# *CITS4407Note*

仅为课堂笔记，配合2022S1课件食用口味更佳。

By 七皇

2023.05.09

看到这篇笔记的学弟学妹们，如果你是在week3之前看到，直接移步B站找个系统的教程看

如果在week7 week8这种时间点还没怎么学的话，就别去看教程了，照着Michael的lab自己敲几遍，哪里不会Google哪里，计算机不是文科，看视频记参数不管用，还得多敲多敲多敲

### Linux文件权限

文件权限：rwx 读 写 执行ugo: user group others

rw-|---|--- 仅user可以读写

chmod <octal mode> <files>

chmod <symbolic mode> <files>

增加/减少权限

chmod ugo+rwx myapp

chmod go-rw myapp

文件夹权限 前面加ddrwx

### Redirection重定向

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件：

* 标准输入文件(stdin)：stdin的文件描述符为0，Unix程序默认从stdin读取数据。
* 标准输出文件(stdout)：stdout 的文件描述符为1，Unix程序默认向stdout输出数据。
* 标准错误文件(stderr)：stderr的文件描述符为2，Unix程序会向stderr流中写入错误信息。

< Redirect standard input

> Redirect standard output

>> Redirect and append standard output to the named file >> 2> Redirect standard error output

2>&1 : 将标准错误输出重定向到标准输出；2表示标准错误信息，1表示标准输出，&表示通道，如果直接2>1就是把标准错误输出重定向到文件名为“1”的文件中里了

[命令] >log.txt 2>&1这里 >log.txt 表示[命令]的标准输出被重定向到log.txt里，然后2>&1标准错误被重定向到标准输出里，而标准输出又被重定向到log.txt里，这样一来[命令]的标准输出和标准错误信息就都会被记录在log.txt里了。

& 单独一个&表示后台进行

### DIY scripts

#!/usr/bin/env bash

Or

#!/bin/bash

### gzip

gzip file\_name 压缩文件

gzip -d file\_name 解压缩

​

gzip large\_file.txt > trace 2> errs &

在后台压缩文件，标准输出写入到trace中，标准错误被写入到errs中

在命令行最后加上 & 表示后台运行

### ps

ps <options> 打印进程信息

-l 打印一个长列表包含更多信息

-f provides more information about the command (which options were used)

### top

列出所有进程和cpu使用

Lists a range of statistics about all the processes, and then lists the processes one by one in order of their CPU use. Useful for finding that process that has gone off the rails.

### kill <options> <PIDs>

kill -9 XXXX

* 1 (HUP)：重新加载进程。
* 9 (KILL)：杀死一个进程。
* 15 (TERM)：正常停止一个进程。

### cut

cut [ -d <delim>] -f <list> <files>

-f The comma separated list following –f specifies the fields that are to remain

-d Specifies an alternate delimiter

​

e.g.

cut –d , -f 1,5-7 datafile.csv 把datafile.csv按逗号分割，显示1和5-7列。

### paste

paste [ -d <sep> ] <files>

Steps through one or more files in parallel. The output is a line containing all the first lines (separated by <sep>, say the Tab char), followed by a line containing all the second lines, and so on.

把两个文件的每一行分别拼成一行显示

### tr

tr <options> <string1><string2>

替换：把string1替换为string2

tr ’[A-Z]’ ’[a-z]’ < textfile 把textfile的大写字母全替换为小写字母

参数：

-C Complement characters in the first string 反选，不符合string1的才会被替换-s Squeeze multiple contiguous occurrences of string1 多个连续的string1被视为一个进行替换

\012 = \n

### comm

comm [<output options>] <file1> <file2>

The two **sorted** files are compared, and three column output is produced. Column 1 contains those lines only found in <file1>, Column 2 contains lines only in <file2>, Column 3 contains lines common to both files.

对比两个排好序的文件，输出三列，输出的第一列只包含在 file1 中出现的，第二列包含在 file2 中出现的，第三列包含在 file1 和 file2 中都出现的。

### uniq

uniq [-cdu][输入文件][输出文件]

uniq 可检查文本文件中重复出现的行列。

* -c或--count 在每列旁边显示该行重复出现的次数。
* -d或--repeated 仅显示重复出现的行列。
* -u或--unique 仅显示出一次的行列。

### wc

参数

-c # 统计字节数，或--bytes：显示Bytes数。

-l # 统计行数，或--lines：显示列数。

-m # 统计字符数，或--chars：显示字符数。

-w # 统计字数，或--words：显示字数。一个字被定义为由空白、跳格或换行字符分隔的字符串。

-L # 打印最长行的长度，或--max-line-length。

wc -l \*       # 统计当前目录下的所有文件行数及总计行数。

wc -l \*.txt    # 统计当前目录下的所有 .txt 后缀的文件行数及总计行数。

find -exec wc -l {} \; # 当前目录以及子目录的所有文件行数及总计行数。

​

​

#查看文件的字节数、字数、行数

wc test.txt

# 输出结果

 7    8     70     test.txt

#行数 单词数 字节数     文件名

### sort

sort [options] files

参数：

-u Removes duplicated entries following the sort (i.e. as if uniq had been applied) 输出的结果是去了重的

-o <输出文件> 将排序的结果写入到文件里

-t <分割字符> 指定排序时所用的栏位分隔字符

-k filed1, field2 Instead of sorting from the first field, sort from the <start> field. If <end> is specified, only the fields from <start> to <end> are sorted. 按照指定的列进行排序，从filed1到field2，如果没有field2，默认到结尾 -k和-t结合使用，表示取那一段为关键字进行排序，后面跟数据，1…n，表示取第几段，也可以是范围如1,3，表示将第1段到第3段作为一个整体来排序

-n 以数字进行排序，默认升序

-r 倒序（降序）

e.g.

sort –u –t , -k 4nr file.csv

参数说明：

Sort on fourth column, removing duplicates, contents of csv file.

-u 去重

-t , 按逗号分隔

-k 4nr 按照第四列以数字进行降序排序

### shell变量

<variable>=<single value>

​

NOTE: NO spaces around =

等号两边不可以加空格

$后加括号：

%  TimeNow=date

% TimeNow

>> 可以输出当前时间，效果和date一样

%  echo "The time now is $TimeNow"

>> The time now is date #这里输出的是“date”而不是时间

%  echo "The time now is $(TimeNow)"

>> bash: TimeNow: command not found

%  echo "The time now is $(date)"

>> The time now is （可正常输出时间）

%  echo "The time now is $($TimeNow)"

>> The time now is （可正常输出时间）

​

%  $(TimeNow)

>> bash: TimeNow: command not found

%  $($TimeNow)

>> bash: Thu: command not found

### $()

$() 的解释：

$()表示将()里的内容作为命令执行后，得到的输出再进行执行返回结果，这里TimeNow不是命令而是自己在前面定义的变量，所以会报错

echo "The time now is $($TimeNow)" 这里$(TimeNow) 在echo中其实是date，所以这里的输出和 echo "The time now is $(date)" 是一样的

而直接在控制台输入 $(TimeNow) ，在控制台的$(TimeNow)就是TimeNow把TimeNow当作命令直接执行，而TimeNow是变量，所以会报 bash: TimeNow: command not found 的错

在控制台输入 $($TimeNow) ，这里括号里的$TimeNow是变量，指向date，而date的执行结果是日期，日期第一个返回值是星期四Thu, 所以会报 bash: Thu: command not found 的错

总结：$表示变量，可直接返回变量的结果，$()表示命令替换，返回括号里命令的执行结果

**命令位置**

In Unix, commands (aka executables, binaries) can live in a number of places, public or private

* /bin top level system commands, cat, echo, ls 系统命令存放在 /bin
* /usr/bin system commands that generally come with the system, but may need to be installed, gzip, make, zip, gcc, python 系统命令（需安装）存放在/usr/bin
* /usr/local/bin programs that you will have installed for all users of your system locally. hmmsearch, pip3, ps2pdf 用户命令/usr/local/bin
* ~/bin perhaps you have your own personal directory for programs 个人程序内部的命令
* . Perhaps the program you need is in the same directory you are in now

**PATH** ：The list of directories the system traverses (in order) to find the named command 一个字符串变量，当出入命令的时候系统会查找在PATH里记录的路径

**HOME** is your personal root directory (i.e. ~)**CD\_PATH** short-cuts when doing cd to a named (sub)directory

.profile 文件：用于设置用户登录后自动执行的命令

/dev/null 空设备

some\_command > ouput\_file 2> /dev/null 相当于把输出全部扔到垃圾箱里不看了

### test

test <test expression>

is a program that doesn’t output anything, but rather executes the test expression and returns the outcome in the exit status

* -d <directory> Tests if the directory exists
* -f <file> Tests if the file exists as ordinary file
* -s <file> Test if the file exists and is not 0 length
* -n <string> Is the string non-zero length
* -z <string> Is the string zero length

if <condition>

then

<statements>

[elif <condition>

<statements>] .....

[else

<statements>]

fi

### Globbing/wildcards

[ <letters> ] – Match any one of these letters (can be a range)[!<letters> ]– Match any one letter BUT NOT any of these (can be a range

\* matches anything in a filename, e.g. ls x\* matches: x, x.bak, x.txt, x1, xfer\_peter (The last is a directory)

? Matches a single letter. In the list above x? onlymatches x1

\* 匹配任何

? 匹配单字母

ls /bin/g[!e]\*t

### case

case <expression> in

<pattern> [ | <pattern> .... ] ) <statements> ;;

<pattern> [ | <pattern> .... ] ) <statements> ;;

.......

esac

* The <expression> is anything (i.e. variable or command) that returns a string
* Each <pattern> can be of the sort used in file-name generation, including wild-cards.
* Alternate patterns, but with the same set of actions, are separated by |.
* The expression is evaluated and the result tested against the patterns top-down, left-to-right across alternative patterns separated by |.
* If a match, the corresponding statements are executed (up to the ;; ).
* If no patterns match, execution proceeds to the next statement
* \*matches every string, so is used as the default pattern (i.e. just like else).

case $DAY in

Mon | Fri ) echo ${DAY}day ;;

Tue | Thu ) echo ${DAY}sday ;;

Wed ) echo ${DAY}nesday ;;

S??) echo WEEKEND! ;;

\*) echo "$DAY is not a day I understand."

echo "April Fool’s Day?" ;;

esac

### for

for <name> in <list>

do

<commands>

done

<list> is just a white-space list of strings

${#var}可计算出变量的长度：

%  a=cxndicd

%  echo ${#a}

>> 7

${!var}引用间接变量

%  v1="v2"

%  v2="hello"

%  echo ${!v1}

>> hello

Write a script, longest\_line which, given a text file, reports the longest line and its length.

#!/bin/bash

​

if [[ $# -ne 1 ]]

then

   echo "Usage $0 <file\_name>." > /dev/stderr

   exit 0

fi

​

max\_length=0

​

for i in $(<$1)

do

   if [[ ${#i} -gt $max\_length ]]

   then

       max\_length=${#i}

   fi

done

​

echo "The length of the longest line is $max\_length"

unset max\_length

遍历files在一个directory里：

for i in \*

do

...

done

### shift

用于对参数的移动(左移)，通常用于在不知道传入参数个数的情况下依次遍历每个参数然后进行相应处理

testshift文件如下:

#!/usr/bin/env bash

echo "$# : $\*"

shift

echo "$# : $\*”

​

​

% test shift 1 two "two 1/2" 2.99994

4 : 1 two two 1/2 2.99994

3 : two two 1/2 2.99994

e.g. Want to generalise count\_occurences to have optional -topN argument (if absent, assume all)

#!/bin/bash

#Exact words in text file, return top N by count

​

if [[ $# == 0 ]]

then

echo "Usage: $0 [<topN N> <text\_file>]" > /dev/stderr

echo "If -topN not specified, assume all"

exit 0

fi

​

topN=-1

while [[ $# -gt 1 ]] #如果参数个数大于1，就开始遍历参数

do

case $1 in

-topN ) topN=$2; sh

ift; shift;; #如果第一个参数是-topN，把第二个参数赋给topN，然后shift两次，这样$1就直接指向了文件

-\*) echo "Unknow option $1" > /dev/stderr; exit 1;; #如果第一个参数是其他的，报错退出

\*) break;; #如果是其他，直接break

esac

done

​

if [[ ! -s $1 ]] #此时$1是文件名，排除此文件不存在或长度为0的情况

then

echo "$1 is not a file or has zero length" >/dev/stderr

exit 1

fi

​

#name=$(basename $1 .txt)

#case $topN in

# -1) tr -cs '[A-Za-z]' '\n' < $1 | sort | uniq -c | sort -k 1nr > ${name}\_counts.txt;;

# \*) tr -cs '[A-Za-z]' '\n' < $1 | sort | uniq -c | sort -k 1nr | head -n$topN >${name}\_counts.txt ;;

#esac

​

case $topN in #如果是-1，就统计到底；如果不是，就传入head里

-1) tr -cs '[A-Za-z]' '\n' < $1 | sort | uniq -c | sort -k 1nr ;;

\*) tr -cs '[A-Za-z]' '\n' < $1 | sort | uniq -c | sort -k 1nr | head -n$topN ;;

esac

### shell的数学运算符

$(( <expression> )) evaluates the expression and returns the result on stdout.

$(( )) 就直接看作十进制数的运算即可，由于shell里最基本的是字符串，shell不会自动区分数字，所以在shell中a=123+1 执行后，再echo $a 返回的是 123+1 而不是 124 。

加上双括号后，((a=123+1)) 运行后，再echo $a 返回的就是124 了。

而 a=$((123+1)) ，就是说明$(( )) 里的是数字而已。

#Example:

a=1

a=$((a + 1))

echo $a

上述例子中，a=$((a + 1)) 也可以写成 a=$(($a + 1)) ，在双括号结构中，所有变量可以不加入$符号前缀。

### 关于带$和不带$的括号总结

原帖：Double parenthesis with and without dollar

* $(...)表示在子shell 中执行括号中的命令并返回其标准输出。

$ echo "The current date is $(date)"

The current date is Mon Jul  6 14:27:59 PDT 2015

* (...)表示在子 shell 中运行括号中列出的命令。

$ a=1; (a=2; echo "inside: a=$a"); echo "outside: a=$a"

inside: a=2

outside: a=1

* $((...)) 表示执行算术并返回计算结果。

$ a=$((2+3)); echo "a=$a"

a=5

* ((...))表示执行算术运算，可能会更改 shell 变量的值，但不返回其结果。

$ ((a=2+3)); echo "a=$a"

a=5

* ${...}表示返回括号中命名的 shell 变量的值。

$ echo ${SHELL}

/bin/bash

* {...}表示将大括号中的命令作为一个组执行。

$ false || { echo "We failed"; exit 1; }

We failed

### find

在指定目录下查找文件，默认当前目录

find <path> <options>

​

find -type d #查找当前目录下的文件夹

find -type f #查找当前目录下的文件

find -perm 700  #查找当前目录下权限为700的文件和目录

find -newer Alice\_in\_Wonderland.txt -type f -name "\*.txt" -print

#在当前目录下查找比 Alice\_in\_Wonderland.txt 新的，后缀是txt的 文件

​

-exec 参数：

格式：-exec <命令> {} \;

#可以在find命令找到对应的文件之后，对每个文件执行-exec后的命令

find . -name "\*.txt" -exec ls -l {} \; #查找当前目录及子目录下的所有.txt文件并用ls -l列出

find . -name "\*.txt" -exec chmod o-x {} \; #查找当前目录及子目录下的所有.txt文件并给others组添加可执行的权限

#对应的可以用mv，cp等命令实现批量移动和复制文件

​

find . -type f -exec ls -l ’{}’ \; #当前目录下所有子目录进行ls -l

find . –perm 600 –exec chmod 644 ’{}’ \; #把当前目录下所有权限为600的文件和文件夹改为644权限

find . -exec wc -l {} \; # 当前目录以及子目录的所有文件行数及总计行数。

### 正则

Wild-card Patterns 通配符

[正则表达手册](https://tool.oschina.net/uploads/apidocs/jquery/regexp.html)

|  |  |  |
| --- | --- | --- |
| ed编辑器 | shell命令行 | 描述 |
| . | ? | 匹配单个字符 |
| [ ] | [ ] | 从一个集合里匹配单个字符 |
| [^ ] | [^ ] | 匹配除包含在这个集合里的字符以外的单个字符 |
| \* |  | 匹配前面的一个字符，0次或多次 |
| ..\* | \* | 匹配任意字符，1次或多次 |
| ^ |  | 起始符，表示String的开头 |
| $ |  | 终止符，表示String的结尾 |
| \ | \ | 转义，比如 \\ 表示单斜杠，\n 表示换行等 |
| \( \) |  | 捕获匹配以便以后重用 |

e.g.

malloc \*( every instance where there was at least one space following the word malloc before the bracket.

.\* 任何字符（包括空串）

[0-9][0-9]\*\.[0-9][0-9]\*

数   数\*   . 数   数\*

表示匹配 一个数+0或多个数+点+一个数+0或多个数

^$ 空串

\<N> 表示第N个记录的匹配

\([a-z]\)\([a-z]\)\1\2

匹配abab，不匹配aBBa，即第一个是[a-z]；第二个是[a-z]；第三个是第一个记录的匹配，即第一个字符；第四个是第二个记录的匹配，即第二个字符；

xxxx也符合

### grep

[grep参数说明](http://linux.51yip.com/search/grep)

grep [<options>] <regular-expression> <file> ...

-i Make comparisons case insensitive (“i” matches “I”) 忽略大小写

-n Prepend the numbers of the matching lines 在显示符合样式的那一行之前，标示出该行的列数编号。

-v Invert the match so only non-matching lines are reported 显示不包含匹配文本的所有行。

• grep ’ˆ\.[VABL][LIE]$’ file

#一个点后接俩字母，第一个VABL之一，第二个LIE之一

​

• grep -v ’warning:’ errs | grep -v ’In function’

#匹配不包含“warning:”的行，再在结果上匹配不包含“In function”的行

​

• find . -type f -exec grep -n awk ’{}’ \; -print

#在当前目录及其子目录下查找所有文件，然后在每个文件里寻找含有“awk”的行，标出行数并打印出对应的文件名。

# find . -type f ... '{}' \;         -exec grep awk           -n           -print

​

# 在当前目录及其子目录下查找所有文件   find . -type f ... '{}' \;

# 然后在每个文件里寻找含有“awk”的行 -exec grep awk

# 标出行数 -n

# 并打印出对应的文件名 -print

### sed

[sed参数说明](http://linux.51yip.com/search/sed)

参数：

sed <options> <file> ...

-e <sed operation>   Apply this one operation to the file(s) 直接处理输入的文本文件

-f <file of sed operations>   A file of sed operations, executed top-down on each line

以选项中指定的script文件来处理输入的文本文件

-n Transformed output not sent to stdout (need p) 仅显示script处理后的结果，关闭默认的输出

-i 直接操作文件

sed操作：

* a ：在当前行的下一行新增行

sed -e 2a\hahahahah testfile

在testfile的第二行下面新增一行

* c ：取代行

sed -e 2c\hahahahaha testfile   把testfile的第二行替换

* d ：删除

sed -e 2,5d testfile   删除testfile的第2-5行

sed -e 2d -e 5d testfile 删除testfile的第2行和第5行

* i ： 在当前行的上一行新增行

sed -e 2i\hahahahah testfile

在testfile的第二行上面新增一行

* p ：打印，亦即将某个选择的数据印出，一般和-n连用

sed -n 1,3p testfile   打印第1-3行

sed -n 'p;n' testfile   打印奇数行

sed -n 'n;p' testfile   打印偶数行

循环执行npnpnp，就会跳过奇数行只打印偶数行

循环执行pnpnpn，就会跳过偶数行只打印奇数行， 同理 sed -n 'p;n;n' testfile 就是打印第1，4，7...行

​

l This option is similar to p, except that nonprintable characters are displayed and long lines are folded

l和p一样都是打印，但l会把转义字符也打印出来

* s ：替换

sed 's/要被取代的字串/新的字串/g' testfile   g表示全局替换

sed 's/要被取代的字串/新的字串/' testfile   表示只替换第一次出现的

查找：

sed -n '/要查找的字符串/p' testfile

-n 仅显示script处理后的结果

$ 表示文件末尾，sed -n '4,$p' testfile 表示打印从第四行到末尾的内容

. 表示当前行

0 表示在第一行之前的行

### awk

[awk参数说明](http://linux.51yip.com/search/awk) [awk官方手册](http://www.gnu.org/software/gawk/manual/gawk.html)

\*笼统的说，sed是用来处理行的，awk是用来处理列的

#### awk fields

$0 refers to the entire line.$1 refers to the first field, $2 to the second, etc.

$0就是整行，$1 就是第一列，$2 就是第二列

awk '{print $0}' test\_file

打印test\_file的全部行

awk默认空格或者制表符作为分割符，$1就是打印每一行的第一列，$2第二列……

指定分隔符 -F

awk –F# ’{print $2 "#" $1}’ test\_file 按#分割，然后先打印第二列，再打印第一列，用#连接

#### awk varibles

<variable> = <expression>

|  |  |
| --- | --- |
| 内置变量 | 说明 |
| ARGC | Command-line argument count 命令行参数个数 |
| ARGV | Array of command-line arguments 命令行参数数组 |
| FILENAME | Name of input file (there may be several) 输入文件名 |
| FNR | Index of line in current file 当前文件的当前行索引 |
| FS | Input field separator (default blank, tab) 输入的分隔符 |
| NF | Number of fields in the current line 当前行有多少列 |
| NR | Index of current line (from start of first file) 从第一个文件开始的当前行索引 |
| OFS | Output field separator (default blank) 输出的分隔符 |

awk '{a=b=c=42;print a "\t" b "\t" c}' 会打印出abc都是42

# add\_this.txt 在虚拟机的week9里

awk '{print $2 "#" $1 "#"| "sort -t# -k2nr"}' add\_this.txt | sed -e 's/#/: /' -e 's/#/ /'

打印add\_this.txt的第二列，，然后#，然后第一列 print $2 "#" $1

将输出用#分割，按第二列的数字排序 sort -t# -k2nr

最后传入sed中，查找含有#的行，把第一次出现的#替换为一个冒号加一个空格 sed -e 's/#/: /'

然后再次查找#的行，再把第一次出现的#替换为一个空格

Two special Rules:– BEGIN { <actions> } is executed before the first line is read– END { <actions> } is executed after the last input line has been read

awk '

BEGIN { actions }

/pattern/ { actions }

/pattern/ { actions }

……….

END { actions }

' filenames

printf：格式化打印（需要括号）awk '{printf("%d:%s\n", ++i , $0)}' test\_fileprint：直接打印（括号可以省略）awk '{print ++i ": " $0}' test\_file

|  |  |
| --- | --- |
| 格式化字符 | 描述 |
| %c | 单字符 |
| %s | 字符串 |
| %d | 整型 |
| %e | 科学计数，默认保留6位小数 ( 345 就是 3.450000e+02 ) |
| %f | 浮点数，默认保留6位小数 （ 345 就是 345.000000 ） |
| %g | 去0，默认6位有效数字 （123.456789 就是 123.456，如果数字很大就会变成保留6位有效数字的科学计数 ） |
| %% | 就是百分号% |

**awk匹配**

<expression> <matchop> <regular expression>

~ 表示匹配， !~ 表示不匹配，后接/<regular-expression>/

awk '$0 ˜ /ˆ$/{sum++}END{print sum}' 打印空行数，用的是正则匹配

awk '$0 == ”” {sum++}END{print sum}' 打印空行数，用的是判断

**awk的布尔值**

* 数值0表示False
* 空字符串表示False
* 其余所有均为True，字符串”0”也是True，小数0.5之类的也是True
* awk中，正则匹配也有返回值，匹配成功则返回1，匹配失败则返回0
* awk中，所有的布尔运算也有返回值，布尔真返回值1，布尔假返回值为0

**awk数组**

格式 <array name>[<array subscript>] = <value> 值与值之间用空格隔开

删除 a[10]="" ，或者delete a[10]

比如project里初始化数组可以：

word=("also" "although" "and" "as" "because" "before" "but" "for" "if" "nor" "of" "or" "since" "that" "though" "until" "when" "whenever" "whereas" "which" "while" "yet")

awk打印不重复的行：awk '!a[$0]++' ，等价于 awk '!a[$0]++{print}'

说明：

awk的基本命令格式 awk 'pattern{action}' ,省略action时，默认action是{print}，如awk '1'就是awk '1{print}'

var++的形式：先读取var变量值，再对var值+1

如果第一行和第二行是一样的话，

awk处理第一行时：先读取a[$0]值再自增，a[$0]值为空(0)，即为awk '!0'，即为awk '1'，即为 awk'1{print}'，打印；

awk处理第二行时：先读取a[$0]值再自增，a[$0]值为1，即为awk '!1'，即为awk '0'，即为awk '0{print}'，不打印；

这样就变成只打印不重复的行了。

e.g. （顺便勘误网站[awk参数说明](http://linux.51yip.com/search/awk)的错误）

我有一个testfile：

apple

apple apple

This is second line.

WWW

WWW

WWW

hello worldd

​

然后执行

% cat testfile | awk '!a[$0]++'

apple

apple apple

This is second line.

WWW

hello world

​

整个过程是，在awk内部新建了一个数组a，a[$0]最开始没有被赋值，所以初始化为0，判断为false，取反为true，于是就打印。在这个例子中，读取第一行时，a[$0]即为a['apple']；读取第二行时，a[$0]即为a['apple apple']

​

再执行

% cat testfile | awk '!a[$1]++'

apple

This is second line.

WWW

hello world

​

整体过程与上述一样，在awk内部新建了一个数组a，但这里与上面不同，是a[$1]。最开始a[$1]没有被赋值，所以初始化为0，判断为false，取反为true，于是就打印。在这个例子中，读取第一行时，a[$1]即为a['apple']；读取第二行时，a[$1]还是a['apple']，所以这里判断依据仅为每一行的第一个field，而testfile文件中第一行和第二行的第一个field都是apple，所以在读取第二行的时候，a[$1]（即为a['apple']）已经被赋值了，并且被加了1，次数a[$1]为1，为true，取反为false，所以不会打印。

​

**awk判断和循环**

* Awk provides a test of whether an item is in an array. <expression> in <array name>
* Evaluates to TRUE if the value of the <expression> is a subscript of the array <array name>.

# 特殊for

for(<variable> in <array name>)

<statement or statement block>

​

# For example:

for(i in surname)

print initials[i] " " surname[i]

# 这里的i是索引，不是surname[i]

​

# 一般for

for(<initial expr> ; <continuation test> ; <update expr>)

<statement or statement block>

​

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

sum += $i

​

**awk Built-in Functions**

* length(<string>) return the length of the string
* substr(s, m, n) substring of s, starting at m, with length n (or till end, if n absent)
* sprintf(fmt, expr, …) Like printf, but returns a string rather than printing. Only way to concatenate strings within Awk
* sub(r, t, s) Substitute t for first regex r in s 把s中符合r的替换为t
* gsub(r, t, s) Substitute t for all regex r in s 全局替换
* system(cmd) 执行Unix命令 awk '{system("ls")}'

导入变量

-v

# report a column specified by command line variable

# e.g., -v COL=3 (default first column)

BEGIN{

if(COL == "")

COL = 1

}

NF >= COL {print $COL}

​

# Example:

% awk -v COL=3 -f my\_cut.awk jab.txt

**User Defined Functions**

function <name>(<parameter list>)

{

<statements>

}

​

function max(a, b)

{

   if(a>=b)

  return(a)

  return(b)

}

{print max($1, $2)}

**awk数值运算**

echo | awk '{print 10/3}' # 计算10/3

​

A=2

B=3

echo "$A $B" | awk '{printf ("%.2f\n",$1\*$2)}' # 计算A\*B

后续的Git和shell function make就不再更新了

Git是很有用的工具，后续有选择CITS5505、CITS5206、CITS5506、CITS4402等课程内容里有需要团队协作project的话，GIT是必修课，最好自己新建一个github仓库练练手。

git的东西还是挺多的，老师讲的太浅了，建议系统的完整的学一下。

有不懂的或者作业懒得自己写的加我vx：GnaH1370

祝大家期末顺利！

2022年5月20日14点55分