Compiled by Aluizio using the book UNIX IN A NUTSHELL, Arnold Robbins, O'Reilly Ed., 4th edition, 2005, ISBN 0-596-10029-9.

UNIX SHELL

The shell is the user interface. Similar when using an equipment that you have the main panel (interface) to operate it, but inside it runs more complicate internal functions. Original shells were developed before the Graphical User Interfaces (GUIs). They are harder to learn, but after you have mastered them, you will find that you can accomplish infinity tasks not possible with the GUI only. The shell is just a program that allows the system to understand your commands.

Purposes of the Shell:

1. Interactive Use: system waits you to type the commands in the Unix prompt. Commands can include special symbols that let you abbreviate filename or redirect input and output.
2. Customization of your Unix session: a Unix shell defines variables to control the system behavior of the session.
3. Programming: shells provide a set of special (built-in) commands tha let you create programs called shell scripts, which are useful for executing a series of individual commands.

Shell Choices:

Use the command *more /etc/passwd* to know what shell you are using:

/bin/sh	Bourne Shell (most popular shell in use today)
/bin/bash	Bash Shell
/bin/ksh	Korn Shell
/bin/csh	C shell or Tenex C Shell (system dependent)
/bin/tcsh	Tenex C shell (Ocean Mapping Group applications)

Example: % **more /etc/passwd**

root:x:0:0:root:/root:/bin/bash
aluizo:x:1000:1000:aluizio,,,:/home/aluizio:/bin/tcsh

Use the command ***exec tcsh*** to change from the original Bourne shell to the Tenex C shell.

Common Features to all shell types:

Symbol	*Action*
>	Redirect output.
>>	Append to a file.
<	Redirect input.
<<	“Here” document (redirect input).
|	Pipe output.
&	Run process in background.
;	Separate commands on same line.
~	Home directory symbol.
*****	Match any character(s) in file name.
?	Match single character in file name.
[ ]	Match any characters enclosed.
( )	Execute in subshell.
{ }	Expend elements in list.
` `	Substitute output of enclosed command.
“ ”	Partial code (allows variable and commands expansion).
' '	Full quote, no expansion.
****	Quote following character.
$var	Use value for variable.
$$	Process ID.
$0	Command name.
$n	nth argument ($0 \leqslant n \leqslant 9$).
$*	All arguments as simple words.
#	Begin comment.
bg	Background execution.
break	Break from loop statements.
cd	Change directory.

Symbol	*Action*
continue	Resume a program loop.
echo	Display output.
eval	Evaluate arguments.
exec	Execute a new shell.
fg	Foreground execution.
history	List previous comments.
jobs	Show active jobs.
kill	Terminate running job.
shift	Shift positional parameters.
suspend	Suspend a foreground job (such as a shell created by *su*).
time	Time a comment.
umask	Set default file permissions for new files.
unset	Erase variable or function definitions.
wait	Wait for a background job to finish.

Shell Differing Features:

bash	*tcsh*	*Action*
$	**$**	Prompt.
>!	**>!**	Force redirection.
	>>!	Force append.
>& file	**>& file**	Combine stdout and stderr.
` `	**` `**	Substitute output of enclosed command.
$ ( )		Substitute output of enclosed command (preferred form).
$HOME	**$home**	Home directory.
var=value	**set var=value**	Variable assignment.
export var=val	**setenv var val**	Set environmnet variable.
${nn}		More than nine args can be referenced.
“$@”		All args are separate words.

bash	***tcsh***	***Action***
$#	**$#argv**	Number of arguments.
$?	**$status**	Exit status.
$!		Last background Process ID.
$-		Current options.
. file	**source file**	Read commands in file.
alias x=y	**alias x y**	Name X stands for Y.
case	**switch/case**	Choose alternatives.
popd/pushd	**popd/pushd**	Switch directories.
done	**end**	End a loop statement.
esac	**endsw**	End case or switch.
exit[n]	**exit[(expr)]**	Exit with a status.
for/do	**foreach**	Loop through values.
echo -E	**glob**	Ignore echo scapes.
hash	**hashstat**	Display hashed commands (tracked aliases).
hash *cmds*	**rehash**	Rememember command locations.
hash -r	**unhash**	Forget command locations.
history	**history**	List previous commands.
fc -s	**!!**	Redo previous commands.
fc -s *str*	**!***str*	Redo command that starts with *str*.
fc -s x=y [cmd]	**!cmd:s/x/y/**	Edit command, then execute.
if ((i==5))	**if ($i==5)**	Sample if statement.
fi	**endif**	End if statement.
ulimit	**limit**	Set resource limits.
pwd	**dirs**	Print working directories.
read	**$<**	Read from standard input.
trap INTR	**onintr**	Ignore interrupts.
unalias	**unalias**	Remove aliasis.
until/do		Begin until loop.

bash	*tcsh*	*Action*
while/do	**while**	Begin while loop.

TCSH SHELL (An Extended C Shell)

This is the shell used for the Ocean Mapping Group applications. The C shell was so named because many of its programming constructions and symbols resamble those of the C programming language.

Special file: ~/.cshrc

The file .cshrc is executed at each instance of shell startup. The **Ocean Mapping Group has its own .cshrc file that has to be copied to your home directory (/home/username)**. After that, use the command: **source /home/username/.cshrc** to be able to start using the OMG binary files.

Filename Metacharacters:

Matacharacters	*Meaning*
*****	Match any string of zero or more characters. % ls new* (match new and new.l)
?	Match any single character. % cat ch? (match ch9 but NOT ch10).
[abc...]	Match any one of the enclosed characters. Also can use hyphen for a range of values (a-z, A-Z, 0-9). % vi [D-R]* (match files that begin with uppercase D through R).
[^abc...]	Match any character NOT enclosed as above.
{abc, xxx, ...}	Expand each comma-separeted string inside braces. The strings need not match actual filenames. % ls {ch, app}? expand, then match ch1, ch2, app1, app2). % mv info{,.old} (expands to mv info info.old).
~	Home directory for the current user. % ~tom (change to tomś home directory).
~name	Home directory of user name.
= n	The nth entry in the directory stack, counting from zero.
= -	The last entry in the directory stack.
^pattern	Matches anything that pattern does not match.To work correctly pattern must

Matacharacters	*Meaning*
	contain *, ?, or [...]. Should NOT contain {...} or ~. % ls ^a* (list nonmatching filenames like bb and cc. But, don't list files like aa).

Quoting:

The following files have special meaning to the tcsh shell.

Characters	*Description*
;	Command separator.
&	Background execution.
( )	Command grouping.
|	Pipe.
*** ? [ ] ~ ^**	Filename metacharacters.
{ }	String expasion characters (usually don't require quoting).
< > & !	Redirection symbols.
! ^	History substitution, quick substitution.
**“ ' **	Used in quoting other characters.
`	Command substitution.
$	Variable substitution.
space tab newline	Word separators.

The following characters can be used for quoting:

CASE 1 **“ ”**	Every between “ and ” is taken lierally except for the following characters, which keep their special meaning: $ Variable substitution will occur. ` Command substitution will occur. “ The end of the double quote. \ Escape next character. ! The history character. newline The newline character
CASE 2 **' '**	Everything between ' and ' is taken literally except for: ! (history), another ' and newline.
CASE 3 ****	The character following a \ is taken literally. Used within “ ” to escape “, $, ` and newline. Used within ' ' to escape itself, spaces or newlines. Always needed to escape a ! (history) character.

Examples:
% echo 'Single quotes “protect” double quotes'
Single quotes protect double quotes

% echo “Don't double quotes protect simble quotes too?”
Don't double quotes protect simble quotes too?

% echo “You have `ls|wc -l`files in `pwd`”
You have 43 files in /home/bob

% echo The value of \$x is $x
The value of $x is 100

Command Forms:

Command	*Action*
cmd &	Execute command in the backgroung.
cmd1; cmd2	Command sequence.
(cmd1; cmd2)	Treat cmd1 and cmd2 as a command group.

Command	*Action*
cmd1 | cmd2	Pipe. Use output from cmd1 as input of cmd2.
cmd1 && cmd2	Execute cmd1, and if it succeds, execute cmd2.
cmd1 || cmd2	Execute either cmd1 or (if cmd1 fails) cmd2.

Redirection Forms:

File descriptor	*Name*	*Common abbreviation*	*Typical default*
0	Standard input	stdin	keyboard
1	Standard output	stdout	screen
2	Standard error	stderr	screen

The usual input source or output destination can be changed with the redirection commands listed in the following sections:

Command	*Action*
cmd > file	Send output of cmd to file (overwrite). % cat part1 > book (Copy part1 to book).
cmd > ! file	Same as proceding, even if noclobber* is set.
cmd >> file	Send output from cmd to file (append). % cat part2 part3 >> book (append parts 2 and 3 to the end of file book).
cmd >> ! file	Same as proceding, even if noclobber is set.
cmd < file	Take input for cmd from file.

* Noclobber is the variable set to disable the redirection of outputs to an existing file, prevents accidental destruction of files.

Variables:

tcsh shell provide an extensive quantity of variables (argv, path, prompt, shell, status, tty, user, etc) and operators (assignent, arithmetic, bitwise, logical and comparison).

The variables can be set in one of the two ways:

- set var = value (assigning a value)
- set var (turning on the variable)

Some valuable **variables substitution** values useful to undersand OMG scripts are listed:

Variable	*Description*
${var}	The value of variable var.
${#var}	The number of words in var.
$#	The namber of arguments.
$0	Name of the program.
$*	All arguments on the command line
$?	Status value.
$<	Read a line from standard input.
$$	Process number of current shell.
$_	Text of the command line of the last command executed.
$!	Process ID number of last background process started by the shell.

Command Line Editing:

Previous executed commands are stored in a history list. The easiest way to take advantage of the command history is to use the arrow keys.

Key	*Description*
↑ UP ARROW	Previous command.
↓ DOWN ARROW	Next command.
← LEFT ARROW	Move left in the command line.
→ RIGHT ARROW	Move to the right in the command line.

Other very useful key is the TAB. When typing commands and arguments such as name of directories and files, you can start typing the first words, then press TAB to fill the end of the words. It speeds up the typing process, and the most important, diminush typing mistakes.

To display the history list, Type *% history* or *% h* (alias predefined in the OMG .cshrc file) and the last commands are listed in the screen, like presented below:

```
History list:   498  23:11   cd aluizio/
                499  23:11   ls -la
                500  23:11   ls -la .c*
```

Useful Commands:

Command	***Description***
! string	Execute the most recent command that starts with string.
! N	Execute the command number N in the history list.
! ? string ?	Most recent command that contains string.
! ? string ? %	Most recent command argument that contains string.

Job Commands:

Command	***Description***
bg	Put a job in the background.
fg	Put a job in the foreground.
jobs	List active jobs.
kill	Terminate a job.
notify	Notify when a background job terminates.
stop	Suspend a background job.
CTRL + Z	Suspend a foreground job.

To specify the job, the following strings can be used:

Job-arguments	*Description*
%n	Job number n.
%s	Job whose command line starts with string s.
%?s	Job whose command line contains string s.
%	Current job.
%-	Previous job.

Built-in Commands:

SCRIPTS are very useful to execute repeated commands. In order to understand how they are written, you must study the built-in commands. The list below presents some built-in commands found in the OMG scripts:

Built-in Command	*Description*
@ variable = expression	Assign the value of the arithmetic expression to variable.
#	Ignore all text that follows on the same line.
# ! /bin/tcsh -f	Used as the first line of a script to invoke the named shell (here the tcsh)
alias [name [command]]	Assign name as the shorthand name, or alias, for command.
bg [jobIDs]	Put the current job or the jobIDs in the background. The job-arguments can also be used here.
break	Resume execution following the *end* command of the nearest enclosing *while* or *foreach*.
case pattern:	Identify a pattern in a *switch*.
cd [options] [dir]	Change working directory.
continue	Resume execution of nearest enclosing *while* or *foreach*.
default	Label the default case in the switch.
echo	Write string to standard output.
else	Reversed word for interior of *if ... endif* statement.
end	Reversed word that ends a *foreach* or *while* statement.
exit (0)	Exit the shell script with the status. (0 means success).

Built-in Command	***Description***
fg [jobIDs]	Bring the current job or the jobIDs to the foreground.
foreach name (wordlist) **command** **end**	Assign variable name to each value in wordlist and execute commands between *foreach* and *end*.
if	Begin a conditional statement.
kill [options] IDs	Terminate each specified ID or job ID.
set [option] variable = value	Set variable to value to value.