重要资讯
- 本站公告
- 新闻事件
- 本站论坛
- 贡献者

子站索引
- Fedora中文
- Debian中文
- LFS 中文
- Slackware
- Gentoo中文
- Arch 中文

项目索引
- 小企鹅输入法
- LumaQQ
- OpenQ
- RelaxBSD

导航

TCSH shell 编程入门

作者：KornLee
来自：LinuxSir.Org
摘要：TCSH shell不同于其他的shell，因为控制结构更符合程序设计语言的格式。例如TCSH 的test条件的控制结构是表达式，而不是linux命令，得到的值是逻辑值true或false，TCSH的表达式与C语言中的表达式基本相同。

目录

第一篇 **TCSH shell**简介

一、**TCSH shell**变量、脚本和参数
二、脚本输入和脚本输出：**$ <**
三、操作符
四、控制结构
1、条件控制结构；功能
2、循环控制结构: 功能
五、测试表达式**;()**
1、字符串比较: 功能
2、逻辑运算符: 功能
六、**TCSH**内建命令
1、数字变量@
2、alias
3、argv
4、bg
5、break

基础知识
- 安装配置
- 基础入门
- 硬件解决
- 软件管理
- 重要资源

软件应用
- 网络工具
- 图形图像
- 音乐视频
- 字体中文
- 软件其它

网络服务器
- 文件服务器
- Web服务器
- 邮件服务器
- 数据库应用
- 服务器其它

导航

- 我的博客
- 注册/登录

论坛快讯

- 中文文件夹显示的问题
- 网络管理软件
- [help]amule搜索出来的结果是乱码？
- 各位大虾你们的use值是怎么设置的呀。
- 請教etch 下mips的交叉編譯環境！！！！

更多

新进会员

- muzhitu
- bingzheaaa
- xsj0810
- shadis_xu
- zzNucker

在线用户

目前共有 *0* 位用户和 *20* 位游客在线。

6、breaksw
7、builtins
8、cd
9、continue
10、dirs
11、echo
12、eval
13、exec
14、exit
15、fg
16、foreach
17、history
18、hup
19、if-then
20、if-then-else
21、jobs
22、kill
23、logout
24、nice
25、nohup
26、notify
27、onintr
28、popd
29、printenv
30、pushd
31、repeat
32、set
33、setenv
34、shift
35、source
36、stop
37、switch
38、time
39、umask
40、unalias
41、unset
42、unsetenv
43、wait

最新评论

- linux有迅雷吗？
  2 年 48 周 前
- 我下载32位的DVD，迅雷
  2 年 48 周 前
- 幸会幸会
  2 年 49 周 前
- 恭喜恭喜..... 天
  2 年 49 周 前
- 恭喜~
  2 年 50 周 前
- 非常不错的一本书，
  2 年 51 周 前
- 配置后所有字体都没
  2 年 51 周 前
- 看不了啊！
  3 年 6 天 前
- setuid,setgid,stickybit的三个权限的详细说明
  3 年 1 周 前
- 如何记录用户在samba服务器上读取和存放了那些文件？
  3 年 1 周 前

44、where
45、which
46、while

第二篇 TCSH配置

一、TCSH shell 特征

1、echo
2、ignoreeof
3、noclobber
4、noglob
5、附录：常用的TCSH shell特征；

二、TCSH shell 变量

1、prompt prompt2 prompt3
2、cdpath
3、history和savehist
4、mail
5、附常用的TCSH shell配置变量

三、TCSH shell初始化文件

1、 .login
2、 .tcshrc
3、 .logout

关于本文
相关文档

++
正文
++

第一篇 TCSH shell简介

TCSH shell不同于其他的shell，因为控制结构更符合程序设计语言的格式。例如TCSH 的test条件的控制结构是表达式，而不是linux命令，得到的值是逻辑值true或false，TCSH的表达式与C语言中的表达式基本相同。

一、TCSH shell变量、脚本和参数

用户可以在shell中定义变量，为变量赋值以及引用脚本参数。TCSH使用set，@，setenv定义一变量，也可以用相同的方法定义数值变量和数组，用户通过@命令定义的数值变量来进行算术运算，用户使用圆括号()和方括号[]定义和引用数组。脚本也可以用相同的方法操作，但是有个例外，尽管可以用echo命令输出提示符，但没有read命令处理输入，相反，必须重定向到一个变量里。

二、脚本输入和脚本输出：$<

用户能够在脚本范围中定义和使用变量。在下例中，使用文本编辑器把赋值操作和echo之类的linux命令放在一个文件中。然后可以产生可执行文件并像其他命令一样在命令行中执行它，要记住添加可执行权限，必须使用带u+x参数的chmod命令或者带绝对参数700的chmod命令。本中，可以使用echo命令去输出数据，但是，必须通过重定向标准输入把输入读入变量。在TCSH中没有linux read命令的比较版本。记住TCSH全部的脚本文件的第一行的第一个字符必须是“#”字符。

比如：

```
#
#display "hello"
set string="hello"
echo The value of string is $string
```

set命令和重定向符号结合$<将用户输入的任何数据读入标准的输入中，下例中,把用户输入读入string变量中。

```
%set string=$<
abc
%echo $string
abc
```

能够把提示符放置到相同的行用作echo的输入。TCSH使用一个特殊的选项-n，将消除输出字符串中的回车符.光标将保留在输出字符串的结尾处。

```
%echo -n please enter a string

%cat hello
#
echo -n "please enter a string:"
```

```
set string=$<
echo "the value of string is $string"
%chmod u+x hello
%hello
please enter a string:hello
the value of string is hello
%
```

三、操作符

TCSH有一系列标准的赋值，算术和关系运算以及重定向和后台操作等函数。

赋值运算符 功能说明

```
= 赋值操作
+= 先加再赋值
-= 先减再赋值
*= 先乘再赋值
/= 先除再赋值
%= 取余再赋值
++ 自增量1
-- 自减量1
```

算术运算符 说明

```
- 负号
+ 加法
- 减法
* 乘法
/ 除法
% 取余
关系运算符 说明
```

```
> 大于
< 小于
>= 大于等于
<= 小于等于
!= 不等于
== 等于
```

重定向和管道符

TCSH支持标准输入和标准输出的重定向和管道操作.如果设置了noclobber特征,要用重定向操作重写当前文件,需要用符号>!代替>

四、控制结构

同其他shell一样，TCSH也有一系列的控制结构控制脚本的命令的执行。while和if控制结构是最常用的控制结构，switch和foreach是更专用的控制结构。switch是if条件的限定形式，检查数值是否等于一系列可能的数值中的一个数值。foreach是循环结构的限定形式.浏览数值列表,给变量赋新的数值。

TCSH不同的控制结构列表**:**

1、条件控制结构；功能

```
if (expression) then  注：如果expression为真,则执行commands
commands
endif

if (expression) then 注：如果expression为真,则执行command1,否则执行
command1 command2.
else
command2
endif
```

```
switch (string)  加注：允许在几条替换命令中选择,string为不同的模式
case pattern:
commands
breadsw
default:
commands
endsw
```

2、循环控制结构：功能

```
while (expression) 只要expression为真，则重复执行commands，
commands 直到expression为假时跳出循环
end
```

foreach variable (argument-list)迭代循环获得和argument-list中一样多的参数
commands (每次循环variable被设置为列表的下一个参数;end 操作方式同BSH)

TCSH中的控制结构有别于其他的shell,因为它更接近编程语言(C).TCSH的条件表达式的值为true/false.BASH和TCSH主要区别在于TCSH的结构不可重定向或进行管道输出.

五、测试表达式;()

if和while控制结构把表达式用作测试.表达式测试的结果为非零(1)表示真,而零(0)表示假(跟BASH相反).测试表达式可由算术/字符串比较,但是字符串只能作相等和不相等的比较.而且表达式必须要在()括号内.

如:

```
if (expression) then
command
endif
```

TCSH有一系列的运算符来分别对字符串进行测试比较.正则表达式可以包含shell脚本的指令的字符串；

如:

```
if ( $var =~[Hh]* ) then #如果变量$var以字母是大写/小写Hh开头,的字
```

```
符串
echo information #执行命令
endif #结束
```

有很多测试文件的操作与BASH完全相同；如:

```
if ( -r myfile ) then #测试myfile是否可读
echo info
endif
```

测试表达式的基本操作:

1、字符串比较:功能

```
== 是否相等,若相等返回真
!= 是否不等,若不等返回真
=~ 字符串和模式进行测试是否相等(模式为任意正则表达式)
!~ 字符串和模式测试是否不等(模式为任意正则表达式)
文件测试: 功能
-e 测试文件是否存在
-r 测试文件是否可读
-w 测试文件是否可写
-x 测试文件是否可执行
-d 测试文件名是否为目录
-f 测试文件是否为普通文件
-o 测试文件是否被用户所拥有
-z 测试文件是否为空
```

2、逻辑运算符:功能

```
&& 与运算,两个条件同时蛮族
|| 或运算,有一个条件满足
! 取反
```

六、TCSH内建命令

1、数字变量@

在TCSH中，用@命令代替set命令来声明数字变量，然后进行算术，关系和位操作，数字和字符串变量是两个不同的对象，需要用不同的方法管理，不能把set用于数值变量的设置@命令由关键词，变量名，赋值运算符和表达式构成。如:

```
%@ num=10 #注意空格
%@ sum=2 * ($num + 3)
%echo $sum
%26
```

2、alias

格式:alias [name[command]]

关键词

```
alias 别名name 引用的命令command
```

如:

```
alias list ls
alias list 'ls -l'
```

3、argv

当脚本被执行时，命令行中的单词被分析并放入argv数组中。
argv[0]，argv[1]...argv[n]，其中argv[0]保存命令名，argv[1]保存命令的第一个参数，argv[n]命令的第n个参数。argv数组元素可缩写元素号码，前面加$。

如：$argv[1]写为$1；$argv[*]写为$*。#argv参数标识符包含输入在命令行中的参数号码,可检查$#argv变量。如:

```
arglist
#
echo "The number of arguments entered is $#argv"
echo "The list of arguments is : $argv[*]"

%tcsh arglist a b c
The number of arguments entered is 3
The list of arguments is : a b c
```

argv变量列表：

```
命令行参数 说明

$argv[0]或$0 命令名
$argv[n]或$n 从1($1-$)开始的第n个命令行参数
$argv[*]或$* 从1开始的所有命令行参数
$#argv或$# 命令行参数的计数
```

4、bg

格式:bg [%job]

bg命令把指定的任务放入后台。如果此任务已经停止，则继续执行，如果没有参数，将当前任务放入后台。(详细用法与BASH相同此处略)

5、break

格式:

```
break
```

break命令用于退出最接近的foreach/while循环过程，执行同一行中其他的命令。

6、breaksw

格式:breaksw

可以从switch语句处中断，在endsw后继续执行。

7、builtins

列出全部shell的内建命令表.

8、cd

格式:cd [-p][-l][-nl -v][name]

如果给出目录名，此命令把name设置为当前目录，如果命令中没有name，当前目录自动设置成用户主目录。用于name的"-"引用上一级目录，如果参数name没有给出子目录，或不是全路径，或使用./或../引用当前目录及父目录,那么就检查在cdpath shell变量中列出的目录来寻找该目录名。如果此操作失败，将检查shell变量中是否保存着的目录路径名。

用-p选项，linux显示目录列表, 用-l，-n，-v选项与用在dirs命令中的选项完全相同。

默认选项-p ；

9、continue

此命令继续执行最靠近while/foreach语句，当前行其余的命令被执行。

10、dirs

格式:dirs [-l][-n|-v]

```
dirs -S|-L [filename]
dir -c
```

如果不带参数，dirs将显示目录列表，列表开头被列在作侧，第一个目录是当前目录。带-l选项，用户主目录中的全部子目录被展开。输入项在到达屏幕边缘时，-n选项隐藏输入项，-v选项显示每一行的输入项入口，-c选项将清除目录列表，-S选项可以把目录列表作为一系列的cd和pushed命令保存在文件中，-L选项可以从指定的文件中将cd和pushed命令读入，该文件包含-S选项所存储的cd和pushed命令。如果没有指定的文件名，将使用赋值到dirsfile shell变量中的文件名。如果没有设置dirsfile，将使用~/.cshdirs，在启动时注册shell将对dirs -L求值，如果设置了savedirs，退出前使用dirs -S,由于在~/.cshdirs之前,仅仅~/.tcshrc是正常来

源,dirsfile应该以~/.tcshrc设置而不是以~/.login设置.

11、echo

格式:

```
echo [-n] word/string
```

此命令把每个单词或字符串写入shell的标准输出.可设置echostyle shell变量来仿真选项以及BSD的换码序列或者echo的System V版本；

12、eval

格式:

```
eval argument...
```

此命令把参数作为shell的输入，执行当前shell的上下文中的结果命令，由于分析发声在替换前，所以该命令通常用于执行命令或变量替代所产生的命令。

13、exec

格式:exec command

此命令代替当前的shell执行指定的命令,并退出终端；

14、exit

格式: exit [expression]

shell可以带指定的表达式的值退出，如果没有包含表达式，也可带状态变量值退出。

15、fg

格式:fg [%job...]

把指定的任务带到前台,如果任务在终止状态，在带到前台的同时使他运行；job参数是使用任务号或者下列字符串之一的任务引用:' % + -

16、foreach

格式:

```
foreach variable(list of values)
commands
end
```

foreach结构被设置为顺序引用数值列表,它类似BASH shell的for in 结构.foreach结构产生两个操作数:一个变量和一组包含在()中的数值列表.将列表中的值赋值到结构中的变量中.循环体的结尾由语句end构成.下例中脚本list输出由项目和foreach循环读取每一项当前的日期构成的一行信息,列表中的每一项被连续赋值到变量object list

```
#
set tdate=`date +%D`
foreach object(milk cookies apples cheese)
echo $object $tdate
end
%tcsh list
milk 04/26/03
cookies 04/26/03
apples 04/26/03
cheese 04/26/03
```

使用foreach循环有助于管理文件,可以模式中的shell特定字符来产生用作数值列表的文件名列表,然后所产生的文件名列表就成为foreach结构引用的列表.如果在列表中不带任何变量,可以使用命令行参数实现循环.

```
list
#set tdate=`date +D%`
foreach object($argv[*])
echo "$object $tdate"
end
%tcsh list a b c
a 04/26/03
b 04/26/03
c 04/26/03
```

使用argv[*]特殊参数变量，可以直接引用命令行参数。下例中，当调用shell脚本cbackuparg时，在命令行下输入C程序文件的列表，在foreach循环中，argv[*]引用命令行中的所有参数，将按顺序把各个参数赋值给变量backfile，变量argnum用于引用每个参数。显示参数和backfile的值来说明它们两个是相同的。

```
cbackuparg
#
@ argnum=1
foreach backfile ($argv[*])
    cp $backfile sourcebak/$backfile
    echo "$backfile $argv[$argnum]"
    @ argnum=$argnum+1
end
```

```
%cbackuparg main.c lib.c io.c
main.c main.c
lib.c lib.c
io.c io.c
```

17、history

格式:

```
history [-hr][n]
history -S|-L|M [filename]
history -c
```

如果history没有任何参数，将显示开行号的历史命令列表。用作参数的号码将列出行的最后号码，如果没有任何选项参数。她将用作历史文件名；否则，将使用hisfile变量值。

用-h选项,将以注释的形式显示不带行号的历史文件列表
用-r选项,将反向显示,以最近的列表开始
用-c选项,则清除历史列表

如果用-S选项，可以把历史列表存入文件，如果savehist shell变量的第一个单词被设置为数值，历史文件将被保存为最大的行数，如果第二单词为'merge'，历史列表将被合并到当前的历史文件中，而不替代历史列表，时间图章排序历史列表。

如果用-L选项，shell将从存储的历史列表文件中读取历史列表并把她追加到当前历史列表中。
如果用-M选项，将从历史文件读取历史列表信息,但用当前历史列表信息合并,排序历史列表。

如果没有用这些选项给出文件名，那么就使用赋值到histfile shell变量的文件名，如果没有设置histfile，就使用~/.history。

18、hup

格式:

hup [command]
如果带有括起来的command参数，hup运行command，根据停机信号退出。当shell退出时，hup安排shell发出停机信号。注意：命令也许设置自己的停机响应，重写hup，如果没有参数(只有在shell脚本中才允许)，hup使shell根据脚本的余项的停机信号退出。

19、if-then

格式:

```
if (expression) then
command
endif
```

if-then结构把条件放入几个linux命令，该条件是expression，如果expression得出非零的数值，那么expression为真，执行if结构内的命令，如果expression得出零值那么expression为假,就不执行if结构内的命令。

if-then结构以if关键词开始，expression表达式用()括起来，关键词then后可以跟任意数量的linux命令，以关键词endif结束if命令。注意：在TCSH中，if (expressiong)和then必须在同一行!

举例:

```
ifls
#
echo -n "Please enter option:"
set option=$<
```

```
if ($option=="s") then
    echo List files by size
    ls -s
endif
```

```
%tcsh ifls
Please enter option: S
List files by size
total 2
1 monday 2 today
%
```

20、if-then-else

格式:

```
if (expressiong) then
command
else
command
endif
```

用户需要经常需要根据expression是真还是假来进行选择；关键词else允许if结构在两者之间选择；如果expression为真，那么执行第一个command，否则执行第二个就是else后面的command。

举例：

```
elsels
#
echo Enter s to list file sizes
echo otherwise all file information is listed
echo -n "please enter option:"
set option=$<
if ($option=="s") then
    ls -s
        else
```

```
    ls -l
endif
echo Good-bey
```

(结果略)

21、jobs

格式:

```
jobs [-l]
```

此命令列出所有活动的任务,如果带-l选项,将列出进程号,以及正常的信息.

22、kill

格式:

```
kill [-signal] %job\pid...
kill -l
```

此命令用于终止进程或者任务，如果带-signal选项，应该指定要发送的信号，默认发送的信号是SIGTERM，如果带有-l选项，则列出信号名，对于带-l选项提供的信号数，将列出与其相关的信号名。

23、logout

此命令用于终止注册shell，如果设置了ignoreeof，它将非常有用；

24、nice

格式:

```
nice [+ number][command]
```

此命令设置shell调整优先为number,如果没有设置number,调整优先设置为4,带有指定的command,nice适当的优先运行command,number值越大,进程获得的CPU的时间就越少.

25、nohup

格式:

```
nohup [command]
```

如果不带command参数,nohup指示shell忽略任何停机信号,如果带command参数,将执行此命令并忽略执行中的任何停机信号.

26、notify

格式:

```
notify [%job...]
```

Linux执行命令后,如果有后台任务在执行,系统将通知到目前为止已经完成的后台任务,该系统不会中断如编辑这样的操作来通知用户关于完成的任务,当某任务完成时,如果想马上知道,无论系统在作什么,可以使用notify命令指令系统通知用户,它的参数作为任务号,当任务完成时,系统将中断当前命令并通知用户任务已经完成,如:

```
%notify %2
```

当2号任务完成时,告诉系统通知用户.

27、onintr

格式：

```
onintr [-|label]
```

此命令控制shell在中断时的动作,不带任何参数,将回复shell中断默认的动作,将终止shell脚本或返回输入级的终止符,如果带-参数,则忽略全部的中断,当接受中断或当子进程终止时,带label的命令将使shell执行goto标号.

28、popd

格式：

```
popd [-p][-l][-n|-v][+n]
```

此命令从目录列表中删除一个目录,不带参数的命令从列表中删除顶层目录,+n删除从左起的第n层的目录.然后popd显示最终的目录列表.pushdsilent shell变量可以设置为支持此特性,-p选项可以重写pushdsilent.

29、printenv

格式:

```
printenv [name]
```

此命令显示环境变量的名和值,如果带name参数,仅仅显示环境变量name的值.

30.pushd

格式:

```
pushd [-p][-l][-n|-v][name|+n]
```

此命令把目录添加到存储的目录列表中,然后显示目录列表,如不带参数,除非目录为空,pushd交换顶层的两个目录,并返回0.

+n旋转列表以便使第n个目录(从左起)列与顶端,但是,如果设置了dextract,pushd +n将展开第n个目录,把它压入堆栈顶层.
-n旋转堆栈以便第n个目录(从右起)列于顶端,dir把dir添加到顶端的目录列表,是她成为新的当前目录.如果设置了pushtohome,不带参数的pushd命令执行pushd ~的功能,像cd的功能一样,如果dunique被设置,pushd在压入堆栈之前从堆栈中删除任何的name历程,可以设置pushdsilent shell变量来取消目录列表的显示,然后可以使用-p选项重写pushdsilent.

31、repeat

格式:

```
repeat count command
```

此命令重复执行command指定的次数count.

32、set

格式:

```
set
set name ...
set name = value ...
set name = (wordlist)...
set name[index] = word ...
set -r
set -r name ...
set -r name = value ...
set -r name = (wordlist) ...
```

不带任何参数的set命令将显示全部的shell变量值,包含许多单词的变量作为放在括号中的词列出,带name参数的set命令定义一个变量并为它赋值null串,带有name和=符号分隔的值,set定义变量并给它赋值,要把wordlist作为值赋值给name变量,把列表中的词作为要赋的值放在括号中,为了将数值赋值给数组元素,使用方括号[]指定元素的index,但是该元素必须已经存在.

-r选项常用于引用只读变量,仅仅带-r选项的set命令将列出只读变量,与变量名一起使用,set将使此变量设置为只读,与赋值的变量一起使用,将初始化该变量,并使该变量成为不能被修改的只读变量.

在TCSH中,用户必须在使用变量之前首先声明它,使用加变量名的set命令声明变量.变量名可以是任何字母字符包括下划线_,也可以含有数字,但是变量名不能以数字为首字符!

33、setenv

格式:

```
setenv [name[value]]
```

setenv常用于定义有特定值的环境变量.如没有带value选项,setenv设置name变量为null串,如果没有带任何参数,将显示全部环境变量的名称和值.

TCSH有两种类型的变量:局部变量和环境变量.局域变量是在shell内部声明的;环境变量是全局域的变量.使用setenv可以定义环境变量,使用setenv命令,变量名,以及被赋值的值,就可给环境变量赋值.其中,没有赋值运算符,如:

```
%setenv greeting hello
```

greeting环境变量被赋值为hello，无论何时调用shell脚本,都将产生自己的shell,如果shell脚本被另外一个shell脚本
执行,它将使自己的shell从第一脚本的shell中分离.现在有两个shell,属于第一个脚本的父shell和当执行第二个脚本是产生的子shell.

在其他的shell内部执行脚本时,这个shell是第一个脚本的子shell,原来的脚本的shell是它的父shell.每个shell都拥有自己的变量,子shell不能引用父shell中的局域变量,但是能引用环境变量.子shell可以引用父shell中声明的任何环境变量.

34、shift

格式:

```
shift [variable]
```

没有参数的shift命令向左移动argv数值,即argv[1]的数值被argv[2]所代替,如果argv没有被设置或者数值少于一个词时将发声错误.带有其数值为列表或数组的variable,该命令将列表和数组的数值向左移动.

35、source

格式:

```
source [-h] name [argument...]
```

source读出并执行以name命名的命令,通常是shell脚本(不能把这些命令放在历史列表中).把任何参数输入argv中.用-h选项,命令被放入历史列表但是不立即执行.

36、stop

格式:

```
stop %job\ job...
```

stop命令停止指定的任务或在后台执行的进程.也可以用一个数字或字符串引用一个任务,如果不存在默认的任务,那么仅仅stop不能停止当前的任务.

37、switch

格式：

```
switch (test-string)
    case pattern:
         commands
breaksw
    case pattern:
         commands
         breaksw
    default:
         commands
         breaksw
endsw
```

switch结构在几个可能的选项中进行选择.此结构与BASH中的case结构非常类似通过把字符串与几个可能的模式进行选择,每个可能的模式都与一系列命令相关联,如果没有相匹配的项,则执行相关的命令。

switch结构以关键词switch和放在圆括号中的test-string开始,字符串常由变量求值导出,然后是一系列的模式,每个模式pattern前是关键词case,以冒号:结束,冒号后列出与选择有关的commands,以breaksw终止command,在所有被列出的模式之后,关键词endsw结束switch结构。

注意：每个测试表达式的命令组的最后一个命令是breaksw，每个测试表达式被逐一测试，直到找到相匹配的内容，如果没有找到相匹配的项，则执行默认项default。用关键词default表示默认选择，default项是可选的。但是它有助于通知用户没有匹配的测试字符串。

举例:

```
lschoice
#
echo s.List sizes
echo l.List files infomation
echo c.List C files
echo -n "Please enter choice:"
set choice=$<
switch ($choice)
  case s:
```

```
    ls -s
    breaksw

  case l:
    ls -l
    breaksw
  case c:
    ls *.c
    breaksw
  default:
    echo Invalid Option
    breaksw
endsw
```

(输出略)

38、time

格式:

time [command]

如果没有参数,此命令显示当前shell的时间累计,用一条命令作为参数,它执行命令并显示时间累计.

39、umask

格式:

```
umask [value]
```

此命令指定用户文件建立掩码,掩码用八进制表示.

40、unalias

格式:

unalias pattern

此命令删除其名称与pattern匹配的所有别名.

41、unset

格式:unset pattern
此命令取消定义shell变量,名称可能是匹配了几个变量的pattern(尽量避免用unset *,因为它将取消所有的变量)

42、unsetenv

格式:

```
unsetenv pattern
```

此命令取消定义环境变量,名称可能是匹配几个变量的pattern,也尽量避免用unsetenv *,因为它将删除所有的环境变量.

43、wait

此命令指示shell等待全部的后台任务的完成,在一个交互的shell环境中,可中断一个wait操作,并显示剩余的任务.

44、where

格式:where command

此命令报告所有已知的命令历程,包括别名,内建和可执行的路径.

45、which

格式:

```
which command
```

此命令显示shell要执行的指定的任务.

46、while

格式:

```
while(expression)
command
```

```
end
```

此循环重复执行命令，以关键词while开始，接着是圆括号括起来的表达式expression以end结束循环。

while循环很容易与switch结构结合，构成菜单的驱动程序；

```
lschoicesw
#
set again=yes
while ($again==yes)
echo "1.List sizes"
echo "2.List files information"
echo "3.List C files"
echo "4.Quit"
echo -n "Please enter choice:"
set choice=$<
switch ($choice)
  case 1:
    ls -s
    breaksw
  case 2:
    ls -l
    breaksw
  case 3:
    ls *.c
    breaksw
  case 4:
    set again=no

    echo good-bey
    breaksw
  default:
    echo Invalid Option
endsw
```

(输出略)

第二篇 TCSH配置

TCSHshell可以使用户使用shell配置变量和特征配置自己的shell.也可以使用set命令设置特征.TCSH也有注册,注消以及何时进入TCSHshell的配置文件.

一、TCSH shell 特征

TCSH有几个特征,允许控制不同的shell操作的方法.TCSH shell特征不仅包括许多它自己的特征,而且还包括KSH/BASH shell中的特征,如,TCSH中有一个noclobber,防止重定向重写文件,一些更常用的特征为:echo,noclobber,ignoreeof和noglob.通过定义和取消定义与该特性相关的变量,打开/关闭TCSH shell特征,为每个特征命名变量.

例如,通过定义noclobber变量,打开noclobber特征.使用set命令定义变量,而使用unset命令取消定义变量.为了运行noclobber特征,使用set noclobber,为了关闭它,使用unset noclobber.如:

```
set variable 定义变量
unset variable 取消变量
```

由于这些变量常用于打开/关闭特征,所以有时也被叫做双态切换.

1、echo

在echo被执行之前,设置echo启动显示命令的特征；

```
set echo 打开echo特征
unset echo 关闭echo特征
```

2、ignoreeof

设置ignoreeof启动防止用户使用CTRL+D注消用户shell的特征,它用于防止突发性的注消操作,如果此特征关闭,可以用CTRL+D注消过程.由于在标准输入中CTRL+D是结束用户输入的操作,所以很容易造成突发注消shell进程.因此,ignoreeof特征就能防止这种突发的注消,当此特征被设置后,用户必须用logout命令明确注消.

3、noclobber

设置noclobber启动预防现有文件不被重定向输出的特征,借助此特征,如果把输出重定向到已经存在的文件中,就不用标准的输出重写文件.原文件被保存.也许会发生这种情况:把现有文件名用作保存被重定向的输入的文件名.因此,noclobber特征使用户不会意外的重写原文件.

```
%set noclobber
%cat oldfile > newfile
newfile: file exist
```

有时,用户想用重定向输出重写某个文件,那么就可在重定向符号操作符之后放置惊叹号!实现重写操作.这将重写noclobber特性,用标准的输出代替文件内容.

```
%cat oldfile >!newfile
```

4、noglob

设置noglob使某特性有效,该特征禁止用户shell中的某些字符.字符 * ? [ ] ~将不再扩展为匹配的文件名.如:

```
%set noglob
%ls myfile?*
myfile?*
```

5、附录：常用的TCSH shell特征；

```
特征 功能
set 打开shell特征
unset 关闭shell特征
echo 执行命令前显示该命令
ignoreeof 禁止用CTRL+D注消
noclobber 不必通过重新定向重写文件
noglob 禁止特殊字符用于文件名扩展:* ? ~ [ ]
notify 当完成后台任务时立即通知用户
verbose 历史命令引用后显示命令
```

二、TCSH shell 变量

正如在BASH shell中一样,也可在TCSH shell中使用特殊shell变量来匹配自己的系统.一些变量在系统初始化中定义,也可以在shell中用新的值重新定义变量,有些必须在系统初始化时定义,如最常用的特殊变量prompt变量,它允许建立自己的命令行提示符另外一个是history变量,该变量确定保存了多少历史命令/事件.

在TCSH shell中,许多特殊变量的名称与函数都与BASH/ZSH shell中的变量名称和函数类似.一些必须用大写表示,但是大多数用小写表示.某些特殊变量起类似的作用,但是具有完全不同的实现方法,如mail变量保存的信息与BASH shell的MAIL,MAILPATH,MAILCHECK变量保存的信息完全相同.

1、prompt prompt2 prompt3

这三个变量为命令行保存提示符.用户可以设置自己想要的符号/字符串作为提示符.为了设置命令行提示符,可以用

```
%set prompt="+"
+
```

这样提示符就变成了+号.也可以使用预定义的一组代码使得配置提示符更容易,在每个代码前加个%号,例如,用%/表示当前的工作目录,%t表示时间,%n表示用户名,%!表示下一个历史事件的号码,如:

```
%set prompt="%/"
/home/dirname
```

代码列表:
代码 说明

```
%/ 当前运行目录
%h,%!,! 当前历史号码
%t 当天时间
%n 用户名称
%d 日期
```

```
%w 当前月份
%y 当前年份
```

当命令分为若干行输入时,变量prompt2被用于特殊情况,为需要输入改命令而增加的行显示prompt2,prompt3提示符用于检查拼写特征是否被激活.

2、cdpath

cdpath变量保持目录的路径名,用cd搜索特定的子目录,这些路径名形成一个数组,就像赋值给TCSH shell的path变量的路径名数组一样,注意在路径名之间要求有空格.

```
%set cdpath=(/usr/bin /var/bin /home/username)
```

3、history和savehist

正如前面了解的那样,history变量常用于确定要保存的历史事件的数量,只需给该变量赋值history应该记录的事件的最大的数值.当达到最大值时,计数器从1开始.但是当注消时,savehist变量保存存储在.history文件中事件的数量,当再次注册时,这些事件就成为初始化历史事件.如:

```
%set history=20
%set savehist=5
```

当注册时,历史列表中将记录20个事件,但是当注消后,仅仅最后5个事件被保存在.history文件中,下一次注册时,历史列表将由以前的最后5个命令组成.

4、mail

在TCSH shell中,mail变量综合了BASH和ZSH shell中的MAIL,MAILCHECK,MAILPATH变量的特征.TCSH shell mail变量值是一个数组,该数组的元素既包括检查电子邮件的时间间隔,又包括检查电子邮箱文件的路径.要给这些元素赋值,可以把赋值的数组赋值给mail变量,用圆括号括起来,空格分割的一系列指定数组的新值.第一个值是再次检查电子邮件前设置等待的秒数,它类似BASH shell的MAILCHECK变量保存的数组.其余的值由电子邮箱文件的路径构成,检查该文件获取邮件.注意,这些变量保存了BASH和ZSH shell MAIL,MAILPATH变量的功能. 如:

```
%set mail ( 1200 /usr/mail/user)
```

mail变量被设置成每1200秒检查一次电子邮件,被检查的电子邮箱文件在/usr/mail/user目录中.

```
%set mail (1200 /usr/mail/user /home/userdir)
```

也可以容易的给mail数组添加更多的邮箱文件路径,注意每个元素之间的空格.

5、附常用的TCSH shell配置变量:

```
变量 说明
home 用户主目录的路径名
user 注册名
pwd 当前运行目录的路径名
shell 用于注册过程的程序路径名
prompt 主提示符
path 目录路径名列表,搜寻目录获取可执行命令
mail mail实用程序检查该变量来获取接受信息的邮件文件名
cdpath cd命令搜索该变量获取子目录的目录路径名
history 历史列表中命令数量
savehist 为下一个注册存储历史列表中的命令数量
EXINIT EX/vi编辑器的初始化命令
TERM 终端名
```

三、TCSH shell初始化文件

TCSH shell有3个初始化文件: .login .tcshrc .logout,.login文件是每次注册时执行的注册初始化文件, .tcshrc文件是每次进入TCSH shell时执行的shell初始化文件, 该文件在注册时执行或者用tcsh命令直接从另一个shell进入TCSH shell时执行,每次注消时执行.logout文件

1、 .login

TCSH shell有自己注册初始化login,该文件用于配置用户shell的shell命令和特殊变量定义, .login文件相当于BASH和ZSH shell中使用的.profile文件。
.login文件包含setenv命令,给TERM之类的特殊环境变量赋值,可以用任何标准的编辑

器编辑它,改变这些值,也可添加新的值.但是要记住:在TCSH shell中,为环境变量赋值的命令是setenv,如:

```
%setenv EXINIT 'set nu ai'
```

定义EXINIT变量并设置vi编辑器行计数和自动缩排的功能.

当编辑.login文件时要十分小心,不经意的编辑可能造成错误的变量设置或者根本没有设置变量,因此,编辑之前最好备份.login文件.如果更改了.login文件,想让这些变化在当前注册期间生效,需要使用source命令重新执行该文件,source命令实际上可执行任何初始化文件,其中包括.tcshrc文件和.logout文件,如:

```
%source .login
```

2、 .tcshrc

每当进入TCSH shell或产生任何子shell时,就执行.tcshrc初始化文件,如果TCSH shell是用户注册的shell,那么当用户注册时, .tcshrc文件和.login文件一起运行,如果从另外一个shell进入TCSH shell, .tcshrc文件被自动执行,文件中包含的变量和别名被定义.

实际上,每当产生一个shell时,如运行shell脚本,就执行.tcshrc文件,也就是说,每当创建一个shell时,就执行.tcshrc文件,这允许在.tcshrc中定义局部变量,在某种意义上说,局部变量可以进入任何子shell,即时像history这样的用户定义的变量也是局部变量,但要为每个产生的子shell定义特殊变量,这样,为每个子shell变量设置history,但是,每个子shell都有自己的局部history变量,用户甚至可在不影响其他子shell的变量的情况下改变一个子shell的局部history变量,在shell初始化文件中定义特殊变量可以看做是BASH shell导出变量,在BASH shell和ZSH shell中的导出变量只把自己的拷贝传送给子shell,改变拷贝不会影响原始定义.

.tchsrc文件也可包含别名变量和用于打开shell特征的特征变量,别名变量和特征变量是局部变量,限定于shell内部.但是.tcshrc文件在每个shell中都将定义这些变量.因此,.tcshrc文件通常保存各种命令定义的别名,如:

.tcshrc

```
#
set shell=/usr/bin/csh
set path=$PATH (/bin /usr/bin .)
```

```
set cdpath=(/home/dirname/filename /home/dirname/files)
set prompt="!$pwd>"
set history=20
set ignoreeof
set noclobber
alias rm 'rm -i'
alias mv 'mv -i'
alias cp 'cp -i'
```

与环境变量不同,局部变量用set命令定义,在.tcshrc文件中的局部变量应该使用set命令定义,用setenv命令定义的环境变量是放在.login文件中的,请记住:当改变path或cdpath的路径名时,数组中包含这些路径名,数组中的每个元素要用空格分开,如果添加新的路径名时,必须保证用空格把它和其他的路径名分开.如果已经改变了.tcshrc文件,希望这些变化在当前注册中立即生效,别忘了用source命令重复执行.tcshrc文件。

3、 .logout

.logout文件也是初始化文件,只不过它在注消时执行,用户可以设置.logout文件以使其完成任何在注消时想要完成的操作,文件中没有定义变量,常包括关闭进程的shell命令.可以向.logout文件中添加自己的shell命令,如:

.logout

```
#
clear
echo "Good-bey..."
```

关于本文

“现在关于BASH的资料,在网上很多,我一直没有找到关于TCSH编程的入门资料,因此,特从我的书中摘抄下来,终于打完啦~~,请喜欢tcsh编程的兄弟们多提意见,有想了解ZSH shell的兄弟也别急,我正在找ZSH方面的书,不久就可以整理出来啦~~~”——KornLee

本文是Lee兄整理的文档，原文参见《【TCSH shell编程入门】》；北南重新整理，把说明性的标点，由英文标点改为中文标点。并对全文进行了序号化，制作出HTML格式，方便大家查看和索引；

感谢Lee兄所做的巨大贡献~~

—— 北南南北

相关文档

由 北南南北 在 2005/12/20 - 16:18 发表 | 命令/SHELL/PERL | 要发表评论，请先登录 或 注册