《TCSH shell 编程》入门简介

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,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.在脚本中,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.

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:

条件控制结构; 功能
if (expression) then 如果 expression 为真,则执行 commands
commands
endif

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

switch (string) 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

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:

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

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.

源码:

```
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:

源码:

```
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:

源码:

```
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;
源码:

```
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:
源码:

```
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.举例:

源码:

```
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)

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?*

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.

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..."
```

注:
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~~~

（完）