟悉，-n 参数用于对数字排序，-r 参数用于倒序排列。

-k 参数用于指定根据哪几列进行排序，这里用 `-k 2` 表示根据第 2 列来排序。

-t 参数用于指定列和列之间用什么作为分隔符，这里用 `-t -` 表示分隔符是 - 。

然后每次我们都要把 tmp.txt 这个临时文件删除，用 `rm tmp.txt` 。

我的最终代码：

```
#!/bin/bash

# Verification of parameter
# 确认参数
if [ -z $1 ]
then
   echo "Please enter the file of dictionary !"
   exit
fi

# Verification of file existence
# 确认文件存在
if [ ! -e $1 ]
then
   echo "Please make sure that the file of dictionary exists !"
   exit
fi

# Definition of function
# 函数定义
statistics () {
  for char in {a..z}
  do
    echo "$char - `grep -io "$char" $1 | wc -l`" | tr /a-z/ /A-Z/ >> tmp.txt
  done
  sort -rn -k 2 -t - tmp.txt
  rm tmp.txt
}

# Use of function
# 函数使用
statistics $1
```

上面只是我的解法，你的解题思路和代码当然不必和我一样。而且我也非常肯定我的代码不够优。

我相信各位能想出更好的解法，欢迎留言补充（如果留言支持代码，可以把你的代码贴出来）。

这个程序虽然短小，但是我们用到了 Linux 中的 grep 命令、sort 命令、wc 命令、rm 命令、echo 命令、exit 命令、管道 （ | ）、重定向（ >> ）。Shell 中的条件语句 （ if ）、循环语句 （ for ）、函数等知识点。

4. 可能的优化

我给出的解法是基础的，你可以自由发挥。

下面提出几点优化的设想：

1. 除了第一个参数，也就是要统计的字典文件的名字，我们还可以添加其它参数，来完成更多任务；
2. 改变输出的形式，使之更美观；
3. 每一行可以输出更多信息；
4. 尝试不借助中间文件 tmp.txt。

其它优化，就有待大家去发挥自己的想象力咯！

今天的课就到这里，一起加油吧！

}

46 Shell实现图片展示网页

48 测试题