第5章 C-shell的交互功能

5.1 UNIX的shell

- UNIX系统的重要特性之一就是提供了大量的公用程序，作为UNIX的命令，利用操作系统内核提供的系统调用，完成处理任务。
- 在这些公用程序中，有一组被称为外壳（shell）的程序，shell程序是用户和系统之间的接口，是一个交互式命令解释器。
- 用户通过它可以输入命令，然后shell调用这些命令，利用kernel的功能，完成用户的任务。

UNIX提供了几种shell，主要有：

（1）/bin/sh

- **B-shell，由Stephen R. Bourne在贝尔实验室开发，是最早被普遍认可的shell，UNIX的标准shell，设计简练。它的风格被后来出现的其他shell所继承，影响很大。**
- **它的命令行编辑功能很弱，交互操作起来非常不方便。**

（2）/bin/csh

- **C-shell，最先由加利福尼亚大学的William N. Joy在20世纪70年代开发，最初运用在BSD 2.0版本的UNIX，是Berkeley UNIX的主要特征之一。**
- **C-shell提供了历史机制和别名替换，相对B-shell来说交互起来更方便。在编程方面也更灵活，许多编程结构的风格类似C语言，所以取名C-shell。**

（3）/bin/ksh

- K-shell，由贝尔实验室的David Korn在1986年开发。
- 是B-shell的超集，支持带类型的变量、数组，等等，与sh相比，它提供了更强的功能。

（4）/bin/bash

- Bourne Again shell，这是Linux上的标准shell，它兼容B-shell，并且在标准B-shell上进行了扩展，吸收了C-shell的某些特点。
- 它的命令行编辑方法非常方便，可以直接使用键盘上的上下箭头等全屏幕编辑操作的功能键，便于交互式操作，得到许多用户的喜爱。

在配置文件/etc/passwd中记录了每个用户的用户名、用户ID、组号、主目录、登录shell等。超级用户直接编辑这个文件，也可以修改用户的注册shell。下面是/etc/passwd中的若干行：

tian:x:1289:100::/usr/tian:/bin/csh

jiang:x:1306:100::/usr/jiang:/bin/sh

liang:x:2167:100::/usr/liang:/bin/ksh

sun:x:1283:100::/usr/sun:/bin/sh

5.2 csh启动与终止

- csh启动时，将自动执行用户主目录下.cshrc文件中命令。作为登录shell运行，执行主目录中.login文件中命令。
- 不同的用户有自己独立的主目录，不同用户有自己独立的.cshrc文件。
- 作为登录shell的csh终止时，执行主目录下.logout文件中的命令。
- 如果系统不是以csh作为登录shell启动的，执行命令csh就可以进入一个交互式的C-shell。

5.3 使用csh的历史机制

所谓历史机制，指的是csh将最近一段时间内输入的命令保存起来，这样就可以重复使用前面已经输入的命令，或者前面的命令有错时，用一种简化的方法修改，而不需要将命令重新输入一遍。

5.3.1 历史表大小

- 以前键入的命令行被存在历史表中，该表的大小由csh的变量history设定。
- 内部命令set history=N，可以设定历史表大小为N个命令行。许多系统默认的历史表大小为1。

- 查看历史表：用csh的内部命令history可列出历史表内容。例如：

```
% history
87  ls -l /etc/passwd
88  vi /etc/passwd
89  ls -l /bin/passwd
90  man passwd
91  vi /etc/security/passwd
92  cd /usr/include
93  grep -n termio *.h
94   history
```

- 引用历史机制：C-shell进行“历史替换”，与历史替换有关的符号是！

表5-1 C-shell历史替换的方法

引 用 方 法	历史替换操作
!!	引用上一命令（如同DOS中按F3键）
!*str*	引用以*str*开头的最近使用过的命令。如:用!v引用刚不久执行过的命令vi disp_stat.c命令；用!fin引用最近用过的find . -name core -print命令；用!cc引用最近用过的cc disp_stat.c -o disp_stat命令；用!.引用最近用过的./disp_stat命令
!45	引用历史表中第45号命令
!20:s/*str1*/*str2*/	把历史表中第20号命令中的*str1*串替为*str2*后执行。这种方法可以修改一个以前用过的命令
^*str1*^*str2*	把历史表最后一行中*str1*串替为*str2*串，并执行。这种方法可以直接修改刚刚键入命令中的错误。例如：刚刚输入的命令find . –nmae core -print，可以用^nmae^name修改
!55:2	引用55号命令的第二个单词。命令行中的单词从左向右编号0，1，2，…。如果55号命令是上述的find命令。那么命令ls-l!!:3实际上执行 ls -l core
!55:^	引用55号命令的第一个单词
!55:$	引用55号命令的最后一个单词
!55:2-4	引用55号命令的第2~4个单词

5.4 别　名

- 使用csh的别名机制，可以为一个经常使用的命令取一个别名，帮助提高效率和减少重复劳动。
- UNIX系统中最常用的命令的命令名长度2~3个字符。可以为那些一段时间内经常使用的命令取一个单字符的别名，简化命令，别名也可以是多个字符。
- 与别名有关的csh内部命令是alias和unalias。

5.4.1 在别名表中增加一个别名

csh的内部命令alias用于定义一个别名。例如：

```
alias h         'history'
alais t         'tail -f /usr/adm/pppd.log'
alias l         'ls -Fl'
alias rm        'rm -i'
alias dir       'ls -Flad'
alias type      'cat'
alias n         'netstat -p tcp -s | head -10'
alias r         'netstat -rn'
alias p         'ping 202.143.12.189'
alias rt        'traceroute 217.226.227.27'
```

5.4.2 查看别名表

使用不带任何参数的alias命令，能打印出当前的别名表。

5.4.3 给别名传递参数

许多命令带有参数，使用别名带有参数时，就将参数传递给命令。例如：

alias dir 'ls -Flad'

dir ~ 那么，实际执行ls -Flad ~

在csh中，~是个特殊符号，它将被csh展开为当前用户的主目录。

【例5-1】 给别名命令传递参数的方法。

下面的命令递归式地在系统头文件目录下检索所有的头文件，查找含有termio字符串的程序行。

```
find /usr/include -name "*.h" -exec grep -n termio {} /dev/null \; | more
```

- 别名参数的使用，用一个惊叹号表示当前的输入。在冒号后边的数字或者^号$号代表参数号。

alias f 'find /usr/include -name "*.h" -exec grep -n \!:$ {} /dev/null \; | more'

- 这里的惊叹号前面加上转义符\，使得csh不再把他们解释为历史替换，而是把惊叹号作为一个字符，传递给alias命令，alias真正得到的是!:$。
- 直接使用f termio就可以了。
- 传递多个参数。

alias scan 'find \!:3 -name "*.h" -exec grep -n \!:1 {} /dev/null \; | more'

- 引用时，使用命令： **scan termio in /usr/include**

5.4.4 取消别名

使用内部命令**unalias**，可以取消别名。用法是：

unalias 别名

例如：

unalias n　在别名表中取消别名**n**

5.5 csh提示符

- csh的提示符默认是%。
- 用户可以使用csh的变量prompt控制，自定义csh提示符。

```
set prompt="[\!]%"
```

5.6 csh的管道和重定向

在交互式命令中，经常会使用到管道和重定向。标准输入和标准输出的输入重定向以及管道，在不同的shell中，用法都相同。例如：

ls -l | more

man ls > man.paper

man rm >> man.paper

tr UVW uvw < file1 > file2

重定向符号>>是向文件中追加。除了标准输入和标准输出的输入重定向和管道之外，标准错误输出的重定向在不同的shell之间有所不同。

例如：C语言的程序员很容易会发现标准错误输出和标准输出的不同之处。

cc myap.c -o myap | more

这样的命令并不能使得编译程序产生的错误信息停下来逐屏显示，而是照常滚动。同样的：

cc myap.c -o myap > err.list

也不能把编译程序产生的错误信息写到文件**err.list**中，而是照常在屏幕上滚动显示。究其原因，上述的两个命令的管道和重定向操作，第一个命令是将**cc**命令的标准输出管道到命令**more**的标准输入，第二个命令是将标准输出重定向到文件**err.list**，而**cc**给出的错误信息是在标准错误输出上输出的，所以根本不起作用。

5.6.1 标准输入，标准输出，标准错误输出

- C语言中操作文件有两种模式，一种是标准的缓冲I/O方式，一种是低级的直接使用系统调用的方式。
- 使用缓冲方式时，在stdio.h头文件中已经声明了3个变量，分别对应标准输入，标准输出，标准错误输出。

  extern FILE *stdin, *stdout, *stderr;
- 直接使用系统调用的方式操作文件，文件描述符0,1,2分别对应标准输入，标准输出和标准错误输出。这三个文件描述符对应的文件，分别与当前终端的键盘输入，屏幕输出相关联。

执行一条命令，在屏幕上产生的输出，到底属于标准输出，还是标准错误输出，要看实现这一命令的程序是如何编制的。

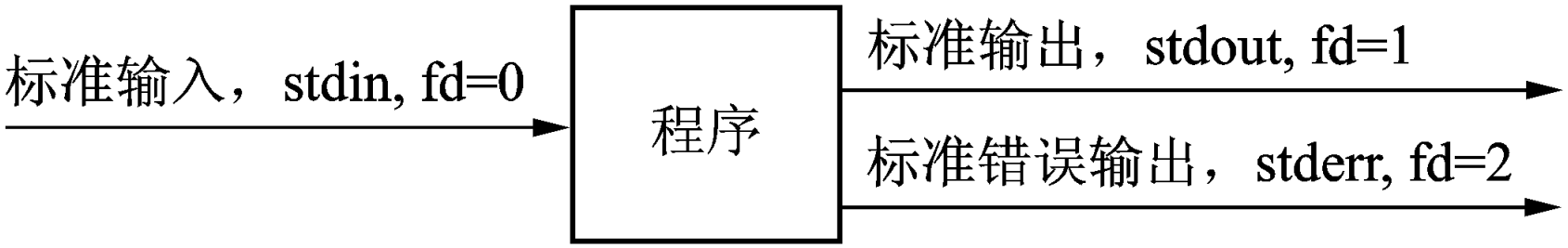

图5-1 标准输入，标准输出，标准错误输出

下面的例子，在屏幕上输出的所有STRING1属于标准输出，所有的STRING2都属于标准错误输出。

【例5-2】使用缓冲I/O标准输出和标准错误输出。

```
#include <stdio.h>
main()   /* 使用缓冲I/O */
{
      static char  *str1= "STRING1\n";
      static char  *str2= "STRING2\n";
      int i;
      for (i=0;i<20;i++) {
        printf(str1);  /* 或:fprintf(stdout,str1); */
        fprintf (stderr,str2);
      }
      exit(0);
}
```

【例5-3】使用原始I/O的标准输出和标准错误输出。本例与例5-2有等价的效果。

```
main()  /* 使用原始I/0 */
{
      static  char  *strl= "string1\n";
      static  char  *str2= "string2\n";
      int i;
      for(i=0;i<20;i++) {
        write(1,strl,strlen(str1));
        write(2,str2,strlen(str2));
      }
      exit(0);
}
```

5.6.2 标准输出和标准错误输出重定向

- 为了防止无声息地覆盖掉重要的文件，csh提供了一个变量noclobber防止这种现象发生，给予重定向的文件一个简单的保护机制。设置和取消这个变量的csh命令分别是：

  set noclobber

  unset noclobber

  阅这个变量值的命令是不带任何参数的set命令。

5.6.2 标准输出和标准错误输出重定向

1. 未设置csh变量noclobber情况下

未设置csh变量noclobber情况下，重定向用法有以下几种：

用法1：>文件

把stdout重定向到一个文件中，但不影响stderr仍然在终端输出，见图5-2。

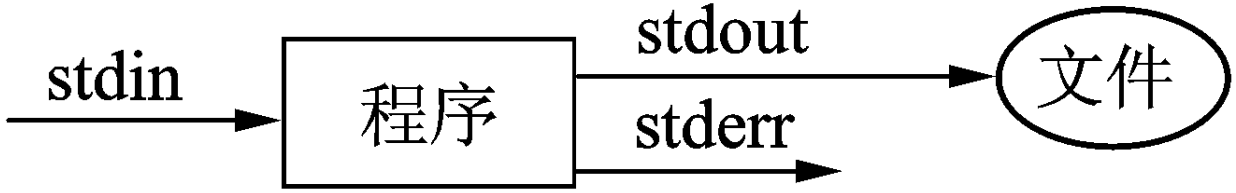

图5-2 标准输出重定向

用法2：>& 文件

把stderr合并到stdout，然后又重定向到一个文件中，见图5-3。

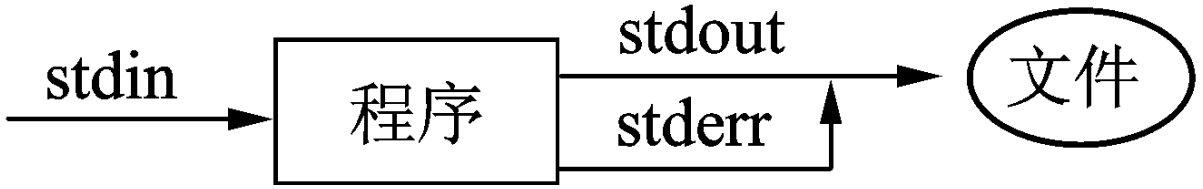

图5-3 标准输出和标准错误输出合并后重定向

例如：cc myop.c -o myop >& err.list

注意，在>和&之间不许有多余的空格字符。

对>和>&来说，若文件不存在则创建新文件，否则覆盖掉原文件。

用法3：>>文件　>>&文件

若文件不存在，则创建；若文件已存在则追加到文件尾。

2. 设置csh变量noclobber情况下

- 为了防止无声息地覆盖掉重要的文件，csh提供了一个变量noclobber防止这种现象发生，给予重定向的文件一个简单的保护机制。设置和取消这个变量的csh命令分别是：

set noclobber

unset noclobber

2. 设置csh变量noclobber情况下

设置csh变量noclobber情况下，重定向用法有以下几种：

用法1：>文件 >&文件

若文件已存在则出错；否则创建。

用法2：>!文件 >&!文件

若文件已存在则强制覆盖掉原文件；否则创建。

用法3：>>文件 >>&文件

若文件不存在则出错；否则追加到尾部。

用法4：>>!文件 >>&!文件

若文件不存在，强制创建；否则追加到尾部。

5.6.3 管道

1. 标准输出管道|

用法：命令1|命令2

例如：ls -l|more

把前面命令的stdout管道成下一命令的stdin，不影响stderr输出，见图5-4。

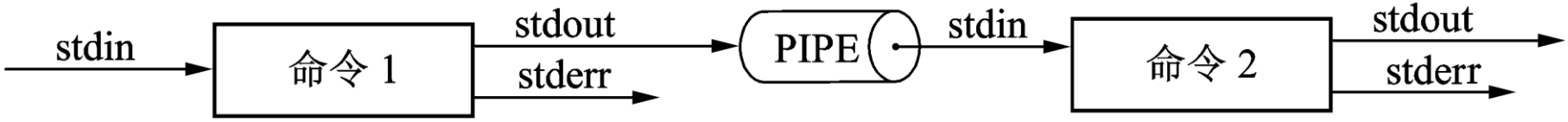

图5-4 将标准输出管道到下个命令

2．标准输出和标准错误输出合并后管道|&

用法：命令1|&命令2

把stderr合并到stdout然后管道到下一命令，见图5-5。

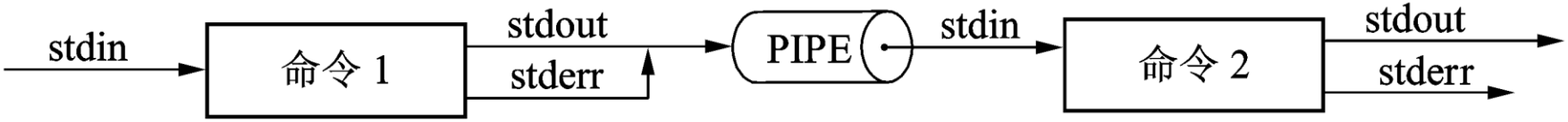

图5-5 将标准输出和标准错误输出合并后管道到下个命令

举例：

cc myjob.c -o myjob | more

若有100个错误行，不能逐屏显示错误信息。

cc myjob.c -o myjob |& more

可以逐屏显示错误信息。