15. UNIX Shells and Environments

Lab 15.1 - Utilities

The UNIX system includes a family of several hundred utility programs, often referred to as *commands*. These utilities perform functions, which are universally required by users.

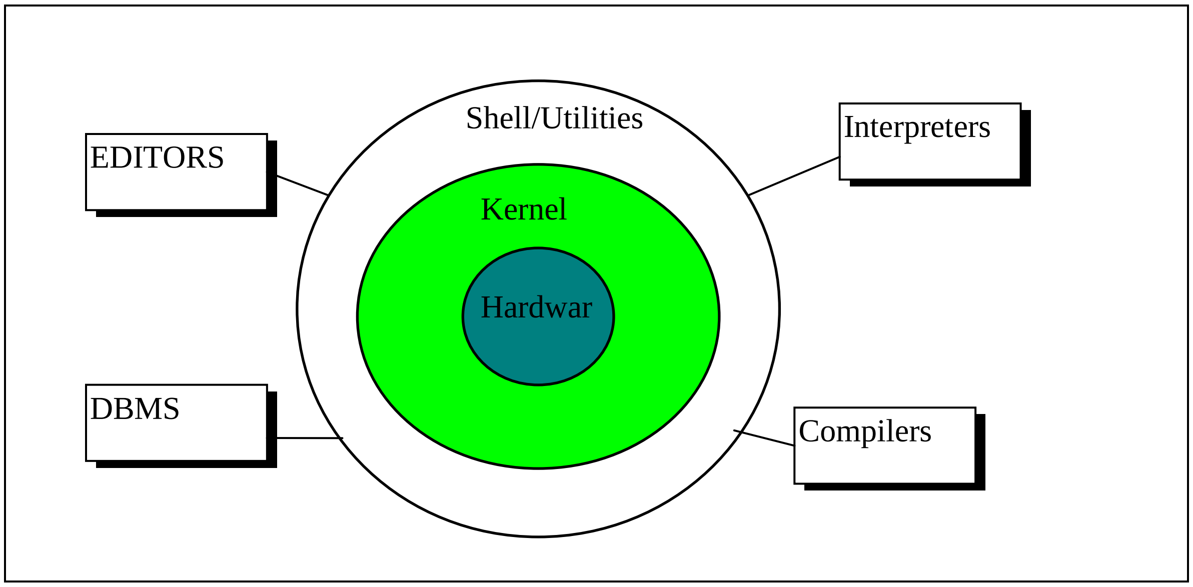


Figure 15.1: UNIX Environment

Lab 15.2 - Shell

The shell is a command interpreter that acts as an interface between users and the operating system. When you enter a command at a terminal, the shell interprets the command and calls the program you want. There are three popular shells in use today:

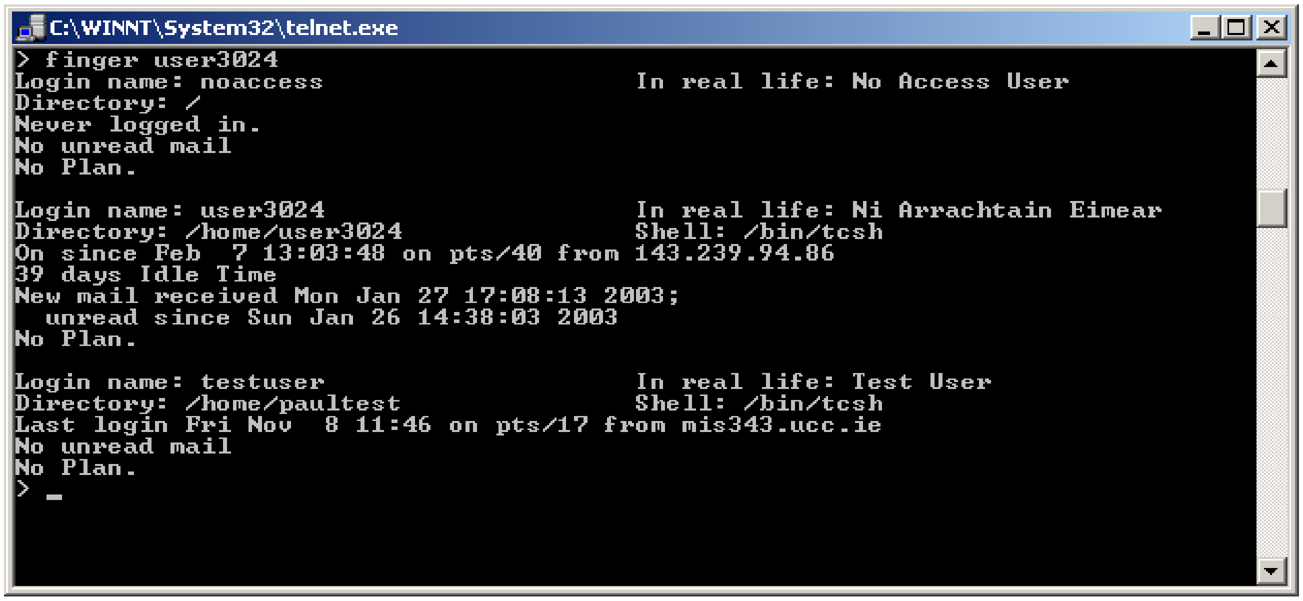
1) C shell.
2) Bourne shell.
3) Korn shell.
4) TC shell
5) Bourne again shell

To identify which shell you are currently using type the following inserting your name instead of yourname:

Lab 15.3 - Finding out which shell you are using

```
finger user3998
```

for example:



C:\WINNT\System32\telnet.exe

```
> finger user3024
Login name: noaccess                    In real life: No Access User
Directory: /
Never logged in.
No unread mail
No Plan.

Login name: user3024                    In real life: Ni Arrachtain Eimear
Directory: /home/user3024               Shell: /bin/tcsh
On since Feb  7 13:03:48 on pts/40 from 143.239.94.86
39 days Idle Time
New mail received Mon Jan 27 17:08:13 2003;
  unread since Sun Jan 26 14:38:03 2003
No Plan.

Login name: testuser                    In real life: Test User
Directory: /home/paultest               Shell: /bin/tcsh
Last login Fri Nov  8 11:46 on pts/17 from mis343.ucc.ie
No unread mail
No Plan.
> _
```

Figure 15.2: Using the finger command to check shell

This shows that the T shell is being used.

Lab 15.4 - Displaying environment variables

For the Bourne shell use the command **set** to display the value of environment variables.

For the C and TC shell use the command **printenv** or **env** to display the value of environment variables.

If the list of variables is so long that it scrolls off the screen and you want to display the information one screenful at a time, pipe the output through a pager. For example:

The command

```
env | more
```

displays the information one screenful at a time.

Information about which shell you are using is held in the SHELL environment variable. The command

```
echo $SHELL
```

displays the value of this variable

You can identify which shell you are presently using from the last part of the pathname.

Pathname	**Shell**
/.../sh	Bourne shell
/.../csh	C shell
/.../tcsh	TC shell
/.../ksh	Korn shell
/.../bash	Bourne Again Shell

Lab 15.5 - Changing environment variables

To make a temporary change to the value of an environment variable enter the command:

setenv VARNAME - ***value_of_variable***

This value will remain until you logout from the system or exit from the shell you are working in. By convention, the names of environment variables are given in UPPER CASE.

To make lasting changes to the value of an environment variable:

1. Use your editor to open the .cshrc (.tcshrc) and add the line:

```
setenv VARNAME  - value_of_variable
```

2. Save the file and leave the editor.

3. Enter the command:

```
source .cshrc (.tcshrc)
```

This adds the new value for the variable to your environment

Lab 15.6 - Switch to using another shell

You can switch to another shell for the remainder of your login session. To do this enter the shell command name at the system prompt. For example:

```
ksh
$
```

This switches you from your current shell to the Korn shell.

Example of changing the login shell with the chsh command

To change your login shell to the Korn shell (ksh):

```
chsh
   Changing login shell for sarah.
   Old shell name: /bin/csh
   New shell: ksh
```

This changes the user's login shell from the C shell (csh) to the Korn shell (ksh).

If you mistype the name of the shell or specify a shell that is not available on your system:

- you will be given a list of all the shells that are available on your system.

or

- a message similar to the following will be displayed.

name is unacceptable as a new shell

Lab 15.7 - Filename completion with the TC shell

To complete a file or directory name press the TAB key.

```
ls -l on<TAB>line_help/
```

If the filename is not completed enter more characters until the filename can be uniquely identified.
Directories are shown by the trailing / character.

Lab 15.8 - List possible names

To list all possible names of files and directories starting with co use CTRL-D. (Press and hold down the CTRL key and type a D).

```
ls co<CTRL-D>
```

Directories are shown by the trailing / character.

Lab 15.9 - Filename completion with the C shell

To turn on filename completion set the shell variable filec by adding the line

```
set filec
```

to your C shell startup file.

To complete a file or directory name press the ESC key.

```
ls -l on<ESC>line_help
```

If the filename is not completed enter more characters until the filename can be uniquely identified.

Directories are shown by the trailing / character.

Lab 15.10 - Why change your shell

The UNIX shell is most people's main access to the UNIX operating system and as such any improvement to it can result in considerably more effective use of the system, and may even allow you to do things you couldn't do before. The primary improvement most of the new generation shells give you is increased speed. They require fewer key strokes to get the same results due to their completion features, they give you more information (e.g. showing your directory in your prompt, showing which files it would complete) and they cover some of the more annoying features of UNIX, such as not going back up symbolic links to directories.

Lab 15.11 - Shell features

This table below lists most features that I think would make you choose one shell over another. It is **not** intended to be a definitive list and does **not** include every single possible feature for every single possible shell. A feature is only considered to be in a shell if in the version that comes with the operating system, or if it is available as compiled directly from the standard distribution. In particular the C shell specified below is that available on SUNOS 4.*, a considerable number of vendors now ship either tcsh or their own enhanced C shell instead (they don't always make it obvious that they are shipping tcsh.

	Sh	csh	ksh	bash	tcsh
Job control	N	Y	Y	Y	Y
Aliases	N	Y	Y	Y	Y
Shell functions	Y(1)	N	Y	Y	N

"Sensible" Input/Output redirection	Y	N	Y	Y	N
Directory stack	N	Y	Y	Y	Y
Command history	N	Y	Y	Y	Y
Command line editing	N	N	Y	Y	Y
Vi Command line editing	N	N	Y	Y	Y(3)
Emacs Command line editing	N	N	Y	Y	Y
Rebindable Command line editing	N	N	N	Y	Y
User name look up	N	Y	Y	Y	Y
Login/Logout watching	N	N	N	N	Y
Filename completion	N	Y(1)	Y	Y	Y
Username completion	N	Y(2)	Y	Y	Y
Hostname completion	N	Y(2)	Y	Y	Y
History completion	N	N	N	Y	Y
Fully programmable Completion	N	N	N	N	Y
Mh Mailbox completion	N	N	N	N(4)	N(6)
Co Processes	N	N	Y	N	N
Builtin artithmetic evaluation	N	Y	Y	Y	Y
Can follow symbolic links invisibly	N	N	Y	Y	Y
Periodic command execution	N	N	N	N	Y
Custom Prompt (easily)	N	N	Y	Y	Y
Sun Keyboard Hack	N	N	N	N	N
Spelling Correction	N	N	N	N	Y
Process Substitution	N	N	N	Y(2)	N
Underlying Syntax	sh	csh	sh	sh	csh
Freely Available	N	N	N(5)	Y	Y
Checks Mailbox	N	Y	Y	Y	Y
Tty Sanity Checking	N	N	N	N	Y
Can cope with large argument lists	Y	N	Y	Y	Y
Has non-interactive startup file	N	Y	Y(7)	Y(7)	Y
Has non-login startup file	N	Y	Y(7)	Y	Y
Can avoid user startup files	N	Y	N	Y	N
Can specify startup file	N	N	Y	Y	N
Low level command redefinition	N	N	N	N	N
Has anonymous functions	N	N	N	N	N
List Variables	N	Y	Y	N	Y
Full signal trap handling	Y	N	Y	Y	N
File no clobber ability	N	Y	Y	Y	Y
Local variables	N	N	Y	Y	N

Lexically scoped variables	N	N	N	N	N
Exceptions	N	N	N	N	N

Key to the table above.

Y	Feature can be done using this shell.
N	Feature is not present in the shell.
F	Feature can only be done by using the shells function mechanism.
L	The readline library must be linked into the shell to enable this Feature.

Notes to the table above

1. This feature was not in the orginal version, but has since become almost standard.
2. This feature is fairly new and so is often not found on many versions of the shell, it is gradually making its way into standard distribution.
3. The Vi emulation of this shell is thought by many to be incomplete.
4. This feature is not standard but unoffical patches exist to perform this.
5. A version called 'pdksh' is freely available, but does not have the full functionality of the AT&T version.
6. This can be done via the shells programmable completion mechanism.
7. Only by specifing a file via the ENV environment variable.
8.

Lab 15.12 - How to change your shell

If you ever look at a UNIX manual page it will say that to change your shell use *chsh* or *passwd -s*; unfortunately it often isn't as simple as this, since it requires that your new shell is recognized as a valid shell by the system and at present many systems do not recognize the newer shells (the normal selection is, */bin/sh*, */bin/csh* and possibly */bin/ksh*). You are thus left with having to do some sort of fudge, changing your effective login shell without changing your official entry in /etc/passwd. You may also be left with the problem that there isn't a compiled binary on your system , so you will have to get hold of the shell's source and compile it yourself (Its generally best to ask around to see if anyones done this

already, since it isn't that easy). Once done you should add in code to your old shells login file so that it overlays your official login shell with your new shell (remember to add the login flags to the command line, and with csh/tcsh ensure that the overlay doesn't happen recursively since they both read the same *.login* file).
The shell can be recognized as a valid shell if the system administrator puts it in the file */etc/shells*. If this file does not exist, it must be created and should contain all valid shells (i.e.don't forget the traditional ones in all their forms).

Lab 15.13 - WARNING

If you do decide to change your shell you must be **very** careful - if handled wrongly it can be almost impossible to correct, and will almost certainly cause you a lot of hassle. Never make a new shell a login shell until you have tested its new configuration files thoroughly and then tested them once again. It is also important that you make a full backup of your previous config files onto a floppy disk (or a different host if you have a second account) if you have to change any of them (which you will probably have to do if you can't change your shell entry in */etc/passwd*). You should also note that your new shell is probably **not** supported by your system admin, so if you have any problems you will probably have to look elsewhere.

Lab 15.14 - Deciding on a shell

Which of the many shells you choose depends on many different things, here is what I consider to be the most important, you may think differently.

How much time do I have to learn a new shell?
There is no point in using a shell with a different syntax, or a completely different alias system if you havn't the time to learn it. If you have the time and are presently using csh or tcsh it is worth considering a switch to a Bourne shell variant.

What do I wish to be able to do with my new shell?
The main reason for switching shells is to gain extra functionality; its vital you know what you are gaining from the switch.

Do I have to be able to switch back to a different shell?
If you may have to switch back to a standard shell, it is fairly important you don't become too dependent on extra features and so can't use an older shell.

How much extra load can the system cope with?
The more advanced shells tend to take up extra CPU, since they work in cbreak mode; if you are on an overloaded machine they should probably be avoided; this can also cause problems with an overloaded network. This only really applies to very old systems nowadays.

What support is given for my new shell?
If your new shell is not supported make sure you have someone you can ask if you encounter problems or that you have the time to sort them out yourself.
What shell am I using already?
Switching between certain shells of the same syntax is alot easier than switching between shells of a different syntax. So if you havn't much time a simple upgrade (eg csh to tcsh) may be a good idea.

Can I afford any minor bugs?
Like most software all shells have some bugs in them (especially csh), can you afford the problems that may occur because of them.

Do you need to be able to use more than one shell?
If you use more than one machine you may discover that you need to use more than one shell regularly. How different are these shells and can you cope with having to switch between these shells on a regular basis. It may be to your advantage to choose shells that are similar to each other.

Lab 15.15 - A brief history of UNIX shells

In the near beginning there was the Bourne shell /bin/sh (written by S. R. Bourne). It had (and still does) a very strong powerful syntactical language built into it, with all the features that are commonly considered to produce structured programs; it has particularly strong provisions for controlling input and output and in its expression matching facilities. But no matter how strong its input language is, it had one major drawback; it made nearly no concessions to the interactive user (the only real concession being the use of shell functions and these were only added later) and so there was a gap for something better.

Along came the people from UCB and the C-shell /bin/csh was born. Into this shell they put several concepts which were new, (the majority of these being job control and aliasing) and managed to produce a shell that was much better for interactive use. But as well as improving the shell for interactive use they also threw out the baby with the bath water and went for a different input language.

The theory behind the change was fairly good, the new input language was to resemble C, the language in which UNIX itself was written, but they made a complete mess of implementing it. Out went the good control of input and output and in came the bugs. The new shell was simply too buggy to produce robust shell scripts and so everybody stayed with the Bourne shell for that, but it was considerably better for interactive use so changed to the C shell, this resulted in the stupid situation where people use a different shell for interactive work than for non-interactive, a situation which a large number of people still find themselves in today.

After csh was let loose on an unsuspecting world various people decided that the bugs really should get fixed, and while they where at it they might as well add some extra features. In came command line editing, TENEX-style completion and several other features. Out went most of the bugs, but did the various UNIX operating system manufacturers start shipping tcsh instead of csh? No, most of them stuck with the standard C-Shell, adding non-standard features as they went along.

Eventually David Korn from AT&T had the bright idea to sort out this mess and the Korn shell /bin/ksh made its appearance. This quite sensibly junked the C shells language and reverted back to the bourne shell language, but it also added in the many features that made the C shell good for interactive work (you could say it was the best of both worlds), on top of this, it also added a some features from other operating. The Korn shell became part of System V but had one major problem; unlike the rest of the UNIX shells it wasn't free, you had to pay AT&T for it.

It was at about this time that the first attempts to standardize UNIX started in the form of the POSIX standard. POSIX specified more or less the System V Bourne Shell (by this time the BSD and System V versions had got slightly different). Later the standard is upgraded, and somehow the new standard managed to look very much like ksh.

Also at about this time the GNU project was underway and they decided that they needed a free shell, they also decided that they wanted to make this new shell POSIX compatible, thus bash (the Bourne again shell) was born. Like the Korn shell bash was based upon the Bourne shells language and like the Korn shell, it also pinched features from the C shell and other operating systems (in my opinion it put them together better; guess which shell I use), but unlike the Korn shell it is free. Bash was quickly adopted for LINUX (where it can be configured to perform just like the Bourne shell), and is the most popular of the free new generation shells.

Meanwhile Tom Duff faced with the problem of porting the Bourne shell to Plan 9, revolts and writes rc instead, he publishes a paper on it, and Byron Rakitzis reimplements it under UNIX. With the benefit of a clean start Rc ended up smalled, simpler, more regular and in most peoples opinion a much cleaner shell.

The search for the perfect shell still goes on and the latest entry into this arena is zsh. Zsh was written by Paul Falstad while he was a student a Princeton and suffers from slight case of feeping creaturism. It is based roughly on the bourne shell (although there are some minor but important differences) and has so many additional features that I don't even think the author even knows all of them.

Additionally rc has been enhanced to produced es, this shell adds the ability for the user to redefine low level functions.