Angling may be said to be so like the mathematics, that it can never be fully learnt.

— IZAAK WALTON

So is UNIX.

— K. Nandakumar

Appendix C

Some basic unix commands

C.1 Introduction to the shell and the desktop

In a command line oriented, interactive environment, a *command shell* (which is a program or a *process*) accepts a command from the keyboard, passes it to the operating system for execution, prints out any error or informational messages generated by the command and displays a prompt string indicating the readiness to accept another command. There are several shells available under AIX. The **Kron shell** or **ksh** is one of the most powerful shells and is the default shell on the AIX machines maintained by the department of chemical engineering.

In the GUI oriented environment, the equivalent of a command shell is the **desktop** which organizes various tools and application programs such as file manager, program manager, printer manager, *etc.* as objects accessible via icons. The interaction takes place through dialogue boxes and forms that must be completed. Program execution begins simply by double clicking on the appropriate icons.

If you have a good reason to change your default shell to something other that **ksh**, you can do so with the **chsh** command,

```
user@machine:dir> chsh
```

This command will display your current shell, and prompt you for the

name of the new shell. The change takes effect when you login the next time. At any time you can invoke a new shell, different from the login shell, *e.g.*,

```
user@machine:dir> csh
```

invokes a C-shell.

You can invoke a desktop at any time on AIX machines by entering

```
user@machine:dir> xdt3
```

Since the use of desktop is supposed to be rather intuitive, you are encouraged to explore its features on your own!

C.1.1 The ".profile" file

The ".profile" file is used to customize the shell environment for **ksh**. The following is a typical example of a ".profile" file.

```
PATH=$PATH:$HOME/bin:$KHOROS_HOME/bin
HOSTNAME=‘hostname‘
PS1=’ $LOGNAME@$HOSTNAME:$PWD>’
EDITOR=emacs

#defines alias for commonly used commands
alias ls=’ls -al’
alias rm=’rm -i’

#export environment variables to other processes ...
export PATH HOSTNAME PS1 EDITOR TERM
```

Here, several environment variables such as PATH, HOSTNAME, PS1 *etc.* have been defined. The concept of the environment is like a bulletin board. You can post definitions of any number of variables there. Application programs that expect specific variables can look for them and use their values, if they are defined. Note that when you set the value of a variable (*i.e.*, left hand side of an equal sign), the name is used without any prefix. When you want to retrieve the value the $ prefix is used. For example try,

```
user@machine:dir> echo $PATH
```

to look at the value of the current path.

In the first line of the example above, a system wide environment variable $PATH, which is already defined, is redefined to add additional paths, such as $HOME/bin separated by colon. Observe the '$' prefix to the name of the environment variable. $HOME itself is an environment variable, containing the value of the home directory. In the 2nd line the variable HOSTNAME is defined to contain the name of the workstation. This name is actually retrieved by the program 'hostname'. The 3rd line redefines the prompt string PS1 using other variables such as LOGNAME, HOSTNAME and PWD. These variables contain, respectively, the values of the *userid, machine name* and *present working directory*. If the variable EDITOR is set to emacs, then command line editing features using emacs keys are enabled.

You can also define aliases for certain commands. In the example above, the "ls" string is defined to be 'ls -al' - so when you enter "ls" at the prompt, the command 'ls -al' is executed. To examine all the currently defined aliases, enter,

```
user@machine:dir> alias
```

By default, the "rm" command removes files without prompting you for confirmation which could result in accidental deletion of files. The alias defined above, assigns 'rm -i' to "rm". The keyword "-i" stands for interactive mode and hence you will always be prompted before removing a file.

The variables defined in a shell environment are available only to that shell environment and not to other shells that you may start from the current one. The **export** command is used to export the variables to all subsequent shells. The last line in the above example *exports* several environment variables.

To look at all of the environment variables defined in the current **ksh** shell, enter,

```
user@machine:dir> set
```

To examine the value of an environment variable, enter,

```
user@machine:dir> echo $PS1
```

To set a new environment variable, use

```
user@machine:dir> DUMMY=junk
```

In addition to assigning values to environment variables, the shell allows programming flow control features. Thus one can write quite powerful *scripts* to execute a complex sequence of commands. A *script* is nothing but a set of AIX instructions placed in a file. By enabling the *execute permission* for this file, and entering its name from the command line you can cause the instructions in that file to be executed.

C.2 Managing files

In managing your files and directories, you need to be able to list the contents of a directory or file, copy and move files, compress and uncompress files, create and delete files and directories, control the ownership and access to files *etc.* Commands to carryout these tasks are illustrated below with specific examples. Try them out at a terminal. To get a complete description of each *command* use the man pages *i.e.*,

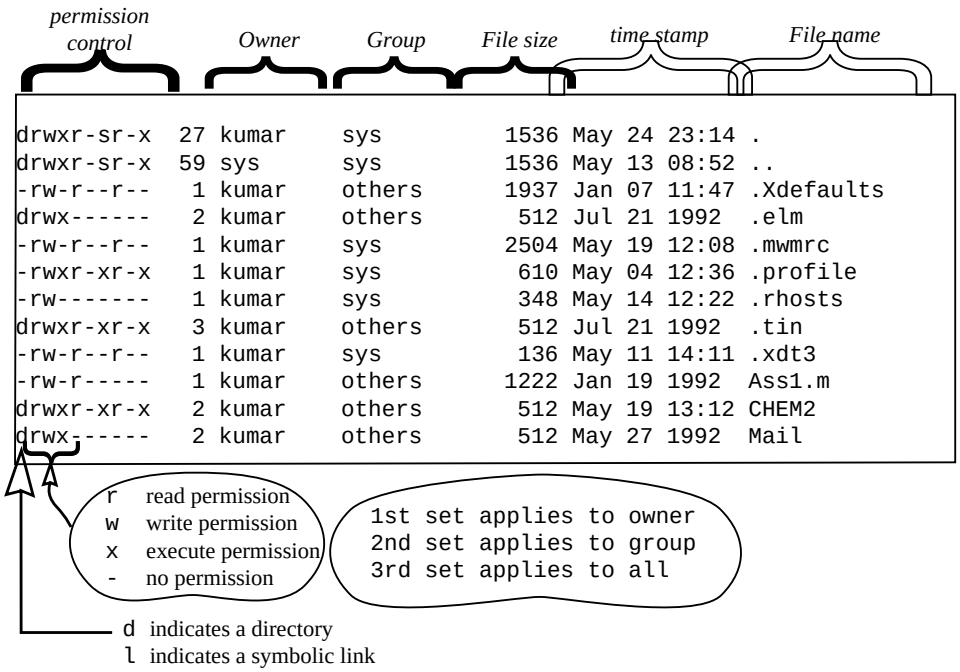
`user@machine:dir>` **man** *command*

C.2.1 Making sense of the directory listing - `the "ls" command`

The `ls` command produces a listing of all the files in the current directory. In its most useful form, you will use the "-al" keywords, *i.e.*,

`user@machine:dir>` **ls -al** *dir*

Typically, files that begin with the "." (*e.g.*, .profile) are treated as hidden files. They keyword "-a" however lists all of the files including the hidden ones. The keyword "-l" produces the long listing, a sample of which is shown in figure C.1. This listing provides information on file access control, ownership, size, time stamp *etc.* Each line contains information for a file or directory. The first character identifies whether it is a file (-), a directory (d) or a symbolic link (l). A symbolic link is a pointer to some other file (think of it as an alias). The next set of nine characters identify the file access control, in groups of three. Since AIX is a multiuser environment, users can control ownership and access of their files to others. The possible access modes are: read (r), write (w) execute (x) or none(-). These modes apply to (user, group, others). The groups are established by the system administrator. The owner and group names are listed next, followed by file size in bytes, the time stamp for last change and the file name.



Examples:

The file **`.profile`** has *(read,write,execute)* permission for *owner* (kumar in this case) and *(read,execute)* permission for both the group (sys in this case) and everyone.

The directory **`Mail`** has *(read,write,execute)* permission for *owner* only

The command

```
chmod g+r file
```

will give *read* access to *group* for file, while

```
chmod o-w file
```

takes away *write* access to *all* for file

Other related Unix commands

`ls -al`	-	detailed listing of directory such as the above
`chmod`	-	change permission on files and directories
`chown`	-	change ownership of files and directories
`rm`	-	remove or delete a file
`rmdir`	-	remove or delete a directory
`mkdir`	-	create a new directory

Figure C.1: Output of the "ls" command

C.2.2 Changing permission on files - `the "chmod" command`

The `chmod` command allows you to modify the access control of files and directories.

Examples

- To give *read* permission to *group* for *file* use,

  `user@machine:dir>` **chmod g+r** *file*

- To give *write* permission to *everyone* for all the files in a directory use,

  `user@machine:dir>` **chmod -R a+w** *dir*

  Note the the "-R" flag stands for recursive use of the command for all files in all subdirectories.

- Note that, in order to give read permission to a directory, the execute permission at the directory level must be set.

  `user@machine:dir>` **chmod a+x** *dir*

C.2.3 Moving files

The **mv** (move) command moves files and directories from one directory to another, or renames a file or directory. You cannot move a file onto itself.
Warning: The **mv** command can overwrite many existing files unless you specify the -i flag. The -i flag prompts you to confirm before it overwrites a file.

Examples

- To rename a file, enter:

  `user@machine:dir>` **mv** *oldname newname*

  This renames file *oldname* to *newname*. If a file named *newname* already exists, its contents are replaced with those of *oldname*.

- To move a directory, enter:

  ```
  user@machine:dir> mv olddir  newdir
  ```

  This moves all files and directories under *olddir* to the directory named *newdir*, if *newdir* exists. Otherwise, the directory *olddir* is renamed to *newdir*.

- To move several files into another directory, enter:

  ```
  user@machine:dir> mv file1  dir1/file2 newdir
  ```

  This moves *file1* to *newdir* and *dir1/file2* to *newdir/file2*.

C.2.4 Copying files

The **cp** command creates a copy of the contents of the file or directory from a source to a target. If the file specified as the target exists, the copy writes over the original contents of the file. If you are coping more than one source file, the target must be a directory.

Examples

- To make a copy of a file in the current directory, enter:

  ```
  user@machine:dir> cp file.old file.new
  ```

  If *file.new* does not already exist, then the **cp** command creates it. If it does exist, then the **cp** command replaces it with a copy of the *file.old* file.

- To copy a file in your current directory into another directory, enter:

  ```
  user@machine:dir> cp file.old  dir/sub.dir/
  ```

  This copies *file.old* to *dir/sub.dir/file.old*.

- To copy all the files in a directory to a new directory, enter:

  ```
  user@machine:dir> cp /home/user/dir1/*  /home/user/dir2
  ```

  This copies all the files in the directory */home/user/dir1/* to the directory */home/user/dir2/*. As a variant, explore the "`-R`" flag to copy not only all of the files, but also all of the subdirectories.

C.2.5 Changing ownership of files - the "chown" command

The chown command changes the owner of the file specified by the File parameter to the user specified by the Owner parameter. The Owner parameter can be specified either as a user ID or as a lo- gin name found in the /etc/passwd file. Optionally, a group can also be specified. The group can be specified either as a group ID or as a group name found in the /etc/group file. The syntax is,

`user@machine:dir>` **chown -R** *owner:group file*

Only the root user can change the owner of a file.

C.2.6 Compressing files - the "compress" command

The compress command reduces the size of files using adaptive Lempel-Zev coding. Each original file specified by the *file* parameter is replaced by a compressed file with a "`.Z`" appended to its name. The compressed file retains the same ownership, modes, and access and modification times of the original file. If compression does not reduce the size of a file, a message is written to standard error and the original file is not replaced. The syntax is,

`user@machine:dir>` **compress** *file*

To restore the file to its original state use the command,

`user@machine:dir>` **uncompress** *file*

Also try the GNU version of compress utility called **gzip** and **gunzip** - they are more efficient in both speed and size.

C.2.7 Removing files - the "rm" command

The **rm** command removes the entries for the specified file or files from a directory. If an entry is the last link to a file, the file is then deleted. You do not need read or write permission for the file you want to remove. However, you must have write permission for the directory containing that file.

Examples

- To delete a file, enter:

```
user@machine:dir> rm myfile
```

If there is another link to this file, then the file remains under that name, but the name *myfile* is removed. If *myfile* is the only link, the file itself is deleted. **Caution:** You are not asked for confirmation before deleting the file. It is useful to set an alias in your "`.profile`" file to redefine "rm" as

alias rm='rm -i'

After each file name is displayed, enter "y" to delete the file, or press the Enter key to keep it.

C.3 Managing processes

Since AIX is a multi-tasking operating system, several tasks (or *processes*) can be running at the same time. So we need a set of tools to monitor the currently running processes and the resources they consume, suspend or terminate specific processes, set priority for certain tasks or schedule some tasks for execution at specified times. Commands to accomplish these tasks are illustrated next.

C.3.1 Examining jobs or processes - `the "ps" command`

The **ps** command displays a set of currently running tasks. In its simplest and most useful form, the command is,

```
user@machine:dir> ps -ael
```

This provides a long listing of all the currently running processes including all of the *daemons* started by the *root* at the time of booting the computer. A typical sample output might look like,

```
     F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
200001 R  21 15101 17095  13  66 20 196d   116           pts/0  0:00 ps
240801 S  21 17095 16070   3  61 20 1dce   108           pts/0  0:00 ksh
260801 S   0  3637  3112   0  60 20  b25   260               -  0:00 sendmail
260801 S   0 12169     1   0  60 20  8a5   152           hft/0  0:00 lmgrd
222801 S   0 12938 12169   0  60 20 16aa   352           hft/0  0:05 CFDSd
 40801 S   0 10342  8542   0  60 20 357a   196               -  0:11 nfsd
```

The process name (or the command name) is shown in the last column. Other useful parameters are the process identification number (PID), the nice value (NI) which determines the priority of the process,

and the cpu time (TIME) used up by the task. In the above example listing, `sendmail` is the mail program, `lmgrd` is the license manager daemon, `CFDSd` is the license server for FLOW3D program, `nfsd` is the NFS daemon; all of these tasks are run by root with a user identification number (UID) of 0. Note that the `ps` command itself is a task.

C.3.2 Suspending jobs or processes

If you started a process like "`emacs`" or "`matlab`" and you want to suspend that task and return to the shell you can do so with the key sequence

`user@machine:dir>` **ctrl-z**

The PID number is displayed at that time. Even if you did not note it down, you can find a list of all suspended jobs with the command

`user@machine:dir>` **jobs**

To resume the job, enter

`user@machine:dir>` **fg %***n*

where *n* is the job number produced by the `jobs` command (and not the PID number!). The "**fg**" command brings a job to the foreground.

C.3.3 Terminating jobs or processes - `the "kill" command`

If you started a process in error and want to terminate it, you can use the "`kill`" command. You need to find out the PID number of the process using "`ps` command.

`user@machine:dir>` **kill -9** *PID*

Except for the super user (or root), one can terminate only those processes that belong to (or initiated by) individual users.

C.3.4 Initiating background jobs - `the "nohup" command`

The **nohup** command runs a command or a script ignoring all hangups and QUIT signals. Use this command to run programs in the background after logging off. To run a **nohup** command in the background, add an & (ampersand) to the end of the command. Its syntax is:

`user@machine:dir>` **nohup** *command_or_script &*

When used in its simplest form as above, any output that would normally appear on the screen will be saved in a file named **nohup.out** in the current directory. Wait before logging off because the **nohup** command takes a moment to start the *command_or_script* you specified. If you log off too quickly, your *command_or_script* may not run at all. Once your *command_or_script* starts, logging off does not affect it. Note that in order to run a script, the script file must have execute permission.

C.3.5 Script files & scheduling jobs - `the "at" command`

If you wish to schedule a job to begin at a specified time (typically late night), use the `at` command. The job should be constructed in the form of a script file. For example a file named *test.bat* contains the following lines and has *execute* permission enabled with the `chmod` command.

```
matlab >out  « 'eof'
secant('ass3a',[0,1],eps,1)
fzero('ass3a',0.5,eps,1)
quit
eof
```

In the above script we start MATLAB in the first line and redirect any output generated by MATLAB for the standard output (*i.e.*, screen during an interactive session) to a file named `out`. During an interactive session, MATLAB expects commands from the standard input (*i.e.*, the keyboard). Such inputs are now taken from the script file itself as seen in the next few lines where we execute some MATLAB functions and finally quit MATLAB.

The contents of such a script file can be executed interactively while logged in to a machine by simply entering the file name as

`user@machine:dir>` **test.bat**

The above example illustrates script programming in its simplest form. It is possible to write very sophisticated scripts in the **Kron shell** or any other shell. When you invoke MATLAB, for example, with the command `matlab`, you are actually executing a powerful script. Browse through the matlab script file using

`user@machine:dir>` **pg /usr/local/matlab/bin/matlab**

to appreciate the power of script programming.

Once a script file has been constructed, you can schedule it to be executed at a specified time using the `at` command as follows

`user@machine:dir>` **at 21:00** *scrit_file*

which will begin executing the *script_file* at 21:00 hours. To examine a listing of all the jobs scheduled use,

`user@machine:dir>` **at -l**

To remove a job that you have accidentally submitted, you can use,

`user@machine:dir>` **at -r** *job_number*

C.4 List of other useful AIX command

A list of less frequently used AIX commands is given in Table C.1. You can use either the man page feature with

`user@machine:dir>` **man** *command*

or the

`user@machine:dir>` **info**

command which starts the InfoExplorer to find out about the syntax and usage of these and other commands. The directory **/usr/bin** contains all of the Unix commands.

command	Function
at	to schedule a task to start at a given time
cat	to list a file
cd	to change directory
diff	compare two files
dosformat	formats a floppy diskette using MS-DOS standards
dosread	copies a DOS file from a floppy
doswrite	copies a unix file to a DOS formatted floppy
find	find a file
info	InfoExplorer - online documentation
ksh	start a Kron Shell
make	a powerful UNIX make facility
mail	read mail
mkdir	create a directory
man	display online manual pages
logout	logout of current AIX session
lpq	list the queue of print jobs
lpr	send a print job to a network printer
lprm	remove a print job from a queue
nice	control job priority
nohup	Don't kill a process upon logout
pg	display a file, one page at a time
ping	to check if another machine is alive
pwd	display present working directory
rlogin	remote login to another machine
rcp	remote copy files from one host to other need to have "`.rhosts`" file setup
rm	remove (delete) files
rmdir	remove directories
rsh	execute a command on a remote machine need to have "`.rhosts`" file setup
rusers	list remote users in the local area network
script	logs a terminal session to a file
talk	talk to another user currently signed on
tar	archive files
telnet	connect to remote hosts
whoami	find out the current user
xinit	start X-server
xlc	c-compiler
xlC	c++ compiler
xlf	Fortran compiler

Table C.1: List of other useful AIX command