csh语法速查

参考：

https://www.jb51.net/article/57770.htm

https://www.computerhope.com/unix/ucsh.htm

https://docslib.org/doc/12944229/c-shell-cookbook

https://starlink.eao.hawaii.edu/devdocs/sc4.pdf --- 推荐看这本书

cshell脚本没有函数的概念，就可以把它当做是批处理脚本。

它的好处是在命令行输入的命令都可以直接放到脚本里面去执行（不用转换），这个对于批处理来说确实挺有好处的。

1. csh脚本执行方式

```
#文件名执行方式
./myscript     #相对路径执行
/home/xxx/myscript    #绝对路径执行

# source xxx脚本执行方式
source ./myscript     #用source的方式执行,

#用source xxx脚本的好处是：在shell环境中定义的alias语句，可以直接在脚本文件中使用；在脚本文件里面定义的alias同样在shell环境中也可以使用。
#其实是当前shell环境下的所有东西都可以直接在脚本文件里面使用，包含变量这些。

#直接文件名执行方式，除了在.cshrc文件里面全局设置的东西之外，当前shell环境下的所有东西都不能在脚本文件中使用。
# --好处是不依赖于当前shell环境下人为设置的变量或alias可能不太容易出错。

#csh脚本执行一旦出错，后续命令将不再执行，直接退出当前脚本执行。

#不管是用那种方式执行脚本，都是建议在脚本文件的第一行加上  #!/bin/tcsh
```

2. csh脚本举例

2.1 例子1

假设有脚本文件a.csh内容如下：

```
#!/bin/csh
```

```
set a = ${#argv}

echo "argv number = $a"

echo $0 $1 $2 $3; #显示脚本名和3个输入参数
echo "argv: $argv[1] $argv[2] $argv[3]"; # $argv[1]和$1的效果是一样的

echo ${0}    #显示脚本名
```

```
执行:
./a.csh 1 2 3
运行结果：
argv number = 3
./a.csh 1 2 3
argv: 1 2 3
./a.csh
```

2.2 例子2

```
#!/bin/csh

set files = ~/*; #用户home目录下的所有文件和目录
foreach tmp ($files)
    if (-f $tmp) then
        set bname = `basename $tmp`
        echo "normal file: $tmp, basename = $bname"
    else if(-d $tmp) then
        echo "dir: $tmp, basename = $bname"
    endif
    sleep 1; #等待1s
    sleep 1;
end
```

3. 变量

```
set x = 5
echo $x;  #输出5
echo ${x}; #输出5
echo ${x}kg; #输出变量x再加kg, 即输出5kg
echo $%x; #$%x表示变量的值的长度

$#name name数组个数
${#argv} argv参数个数
$#argv,   argv参数个数
$argv[1],  argv参数1, 是第一个输入arg
$argv[0] argv参数0, 是指脚本的名字
```

```
set i = 0
@ i = $i + 1 #使用算术命令
@ i = $i - 1 #使用算术命令
@ i = $i * 2 #使用算术命令
@ i = $i / 2 #使用算术命令
#类似c++的自加减乘除操作
@ i++ #自加操作
@ i-- #自减操作
@ i+=2 #自加2
@ i-=2 #自减2
@ i*=2 #自乘2
echo $i
```

3.1 系统变量

- argv 脚本变量参数。 $0表示shell名，$1会被替换为$argv[1]
- cwd, 当前工作目录
- path path变量
- home 等同于~,用户home目录
- prompt 命令行提示符变量

3.2 cshell prompt

```
prompt          The string which is printed before reading each command from
the terminal. prompt may
include any of the following formatting sequences (+), which are replaced by
the given information:

%/              The current working directory.
%?              The current working directory, but with one’s home directory
represented by ‘?’ and other users’ home directories represented by ‘?user’
as per Filename substitution. ‘?user’ substitution happens only if the shell
has already used ‘?user’ inapathname in the current session.

%c[[0]n], %.[[0]n]     The trailing component of the current working
directory, or n trailing components if a digit n is given. If n begins with
‘0’, the number of skipped components precede the trailing component(s) in
the format ‘/<skipped>trailing’. If the ellipsis shell variable is set,
skipped components are represented by an ellipsis so the whole becomes
‘...trailing’. ‘?’ substitution is done as in ‘%?’ above, but the ‘?’
component is ignored when counting trailing components.

%C Like %c,     but without ‘?’ substitution.
%h, %!, !       The current history event number.
%M              The full hostname.
%m              The hostname up to the first ‘.’.
%S (%s)         Start (stop) standout mode.
%B (%b)         Start (stop) boldfacing mode.
%U (%u)         Start (stop) underline mode.
```

```
%t, %@          The time of day in 12-hour AM/PM format.
%T      Like ‘%t’, but in 24-hour format (but see the ampm shell variable).
%p      The ‘precise’ time of day in 12-hour AM/PM format, with seconds.
%P      Like ‘%p’, but in 24-hour format (but see the ampm shell variable).
\c      c is parsed as in bindkey.
?c      c is parsed as in bindkey.
%%      A single ‘%’.
%n      The user name.
%d      The weekday in ‘Day’ format.
%D      The day in ‘dd’ format.
%w      The month in ‘Mon’ format.
%W      The month in ‘mm’ format.
%y      The year in ‘yy’ format.
%Y      The year in ‘yyyy’ format.
%l      The shell’s tty.
%L      Clears from the end of the prompt to end of the display or the end of
the line.
%$      Expands the shell or environment variable name immediately after the
‘$’.
%#      ‘>’ (or the first character of the promptchars shell variable) for
normal users, ‘#’ (or the second character of promptchars) for the
superuser.
%{string%}
Includes string as a literal escape sequence. It should be used only to
change terminal
attributes and should not move the cursor location. This cannot be the last
sequence in
prompt.
%?      The return code of the command executed just before the prompt.
%R      In prompt2, the status of the parser. In prompt3, the corrected
string. In history, the history string.  ‘%B’, ‘%S’, ‘%U’ and ‘%{string%}’
are available in only eight-bit-clean shells; see the version shell
variable.


The bold, standout and underline sequences are often used to distinguish a
superuser shell. For
example,
> set prompt = "%m [%h] %B[%@]%b [%/] you rang? "
tut [37] [2:54pm] [/usr/accts/sys] you rang? _

If ‘%t’, ‘%@’, ‘%T’, ‘%p’, or ‘%P’ is used, and noding is not set, then
print ‘DING!’ on the change of hour (i.e, ‘:00’ minutes) instead of the
actual time.

Set by default to ‘%# ’ in interactive shells.
```

3.3 通过perl处理变量

```
set a = "123"
```

```
set a = `echo $a | perl -pe "s#123#456#"`

echo $a
#现在变量a的值变为456，通过perl脚本被修改了，有时候这个方法还是比较有用。


set cl = "123"
set cl_type = `echo $cl | perl -e '$_ = <>; if (/^$/) { print "none"} elsif (/^\d\+$/) { print "num"} else { print "else" }'`
echo $cl_type
# cl_type当前为num
```

4. 数组

```
#初始化，空数组
set arr = ()

#初始化，带有3个元素的数组
set myarr = (str1, str2,str3)
echo $myarr[2]
echo $myarr
echo $myarr[*]

set files =  *
```

追加元素

```
set myarr = ($myarr str4)
```

shift操作

```
# array test
set ar = (0 1 2 3 4)
echo $#ar
echo ${#ar}
echo $ar[1]   # index [1] 才是第开头的数据
echo $ar[1-3]
echo $ar[*]
shift ar
echo $ar[*]
```

执行结果：

```
5
5
0
0 1 2
0 1 2 3 4
1 2 3 4
```

5. if/else/switch/case

```
if ($n < 5 || 20 <= $n) then
endif

if ($n < 5 && 20 <= $n) then
endif

if(expression)then
  commands
endif

if {(command)} then
  commands
endif


if(expression) then
  commands
else if(expression) then
  commands
else
  commands
endif
```

6. 判断上一条命令退出值

```
echo "haha"
if($? != 0) then
  echo "run failed 1"
  exit 1
endif

rm "jjjjjjjjjjjjjjjjj"
if($? != 0) then
  echo "run failed 2"
  exit 1
endif
```

7. switch/case

```
switch("$value")
case pattern1:
  commands
  breaksw
case pattern2:
```

```
  commands
  breaksw
default:
  commands
  breaksw
endsw
```

8. while/foreach

```
while(expression)
  #类似c++的自加减乘除操作
  @ i++ #自加操作
  @ i-- #自减操作
  @ i+=2 #自加2
  @ i-=2 #自减2
  @ i*=2 #自乘2

  commands
  continue
  break

end

set wordlist = (a b c)
foreach var ($wordlist)
  echo $var
end
```

9. 判断文件类型

注意: 在使用类似-e -f的时候，后面建议跟普通路径名，而不是带通配的那种路径名。

```
#!/bin/tcsh

#比如建议使用
if (! -e ./abc/abc1) then
endif

#不建议使用如下的方式,当找不到相应文件时，脚本会报错，从而会异常退出，这不是期望的结果。
if (! -e ./*/abc1) then
endif


-f 和-e的区别
Conditional Logic on Files

-a file exists.
-b file exists and is a block special file.
```

```
-c file exists and is a character special file.
-d file exists and is a directory.
-e file exists (just the same as -a).
-f file exists and is a regular file.
-g file exists and has its setgid(2) bit set.
-G file exists and has the same group ID as this process.
-k file exists and has its sticky bit set.
-L file exists and is a symbolic link.
-n string length is not zero.
-o Named option is set on.
-O file exists and is owned by the user ID of this process.
-p file exists and is a first in, first out (FIFO) special file or
named pipe.
-r file exists and is readable by the current process.
-s file exists and has a size greater than zero.
-S file exists and is a socket.
-t file descriptor number fildes is open and associated with a
terminal device.
-u file exists and has its setuid(2) bit set.
-w file exists and is writable by the current process.
-x file exists and is executable by the current process.
-z string length is zero.
```

是用 `-s` 还是用 `-f` 这个区别是很大的！