# 第一章 sed与awk 初识



1. **Sed**

**1, sed 对于文件内容替换**

Abc.txt

Sed ‘s/ good/bad/’ abc.txt

1. **对于多命令解析 -e**

Sed -e ‘s/ good/bad/’ -e ‘s/name/life/’ abc.txt

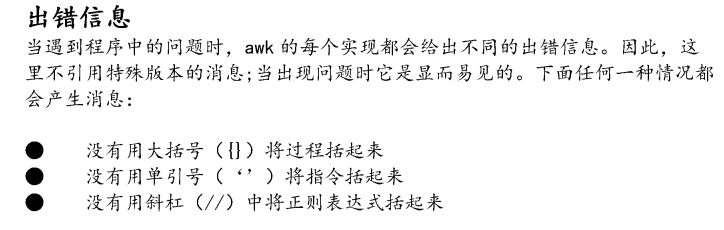
1. **对于解析文件指令**

Sed -f file abc.txt

这里注意两个问题 1.file文件中的换行 不再需要 \ (仅作为cmd换行) 2.文件中不再需要单引号 **3.输出的结果只做展示，实际内容并未改变**

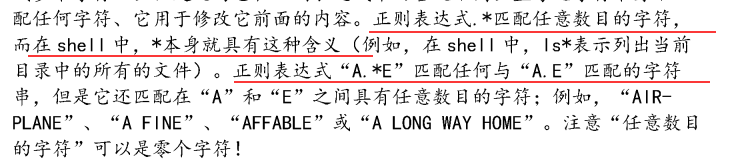
**4 sed -n 's/hello/get/p' abc 注意-n 配合 p 可以输出 只做修改的行**

1. **Awk 命令**
2. **awk 解析文件参数 -f**
3. **Awk ‘/hello/’ abc.txt 会解析所有匹配到的行**
4. **Awk ‘/hello/{print $1}’ 输出匹配到的行的第一个字段**
5. **Awk -F, ‘/hello/{print $1}’ 注意 -F 转换分割符为,**



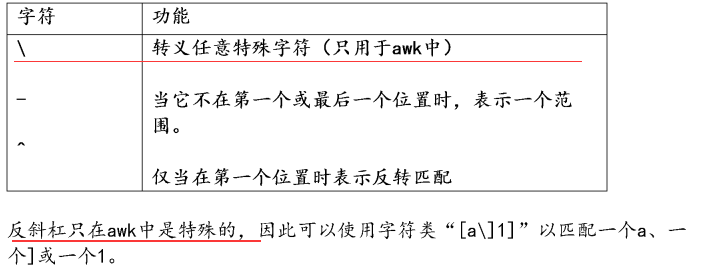


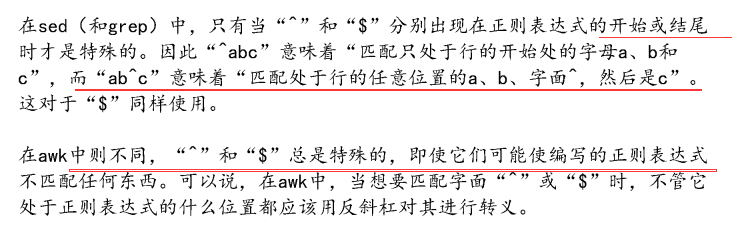
**3.正则表达式**



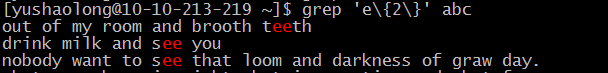
注意 ： sed 与 awk 的正则表达式区分 大小写 ，某些程序使用i 不区分大小写

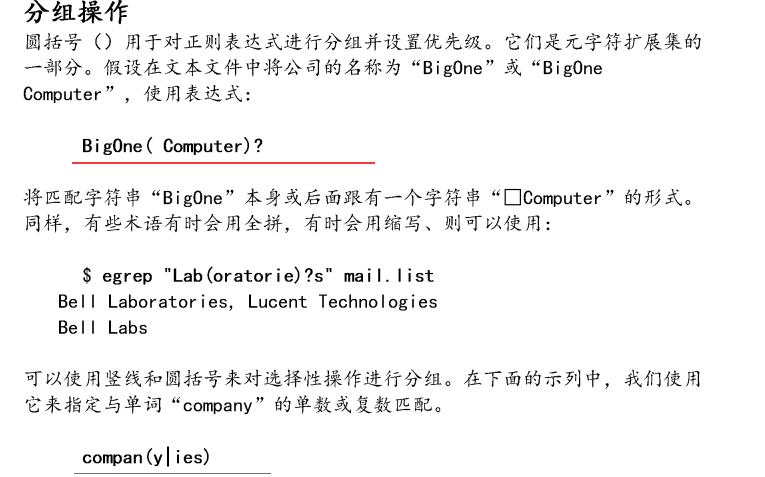
4





这里注意 对于 grep指令使用 { } 时，使用 \ \





1. 对于shell中的标准输出

**nohup /mnt/Nand3/H2000G >/dev/null 2>&1 &**

对于& 1 更准确的说应该是文件描述符 1,而1

    一般代表的就是STDOUT\_FILENO,实际上这个操作就是一个dup2(2)调用.他标准输出到all\_result

,然后复制标准输出到文件描述符2(STDERR\_FILENO),其后果就是文件描述符1和2指向同一个文件表项,也可以说错误的输出被合并了.

**其中0 表示键盘输入 1表示屏幕输出 2表示错误输出.**

把标准出错重定向到标准输出,然后扔到/DEV/NULL下面去。通俗的说，就是把所有标准输出和标准出错都扔到垃圾桶里面。

**command >out.file 2>&1 & --> command 1>out.file 2>&1 &**

     command

    >out.file是将command的输出重定向到out.file文件，即输出内容不打印到屏幕上，而是输出到out.file文件中。

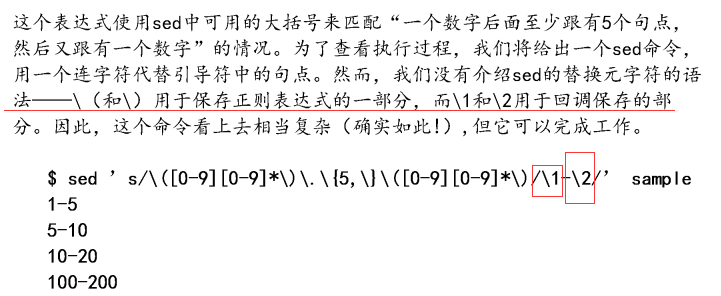
    2>&1

    是将标准出错重定向到标准输出，这里的标准输出已经重定向到了out.file文件，即将标准出错也输出到out.file文件中。最后一个&

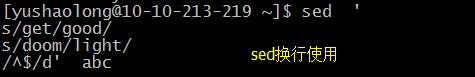
    ， 是让该命令在后台执行。

好了，现在的基本知识都已经具备了，让我们来讲讲开头的那个例子吧。

3>&1表示将标识符3复制成和1一样，都是标准输出。  
1>&2表示将标识符1复制成和2一样，此时1成了标准错误输出。  
2>&3表示将标识符2复制成和3一样，由于前面的第一步我们将3变成了标准输出，现在2和3一样，也是标准输出。  
最后的3>&-将标识符3关闭。也就是说3相当于是一个中间的临时变量，用来交换程序的标准输出和标准错误输出。

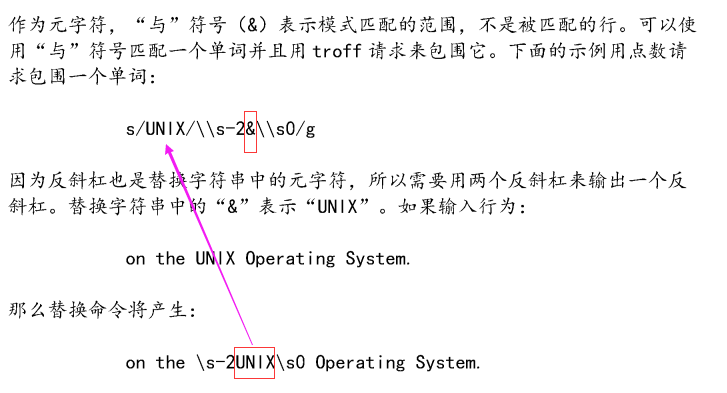


7.

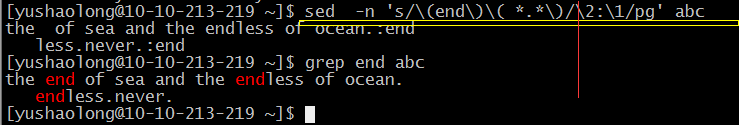


# 第二章.Sed 命令

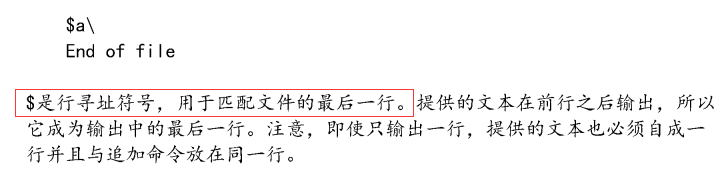
1.对于 & 使用可以提取匹配的第一个参数



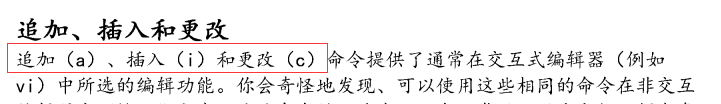
1. **对于sed中的 / 1 / 2 / 1使用正则表达式， 2内部 直接写字符串即可 如 . 可以不用考虑使用\转义**

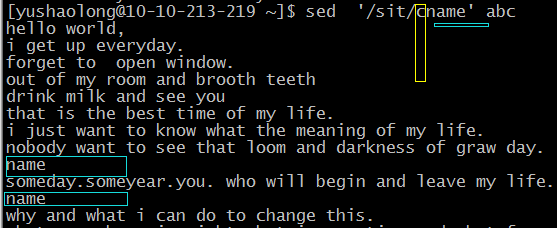


1. **/^$/d s/ / /**

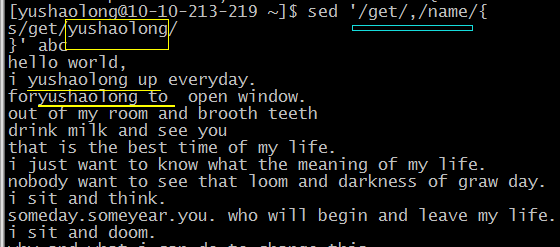


1. **A i c**

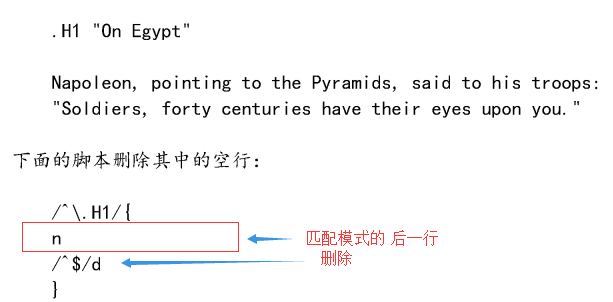




**对于内嵌 { } 方式 实施**



**5. N （next ） 后一行**

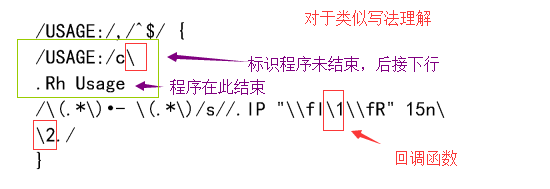


1. **读写命令 r w ??（存在问题）**

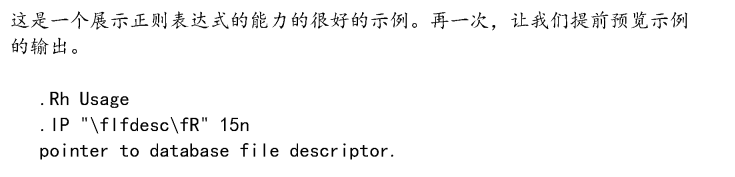




**7.对于 换行 \ 的 理解**

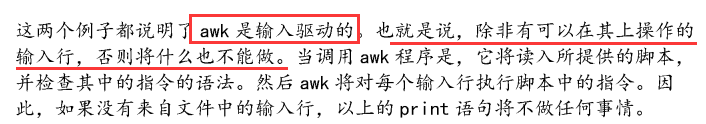


输出如下:

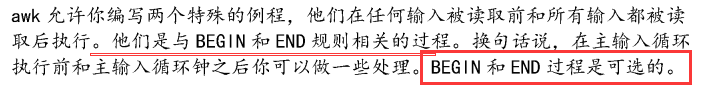


# .AWK命令

**1.输入驱动 ，读取文件 或者 键入字符串**



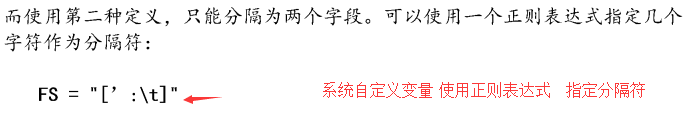
**注意 BEGIN 与 END 是两种不同的规则**



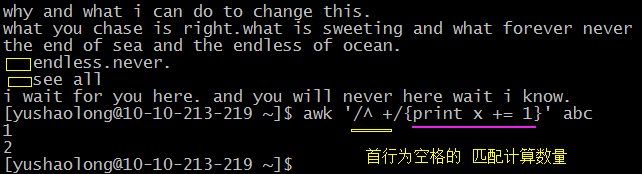
**2.对于 $2 ~ /sit/ 匹配 第二个为 sit 的 行 , (！~) 表示不等于**



**3 对于系统自定义变量 FS**



**4. 变量自动初始化为 0 ，内部循环相加**



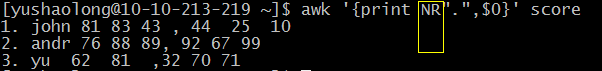
1. **对于awk 系统变量**

**FS 分隔符变量，可以为一个符号 或者正则表达式**

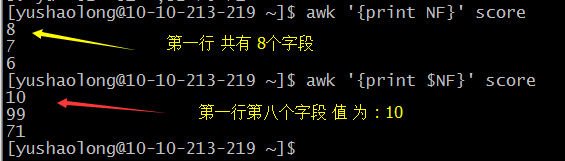
**OFS 输出分隔符重定向**

**FILENAME 文件名称**

**NR 指代每一行的行数**



**NF 每一行最后一个字段的 变量数**



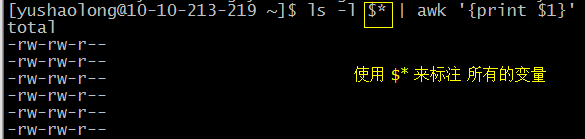
**6 awk 的运算符**

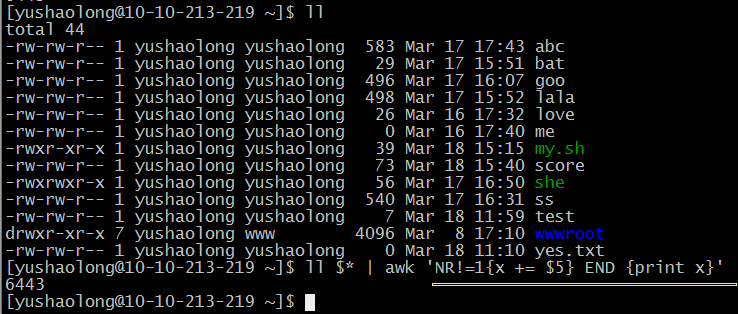
== , != , ~ , !~ , > , >= , || , && , !

++ -- += %=

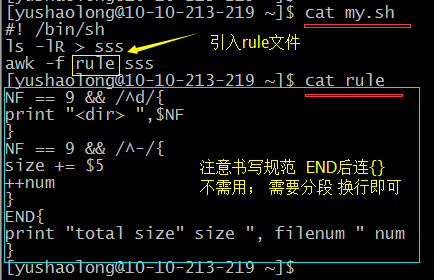
注意 ： 1. ~ 匹配 2. !~ 不匹配

注意 ： && 优先级 比 || 高

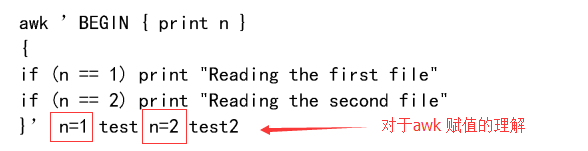




**7，一个简单脚本的编写**

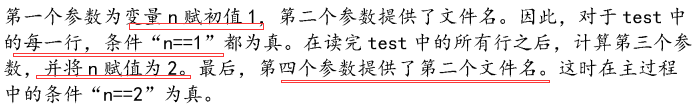


**8 awk 赋值**

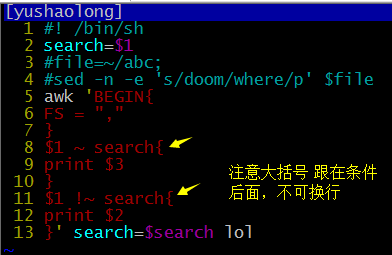


对于上面代码, BEGIN会先执行 ，所以 n=1 赋值与之无关 ， 所以会首先输出空行

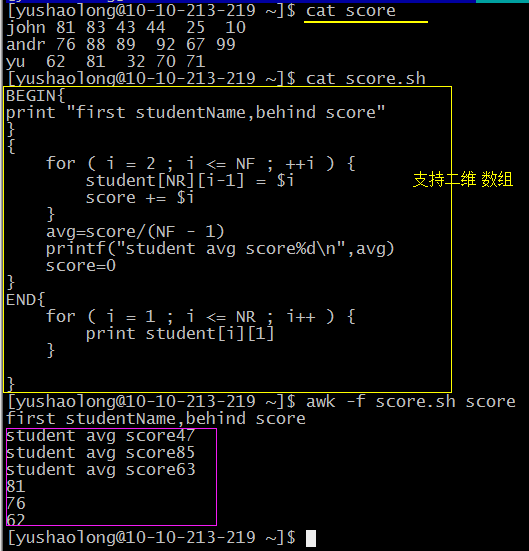
然后 开始解析赋值 n=1 ，然后 test .......，如下说明



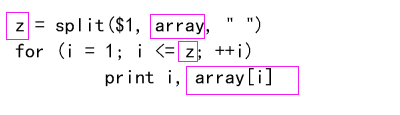
9，编写awk脚本，，注意变量$search 在  **awk 命令行中** 解析，以及 **大括号在条件后面**



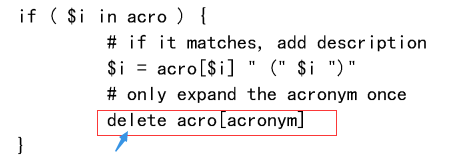
1. **二维数组**



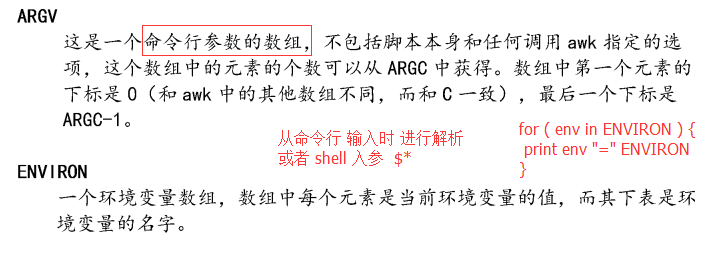
1. **split 函数的使用**



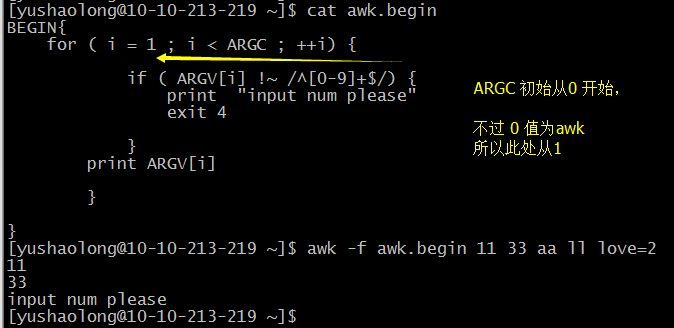
1. Delete 删除数组下标



1. **数组系统变量**

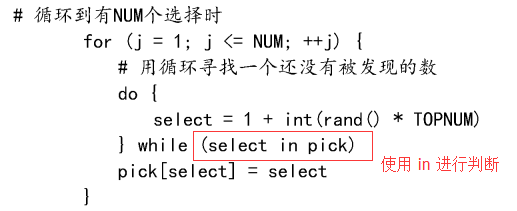


**14 . awk用例**

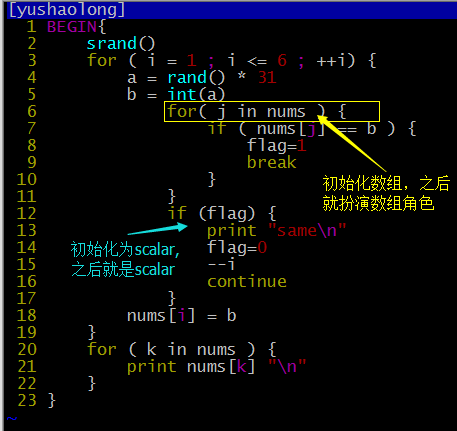


15 编写 0 -30 不同六个随机数 awk 脚本

1. 先用 srand() 生成种子
2. 然后随机生成
3. 以下示例可以用while循环



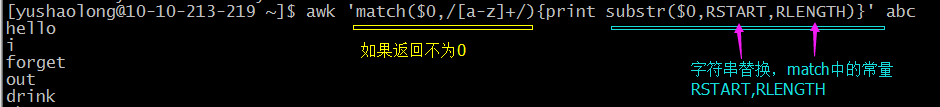
以下使用for 循环

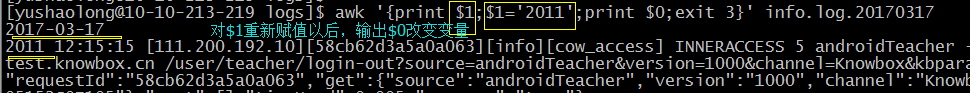


**16 awk 函数**

(1) Substr()

(2) Match() ==> match 常量 **RSTART**初始匹配的位置； **RLENGTH** 匹配的字符串长度

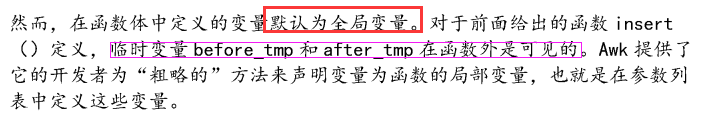




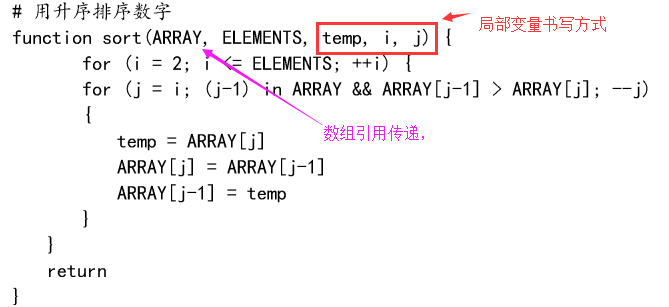
**17 awk 自定义函数**

**自定义脚本里 注意使用 “” 而不是‘’进行字符串传参**

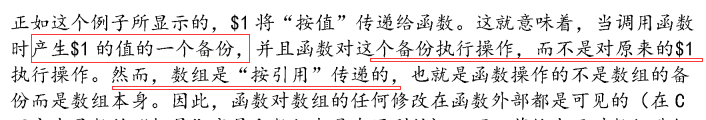
(1)注意 awk 自定义函数，内部变量默认为全局变量；



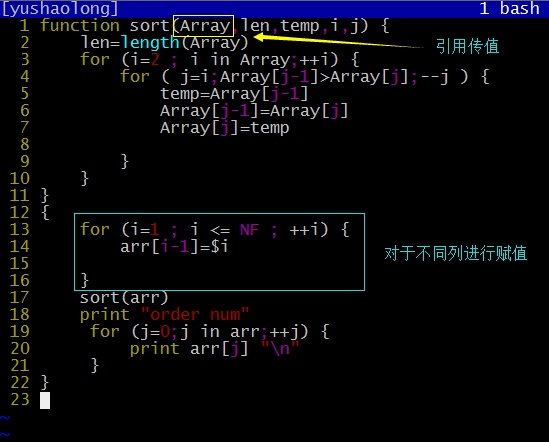
(2)可以使用传参的方式进行 局部变量的书写



**(3)注意 字符串传值是 拷贝传递，数组传值是引用传递**



Awk脚本示例



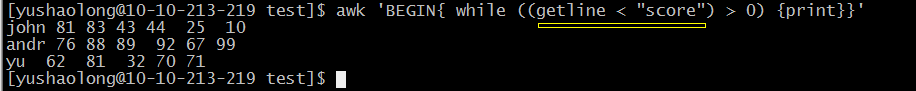
**18 底部抽屉**

**getline函数**

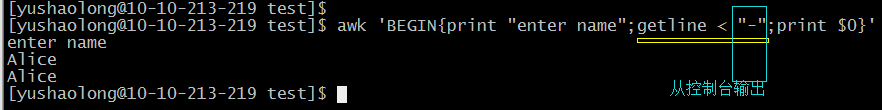
1. getline函数 输出下一行 : 获取成功，则返回 1 ；行末尾，返回0；遇到错误 -1；

可以使用 getline 读取文件中的数据，使用输入重定向符号 <

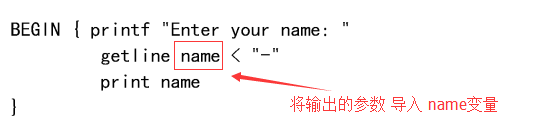
**(1)如下 读取 score 文件的内容，并输出**



**(2)从控制台中输出 变量， 进行解析： 符号 < “-”**



**(3)将控制台参数导入变量 name**

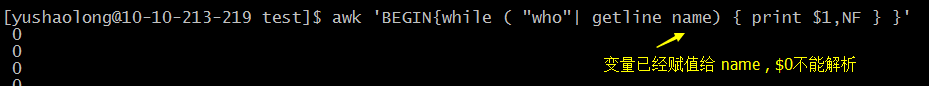


注意 getline 与 next 区别:

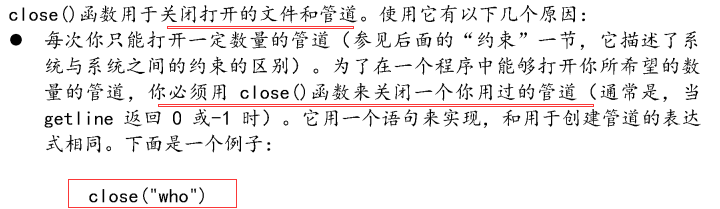
Getline 获取下一行的输入行，不改变程序控制；而next则从文件控制传递到脚本顶部

**对于getline的应用实践**





**Close函数**



**对于 -v 的语法学习**

awk 'BEGIN{print abc}' abc=2 //在BEGIN之后才执行，输出abc为空

awk  **-v abc=2**  'BEGIN{print abc}' //-v abc 可以在 BEGIN前赋值