

Unix and SAS: Getting Started
Sara Hickson, Brigham and Women's Hospital, Boston, MA

ABSTRACT

The Unix operating system is a powerful way to run programs efficiently on very large datasets, and has advantages in flexibility over a Windows system when combined with SAS[®]. The introduction to Unix and Unix scripting can be quite a shock, however, compared to the relatively intuitive interface of the PC. Although each environment has its own versions and added software, this paper will touch on the more universal challenges of getting started using SAS in Unix, including the basics of getting started in Unix, running programs in batch mode and how to use the most commonly available editors to write programs. We will also look at some simple Unix scripting to investigate files.

INTRODUCTION

Unix systems can vary widely, depending on the type of Unix and servers you are using. This paper was written using a Sun SPARC Server E3000, using Sun Microsystems' Solaris. In addition to standard Solaris, various other commands and software packages have been installed or modified, so what is described in this paper might differ on another UNIX computer system. Your system administrator should be able to tell you how to do a certain task on your system.

ASSUMPTIONS

I am assuming prior knowledge of SAS in a PC-based environment. Though there are programs such as X-Windows which can produce an interactive environment on some Unix systems, this paper assumes there is not access to this option (you won't need many of these hints if you do have X access!) But speed can be compromised quite a bit when using X-Windows, and the utility of this application varies widely depending on your system. Also, it is possible to run programs in batch mode from PC SAS, but it is not very common. Resources listed at the end of the paper which can give you more info on those options. I am also assuming little or no knowledge of Unix.

UNIX SHELLS

A shell is a command interpreter-a method of interacting with the computer. The shell only changes how you accomplish tasks, though; it does not change the underlying capabilities of Unix. There are a number of different shells: csh, bash and ksh, for example. The most common shell is bash. You can change between shells easily, depending on your system either by typing chsh and the name of the shell at the following prompt, or by typing the name of the shell at the command prompt. It is important to find out which shell you are using before getting started, and to use references specific to that shell. Most of the commands below are tcsh shell.

UNIX COMMANDS

Unix is accessed from a command line. After you log-in to the system, you will see a command prompt designated with \$ or %. Below are a few simple commands to give you some examples. Each of the following commands is typed at the command prompt.

- **man** *command name* displays manual (help) for specified command-very helpful!
- How to run a SAS® program is set during installation at your site, so your administrator should let you know what that is. Commonly, the command is **sas filename.sas**.
- To search files for a string, use the grep command followed by the string and then the filename. You can also search all files in a directory: **grep datastring ./***, (* is a wildcard).
- Use the **tab** key to auto complete a file or directory name after typing the first few letters, making moving between directories much easier. This may need to be set up, which you can do by typing **set filec** at the command prompt.
- **ls** lists the files in the current directory
- **more filename** displays a file on the screen
- **head filename** displays the first few records of a file
- **tail filename** displays the last few records of a file
- **cp oldfile newfile** makes a copy of oldfile under the name newfile
- **cp -p oldfile newfile** makes a copy of oldfile under the name newfile, while preserving date/time information from oldfile
- **mv oldfile newfile** moves (renames) a file
- **rm filename** deletes a file
- **rm -r** deletes directory, even if the directory is not empty
- **cd directory name** moves you from your current directory to directory
- **cd ../** moves you up one directory, to the parent directory
- **pwd** shows the name of your current directory
- **mkdir directory name** creates directory
- **rmdir directory name** deletes directory, but this will only work if the directory is empty
- **diff fileone filetwo** To find the differences between two files. Lines unique to fileone will be preceded with a <, files unique to filetwo with a >
- **!** followed by the first few letters of the command will repeat a previously typed command.

Commands can be strung together through the use of pipes: |. On a qwerty keyboard, it is located at shift-backslash. This will pass the results of one command to the next command.

Example:

To find all files containing ID numbers in a range, sort and print the results:

```
grep '100???' .* | sort | qpr
```

SCRIPTING

There are a number of utilities available in UNIX depending on the setup of the system. As stated before, there will be at least one shell, which allows commands to be passed from the user to the computer (or kernel). It is, in effect, a programming language: a simple, limited language; but a language nonetheless. UNIX systems will always come with many other languages such as a mail program and VI. Typically they will also have such tools as SED (which is a complex, non-interactive editor) and AWK (which is a language). SED and AWK are two powerful tools for anyone using the UNIX System that allow for more flexibility than the base UNIX system commands for editing and reporting.

SED works on source files without changing them, so it can be used for most searching and editorial tasks, as well as much more complicated work.

```
$ sed -e 's/oldinfo/newinfo/' fileone > filetwo
```

This would change all oldinfo references in fileone with newinfo in filetwo. This would work the same as search and replace in any typical editor with the built in safety feature of never changing the original file.

```
$ sed '1,4d' infile > outfile
```

This would get rid of a header on infile and put it in outfile.

```
$ sed -n '/string/p' fileone
```

Print all lines from fileone with string to the screen.

AWK is a programming language for dealing with output and reporting.

```
$ date | awk '{print "The Time is " $4}'
```

Piping the date command's output to AWK, we can print the time to the screen.

```
The Time is 15:08:52
```

```
$ ls -ltr | awk '{print $9}' | sort -r
```

Sending a long list of all files in a directory, awk will pick out the names of the files and send it to be sorted in reverse order.

Small commands can be helpful, but it is by combining them that we can complete complex tasks. And this is called Shell Scripting. The strength of Scripting is that it allows you to work with AWK, SED, shell commands and Unix commands in the same file.

Before you start scripting, there are a few points to keep in mind:

1. Changes to a file cannot be reversed, so creating a new file to check the results is a good idea.
2. Work through what you need to do and the best way to do it before starting to program.
3. Test and check thoroughly before removing or changing any original files.

Here is an example of a search and replace script called newgrep, that utilizes shell commands, unix commands (found between `` marks), sed commands and awk commands. For the purpose of this tutorial this is exactly the same as using the first command above followed by a head, but this shows the ability to mix and match commands as a started. The # preceded and followed by a space indicates a comment, and is not read for the program. The symbol may, however, be used in other places.

```
# first line tells UNIX to use the bash shell
#!/bin/sh

# if then statement ensures that three parameters are entered
if [ $# -lt "3" ]
then
    echo Usage: newgrep pattern replacement file
    exit 1
fi

# set first pattern to what will be copied, set second pattern to what will be pasted
pattern=$1
replacement=$2

# ensures a file has been sent in as the third parameter. Sets it file or creates an error
if [ -f $3 ]
then
    file=$3
else
    echo $3 is not a file.
    exit 1
fi

# the following used UNIX commands to replace 1st pattern with 2nd pattern
`tr $pattern $replacement < $file > $file.new`

# uses awk and sed to print the first 12 lines from the new file to the screen
sed '12,$d' $file.new | awk {print}

rm -f $file.temp
```

To run the script:

```
./scriptname.sh
```

SAS: BATCH MODE VERSUS INTERACTIVE MODE

Interactive mode means running SAS in a Windows-type environment. Coming from a PC SAS® platform, you are probably used to working with Display Manager in interactive mode. You can save datasets in a working directory between runs, and run individual DATA or PROC steps on those datasets. You can then see the output from that code dynamically. Many tasks, such as running a program or switching between program and log files are either point and click or keyboard shortcuts. PC interactive SAS also uses syntax-specific color highlighting (Figure 1), making it easier to find mistakes and errors.

However, since PC SAS® usually runs on a local machine, your resources can be limited in the amount of data the system can handle, and it can require either copying files across a network or storing them on your local machine. You must also be aware of when each piece of your code will run: putting *run;* at the end of each step is the common way to handle this issue.

Batch mode is a way of submitting a complete file containing SAS code for processing. A batch queue is a way of running a job in the background at a lower priority, so that only a limited number of jobs are running at any single time, essential to maintaining a multi-user system. Systems use batch queues to process computationally intensive jobs (e.g., SAS programs using large data files). Running SAS statistical jobs in batch helps keep the system responsive to interactive needs, such as editing. When processing is complete, the output will consist of one or two files saved in the same directory as the program file. Batch mode involves creating a file containing your SAS commands, sending that file to a queue for processing, and reading or printing the results from an output file. The batch sequence has two advantages over interactive use: it maintains an exact printed record of the procedures and data steps used, which is necessary for reproducing the results; and it restricts the number of SAS® jobs that can be run simultaneously, which ensures that the shared computer system is not overburdened.

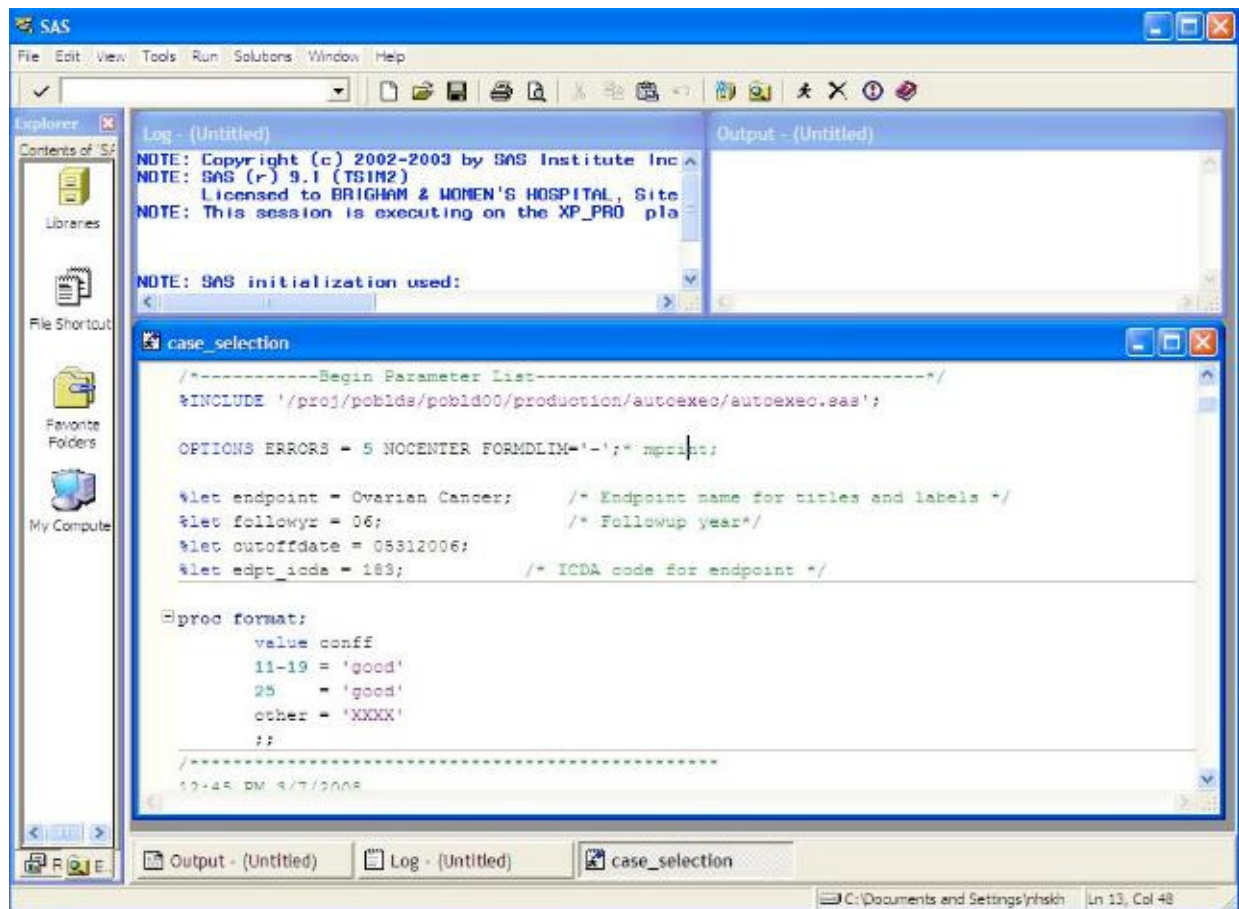


Figure 1: A Screen shot of PC SAS

EDITING CODE BY USING EDITORS

A SAS program to be submitted using batch mode can be written in any text editor. There are a number of different editors within Unix, but the two most common are Emacs and VI. The main difference between the two is that VI is modal and Emacs is modeless. Modes are a way of having different commands available using the same keys. So, the function of a certain command depends on the mode currently in use. It comes down to personal preference, though each editor has its ardent supporters, and there is much debate about which is easier to use. I have included Pico as well, because it is much simpler and runs more quickly, and nice to know if you just need to make a small change to a program. Each of them is started by typing the program name followed by the filename at the command prompt. For example: **pico filename.sas**.

```

UW PICO(tm) 4.6      File: case_selection.sas

/*****
Program for '06 Ovarian Cancer Selection
Sara Hickson
March 2008

*****/

/*-----Begin Parameter List-----*/
%INCLUDE '/proj/pobids/pob1d00/production/autoexec/autoexec.sas';

OPTIONS ERRORS = 5 NOCENTER FORMDLIN='-' ;* mprint;

%let endpoint = Ovarian Cancer;      /* Endpoint name for titles and labels &
%let followyr = 06;                  /* Followup year*/
%let cutoffdate = 05312006;
%let edpt_icda = 183;                /* ICDA code for endpoint */

^G Get Help   ^O WriteOut  ^R Read File  ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where is  ^V Next Pg   ^U UnCut Text ^I To Spell

Ready          sp2: AES 256      3, 1      24 Rows, 80 Cols  VT100      NUM

```

Figure 2: A Screen shot of the Pico editor.

PICO

The simplest editor to use is Pico. It is a full-screen editor (Figure 2). It comes with Pine, a Unix email reading program, but can be used as a stand-alone editor. Depending on your system, you can often use the same keyboard commands as your computer (ie, up, down, delete). Other commands are accessed using the Control key with another key. Common commands are displayed at the bottom of the editor, and to view the help files, which provides a more extensive list of shortcut keys, press ctrl-G.

Ctrl-W	activates a word-search feature
Ctrl-X	exits the editor, and type Y to save your program
Ctrl-F	move Forward a character.
Ctrl-B	move Backward a character.
Ctrl-P	move to the Previous line.
Ctrl-N	move to the Next line.
Ctrl-A	move to the beginning of the current line.
Ctrl-E	move to the End of the current line.
Ctrl-V (F8)	move forward a page of text.
Ctrl-Y (F7)	move backward a page of text.

EMACS

```
capecod.bwh.harvard.edu - PuTTY
OPTIONS ERRORS = 5 NOCENTER FORMDLIN='-' ; * mprint;

%let endpoint = Ovarian Cancer;      /* Endpoint name for titles and labels \
 */
%let followyr = 06;                 /* Followup year*/
%let cutoffdate = 05312006;
%let edpt_icda = 183;               /* ICDA code for endpoint */

proc format;
  value conf
--**-XEmacs: case_selection.sas (SAS PenDel Font)---L21--2%-----
----XEmacs: white (Fundamental PenDel)---L1--A11-----
C-C -
```

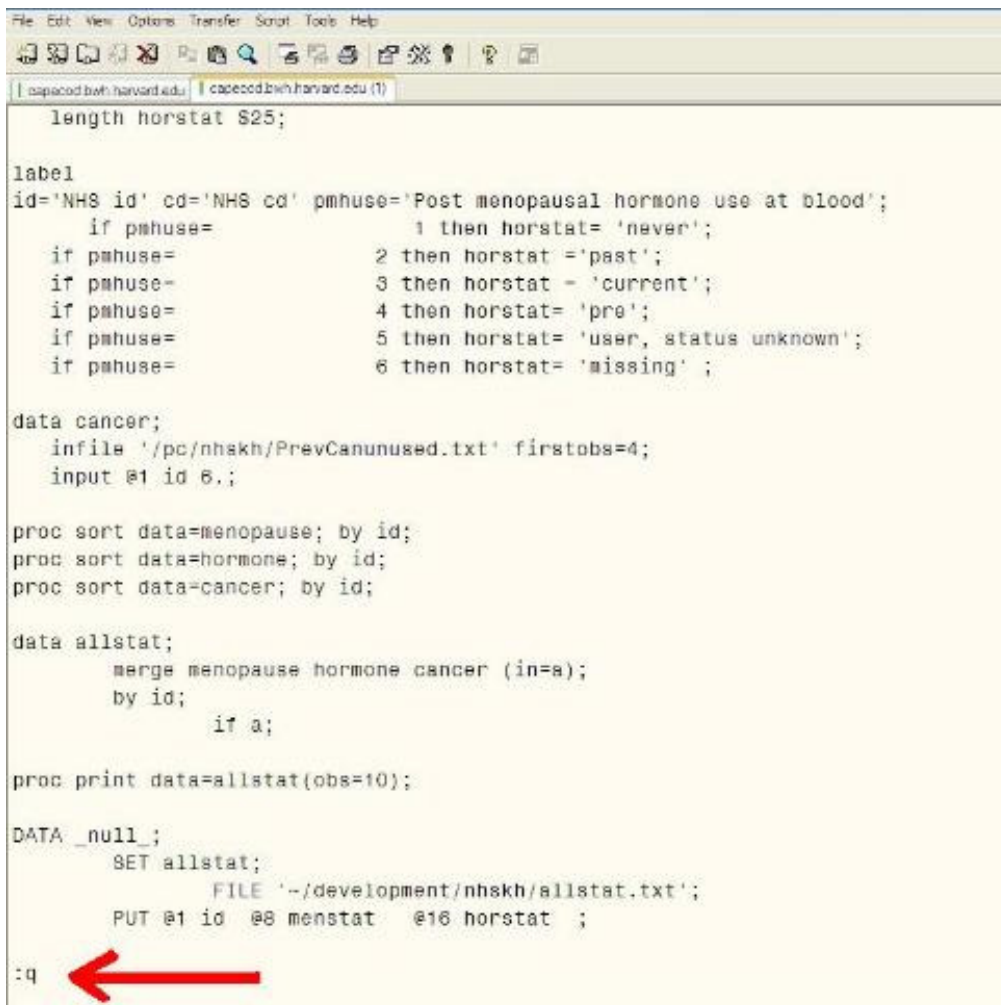
Figure 3: A Screen shot of Emacs

Emacs is more complex and more powerful than Pico. On most systems, the arrow keys are enabled, but you can also move around and edit with keyboard commands. Commands in Emacs are accessed using the control key (Ctrl) and a meta key (M), and are shown in a buffer area of the screen, usually along the bottom (Figure 3). This meta key can be set to the ALT or Escape key.

Ctrl-H, then T runs the online tutorial.
 Ctrl-X Ctrl-C exits Emacs with an option to save.
 M-> moves to the end of the file.
 Ctrl-n moves to the next line
 Ctrl-G cancels any pending commands in the buffer area.

VI

The standard editor available on all Unix systems is VI. It is also the most complex, with a steep learning curve. Once mastered, it can be a much faster way to edit, as you are able to keep your hands on the keyboard's home row most of the time, regardless of the task. To start a session, type: **vi filename** at the command prompt. VI has different modes: command mode, in which keys you type are commands to the editor; and insert mode, in which keys you type are inserted as text. VI starts in command mode; type 'i' to enter insert mode; press ESC (ESCAPE) to get back to command mode. The sequence shift q will result in a prompt to the bottom left-hand corner that looks like a colon (Figure 4). There are few other screen indications that you are in VI, or which mode you are using. At the : prompt, q! will quit vi without saving any changes; : wq will save changes and then quit. For a quick tutorial in VI, type **vi tutorial** at the command prompt. Below are a few of the basic commands, used in the command mode.



```

length horstat $25;

label
id='NHS id' cd='NHS cd' pmhuse='Post menopausal hormone use at blood';
  if pmhuse=          1 then horstat= 'never';
  if pmhuse=          2 then horstat= 'past';
  if pmhuse=          3 then horstat= 'current';
  if pmhuse=          4 then horstat= 'pre';
  if pmhuse=          5 then horstat= 'user, status unknown';
  if pmhuse=          6 then horstat= 'missing' ;

data cancer;
  infile '/pc/nhskh/PrevCanunused.txt' firstobs=4;
  input @1 id @6.;

proc sort data=menopause; by id;
proc sort data=hormone; by id;
proc sort data=cancer; by id;

data allstat;
  merge menopause hormone cancer (in=a);
  by id;
  if a;

proc print data=allstat(obs=10);

DATA _null_;
  SET allstat;
  FILE '-/development/nhskh/allstat.txt';
  PUT @1 id @8 menstat @16 horstat ;

:q

```

Figure 4: A Screen shot of VI

k	move up a line
j	move down a line
h	move left one character
l or spacebar	move right one character
x	delete a character
dd	delete a line
r	replace a character
shift-r	overwrite character in the current word
w	move to the first character of the next word

CONCLUSION

Moving to Unix from a PC system can be a big change, but there are distinct advantages in using a Unix. The ability to process much larger amounts of data and the flexibility of the operating system can increase productivity once the initial learning curve is achieved. This paper gives a brief introduction to some basic Unix knowledge, and opens a door for further exploration.

REFERENCES

Taylor, D. Teach yourself Unix in 24 hours. Sams, Indianapolis, Indiana: 2005.
 Dougherty, D. sed & awk. O'Reilly and Associates, Sebastopol, CA: 2000.
 An online guide to pico: <http://www.indiana.edu/~uitspubs/b103>
 An online guide to vi: <http://staff.washington.edu/rells/r110/>
 an online guide to emacs: <http://cmgm.stanford.edu/classes/unix/emacs.html>
 A good discussion about the benefits of emacs vs. Vi: <http://c2.com/cgi/wiki?emacsvsvi>
 Different shells available: <http://www.peachpit.com/articles/article.aspx?p=659655&seqNum=2>
 Shell Variables: <http://steve-parker.org/sh/variables1.shtml>

ACKNOWLEDGMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

Support for this paper was from NIH grants P01 CA67262 and CA49449.

Much thanks to Chris Murphy for help with the scripting section-your help made it possible!

Contact Information

Please contact me if you have any questions!
 Sara Hickson, MPH
 Data Manager/SAS® Programmer
 Nurses' Health Study
 Channing Laboratory
 Brigham and Woman's Hospital
 nhskh@channing.harvard.edu
