<http://blog.sina.com.cn/s/blog_4ba5b45e0102e3wn.html>

**1. 前言**

Unix 界有一句名言：“一行shell脚本胜过万行C程序”，虽然这句话有些夸张，但不可否认的是，借助脚本确实能够极大的简化一些编程工作。比如实现一个 ping程序来测试网络的连通性，实现ping函数需要写上200~300行代码，为什么不能直接调用系统的ping命令呢？通常在程序中通过 system函数来调用shell命令。但是，system函数仅返回命令是否执行成功，而我们可能需要获得shell命令在控制台上输出的结果。例如， 执行外部命令ping后，如果执行失败，我们希望得到ping的返回信息。

**2. 使用临时文件**

首先想到的方法就是将命令输出重定向到一个临时文件，在我们的应用程序中读取这个临时文件，获得外部命令执行结果，代码如下所示：

    #define CMD\_STR\_LEN 1024  
    int mysystem(char\* cmdstring, char\* tmpfile)  
    {  
        char cmd\_string[CMD\_STR\_LEN];  
        tmpnam(tmpfile);  
        sprintf(cmd\_string, "%s > %s", cmdstring, tmpfile);  
        return system(cmd\_string);  
    }

这种使用使用了临时文件作为应用程序和外部命令之间的联系桥梁，在应用程序中需要读取文件，然后再删除该临时文件，比较繁琐，优点是实现简单，容易理解。有没有不借助临时文件的方法呢？

**3. 使用匿名管道**

在<<UNIX 环境高级编程>>一书中给出了一种通过匿名管道方式将程序结果输出到分页程序的例子，因此想到，我们也可以通过管道来将外部命令的结果同应用 程序连接起来。方法就是fork一个子进程，并创建一个匿名管道，在子进程中执行shell命令，并将其标准输出dup到匿名管道的输入端，父进程从管道 中读取，即可获得shell命令的输出，代码如下：

int mysystem(char\* cmdstring, char\* buf, int len)  
{  
      int   fd[2];  
      pid\_t pid;  
      int   n, count;   
      memset(buf, 0, len);  
      if (pipe(fd) < 0)  
          return -1;  
      if ((pid = fork()) < 0)  
          return -1;  
      else if (pid > 0)       
      {  
          close(fd[1]);       
          count = 0;  
          while ((n = read(fd[0], buf + count, len)) > 0 && count > len)  
              count += n;  
          close(fd[0]);  
          if (waitpid(pid, NULL, 0) > 0)  
              return -1;  
      }  
      else                    
      {  
          close(fd[0]);       
          if (fd[1] != STDOUT\_FILENO)  
          {  
              if (dup2(fd[1], STDOUT\_FILENO) != STDOUT\_FILENO)  
              {  
                  return -1;  
              }  
              close(fd[1]);  
          }   
          if (execl("/bin/sh", "sh", "-c", cmdstring, (char\*)0) == -1)  
              return -1;  
      }   
      return 0;  
}

**4. 使用popen**

在学习unix编程的过程中，发现系统还提供了一个popen函数，可以非常简单的处理调用shell，其函数原型如下：

FILE \*popen(const char \*command, const char \*type);

该函数的作用是创建一个管道，fork一个进程，然后执行shell，而shell的输出可以采用读取文件的方式获得。采用这种方法，既避免了创建临时文件，又不受输出字符数的限制，推荐使用。

popen使用FIFO管道执行外部程序。  
  
#include <stdio.h>  
FILE \*popen(const char \*command, const char \*type);  
int pclose(FILE \*stream);  
  
popen 通过type是r还是w确定command的输入/输出方向，r和w是相对command的管道而言的。r表示command从管道中读入，w表示 command通过管道输出到它的stdout，popen返回FIFO管道的文件流指针。pclose则用于使用结束后关闭这个指针。  
  
下面看一个例子：  
  
  
#include <sys/types.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
  
int main( void )  
{  
FILE   \*stream;  
FILE   \*wstream;  
char   buf[1024];  
  
memset( buf, '\0', sizeof(buf) );//初始化buf,以免后面写如乱码到文件中  
stream = popen( "ls -l", "r" ); //将“ls －l”命令的输出 通过管道读取（“r”参数）到FILE\* stream  
wstream = fopen( "test\_popen.txt", "w+"); //新建一个可写的文件  
  
fread( buf, sizeof(char), sizeof(buf), stream); //将刚刚FILE\* stream的数据流读取到buf中  
fwrite( buf, 1, sizeof(buf), wstream );//将buf中的数据写到FILE    \*wstream对应的流中，也是写到文件中  
  
pclose( stream );    
fclose( wstream );  
  
return 0;  
}   

**5. 小结**

有 统计数据表明，代码的缺陷率是一定的，与所使用的语言无关。Linux提供了很多的实用工具和脚本，在程序中调用工具和脚本，无疑可以简化程序，从而降低 代码的缺陷数目。Linux shell脚本也是一个强大的工具，我们可以根据需要编制脚本，然后在程序中调用自定义脚本。