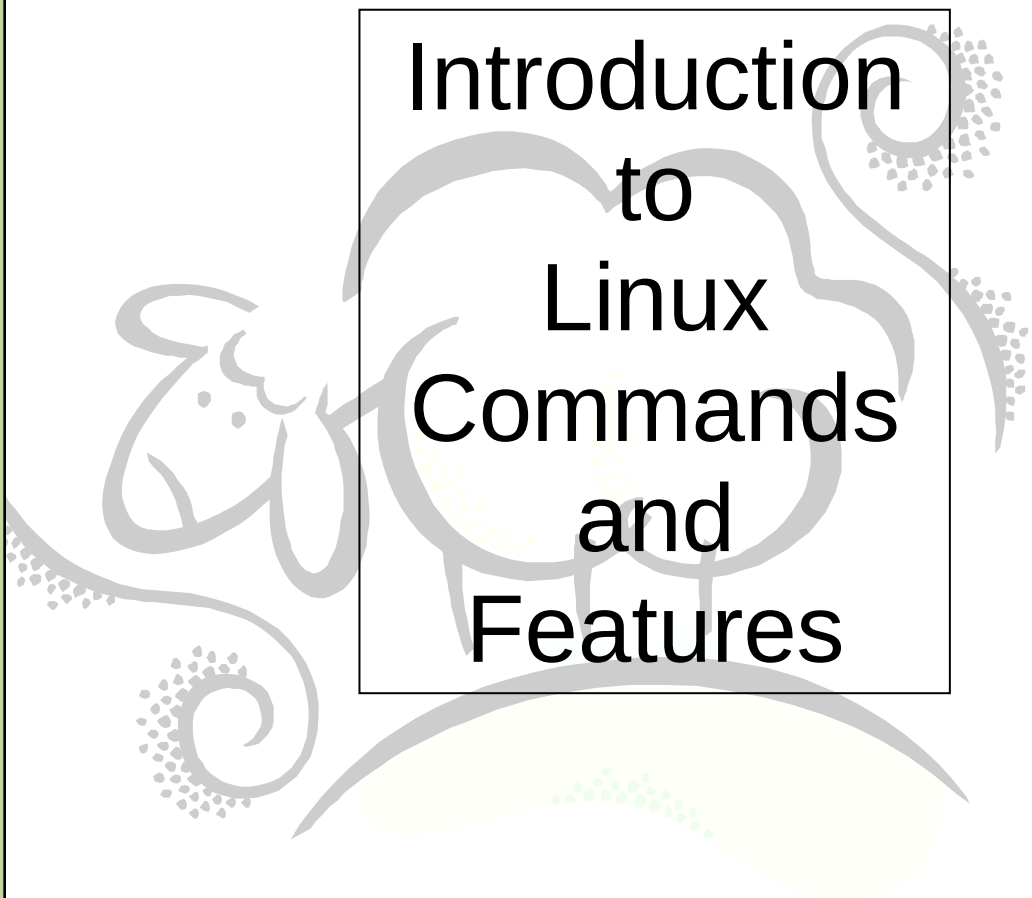
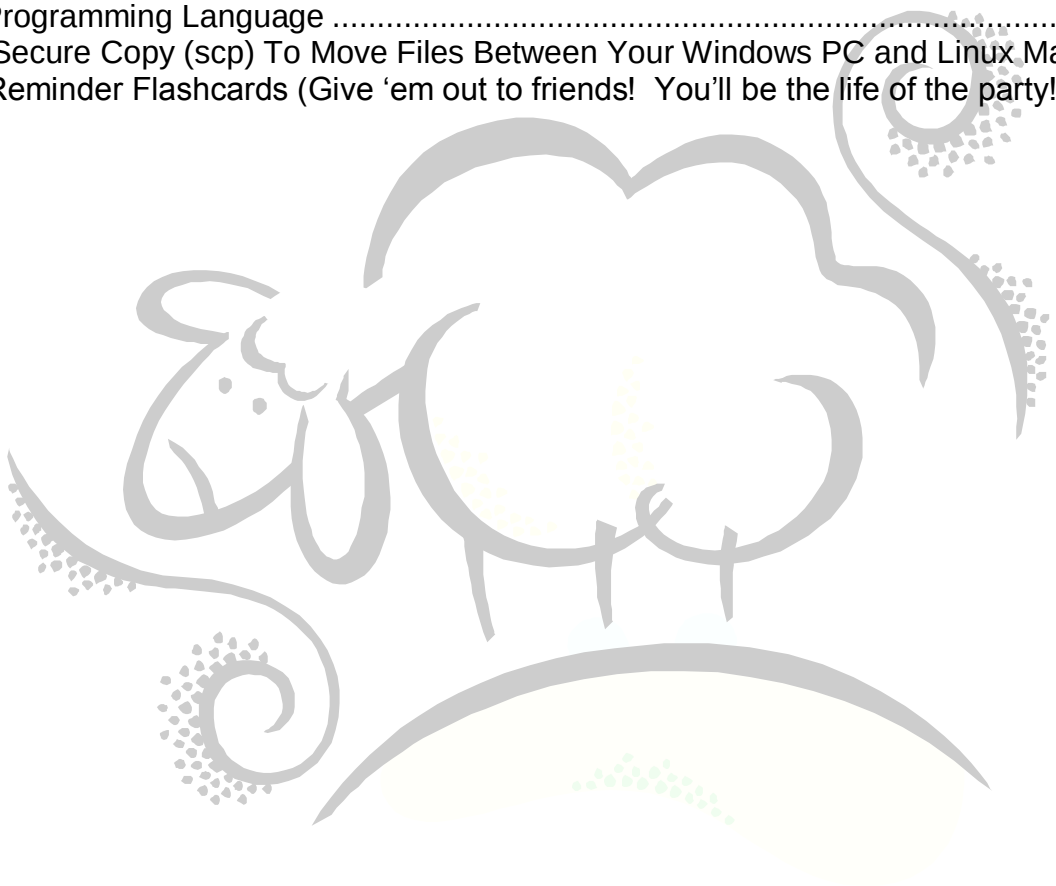
www.sheepsqueezers.com



Introduction to Linux Commands and Features

Table of Contents

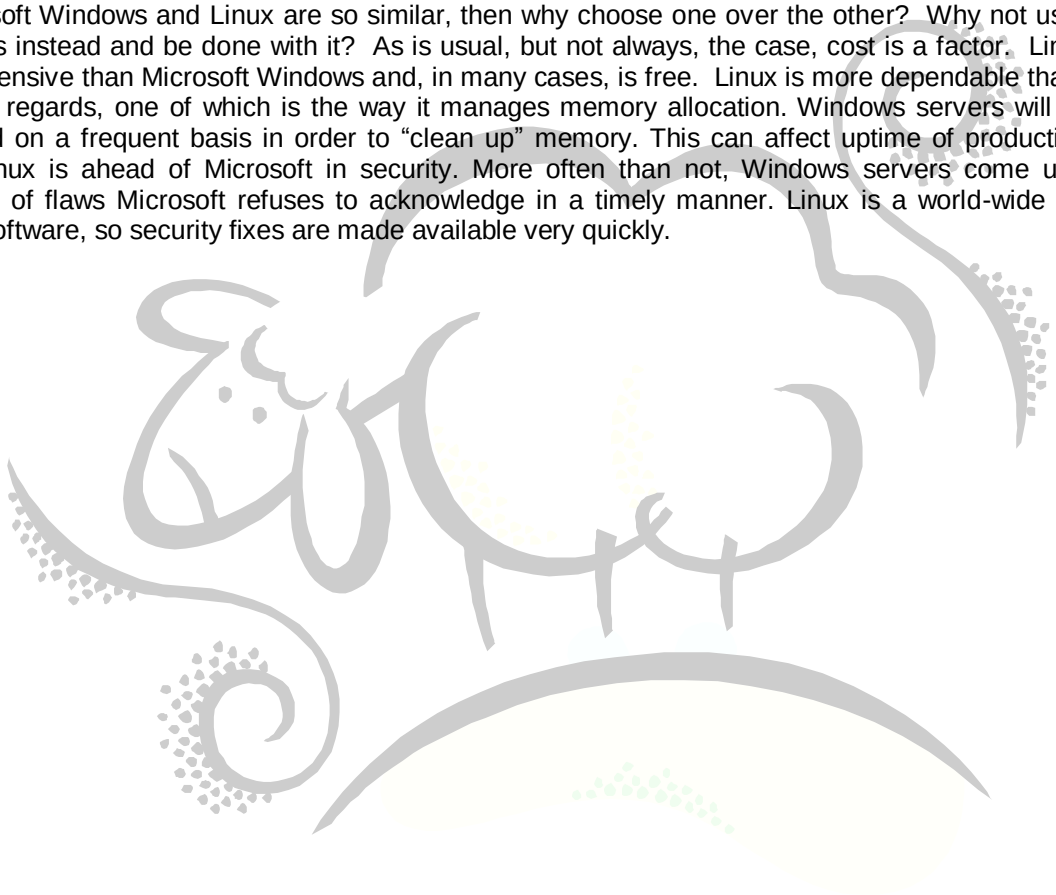


Introduction

This document is a brief introduction on Linux commands. Rather than trying to make you Linux experts, this document focuses on getting you up-and-running as fast as possible without all the gory details. There are numerous websites and books available to teach you Linux in more detail, and we mention them at the end of this document.

You can use Linux in much the same way as you use Microsoft Windows. There is a Graphical User Interface (GUI) which allows you to use graphical applications such as an editor, file manager, web access, SAS Display Manager, etc. There is also a command line interface -- much like MS-DOS -- that allows you to enter commands at a command line to copy files from one place to another, delete files, run batch jobs, etc. Almost all of the commands you execute at the command line you can do using the GUI interfaces, but many people feel that entering commands at the command line is faster than using a GUI tool. We'll show you both methods and let you make that decision yourself.

If Microsoft Windows and Linux are so similar, then why choose one over the other? Why not use Microsoft Windows instead and be done with it? As is usual, but not always, the case, cost is a factor. Linux is much less expensive than Microsoft Windows and, in many cases, is free. Linux is more dependable than Windows in many regards, one of which is the way it manages memory allocation. Windows servers will need to be rebooted on a frequent basis in order to "clean up" memory. This can affect uptime of production servers. Also, Linux is ahead of Microsoft in security. More often than not, Windows servers come under attack because of flaws Microsoft refuses to acknowledge in a timely manner. Linux is a world-wide community-based software, so security fixes are made available very quickly.



Linux Definitions (pids, directories, files, links, commands, permissions)

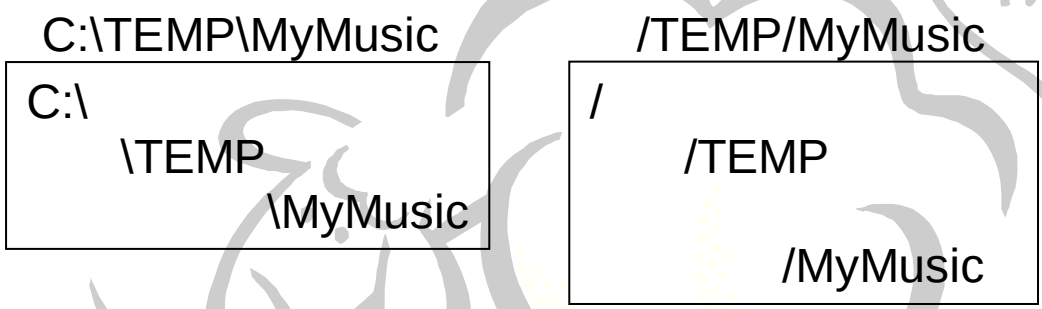
Before we get into Linux commands, we want to start with some definitions.

Whenever you run a command, either at the command prompt or by starting a GUI application, you are starting a *process,* which is a fancy name for an executing program. Each *process* is given a unique number called its *process ID*, or *PID*. Each user can have one or more running processes on a Linux machine. When a user logs into a Linux machine, a process is automatically started – called the parent process -- and any other command the user runs is a *child process* of that *parent process*.

A *file* is a collection of data that's stored on disk and that can be manipulated as a single unit based on its name.

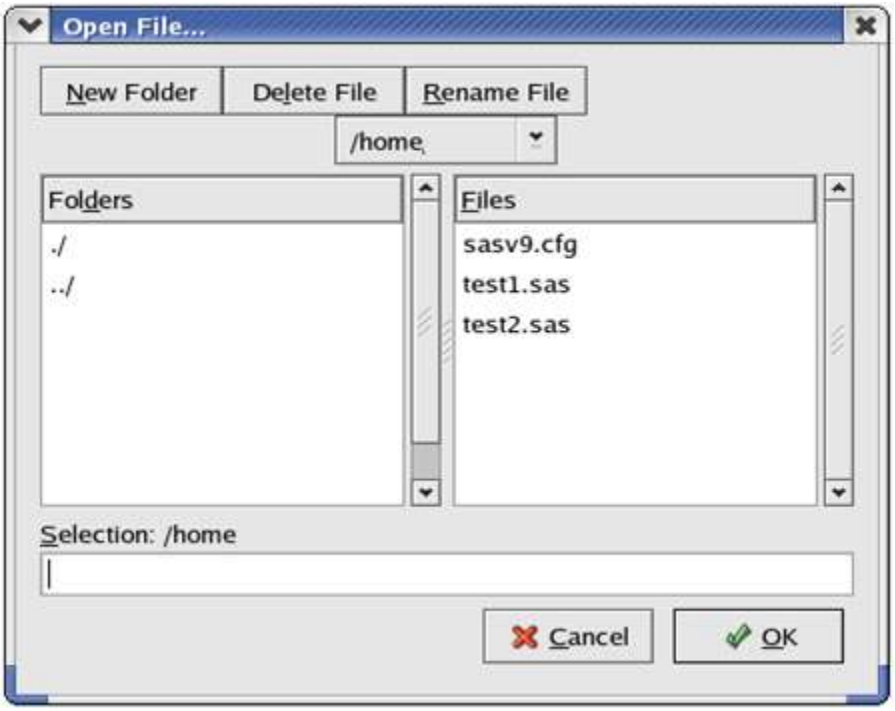
A *directory* acts as a folder for files. A directory can also contain other directories (called *subdirectories*); a directory that contains another directory is called the *parent* directory of the directory it contains.

Both Microsoft Windows and Linux file systems look like an inverted tree. For example, on Windows, you usually start at the C-Drive (or C:\). Then you can move down to a subdirectory, like C:\TEMP. And further down, say, to C:\TEMP\MyMusic. The Linux file system starts at the root of the inverted tree, but is designated as a forward slash (/) rather than C:\. Moving down to the TEMP directory, in Linux it would be designated /TEMP. And futher still to /TEMP/MyMusic. You can compare the two:



When a user logs into a Linux machine, the user is (usually) placed in his or her *home directory*; that is, a folder where the user can store his or her files, such as SAS programs, SAS datasets, text files, perl programs, etc. This home directory is usually designated `/home/`*`your-user-name`*, like `/home/bsmith`. You can create as many subdirectories as you want. You can place your SAS programs in `/home/`*`your-user-name`* or any subdirectory you decide to create. An interesting side note, in order to create a subdirectory in Linux you use the command `mkdir` *`subdirectory-name`*, which is the same command for Microsoft Windows DOS. The same goes for changing directories: `cd` *`new-directory`*.

As you move up and down subdirectories during the course of your work, the directory you are currently in is known as your *current working directory*. The current working directory is sometimes denoted by a period (**.**). The *parent directory*, or the directory one level up from the current working directory, is sometimes denoted by two periods (**..**). You can use this period notation in nearly any command you run at the command line when you want to refer to the current working directory or the parent directory. Some of the GUI tools occasionally show a single or double period in the open or save dialog box:



A *symbolic link* is a way to represent a file or subdirectory, but that file or subdirectory is actually located in a completely different location on disk. The user can place files in this subdirectory and they are redirected automatically to a completely different location. The user then doesn't have to worry about managing disk space…well…at least not too much.

Unix permissions are defined by 9 attributes for each file or directory—read, write and execute for the file ***owner***, read, write and execute for the ***group,*** and read, write and execute for ***all others (or world)***. The `ls -l` command shows the file type attribute followed by the list of permissions—`rwxrwxrwx`. Permissions NOT granted on the file will appear as a hyphen rather than a letter. For example, file with permissions `r-x------` can only be read or executed by the owner and no one else. To set permissions, see the ***chmod*** command below.

A *group* is a set of users created to share files and to facilitate collaboration. Each member of a group can work with the group's files and make new files that belong to the group. We talk more about groups later on.

A Linux *command* is a piece of software that is run at the Linux command prompt to perform an action. Following many of the Linux commands, you can have one or more *options*, usually a single dash followed by a single letter. The options modify how the command operates. For example, at the MS-DOS command prompt you can type `DIR /O-S` to display file sorted in descending size order, while in Linux you would use `ls -lS`. Here "`-lS`" is made up of two options, the `l` and the `S`.

Basic Linux Commands (pwd, ls, mkdir, rm ,mv, cd)

In this section, we show you several useful Linux commands you can run at the Linux command prompt. This list is by no means exhaustive. Before we get started, log into your Linux machine using either an X Windows session or a terminal session via Putty or other software.

Remember that when you first log into your Linux session, you are placed in your `/home/your-username` directory. You can see what directory you are in by entering the command `pwd` and hitting the Enter key. PWD stands for **p**rint **w**orking **d**irectory:

```
[bsmith@biglinux bsmith]$ pwd
/home/bsmith
[bsmith@biglinux bsmith]$
```

If you want to see all of your files, type `ls` and hit the Enter key. LS stands for **L**ist **S**tuff:

```
[bsmith@biglinux bsmith]$ ls
gskvaccine  sasdata  sas_renewal  sasuser.v91  sasv9.cfg  test1.sas  test2.sas  tests  tmp
[bsmith@biglinux bsmith]$
```

From the list above, you could never tell which name indicates a subdirectory and which indicates a file. You can make `ls` give you more information by entering the command `ls -l` (or its alias `ll` command) and hitting the Enter key. The `-l` option forces ls to return a **l**ong **l**isting:

```
[bsmith@biglinux bsmith]$ ls -l
total 24
drwxrwxr-x    2 bsmith   sasusers     4096 May 19 09:21 vaccine
lrwxrwxrwx    1 bsmith   sasusers       15 Mar 10 14:53 sasdata -> /sasdata/bsmith
drwxrwxr-x    2 bsmith   sasusers     4096 Mar 11 15:43 sasuser.v91
-rw-r--r--    1 bsmith   sasusers     2364 Mar 10 14:52 sasv9.cfg
-rw-r--r--    1 bsmith   sasusers        0 May 22 15:32 test1.sas
-rw-r--r--    1 bsmith   sasusers        0 May 23 11:55 test2.sas
drwxrwxr-x    2 bsmith   sasusers     4096 May 18 09:11 tests
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 tmp
[bsmith@biglinux bsmith]$
```

If you take a look at the first letter of the first column, you'll see the letter "d" for some of the entries. This stands for "directory". If you see a dash "-", then that row represents a file rather than a directory. If you see the letter "l", then this is a *symbolic link* to another directory, as you see above for the sasdata folder.

If you want to make a directory, say MyStuff, you use the `mkdir subdirectory-name` command. MKDIR stands for **m**a**k**e **dir**ectory. For example, type `mkdir MyStuff` at the command line and hit the Enter key. You can then use `ls -l` to show the new subdirectory:

```
[bsmith@biglinux bsmith]$ mkdir MyStuff
[bsmith@biglinux bsmith]$ ls -l
total 28
drwxr-xr-x    2 bsmith   sasusers     4096 May 24 15:35 MyStuff
lrwxrwxrwx    1 bsmith   sasusers       15 Mar 10 14:53 sasdata -> /sasdata/bsmith
drwxrwxr-x    2 bsmith   sasusers     4096 Mar 11 15:43 sasuser.v91
-rw-r--r--    1 bsmith   sasusers     2364 Mar 10 14:52 sasv9.cfg
-rw-r--r--    1 bsmith   sasusers        0 May 22 15:32 test1.sas
-rw-r--r--    1 bsmith   sasusers        0 May 23 11:55 test2.sas
drwxrwxr-x    2 bsmith   sasusers     4096 May 18 09:11 tests
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 tmp
[bsmith@biglinux bsmith]$
```

As you see from the listing above, our two test SAS programs, test1.sas and test2.sas, are still there. Let's delete one of these files. You can remove a file or files by using the `rm filename` command. RM stands for **rem**ove. Type `rm test1.sas` at the command prompt and hit the Enter key:

```
[bsmith@biglinux bsmith]$ rm test1.sas
rm: remove regular empty file `test1.sas'? y
[bsmith@biglinux bsmith]$ ls -l
total 28
drwxr-xr-x    2 bsmith   sasusers     4096 May 24 15:35 MyStuff
lrwxrwxrwx    1 bsmith   sasusers       15 Mar 10 14:53 sasdata -> /sasdata/bsmith
drwxrwxr-x    2 bsmith   sasusers     4096 Mar 11 15:43 sasuser.v91
-rw-r--r--    1 bsmith   sasusers     2364 Mar 10 14:52 sasv9.cfg
-rw-r--r--    1 bsmith   sasusers        0 May 23 11:55 test2.sas
drwxrwxr-x    2 bsmith   sasusers     4096 May 18 09:11 tests
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 tmp
[bsmith@biglinux bsmith]$
```

Notice that the operating system asks you if you really want to remove `test1.sas`. If you enter the letter "y" and hit the Enter key, the file is removed. **Note that once you remove a file, there is NO way to undelete it!!** Also note that not all Linux systems are setup to query the user when removing a file. If you feel this is appropriate for you, ask your Linux System Administrator to help you set up an alias for the `rm` command.

Let's say that you want to rename test2.sas to test1.sas. To do that you use the `mv` *`from-filename to-filename`* command. Type `mv test2.sas test1.sas` at the command prompt and hit the Enter key:

```
[bsmith@biglinux bsmith]$ mv test2.sas test1.sas
[bsmith@biglinux bsmith]$ ls -l
total 28
drwxr-xr-x    2 bsmith   sasusers     4096 May 24 15:35 MyStuff
lrwxrwxrwx    1 bsmith   sasusers       15 Mar 10 14:53 sasdata -> /sasdata/bsmith
drwxrwxr-x    2 bsmith   sasusers     4096 Mar 11 15:43 sasuser.v91
-rw-r--r--    1 bsmith   sasusers     2364 Mar 10 14:52 sasv9.cfg
-rw-r--r--    1 bsmith   sasusers        0 May 23 11:55 test1.sas
drwxrwxr-x    2 bsmith   sasusers     4096 May 18 09:11 tests
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 tmp
[bsmith@biglinux bsmith]$
```

MV stands for **mov**e. This command is not just for renaming, but for moving files from one location to another. For example, you can move your `test1.sas` program to the `/home/`*`your-username`*`/MyStuff` directory. The syntax in this case is `mv` *`from-filename to-directory`*. For example, type `mv test1.sas /home/`*`your-username`*`/MyStuff` at the command prompt and hit the Enter key:

```
[bsmith@biglinux bsmith]$ mv test1.sas /home/bsmith/MyStuff
[bsmith@biglinux bsmith]$
```

Are you completely amazed by the lack of information when the `mv` command completes? Just to make sure our file did actually make it to the sasdata subdirectory, let's move to the subdirectory. To move to another subdirectory, you use the `cd` *`subdirectory`* command. CD stands for **c**hange **d**irectory. Let's move down into the `MyStuff` subdirectory and see if the file exists:

```
[bsmith@biglinux bsmith]$ cd MyStuff
[bsmith@biglinux MyStuff]$ pwd
/home/bsmith/MyStuff
[bsmith@biglinux MyStuff]$ ls -l
total 0
-rw-r--r--    1 bsmith   sasusers        0 May 23 11:55 test1.sas
[bsmith@biglinux MyStuff]$
```

Recall above that we mentioned that two periods represent the parent directory. If you want to move one subdirectory above where you are (which would be the parent directory), you type in `cd ..` and hit the Enter key. Notice that there is a single space between the "cd" and the two periods. If you do not leave a space, Linux will complain that the command is not recognized. Let's try that now and then show the current working directory using the `pwd` command:

```
[bsmith@biglinux MyStuff]$ cd ..
[bsmith@biglinux bsmith]$ pwd
/home/bsmith
[bsmith@biglinux bsmith]$
```

You can flip back-and-forth between the directory you `cd`'d to and the directory you `cd`'d from by using the `cd -` command; that is cd followed by a single dash.

```
[bsmith@biglinux bsmith]$ pwd
/home/bsmith
[bsmith@biglinux bsmith]$ cd MyStuff
[bsmith@biglinux MyStuff]$ cd -
/home/bsmith
[bsmith@biglinux bsmith]$
```

If you just type `cd` followed by the Enter key, you will be take back to your home directory, `/home/`*`your-username`*. Also, you are probably getting tired of all of this typing. At the command prompt, you can hit the tab key while you are in the middle of typing a filename or directory, and if you have supplied enough of the letters, Linux will try to complete the filename or directory name:

```
[bsmith@biglinux bsmith]$ ll
total 28
drwxrwxr-x    2 bsmith   sasusers     4096 May 24 15:12 gskvaccine
drwxr-xr-x    2 bsmith   sasusers     4096 May 26 13:41 MyStuff
lrwxrwxrwx    1 bsmith   sasusers       15 Mar 10 14:53 sasdata -> /sasdata/bsmith
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 sas_renewal
drwxrwxr-x    2 bsmith   sasusers     4096 Mar 11 15:43 sasuser.v91
-rw-r--r--    1 bsmith   sasusers     2364 Mar 10 14:52 sasv9.cfg
drwxrwxr-x    2 bsmith   sasusers     4096 May 18 09:11 tests
drwxrwxr-x    2 bsmith   sasusers     4096 May 17 13:32 tmp
[bsmith@biglinux bsmith]$ cd MyS<HIT THE TAB KEY>
[bsmith@biglinux bsmith]$ cd MyStuff   ← Linux finished the subdirectory name for you!!
```

If you have not supplied enough of the text for Linux to complete the filename or subdirectory name, continue to hit the tab key until Linux gives you a list of matching entries:

```
[bsmith@biglinux MyStuff]$ ll test<HIT TAB KEY SEVERAL TIMES>
test1.log       test1.lst       test1.sas       test2_BACKUP.sas  test2.sas
```

If the command `ll` does not work for you, just use `ls -l` instead.

Running SAS from the Linux Command Prompt

In this section, we show you how to run SAS from the Linux command prompt. Although you might favor running SAS Display Manager, you might have to run SAS from the command prompt if you want to run SAS programs at night or on a timely basis in batch

First, make sure you are in the `MyStuff` subdirectory by typing `pwd` at the command prompt and hitting the Enter key. Next, we want to put a small piece of SAS code into the file test1.sas. You can either use the GUI Text Editor, but this is a chance for us to show you some more Linux commands. Type the following command at the command prompt and hit the Enter key:

```
echo "proc setinit noalias;" > test1.sas
```

The `echo` command just spits out – or echoes – the text in the double-quotes. The greater-than symbol says "take the output of the command appearing to the left of me and place it into what appears to the right of me". Thus, the text "`proc setinit noalias;`" is put into the file `test1.sas`. Note that if there was any text in `test1.sas` before you ran this command, it **all** would be replaced by "`proc setinit noalias;`".

You can use the `cat` command in order to see what is currently in the file test1.sas. Type cat test1.sas at the command prompt and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ echo "proc setinit noalias;" > test1.sas
[bsmith@biglinux MyStuff]$ cat test1.sas
proc setinit noalias;
[bsmith@biglinux MyStuff]$
```

In order to append to `test1.sas`, instead of using one greater-than symbol, use two greater-than symbols. At the command line, type `echo "run;" >> test1.sas` and then cat this file to see what's in it:

```
[bsmith@biglinux MyStuff]$ echo "run;" >> test1.sas
[bsmith@biglinux MyStuff]$ cat test1.sas
proc setinit noalias;
run;
[bsmith@biglinux MyStuff]$
```

Next, we want to run this SAS program at the command prompt and view any SAS Log and Listing files the program produces. In order to run a SAS program at the command prompt, type in the following and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ sas test1.sas
[bsmith@biglinux MyStuff]$
```

Do a long listing (`ls -l`) to see what the SAS program produced:

```
[bsmith@biglinux MyStuff]$ ls -l
total 8
-rw-r--r--    1 bsmith   sasusers     2131 May 24 16:29 test1.log
-rw-r--r--    1 bsmith   sasusers       27 May 24 16:27 test1.sas
[bsmith@biglinux MyStuff]$
```

Notice that our SAS program only produced a SAS Log file and no SAS Listing file. (The PROC SETINIT puts its output in the Log file and does not produce any Listing file.) In order to see the text in the Log file, type `cat test1.log` at the command prompt followed by the Enter key:

```
[bsmith@biglinux MyStuff]$ cat test1.log
1                                                          The SAS System
16:29 Tuesday, May 24, 2005

NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) 9.1 (TS1M3)
      Licensed to YOUR COMPANY, Site 0000000000.
NOTE: This session is executing on the Linux 2.4.21-27.0.4.ELsmp platform.
NOTE: SAS 9.1.3 Service Pack 2
```

```
You are running SAS 9. Some SAS 8 files will be automatically converted
by the V9 engine; others are incompatible.  Please see
http://support.sas.com/rnd/migration/planning/platform/64bit.html

PROC MIGRATE will preserve current SAS file attributes and is
recommended for converting all your SAS libraries from any
SAS 8 release to SAS 9.  For details and examples, please see
http://support.sas.com/rnd/migration/index.html

This message is contained in the SAS news file, and is presented upon
initialization.  Edit the file "news" in the "misc/base" directory to
display site-specific news and information in the program log.
The command line option "-nonews" will prevent this display.

NOTE: SAS initialization used:
      real time           0.05 seconds
      cpu time            0.05 seconds

1          proc setinit noalias;
2          run;

NOTE: PROCEDURE SETINIT used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds

Original site validation data
Site name:    'YOUR COMPANY'.
Site number:  00000000.
Expiration:   15MAR2006.
Grace Period:  45 days (ending 29APR2006).
Warning Period: 45 days (ending 13JUN2006).
System birthday:   01JAN1989.
Operating System:   LINUX    .
Product expiration dates:
---Base Product                                                  15MAR2006
---SAS/CONNECT                                                  15MAR2006
---SAS/ACCESS Interface to MYSQL                                15MAR2006

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.06 seconds
      cpu time            0.07 seconds

[bsmith@biglinux MyStuff]$
```

In some cases, you may want to pass parameters to your SAS program to control what is happening within the SAS program. To do this, you add the `-sysparm` option to the `sas` command and add your parameters in double-quotes:

```
sas -sysparm "ABC" -sysin test1.sas
```

Within your SAS program, you can capture the ABC by using the automatic SAS macro variable SYSPARM. Let's add one more line to our SAS program, run it and then cat the Log file:

```
[bsmith@biglinux MyStuff]$ echo "%put ==>&sysparm.<==;" >> test1.sas
[bsmith@biglinux MyStuff]$ cat test1.sas
proc setinit noalias;
run;
%put ==>&sysparm.<==
[bsmith@biglinux MyStuff]$ sas -nodms -sysparm "ABC" -sysin test1.sas
[bsmith@biglinux MyStuff]$ cat test1.log
   1                                                          The SAS System
   16:53 Tuesday, May 24, 2005

   NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
   NOTE: SAS (r) 9.1 (TS1M3)
```

```
      Licensed to YOUR COMPANY, Site 0000000000.
NOTE: This session is executing on the Linux 2.4.21-27.0.4.ELsmp platform.
NOTE: SAS 9.1.3 Service Pack 2

You are running SAS 9. Some SAS 8 files will be automatically converted
by the V9 engine; others are incompatible.  Please see
http://support.sas.com/rnd/migration/planning/platform/64bit.html

PROC MIGRATE will preserve current SAS file attributes and is
recommended for converting all your SAS libraries from any
SAS 8 release to SAS 9.  For details and examples, please see
http://support.sas.com/rnd/migration/index.html


This message is contained in the SAS news file, and is presented upon
initialization.  Edit the file "news" in the "misc/base" directory to
display site-specific news and information in the program log.
The command line option "-nonews" will prevent this display.

NOTE: SAS initialization used:
      real time           0.04 seconds
      cpu time            0.04 seconds

1          proc setinit noalias;
2          run;

NOTE: PROCEDURE SETINIT used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds

Original site validation data
Site name:    'YOUR COMPANY'.
Site number:  00000000.
Expiration:   15MAR2006.
Grace Period:  45 days (ending 29APR2006).
Warning Period: 45 days (ending 13JUN2006).
System birthday:   01JAN1989.
Operating System:   LINUX   .
Product expiration dates:
---Base Product                                                  15MAR2006
---SAS/CONNECT                                                  15MAR2006
---SAS/ACCESS Interface to MYSQL                                15MAR2006

3          %put ==>&sysparm.<==;
==>ABC<==
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           0.05 seconds
      cpu time            0.06 seconds

[bsmith@biglinux MyStuff]$
```

Notice that the `%put` spits out ABC.

More Linux Commands (touch, cp, mkdir, groups, du, df, gzip, head , tail, man, cat, grep)

In this section we introduce more advanced Linux commands that will send you screaming into the night.

Ensure you are in the `MyStuff` subdirectory.

If you want to create a blank file, you can use the `touch` *`filename`* command. At the command prompt, type `touch test2.sas` and hit the Enter key, then show a long listing of the files:

```
[bsmith@biglinux MyStuff]$ touch test2.sas
[bsmith@biglinux MyStuff]$ ls -l
total 8
-rw-r--r--    1 bsmith   sasusers     2173 May 24 16:53 test1.log
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
```

If you would like to show the most recently created file first, use the -t option. At the command line, type ls –lt and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ ls -lt
total 8
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
-rw-r--r--    1 bsmith   sasusers     2173 May 24 16:53 test1.log
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
[bsmith@biglinux MyStuff]$
```

Notice that the newly created file `test2.sas` appears first. If you want to sort by largest file first, use the –S option instead of the –t option.

Next, let's make a backup copy of the SAS program test2.sas and call it test2_BACKUP.sas. You use the Linux `cp` *`filename new-filename`* to make a copy. CP stands for **co**p**y**. At the command prompt, type `cp test2.sas test2_BACKUP.sas` and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ cp test2.sas test2_BACKUP.sas
[bsmith@biglinux MyStuff]$ ls -l
total 8
-rw-r--r--    1 bsmith   sasusers     2173 May 24 16:53 test1.log
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:31 test2_BACKUP.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
[bsmith@biglinux MyStuff]$
```

Next, let's make a sacrificial subdirectory called `StarTrekExtra`. At the command prompt type `mkdir StarTrekExtra` and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ mkdir StarTrekExtra
[bsmith@biglinux MyStuff]$ ls -lt
total 12
drwxr-xr-x    2 bsmith   sasusers     4096 May 24 17:33 StarTrekExtra
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:31 test2_BACKUP.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
-rw-r--r--    1 bsmith   sasusers     2173 May 24 16:53 test1.log
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
[bsmith@biglinux MyStuff]$
```

Next, in order to remove that directory, you first have to ensure that there are no files within that directory. If there are, you can use the `rm` command to remove the files. To remove a subdirectory, use the `rmdir` *`subdirectory-name`* command. RMDIR stands for **r**e**m**ove **dir**ectory. At the command prompt, type `rmdir StarTrekExtra` and hit the phasers…er…Enter key:

```
[bsmith@biglinux MyStuff]$ rmdir StarTrekExtra
[bsmith@biglinux MyStuff]$ ls -lt
total 8
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:31 test2_BACKUP.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
-rw-r--r--    1 bsmith   sasusers     2173 May 24 16:53 test1.log
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
[bsmith@biglinux MyStuff]$
```

As mentioned above, each user can be a part of one or more groups. In order to the groups you are in, type `groups` at the Linux command line and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ groups
sasusers
[bsmith@biglinux MyStuff]$
```

If you want to know the total number of bytes in the current working directory, you use the `du` command. DU stands for **d**isk **u**sage. At the command line, type `du -h .` and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ du -h .
12K     .
[bsmith@biglinux MyStuff]$
```

Notice that we are using the single period to represent the current working directory. As you see, the `MyStuff` subdirectory contains about 12 kilobytes of data.

If you would like to see the current disk used and free space across your Linux machine, use the `df -h` command. DF stands for **d**isk **f**reespace. At the command line, type `df -h` and hit the Enter key:

```
[bsmith@biglinux bsmith]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/cciss/c0d0p1      34G  4.9G   27G  16% /
none                  3.5G     0  3.5G   0% /dev/shm
/dev/cciss/c0d1p1     201G  2.1G  189G   2% /home
/dev/cciss/c1d1p1     234G   33M  222G   1% /saswork
/dev/cciss/c1d0p1     401G  113G  269G  30% /sasdata
[bsmith@biglinux bsmith]$
```

This means that the `/sasdata` drives are 30% used (or 70% free). The `/saswork` drives are 1% used (99% free) and the home directories (`/home`) are 2% used (98% free).

Sometimes we can't remember where we put a file. In order to help find where we put that file, we use the Linux find command. The syntax is `find . -name "filename"`. Note that we are using the single period to represent the current working directory, but the find command automatically searches any subdirectories below the working directory. Let's move back up to our `/home/your-username` directory. We can do that by issuing the `cd /home/your-username` command, or we can just type `cd` and hit the Enter key. Either way we'll be back in Kansas! Next, type `find . -name "test2_BACKUP.sas"` and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ cd
[bsmith@biglinux bsmith]$ pwd
/home/bsmith
[bsmith@biglinux bsmith]$ find . -name "test2_BACKUP.sas"
./MyStuff/test2_BACKUP.sas
[bsmith@biglinux bsmith]$
```

Notice that the ever-present single period rears its ugly head again. You can think of `./MyStuff/test2_BACKUP.sas` as really meaning `/home/bsmith/MyStuff/test2_BACKUP.sas`. Why? If you take a look at the result of the `pwd` command, you'll see that the current working directory is `/home/bsmith`, and as we mentioned above, the single period is just short-hand for the current working directory. Instead of using the single period in the find command, you could actually put in any subdirectory name you want: `find /home/bsmith -name "test2_BACKUP.sas"`, so you are not limited to just the current working directory and below:

```
[bsmith@biglinux bsmith]$ find /home/bsmith -name "test2_BACKUP.sas"
/home/bsmith/MyStuff/test2_BACKUP.sas
[bsmith@biglinux bsmith]$
```

Also, you can use the asterisk (*) in the double-quoted string after the `-name` option. The asterisk has the same meaning in Linux as it does in Microsoft Windows DOS: zero, one, or more characters or numbers. So, if we wanted to find all our SAS programs, we could type `find . -name "*.sas"` and hit the Enter key:

```
[bsmith@biglinux bsmith]$ find . -name "*.sas"
./MyStuff/test1.sas
./MyStuff/test2.sas
./MyStuff/test2_BACKUP.sas
[bsmith@biglinux bsmith]$
```

If you would like to compress a file to save space, just like WinZip on Microsoft Windows, you can use the `gzip` *`filename`* command. Let's move back to the `MyStuff` subdirectory (`cd MyStuff` and hit Enter), and let's compress the test1.log file:

```
[bsmith@biglinux MyStuff]$ gzip test1.log
[bsmith@biglinux MyStuff]$ ls -l
total 8
-rw-r--r--    1 bsmith   sasusers     1017 May 24 16:53 test1.log.gz
-rw-r--r--    1 bsmith   sasusers       48 May 24 16:53 test1.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:31 test2_BACKUP.sas
-rw-r--r--    1 bsmith   sasusers        0 May 24 17:25 test2.sas
[bsmith@biglinux MyStuff]$
```

Note that the file `test1.log` was replaced with `test1.log.gz`. The `.gz` indicates that this file was compressed with the `gzip` command. If you would like to decompress this file, type `gzip -d test1.log.gz` and you will get back `test1.log`. The `-d` option means to decompress. Go ahead and decompress `test1.log.gz` now.

If you would like to see just the last 10 lines of, for example, a SAS Log file while the SAS program is running, you can use the `tail filename` command. Type `tail test1.log` at the command prompt and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ tail test1.log
---SAS/CONNECT                                                    15MAR2006
---SAS/ACCESS Interface to MYSQL                                  15MAR2006

3          %put ==>&sysparm.<==
==>ABC<==
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time               0.05 seconds
      cpu time                0.06 seconds

[bsmith@biglinux MyStuff]$
```

In a similar fashion to `tail`, there is the `head` *`filename`* command which shows you the top 10 lines of the *`filename`*. For both `head` and `tail`, you can add an optional parameter to increase the number of lines shown. Normally, both `head` and `tail` show 10 lines, but `tail -200` *`filename`* will show 200 lines from the bottom of *`filename`*, while `head -50` *`filename`* will show the top 50 lines from *`filename`*.

Recall that when you use the `cat` *`filename`* command, the entire contents of the file is spewed out onto the screen. This makes it very difficult to see anything if the file is very large. Instead, you can use the `more` *`filename`* command to view the file one page at a time. At the command prompt, type `more test1.log` and hit the Enter key. To move to the next page, hit the spacebar; to scroll down one line at a time, hit the Enter key; to quit, hit the letter `q`.

At this point, all of the commands and their options are swimming around in your head. Built into Linux is a help-like functionality which can be accessed by the command `man` *`Linux-command`*. For example, if you type in `man cat` at the command prompt and hit the Enter key, you will see this information:

```
[bsmith@biglinux MyStuff]$ man cat
CAT(1)                                FSF                                CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION] [FILE]...

DESCRIPTION
       Concatenate FILE(s), or standard input, to standard output.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonblank output lines

       -e     equivalent to -vE

       -E, --show-ends
              display $ at end of each line

       -n, --number
              number all output lines

       -s, --squeeze-blank
              never more than one single blank line

       -t     equivalent to -vT

       -T, --show-tabs
              display TAB characters as ^I

       -u     (ignored)

       -v, --show-nonprinting
              use ^ and M- notation, except for LFD and TAB

       --help display this help and exit

       --version
              output version information and exit

       With no FILE, or when FILE is -, read standard input.

AUTHOR
       Written by Torbjorn Granlund and Richard M. Stallman.

REPORTING BUGS
       Report bugs to <bug-coreutils@gnu.org>.

COPYRIGHT
       Copyright © 2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full  documentation for cat is maintained as a Texinfo manual.  If
       the info and cat programs are properly installed at your site, the com-
       mand
```

As you can see in the DESCRIPTION section, the `cat` command has several options we haven't even explored yet! Type `q` to quit out.

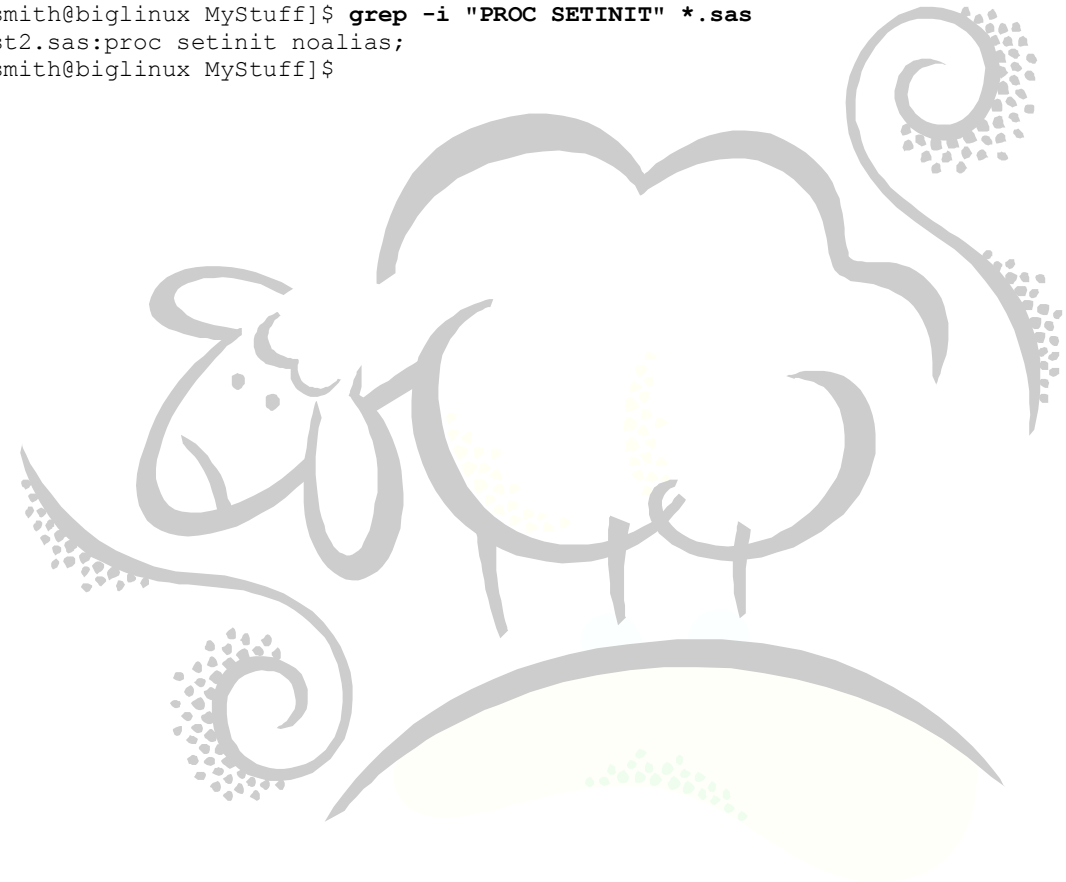
Occasionally, you'll transfer text files – like SAS programs, flat-files containing data, etc. -- from your Windows PC to your Linux machine. In some cases, you may need to run the `dos2unix` command to convert from MS-DOS to Linux. I tend to run this command as a matter of course when transferring text files to the Linux machine. The command is `dos2unix` *`filename`*.

If you would like to search for a string of text in your files, you can use the `grep` command. They syntax is `grep "search-text"` *`filename`*. For example, if you want to search for "proc setinit" in all your SAS programs, you would type the following at the command line and hit the Enter key:

```
[bsmith@biglinux MyStuff]$ grep "proc setinit" *.sas
test2.sas:proc setinit noalias;
[bsmith@biglinux MyStuff]$
```

You'll notice that grep returns the filename containing the search text. By default, grep is case sensitive, but you can disable that using the –i option:

```
[bsmith@biglinux MyStuff]$ grep -i "PROC SETINIT" *.sas
test2.sas:proc setinit noalias;
[bsmith@biglinux MyStuff]$
```



Submitting a SAS Job to Batch

At this point, we'd like to continue our discussion about running SAS jobs on a Linux machine. As mentioned above, you can run any SAS program from the Linux command prompt like this:

```
sas -nodms -sysin SAS-program-name.sas
```

or more easily

```
sas SAS-program-name.sas
```

If your SAS program runs for a long time, Linux will not return a command prompt to you. You'll just have to wait until the SAS program finishes before doing any more work. In this case, you are running your job in the *foreground*.

Linux has the ability to run a job in the *background* as a *batch job* – a program running behind the scenes -- and return the command prompt back to you immediately. In order to do this, just place an ampersand (`&`) at the end of your `sas` command and hit the Enter key. The ampersand indicates to Linux to run the SAS program in the background and return the command prompt to the user immediately. For example, let's rerun our `test1.sas` program in the `MyStuff` subdirectory in the background:

```
[bsmith@biglinux MyStuff]$ sas test1.sas &
[1] 1610
[bsmith@biglinux MyStuff]$
[1]+  Done                    sas test1.sas
[bsmith@biglinux MyStuff]$
```

Note that when you submit the SAS job like this, you may have to hit the Enter key a few times in order for your command prompt to show up.

Notice the number 1610? This is your process ID (or PID) for the SAS program you just submitted to batch. Each time you submit a job to batch, you are given a PID. Knowing the PID allows you to kill a batch job, if need be. You can see the status of your running batch job(s), by using the process status command `ps` at the command line. Type `ps` at the command line and hit enter:

```
[bsmith@biglinux MyStuff]$ ps
  PID TTY          TIME CMD
 5762 pts/5    00:00:00 bash
 1610 pts/5    00:00:00 sas
 5820 pts/5    00:00:00 ps
[bsmith@biglinux MyStuff]$
```

Notice the row for PID 1610; do you see the word `sas` at the end of that line? This is your SAS program running in batch. If you decide you want to kill this running process – maybe you determined your SAS code is wrong (highly unlikely, though…tee-hee…) – you can kill it by using the `kill` *PID* command followed by two Enter key hits:

```
[bsmith@biglinux MyStuff]$ ps
  PID TTY          TIME CMD
 5762 pts/5    00:00:00 bash
 5847 pts/5    00:00:03 sas
 5859 pts/5    00:00:00 ps
[bsmith@biglinux MyStuff]$ kill 5847
[bsmith@biglinux MyStuff]$
[1]+  Exit 5                  sas test1.sas
[bsmith@biglinux MyStuff]$
```

At this point, your job is killed. If you have trouble killing a running batch job, contact your system administrator.

NOTE: Be very careful what you kill! If you kill the process named "bash", you may be logged out!

If you are planning on submitting a long-running job and then logging out, Linux will complain that you have a job running. You can prevent this by using the keyword `nohup` before your SAS command:

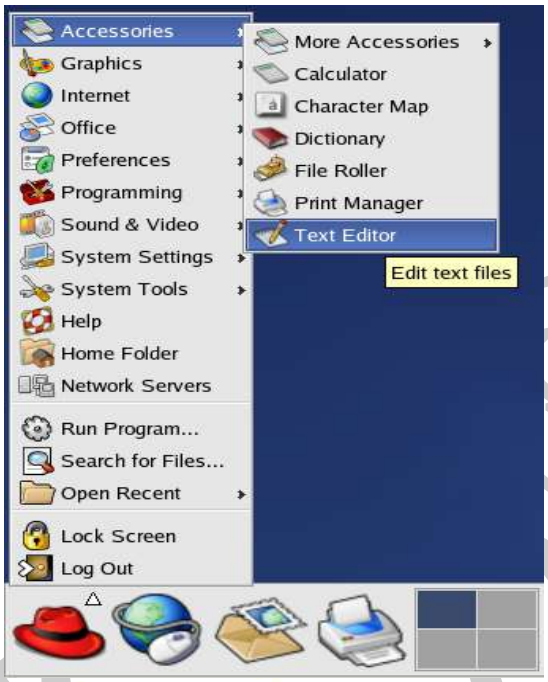
```
nohup sas SAS-program-name.sas &
```

NOHUP stands for **no h**ang**up** and will not "hang up" your running SAS programs once you have logged out. This means that you can submit a large SAS job to batch and then log out and go home secure in the fact that your program is actually running.

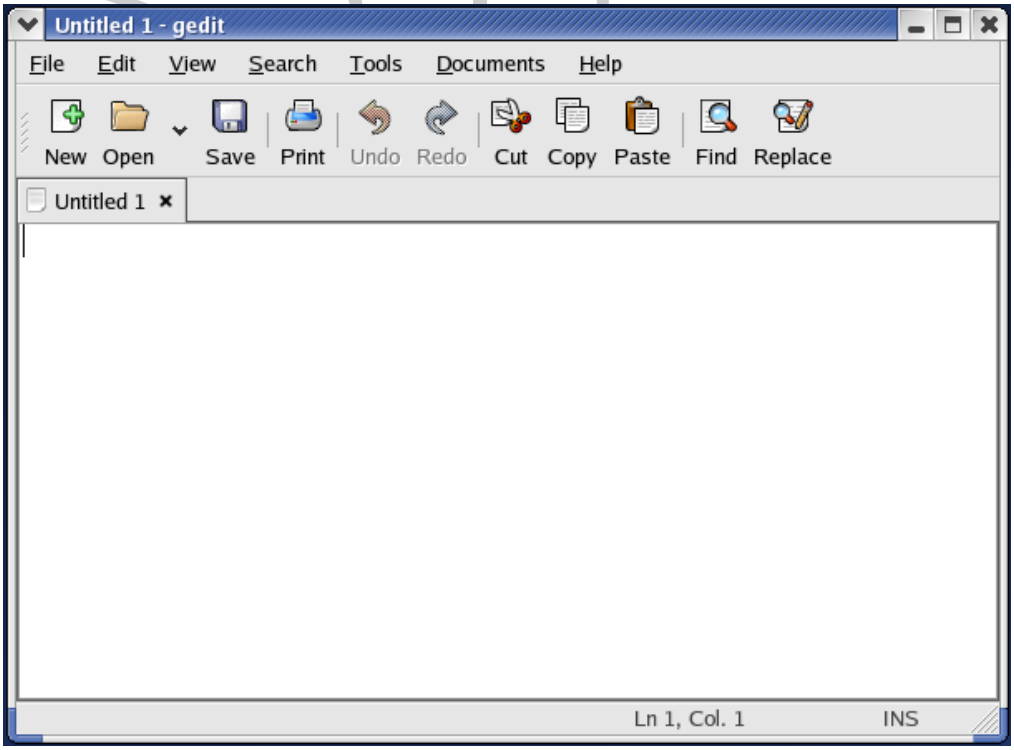
Note that when using the `nohup` command, Linux will create/append information about running jobs to a file called `nohup.out` in your current working directory. If there are any operating system errors, they can be found in `nohup.out`.

Editing Text Files

Recall that we created a SAS program using the `echo` command above. This is not very user-friendly. Just like Notepad for Microsoft Windows, Linux gives you the Text Editor *gedit*. Depending on your Linux version, you may have to use another available text editor. To start the Text Editor, click on the Red Hat icon at the bottom left of the screen, then click on Accessories, then on Text Editor:



Here is what you will see:



The Text Editor works just like Notepad: New, Open, Save, etc. Let's open up our test1.sas program. Click on the folder icon above the word Open, navigate to `/home/your-username/MyStuff` and then select `test1.sas`. Whatever is in test1.sas, delete it and replace it with the following SAS code:

```
options nocenter;
run;

data testdata;
 do i=1 to 10000;
  z=i*2;
  output;
 end;
run;

proc univariate data=testdata;
 var z;
run;
```

Next, save this file as `test3.sas` (don't forget the `.sas`!!) by clicking on File…Save As..., entering `test3.sas` for the filename. Next, go back to your Terminal window and ensure that test3.sas is there in the MyStuff directory by doing an `ls`. Next, run this SAS program in batch mode. Once the program is completed – you may have to hit the Enter key a few times for the completion message to come up – open the SAS Log `test3.log` in the Text Editor and review for errors, then open up the SAS Listing `test3.lst`. Next, to quit out, click on File…Quit.

So, you have two choices to edit your SAS programs: using SAS Display Manager or using the Text Editor along with a terminal session so that you can run your SAS programs in batch.

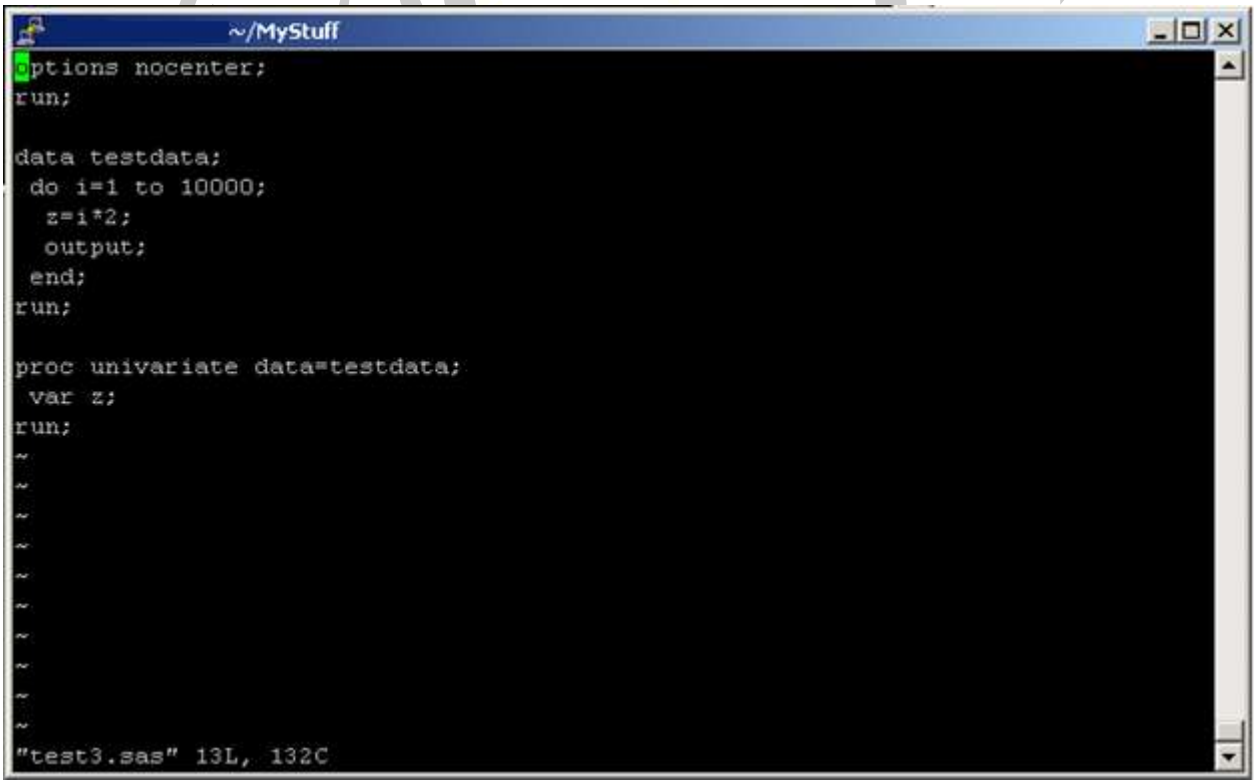
Some of you may have used the old MS-DOS editor EDIT and may wonder if there is something similar for the Linux terminal session. There is, and it is called `vi` – pronouced "v" then "i". While `vi` is the lowest common denominator editor across all Linux/Unix platforms, its finger-twisting sequences to insert, delete and move lines around could prevent you from ever playing the piano again and, thus, makes it a bad choice for novice Linux users such as yourself. Nevertheless, I have included information on `vi` just in case you're curious.

Using the VI Editor

As mentioned above, the `vi` editor can be used in a terminal session to edit files. Here's a list of useful `vi` commands. This list is by no means exhaustive!

Command Name	Command Description
i	Insert Text (puts you in insert mode)
ESC (hit the Escape button)	Brings you out of insert mode
x	Delete a single character
dw	Delete a single word
o (small letter oh)	Places a blank line below cursor and puts you in insert mode
O (capial letter oh)	Places a blank line above cursor and puts you in insert mode
ESC followed by :wq	Saves the file and quits out returning you to the command prompt
d$	Delete to the end of the line
Dd	Deletes one single line
CTRL-f	Moves down a page (toward the bottom)
CTRL-b	Move up a page (toward the top)
0 (number zero, not letter oh)	Brings you to column one on the current line
Shift-R	Overwrite mode (use ESC to come out)
u	Undo (undoes what you just did)

Let's try using the commands above to edit a file. Recall that we created the file test3.sas. Let's use vi to edit this SAS program. At the command prompt, type `vi test3.sas` and hit the Enter key. You will see something like this:



At this point, you can use your arrow keys to move up, down, left, and right just as in the Text Editor. Move your cursor to the blank line just above "`data testdata;`" and insert the SAS comment /* Create test data */. You

do this by using your arrow keys to get to the blank line, then you hit the “i” key, type in the comment, and then hit the ESC button to bring you out of insert mode.

Next, let’s insert a blank line just above this comment so that there is a nice space between the comment and the run; following the options statement. All you have to do it hit a capital letter O followed by the ESC button. Why did we hit the ESC button? When you use o or O you are placed in insert mode – that is, you can type text – so to get out of insert mode, we hit the ESC button.

Next, let’s change that 10000 to 50000. Move your cursor to the left of 10000 and hit Shift-R. At this point, you are in replace mode meaning you can overwrite the text that is there. Type 50000 then hit the ESC button. If you made a mistake, you can hit the letter u to undo what you just did.

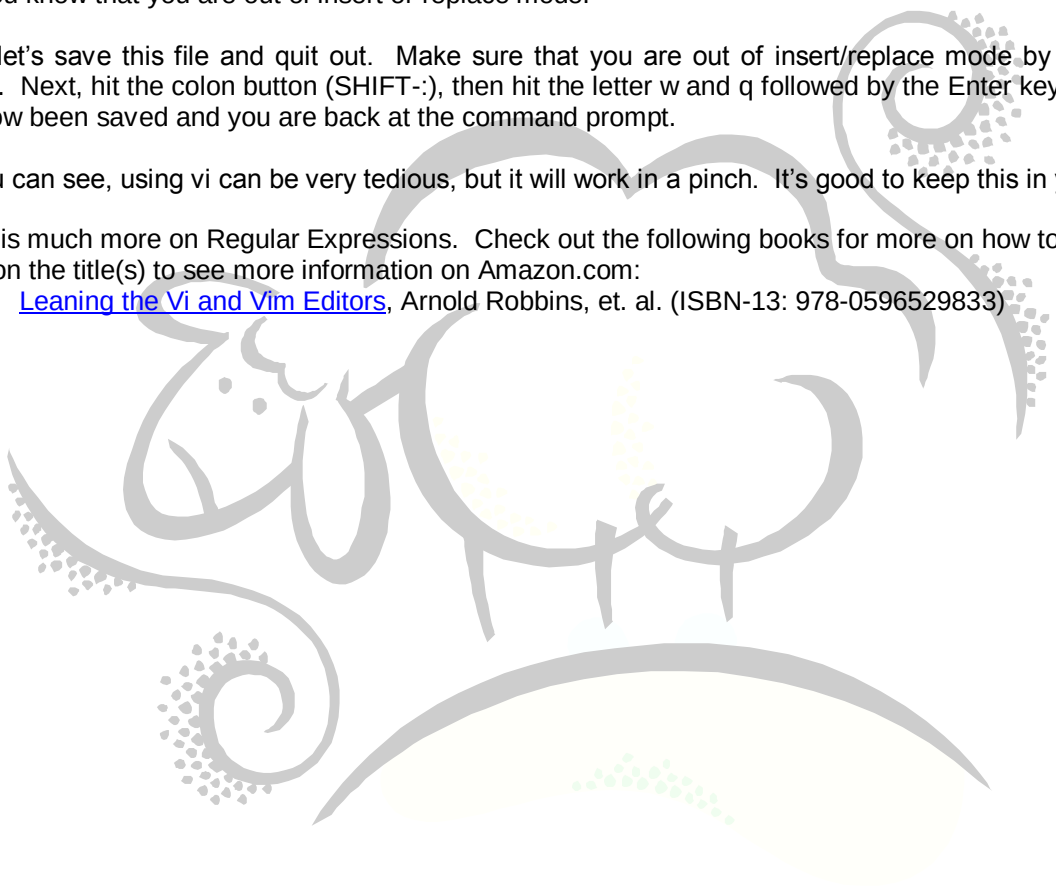
Hey, have you noticed yet that the word INSERT appears at the bottom of the screen when you hit the letter i, o, or O; and the word REPLACE appears when you hit SHIFT-R? If you hit the ESC button, these words disappear
and you know that you are out of insert or replace mode.

Next, let’s save this file and quit out. Make sure that you are out of insert/replace mode by hitting the ESC button. Next, hit the colon button (SHIFT-:), then hit the letter w and q followed by the Enter key. Your program has now been saved and you are back at the command prompt.

As you can see, using vi can be very tedious, but it will work in a pinch. It’s good to keep this in your toolbox!

There is much more on Regular Expressions. Check out the following books for more on how to automate tasks (click on the title(s) to see more information on Amazon.com:

- Leaning the Vi and Vim Editors, Arnold Robbins, et. al. (ISBN-13: 978-0596529833)



Regular Expressions

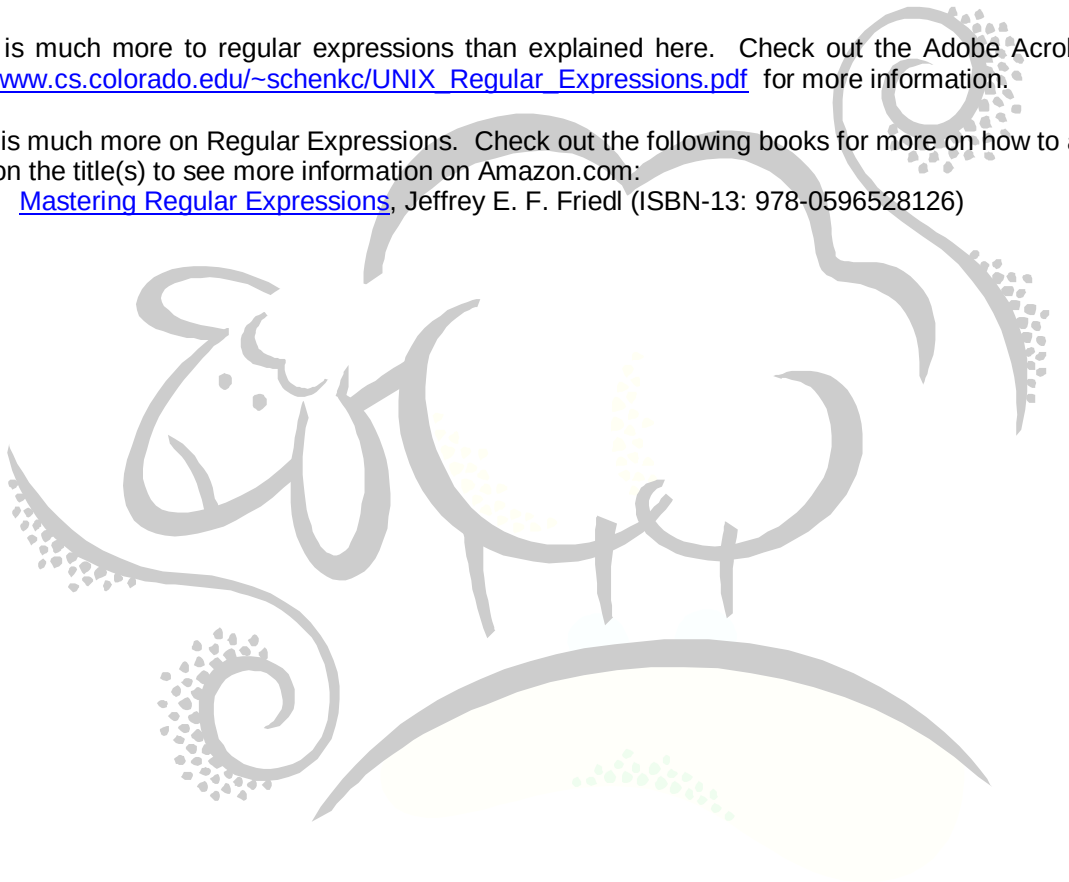
In many places throughout this document, we have used the asterisk (*) as a wildcard; that is, a representation for any amount of letters and numbers. For example, if we had three SAS programs `test1.sas`, `test2.sas` and `test3.sas` in our `MyStuff` subdirectory, we could do `ll test*.sas` to get a long listing of all SAS programs beginning with the word `test` and ending with `.sas`. A *regular expression* is a formula for matching strings that follow some pattern. In our example, the pattern is `test*.sas`. Regular expressions may initially look confusing, but once you are used to all of the wildcards – known as metacharacters in regular expression lingo – you'll be using them all the time.

Suppose, instead of matching all SAS programs beginning with `test` and ending in `.sas`, we want to match only SAS programs beginning with `test` followed by either a `1` or `2` followed by `.sas`. Here is what the regular expression would look like: `test[12].sas`. The square brackets will match any one character at a time: `test1.sas`, `test2.sas`. Note that you can place letters and/or numbers in the square brackets: test[12ab].sas will match `test1.sas`, `test2.sas`, `testa.sas`, and `testb.sas`.

There is much more to regular expressions than explained here. Check out the Adobe Acrobat Reader file http://www.cs.colorado.edu/~schenkc/UNIX_Regular_Expressions.pdf for more information.

There is much more on Regular Expressions. Check out the following books for more on how to automate tasks (click on the title(s) to see more information on Amazon.com:

- Mastering Regular Expressions, Jeffrey E. F. Friedl (ISBN-13: 978-0596528126)



Stream Editor

With a SAS macro, you can automatically replace text by the parameters you pass to the macro program. If you have created a "template" SAS program -- or other text file -- you can use the Linux `sed` command to replace key words or numbers that must be changed to create a final program or text file. SED stand for **s**tream **ed**itor. You pass `sed` a substitution string (`from-string` and `to-string`) and a filename and it looks for `the from-string` in filename and replaces it with `to-string`. Your original file is not damaged. For example, in test3.sas we loop around 50000 times. Suppose we want to loop around 15000 time instead. Do this at the command line:

```
sed "s/50000/15000/g" test3.sas
```

The 50000 is the `from-string` and the 15000 is the `to-string`. The letter `s` means substitute. The `g` means global substitution.

```
[bsmith@biglinux MyStuff]$ sed "s/50000/15000/g" test3.sas
options nocenter;
run;

/* Create test data */
data testdata;
 do i=1 to 15000;
  z=i*2;
  output;
 end;
run;

proc univariate data=testdata;
 var z;
run;
[bsmith@biglinux MyStuff]$
```

Notice that 50000 was replaced by 15000 above. If you want to save your new program, use the greater-than symbol to save to a new file:

```
sed "s/50000/15000/g" test3.sas > test3A.sas
```

You can find much more at http://sed.sourceforge.net/grabbag/tutorials/ or see the book references in the AWK section below.

Linux Scripting

There may be times you need to submit multiple SAS jobs, or rename multiple SAS programs, or make backup copies of multiple SAS datasets, but you don't want to type them all in by hand because it is tedious and error prone. This section explains how to create a Linux *script* which performs multiple actions just like a Microsoft DOS BAT file. For example, using the Text Editor or vi, create a text file called `doit` with the following three lines in it:

```
cp test3.sas test3_BACKUP1.sas
cp test3.sas test3_BACKUP2.sas
cp test3.sas test3_BACKUP3.sas
```

in order to "run" this script, you have to let Linux know that `doit` is a script. You do this by using the `chmod` command to indicate that the file `doit` is a script: `chmod +x doit`. Next, to run this script, at the command line, type in `./doit` and hit the Enter key. Do a long listing and you will see the three backup files are there.

There is **much** more on Linux scripting. Check out the following books for more on how to automate tasks (click on the title(s) to see more information on Amazon.com:

- Linux Shell Scripting with Bash, Ken Burtch (ISBN-13: 978-0672326424)
- Linux Command Line and Shell Scripting Bible, Richard Blum (ISBN-13: 978-0470251287)

AWK Programming Language

AWK is a small programming language we use mostly for creating many Linux commands at once or for chopping out columns of text data. For example, if you have 10 SAS programs you want to make backup copies of but you don't want to actually type them in by hand, you can use awk to create the commands for you and then save them to a script which you can run later on. For example:

```
ll test[123].sas | awk '{printf("cp %s %s_BACKUP\n",$9,$9);}'
```

Let's take this one piece at a time. Here is what `ll test[123].sas` looks like:

```
[bsmith@biglinux MyStuff]$ ll test[123].sas
-rw-r--r--    1 bsmith   sasusers      118 May 26 13:41 test1.sas
-rw-r--r--    1 bsmith   sasusers       27 May 27 08:57 test2.sas
-rw-r--r--    1 bsmith   sasusers      166 May 27 11:31 test3.sas'
[bsmith@biglinux 'MyStuff]$
```

Notice that the text test1.sas, test2.sas and test3.sas are in the 9th column of each long listing line. We'll use this fact next.

The symbol `|` is called a *pipe* and indicates that the *output* coming from the command on the left is passed as *input* to the command on the right. This is different from the greater-than or double-greater-than symbols since these just replace or add to the end of a file. You can actually string as many pipes as you want. An `awk` program always looks like this:

```
awk '{ stuff }'
```

That is, the command `awk` is followed by an apostrophe, the left curly brace, then the programming stuff, followed by the right curly brace followed by an apostrophe. Next, let's look at the stuff. In this case, stuff is the `printf()` function followed by a semicolon. Just like SAS, each line of an AWK program ends in a semicolon.

Inside the parentheses of the printf() function are two sections delimited by a comma:

```
"text-to-output",column-you-want-separated-by-commas.
```

In this case, the text-to-output is the copy command `cp` and the column-you-want is the 9th column because that column contains our SAS program names. Let's look more closely at the text-to-output:

```
"cp %s %s_BACKUP\n"
```

Recall the general syntax for the copy command is `cp from-filename to-filename`. If you were to make the backup copy by hand, you would do this: `cp test1.sas test1.sas_BACKUP`, and so on. As you can see, I am replacing test1.sas with a `%s`. The `%s` just means a string is going to be placed at that point. Notice that `%s` appears twice, once for the first `test1.sas` and again for `test1.sas_BACKUP`. Just before the last double-quote, we have `\n` which is Linux short-hand for *go to the next line*.

Next, for each `%s` in our text-to-output, we must have a corresponding column name, indicated by a dollar-sign followed by the column number. In our case, we have two `$9` because we have two `%s`. In AWK, columns of data are indicated by a dollar sign follwed by the column number. By default, columns are delimited by spaces. If your data is delimited by something else, like a tilde (~), then you can run the AWK command with the `-Fdelimiter` option. For example, try this out and see if you can guess what comes out:

```
echo "ABC~DEF~GHI" | awk -F~ '{printf("%s\n",$2);}'
```

If you guessed `DEF`, then you would be correct!
There is much more on AWK scripting. Check out the following books for more on how to automate tasks (click on the title(s) to see more information on Amazon.com:

- sed & awk, Ken Burtch (ISBN-13: 978-1565922259)
- Effective awk Programming, Arnold Robbins (ISBN-13: 978-0596000707)

Using Secure Copy (scp) To Move Files Between Your Windows PC and Linux Machine

Many of you may be aware of a way to transfer files called FTP (File Transfer Protocol). Sadly, FTP is not the most secure way of transferring data between two machines which is why many companies have decided to use SCP (Secure Copy) instead.

In order to use SCP to transfer files from your Windows PC to your Linux machine, start a DOS command prompt by clicking on Start…Programs…Accessories…Command Prompt. Change directories to the directory containing the data you want to transfer to your Linux machine. The general syntax for scp is:

```
scp from-filename your-username@biglinux:/home/your-username/to-filename
```

This will copy *from-filename* from your Windows PC to your /home/*your-username* subdirectory on your Linux machine. Once you hit the Enter key, scp will request your Linux machine password. Enter the password and then hit enter. Scp will give you a percent complete status as it's transferring the file(s) to your Linux machine.

```
C:\>scp sqlnet.log bsmith@biglinux:/home/bsmith/sqlnet.log
bsmith@biglinux's password: <ENTER YOUR PASSWORD HERE>
sqlnet.log                                    100% 5021     4.9KB/s   00:00

C:\>
```

To transfer from your Linux machine to your Windows PC, reverse the syntax:

```
scp your-username@biglinux:/home/your-username/from-filename to-filename
```

Note that if you do not have scp installed on your Windows PC, email your Help Desk. They may be able to provide more assistance.

Linux Reminder Flashcards (Give 'em out to friends! You'll be the life of the party!)

Pwd Shows the current directory.	**ls** Lists files in directory.	**ls -l** Long listing of files in directory. Same as ll.
Mkdir directory-name Creates a directory directory-name.	**rm *filename*** Removes the file *filename*.	**mv *from-file to-file*** Renames from-file to to-file.
Cd *subdir* Changes directory to *subdir*.	**cd ..** Changes to parent directory one level above.	**cd -** Return to previous directory.
echo "text" Prints out text.	**sas *sas-program.sas*** Runs a SAS program.	**> *filename*** (Greater-than) Replace text in *filename* with output of previous command.
>> *filename* (Greater-thans) Append text in *filename* with output of previous command.	**cat *filename*** Dumps *filename* to screen.	**touch *filename*** Creates a blank file *filename*.
Ls -lt Long listing sorted descending date order.	**cp *from-filename to-filename*** Copies *from-filename* to *to-filename*.	**rmdir *directory-name*** Removes directory *directory-name*
groups Shows all Linux groups you belong to.	**du -h .** Disk usage for current working directory.	**cd** Returns you back to the /home/*username* directory.
df -h Show disk usage across system	**find *subdir* -name "*search*"** Searches *subdir* down looking for files matching *search*.	**gzip *filename*** Compresses *filename*. Can use wildcard (*) here as well.
tail -*# filename* Prints # lines from the bottom of *filename*.	**head -*# filename*** Prints # lines from the top of *filename*.	**more *filename*** Prints *filename* to screen one page at a time.
man *Linux-command* Shows manual page for *Linux-command*.	**dos2unix *filename*** Converts DOS text in *filename* to Linux text format.	**grep "*search*" *filename*** Searches *filename* for *search*. Use -i for case insensitive.
& (Ampersand) End commands with & to submit to batch.	**kill *PID*** Kills a running batch job identified by process *PID*.	**nohup *command* &** No hangup mode when submit to batch. Can logout safely.
vi *filename* Begins VI text editor with file *filename*.	**sed "s/*from*/*to*/" *filename*** Replaces *from* with *to* in *filename*. Prints to screen.	**chmod +x *filename*** Makes *filename* an executable script.
scp *from to* Secure copy between machines.		

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at comments@sheepsqueezers.com.