统计词频降序输出

grep -oE '[a-z]+' words.txt | sort | uniq -c | sort -r | awk '{print $2" "$1}'

有效电话号码

grep -Eo '^(\([0-9]{3}\)){1}[0-9]{3}-[0-9]{4}$|^([0-9]{3}-){2}[0-9]{4}$' file.txt

文件第10行

sed -n '10p' file.txt

---

^　这个符号在规则表达式中，代表行的 “开头” 位置。

 # 井号 (comments)

这几乎是个满场都有的符号。

#!/bin/bash

井号也常出现在一行的开头，或者位于完整指令之后，这类情况表示符号后面的是注解文字，不会被执行。

This line is comments.

echo “a = $a” # a = 0

由于这个特性，当临时不想执行某行指令时，只需在该行开头加上 # 就行了。这常用在撰写过程中。

#echo “a = $a” # a = 0

如果被用在指令中，或者引号双引号括住的话，或者在倒斜线的后面，那他就变成一般符号，不具上述的特殊功能。

~ 帐户的 home 目录

算是个常见的符号，代表使用者的 home 目录：cd ~；也可以直接在符号后加上某帐户的名称：cd ~user

或者当成是路径的一部份：~/bin；~+ 当前的工作目录，这个符号代表当前的工作目录，她和内建指令 pwd 的作用是相同的。

echo ~+/var/log

~- 上次的工作目录，这个符号代表上次的工作目录。

echo ~-/etc/httpd/logs

; 分号 (Command separator)

在 shell 中，担任”连续指令”功能的符号就是”分号”。譬如以下的例子：cd ~/backup ; mkdir startup ; cp ~/.* startup/.

;; 连续分号 (Terminator)

专用在 case 的选项，担任 Terminator 的角色。

case “$fop” inhelp) echo “Usage: Command -help -version filename” ;;version) echo “version 0.1″ ;;esac

. 逗号 (dot)

在 shell 中，使用者应该都清楚，一个 dot 代表当前目录，两个 dot 代表上层目录。

CDPATH=.:~:/home:/home/web:/var:/usr/local

在上行 CDPATH 的设定中，等号后的 dot 代表的就是当前目录的意思。

如果档案名称以 dot 开头，该档案就属特殊档案，用 ls 指令必须加上 -a 选项才会显示。除此之外，在 regular expression 中，一个 dot 代表匹配一个字元。

‘string' 单引号 (single quote)

被单引号用括住的内容，将被视为单一字串。在引号内的代表变数的$符号，没有作用，也就是说，他被视为一般符号处理，防止任何变量替换。

heyyou=homeecho ‘$heyyou' # We get $heyyou

“string” 双引号 (double quote)

被双引号用括住的内容，将被视为单一字串。它防止通配符扩展，但允许变量扩展。这点与单引数的处理方式不同。

heyyou=homeecho “$heyyou” # We get home

`command` 倒引号 (backticks)

在前面的单双引号，括住的是字串，但如果该字串是一列命令列，会怎样？答案是不会执行。要处理这种情况，我们得用倒单引号来做。

fdv=`date +%F`echo “Today $fdv”

在倒引号内的 date +%F 会被视为指令，执行的结果会带入 fdv 变数中。

, 逗点 (comma)

这个符号常运用在运算当中当做”区隔”用途。如下例

#!/bin/bashlet “t1 = ((a = 5 + 3, b = 7 – 1, c = 15 / 3))”echo “t1 = $t1, a = $a, b = $b”

/ 斜线 (forward slash)

在路径表示时，代表目录。

cd /etc/rc.dcd ../..cd /

通常单一的 / 代表 root 根目录的意思；在四则运算中，代表除法的符号。

let “num1 = ((a = 10 / 2, b = 25 / 5))”

\ 倒斜线 (escape)

在交互模式下的escape 字元，有几个作用；放在指令前，有取消 aliases 的作用；放在特殊符号前，则该特殊符号的作用消失；放在指令的最末端，表示指令连接下一行。

type rm

rm is aliased to `rm -i'

\rm .*.log

上例，我在 rm 指令前加上 escape 字元，作用是暂时取消别名的功能，将 rm 指令还原。

bkdir=/home

echo “Backup dir, \$bkdir = $bkdir”

Backup dir, $bkdir = /home

上例 echo 内的 \$bkdir，escape 将 $ 变数的功能取消了，因此，会输出 $bkdir，而第二个 $bkdir 则会输出变数的内容 /home。

| 管道 (pipeline)

pipeline 是 UNIX 系统，基础且重要的观念。连结上个指令的标准输出，做为下个指令的标准输入。

who | wc -l

善用这个观念，对精简 script 有相当的帮助。

! 惊叹号(negate or reverse)

通常它代表反逻辑的作用，譬如条件侦测中，用 != 来代表”不等于”

if ["$?" != 0]thenecho “Executes error”exit 1fi

在规则表达式中她担任 “反逻辑” 的角色

ls a[!0-9]

上例，代表显示除了a0, a1 …. a9 这几个文件的其他文件。

: 冒号

在 bash 中，这是一个内建指令：”什么事都不干”，但返回状态值 0。

:

echo $? # 回应为 0

: > f.$$

上面这一行，相当于 cat /dev/null > f.$$。不仅写法简短了，而且执行效率也好上许多。

有时，也会出现以下这类的用法

: ${HOSTNAME?} ${USER?} ${MAIL?}

这行的作用是，检查这些环境变数是否已设置，没有设置的将会以标准错误显示错误讯息。像这种检查如果使用类似 test 或 if 这类的做法，基本上也可以处理，但都比不上上例的简洁与效率。

除了上述之外，还有一个地方必须使用冒号

PATH=$PATH:$HOME/fbin:$HOME/fperl:/usr/local/mozilla

在使用者自己的HOME 目录下的 .bash_profile 或任何功能相似的档案中，设定关于”路径”的场合中，我们都使用冒号，来做区隔。

? 问号 (wild card)

在文件名扩展(Filename expansion)上扮演的角色是匹配一个任意的字元，但不包含 null 字元。

ls a?a1

善用她的特点，可以做比较精确的档名匹配。

*** 星号 (wild card)**

相当常用的符号。在文件名扩展(Filename expansion)上，她用来代表任何字元，包含 null 字元。

ls a*a a1 access_log

在运算时，它则代表 “乘法”。

let “fmult=2*3″

除了内建指令 let，还有一个关于运算的指令 expr，星号在这里也担任”乘法”的角色。不过在使用上得小心，他的前面必须加上escape 字元。

**** 次方运算**

两个星号在运算时代表 “次方” 的意思。

let “sus=2**3″echo “sus = $sus” # sus = 8

$ 钱号(dollar sign)

变量替换(Variable Substitution)的代表符号。

vrs=123echo “vrs = $vrs” # vrs = 123

另外，在 Regular Expressions 里被定义为 “行” 的最末端 (end-of-line)。这个常用在 grep、sed、awk 以及 vim(vi) 当中。

${} 变量的正规表达式

bash 对 ${} 定义了不少用法。以下是取自线上说明的表列

${parameter:-word} ${parameter:=word} ${parameter:?word} ${parameter:+word} ${parameter:offset} ${parameter:offset:length} ${!prefix*} ${#parameter} ${parameter#word} ${parameter##word} ${parameter%word} ${parameter%%word} ${parameter/pattern/string} ${parameter//pattern/string}

$*引用script 的执行引用变量，引用参数的算法与一般指令相同，指令本身为0，其后为1，然后依此类推。引用变量的代表方式如下：

$0, $1, $2, $3, $4, $5, $6, $7, $8, $9, ${10}, ${11}…..

个位数的，可直接使用数字，但两位数以上，则必须使用 {} 符号来括住。

$* 则是代表所有引用变量的符号。使用时，得视情况加上双引号。

echo “$*”

还有一个与 $* 具有相同作用的符号，但效用与处理方式略为不同的符号。

$@

$@ 与 $* 具有相同作用的符号，不过她们两者有一个不同点。

符号 $* 将所有的引用变量视为一个整体。但符号 $@ 则仍旧保留每个引用变量的区段观念。

$#

这也是与引用变量相关的符号，她的作用是告诉你，引用变量的总数量是多少。

echo “$#”

$? 状态值 (status variable)

一般来说，UNIX(linux) 系统的进程以执行系统调用exit() 来结束的。这个回传值就是status值。回传给父进程，用来检查子进程的执行状态。

一般指令程序倘若执行成功，其回传值为 0；失败为 1。

tar cvfz dfbackup.tar.gz /home/user > /dev/nullecho “$?”$$

由于进程的ID是唯一的，所以在同一个时间，不可能有重复性的 PID。有时，script 会需要产生临时文件，用来存放必要的资料。而此script 亦有可能在同一时间被使用者们使用。在这种情况下，固定文件名在写法上就显的不可靠。唯有产生动态文件名，才能符合需要。符号$$ 或许可以符合这种需求。它代表当前shell 的 PID。

echo “$HOSTNAME, $USER, $MAIL” > ftmp.$$

使用它来作为文件名的一部份，可以避免在同一时间，产生相同文件名的覆盖现象。

ps: 基本上，系统会回收执行完毕的 PID，然后再次依需要分配使用。所以 script 即使临时文件是使用动态档名的写法，如果 script 执行完毕后仍不加以清除，会产生其他问题。

() 指令群组 (command group)

用括号将一串连续指令括起来，这种用法对 shell 来说，称为指令群组。如下面的例子：(cd ~ ; vcgh=`pwd` ; echo $vcgh)，指令群组有一个特性，shell会以产生 subshell 来执行这组指令。因此，在其中所定义的变数，仅作用于指令群组本身。我们来看个例子

cat ftmp-01#!/bin/basha=fsh(a=incg ; echo -e “/n $a /n”)echo $a# ./ftmp-01incgfsh

除了上述的指令群组，括号也用在 array 变数的定义上；另外也应用在其他可能需要加上escape 字元才能使用的场合，如运算式。

(())

这组符号的作用与 let 指令相似，用在算数运算上，是 bash 的内建功能。所以，在执行效率上会比使用 let 指令要好许多。

#!/bin/bash((a = 10))echo -e “inital value, a = $a/n”((a++))echo “after a++, a = $a”

{} 大括号 (Block of code)

有时候 script 当中会出现，大括号中会夹着一段或几段以”分号”做结尾的指令或变数设定。

cat ftmp-02#!/bin/basha=fsh{a=inbc ; echo -e “/n $a /n”}echo $a# ./ftmp-02inbcinbc

这种用法与上面介绍的指令群组非常相似，但有个不同点，它在当前的 shell 执行，不会产生 subshell。

大括号也被运用在 “函数” 的功能上。广义地说，单纯只使用大括号时，作用就像是个没有指定名称的函数一般。因此，这样写 script 也是相当好的一件事。尤其对输出输入的重导向上，这个做法可精简 script 的复杂度。

此外，大括号还有另一种用法，如下

{xx,yy,zz,…}

这种大括号的组合，常用在字串的组合上，来看个例子

mkdir {userA,userB,userC}-{home,bin,data}

我们得到 userA-home, userA-bin, userA-data, userB-home, userB-bin, userB-data, userC-home, userC-bin, userC-data，这几个目录。这组符号在适用性上相当广泛。能加以善用的话，回报是精简与效率。像下面的例子

chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}

如果不是因为支援这种用法，我们得写几行重复几次呀！

[] 中括号

常出现在流程控制中，扮演括住判断式的作用。if ["$?" != 0]thenecho “Executes error”exit 1fi

这个符号在正则表达式中担任类似 “范围” 或 “集合” 的角色

rm -r 200[1234]

上例，代表删除 2001, 2002, 2003, 2004 等目录的意思。

[[]]

这组符号与先前的 [] 符号，基本上作用相同，但她允许在其中直接使用 || 与 && 逻辑等符号。

#!/bin/bashread akif [[$ak > 5 || $ak < 9]]thenecho $akfi || 逻辑符号 这个会时常看到，代表 or 逻辑的符号。 && 逻辑符号 这个也会常看到，代表 and 逻辑的符号。 & 后台工作 单一个& 符号，且放在完整指令列的最后端，即表示将该指令列放入后台中工作。 tar cvfz data.tar.gz data > /dev/null &

/ 单字边界

这组符号在规则表达式中，被定义为”边界”的意思。譬如，当我们想找寻 the 这个单字时，如果我们用

grep the FileA

你将会发现，像 there 这类的单字，也会被当成是匹配的单字。因为 the 正巧是 there 的一部份。如果我们要必免这种情况，就得加上 “边界” 的符号

grep ‘/' FileA

+ 加号 (plus)

在运算式中，她用来表示 “加法”。

expr 1 + 2 + 3

此外在规则表达式中，用来表示”很多个”的前面字元的意思。

grep '10/+9′ fileB1091009100009100009310100O9#这个符号在使用时，前面必须加上 escape 字元。

- 减号 (dash)

在运算式中，她用来表示 “减法”。

expr 10 – 2

此外也是系统指令的选项符号。

ls -expr 10 – 2

在 GNU 指令中，如果单独使用 – 符号，不加任何该加的文件名称时，代表”标准输入”的意思。这是 GNU 指令的共通选项。譬如下例

tar xpvf -

这里的 – 符号，既代表从标准输入读取资料。

不过，在 cd 指令中则比较特别

cd -

这代表变更工作目录到”上一次”工作目录。

% 除法 (Modulo)

在运算式中，用来表示 “除法”。

expr 10 % 2

此外，也被运用在关于变量的规则表达式当中的下列

${parameter%word}${parameter%%word}

一个 % 表示最短的 word 匹配，两个表示最长的 word 匹配。

= 等号 (Equals)

常在设定变数时看到的符号。

vara=123echo ” vara = $vara”

或者像是 PATH 的设定，甚至应用在运算或判断式等此类用途上。

== 等号 (Equals)

常在条件判断式中看到，代表 “等于” 的意思。

if [$vara == $varb]

…下略

!= 不等于

常在条件判断式中看到，代表 “不等于” 的意思。

```
if [ $vara != $varb ]
```

…下略

^

这个符号在规则表达式中，代表行的“开头”位置。

来源： <http://www.jb51.net/article/51342.htm>