**Shells**

The shell sits between you and the kernel, acting as a command interpreter. It reads your terminal input and translates the commands into actions taken by the system. The shell is analogous to command in DOS. When you log into the system you are given a default shell. When the shell starts up it reads its startup files and may set environment variables, command search paths, and command aliases, and executes any commands specified in these files.

The original shell was the Bourne shell, sh. Every Linux platform will either have the Bourne shell, or a Bourne compatible shell available. It has very good features for controlling input and output, but is not well suited for the interactive user. To meet the latter need the C shell, csh, was written and is now found on most, but not all, Linux systems. It uses C type syntax, the language Unix is written in, but has a more awkward input/output implementation. It has job control, so that you can reattach a job running in the background to the foreground. It also provides a history feature which allows you to modify and repeat previously executed commands.

The default prompt for the Bourne shell is $ (or #, for the root user). The default prompt for C shell is %.

Numerous other shells are available from the network. Almost all of them are based on either sh or csh with extensions to provide job control to sh, allow in-line editing of commands, page through previously executed commands, provide command name completion and custom prompt, etc. Some of the more well known of these may be on your favorite Linux system: the Korn shell, ksh, by David Korn and the Bourne Again Shell, bash, from the Free Software Foundations GNU project, both based on sh, the T-C shell, tcsh, and the extended C shell, cshe, both based on csh. Below we will describe some of the features of sh and csh so that you can get started.

**Built-in Commands**

The shells have a number of built-in, or native commands. These commands are executed directly in the shell and don’t have to call another program to be run. These built-in commands are different for the different shells.

**sh**

For the Bourne shell some of the more commonly used built-in commands are:

: null command

. source (read and execute) commands from a file

case case conditional loop

cd change the working directory (default is $HOME)

echo write a string to standard output

eval evaluate the given arguments and feed the result back to the shell

exec execute the given command, replacing the current shell

exit exit the current shell

export share the specified environment variable with subsequent shells

for for conditional loop

if if conditional loop

pwd print the current working directory

read read a line of input from stdin

set set variables for the shell

test evaluate an expression as true or false

trap trap for a typed signal and execute commands

umask set a default file permission mask for new files

unset unset shell variables

wait wait for a specified process to terminate

while while conditional loop

**csh**

For the C shell the more commonly used built-in functions are:

alias assign a name to a function

bg put a job into the background

cd change the current working directory

echo write a string to stdout

eval evaluate the given arguments and feed the result back to the shell

exec execute the given command, replacing the current shell

exit exit the current shell

fg bring a job to the foreground

foreach for conditional loop

glob do filename expansion on the list, but no "\" escapes are honored

history print the command history of the shell

if if conditional loop

jobs list or control active jobs

kill kill the specified process

limit set limits on system resources

logout terminate the login shell

nice command lower the scheduling priority of the process, command

nohup command do not terminate command when the shell exits

set set a shell variable

setenv set an environment variable for this and subsequent shells

stop stop the specified background job

umask set a default file permission mask for new files

unalias remove the specified alias name

unset unset shell variables

while while conditional loop

**Environment Variables**

Environmental variables are used to provide information to the programs you use. You can have both

global environment and local shell variables. Global environment variables are set by your login

shell and new programs and shells inherit the environment of their parent shell. Local shell variables

are used only by that shell and are not passed on to other processes. A child process cannot pass a

variable back to its parent process.

The current environment variables are displayed with the "env" or "printenv" commands. Some

common ones are:

• DISPLAY The graphical display to use, e.g. nyssa:0.0

• EDITOR The path to your default editor, e.g. /usr/bin/vi

• GROUP Your login group, e.g. staff

• HOME Path to your home directory, e.g. /home/frank

• HOST The hostname of your system, e.g. nyssa

• IFS Internal field separators, usually any white space (defaults to tab, space

and <newline>)

• LOGNAME The name you login with, e.g. frank

• PATH Paths to be searched for commands, e.g. /usr/bin:/usr/ucb:/usr/local/bin

• PS1 The primary prompt string, Bourne shell only (defaults to $)

• PS2 The secondary prompt string, Bourne shell only (defaults to >)

• SHELL The login shell you’re using, e.g. /usr/bin/csh

• TERM Your terminal type, e.g. xterm

• USER Your username, e.g. frank

Many environment variables will be set automatically when you login. You can modify them or define

others with entries in your startup files or at any time within the shell. Some variables you might want

to change are PATH and DISPLAY. The PATH variable specifies the directories to be automatically

searched for the command you specify. Examples of this are in the shell startup scripts below.

You set a global environment variable with a command similar to the following for the C shell:

% setenv NAME value

and for Bourne shell:

$ NAME=value; export NAME

You can list your global environmental variables with the env or printenv commands. You unset them

with the unsetenv (C shell) or unset (Bourne shell) commands.

To set a local shell variable use the set command with the syntax below for C shell. Without options

set displays all the local variables.

% set name=value

For the Bourne shell set the variable with the syntax:

$ name=value

The current value of the variable is accessed via the "$name", or "${name}", notation