

A Couple of Unix Tips

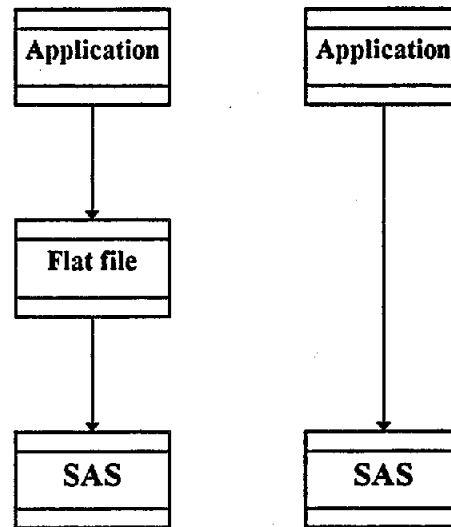
By Patrick Coburn
Case Western Reserve University

Mr. Coburn is the recently appointed Systems Manager for the General Clinical Research Department, School of Medicine, Case Western Reserve University. Prior to that he was the Systems Manager for the Center for Regional Economic Issues, Weatherhead School of Management, Case Western Reserve University. In these capacities, he was responsible for maintaining SAS® on platforms ranging from PC's to workstations to mainframes for researchers using anything from simple statistics on their databases to complex manipulations on large databases.

The first tip: using the 'pipe' engine.

I. What does a UNIX pipe do? It allows you to move data directly between two applications without "pausing" to write out a file. It does this by issuing a command to the host operating system, and then reading the output of that command as input in a SAS data step. It is the poor man's approach to creating an interface on UNIX when one doesn't exist.

For example, if there was an application that SAS® does not have an interface to, the average user might easily extract data from this application into a file and then read this file into SAS. The pipe allows the output from one program to become the input into another program.



II. How does this differ from the display manager X command?

The x command in display manager also allows you to pass a command to the host system, but will do so independent of what is happening in the SAS system.

For example, we would like to do a directory listing, reading in the result and making a permanent SAS dataset from it. As a little extra challenge, we would like to repeat the data step and see what happens.

To do so with a x command you would issue this series of commands:

```
libname mylib '/homes/userid';
dm x'ls-l /homes/userid >
temp';
filename dirt 'temp';
```

```
data mylib.dirt;
  infile dirt lrecl=80 pad;
  input @ 1 line $char80.;
```

```
data mylib.dirt;
  infile dirt lrecl=80 pad;
```

```
input @ 1 line $char80.;

proc print; run;
```

To do so with a pipe command you would issue these commands:

```
libname mylib '/homes/userid';
filename dirt pipe 'ls -l';

data mylib.dirt;
  infile dirt lrecl=80 pad;
  input @ 1 line $char80.;

data mylib.dirt;
  infile mylib.dirt lrecl=80
pad;
  input @ 1 line $char80.;

proc print; run;
```

Two things are immediately obvious:

- 1) With the *x* command, we need to write our output to a file before reading that file in.
- 2) With the pipe command SAS is actually reading as input the directory information as it is being created.

III. How (and why) does this happen?

UNIX has a feature called standard in and standard out. The easiest way to think of these features is that standard in (stdin) is the keyboard and standard out (stdout) is the screen. When linked together, the stdout from one program can become the stdin of another. Some programs or languages, like C and Fortran, allow a user to read/write directly to the stdin and stdout, while others will allow a user indirect access by the "keyboard/screen" approach - that is, if it can be read from the keyboard, or written to the screen you have stdin and stdout.

IV. How to use it.

Pipe is an engine you would use in the filename command:

```
filename myface pipe 'UNIX
program';

data test;
  infile myface;
  input @ 1 var1 @ 10 var2;
```

A simple example might be reading in the directory structure:

```
filename dirstr pipe 'ls -l';

data dir;
  infile dirstr lrecl=80;
  input @ 1 attrib $char10.
```

V. How we used the pipe feature.

We received a database from the Center for Research on Security Prices (CRSP)TM, University of Chicago, which, in binary form, takes up about 2 gigabytes. Simple tests indicated that creating a SAS dataset from this would run 32 gigabytes. SAS offers a procedure that would read this (PROC DATASOURCE), but we found that to be undesirable for two reasons: 1) It had some bugs in it using SUN® binary, and 2) it did not allow for an input dataset of companies to be extracted. But the CRSP supplied access routines were written in Fortran, programming in which is rapidly becoming a lost art on our campus. So, how do we use SAS to access this database?

Step 1: We wrote an generic Fortran access routine that would allow for an input dataset, and write out to the stdout (Fortran unit 6) a list of commonly used variables. We also included an option to allow the user to extract all the variables for that company. We included options that would allow the

user to access one of three main CRSP databases.

Step 2. We wrote a SAS input statement that would read in the stdout from the Fortran based routine. We read in the variables according to what we knew we would receive, cleaned them up, and merged them back together. Because the size of the extracts can be quite large, the ability to skip the step of writing out an intermediate file became very important.

Step 3. We placed the filename statement with the pipe command at the top with instructions on how to modify it:

```
filename crsp pipe 'crsp -
daily -max mydata.inp
mydata.out';
```

Step 4. We then placed the entire SAS program in a common area for users to access.

The final code looked something like this:

```
filename crsp pipe 'crsp -
daily myinput.dat';

data crsp;
    infile crsp lrecl = 250
    pad missover;
```

VI. Some other important applications.

SAS/AF and SAS/Connect: Allows a frame entry to be built that would give a local user a chance to do some simple file management, such as directories, delete a file, rename a file, as well as some more powerful customized features.

The second tip: Using the UNIX display redirection to leverage our equipment.

The problem: We have a very fast SUN Server 1000™ with no monitor on it, and a mixture of less powerful workstations and PC's connecting into it. For the PC's we were using SAS/Connect, but for the workstations we would like to use the CPU power of the SUN Server, but the monitor of the workstation.

The solution: Redirecting the display to the workstation.

To do this, start the OpenWin or Xwindows session on the local workstation. Open a rlogin session on the remote workstation. Start SAS there by typing:

```
sas -fdisplay local:0.0 &
```

where local is the name of the local workstation. This will direct the display back to the local workstation.

Note 1: You may choose to use SAS/Connect between the two workstations, but this is wasting resources, and only get the same functionality as the above solution.

Note 2: This solution would work with PC's running an Xwindow's emulator.

Note 3: Redirecting is a handy tool for any UNIX work. You may read and write tapes or CD's using a drive attached to another machine.

Contact information:

Patrick Coburn
General Clinical Research Center
University Hospitals
Cleveland, Ohio 44106-5041
Phone: (216) 844-1546
e-mail: pxc5@po.cwru.edu