An Overview of the Unix System

Contents

I. Introduction: What is Unix?

Unix is an operating system. An operating system tells a computer how to process commands and programs that the users give it. An operating system is the foundation software of a machine; it schedules tasks, allocates storage, and presents a default interface to the user between applications. Some other operating systems are DOS, Windows, BeOS, Linux, and the Macintosh OS.

Unix is a highly developed, mature, very stable, complex, and immensely powerful operating system. Unix provides multi-tasking, multi-user abilities that allow multiple programs to run on one computer simultaneously, and multiple users to use one computer simultaneously; while most operating systems only work on a specific type of machine, Unix has different *flavors* so that it can be run on many different types of machines. In addition to the operating system, there is a *shell*; the shell takes commands from the user and modifies them to suit the operating system. Since there are several different shells, this allows you to use Unix for different purposes, since each behaves in different ways. This is a powerful aspect of Unix which is not shared by Windows or the Macintosh Operating Systems.

If you have used the DOS command line, some of what follows will be familiar. If you have mostly worked with graphical user interface systems such as Windows or Mac OS, it may take a while before you become comfortable working with Unix.

II. Commands and Jobs

Some introductory notes about Unix commands and this guide:

- File commands are designated by **`bold type`**; parameters are designated by *`italic type`*; optional parameters are shown enclosed in brackets **`[]`**. Unix makes use of extra keys called control characters. These are entered by holding down on the control key and then pressing the key you want. These are denoted with a **`^`** in front of the character (so Control-c is written as **`<^c>`**).

Note: Some of these commands may conflict with keys used by the software on your terminal computer. For instance, if you type **`<^s>`** while using NCSA Telnet on the Macintosh, the screen will freeze until you type **`<^q>`**. To prevent this, go to the **Session** menu, select **Setup keys...** and delete the suggested keys.

- Unix is case-sensitive. *Always* use lower case unless otherwise specified.
- Unix commands take two forms of parameters: plain parameters, like file names; and flags, which are preceded by a hyphen (-). Multiple flags can be appended to each other and require only one hyphen. Most commands accept file name wildcards. The wildcards are **`*`** and **`?`**. For example:

`ls -la *txt`	l and a are flags; * is a wildcard for any files ending with txt.

- To get online help with any command, use the **`man`** command (for manual).

`man` `[command]`	This will scroll the instructions on how to use the command, pausing after each page until you type <**space**>**.**
`man -k` *`pattern`*	This will search through the manual pages for any containing the pattern specified. For example, any command involving formatting could be found with **`man -k format`**.

- To abort an ongoing process in Unix, **`<^c>`** is used, while **`<^z>`** is used to suspend a process so that it can be restarted later.

Suspended Jobs

Whenever you start a program or give a command on Unix, you start a job or process that has an associated number and Process ID. If you quit cleanly from this, either by choosing to exit/quit or by typing **`<^c>`**, the computer kills the process and finishes the job. However, if you do not exit cleanly, usually by pressing **`<^z>`**, the computer keeps the job running. Jobs can be looked at with their associated process IDs by using **`jobs -l`** (el, not a one). The job can then be killed using **`kill -9 pid#`**.

If you try to log out when you have suspended jobs, the machine will tell you that they are there. In order to log out, type **`logout`** or **`exit`** again. This may give you problems, since some processes (particularly Elm) leave files that will have to be removed for them to work again. You can restart jobs by typing **`fg`** (for *foreground*); this will bring them to the screen and will allow you to quit from it properly. You can also restart them in the background by typing **`bg`**. This allows them to run without tying up your screen. When they begin in the background, the job number with the Process ID is printed on your screen. When they are finished, the job number comes up with the word “Done”.

III. Files and Paths

The C-shell

Unix uses shells that take commands from the user and supply them to the operating system. Unix has many different shells, such as sh, sh5, csh, Bourne, zsh, and ksh. All of these have somewhat different characteristics. The shell which we use almost exclusively at Haverford is the C shell, or csh. This is the shell which provides you with the login prompt, lets you change directories, write interactive scripts, and much more. **Only csh is covered in this overview**. Look in the manual pages for information on the other shells.

Special Files

Unix, like most other operating systems, employs a hierarchy of files and directories. Unix knows only one structuring element for this hierarchy: files. Directories are as much files as data or executable files are. The data of directory entries are links to superordinate and subordinate directories in the hierarchy. Directories fit inside directories, forming directory trees. In order to describe a file completely, it is necessary to point out where in the directory tree the file resides. This is done by giving the **path** of the file, a long string of directories and subdirectories above the file. For example:

```
/home/venus/jbloggs/doc.txt
```

Is the file doc.txt that resides within the directory jbloggs, which in turn resides within the directory venus, and so on. The directory "/" is referred to as the **root** directory. Both directories and files can be "hidden", i.e., they do not show up when other files are listed. This is achieved by having a "." as the first character of the filename.

The system has a record which stores the user's username, password, real name, and home directory. When a user logs on, the current working directory (**cwd**) is set to the registered home directory. For most users on Unix, their home directory will be **/home/venus/<username>**. All files which contain the abbreviation "rc" are startup files.

`.login`	This file is shell independent. It is read every time a user logs in. It sets general variables, such as the search paths (where it looks for programs), the terminal type (for how to show characters on the screen), and the mail directory. This script executes after the shell-specific startup (see below).
`.profile`	This file is read with every new shell that is opened. It is the equivalent of .login for each individual shell, while .login sets its variables only once per login. This file is shell-independent.
`.cshrc`	This file gets read with the startup of every new csh. If **/bin/csh** is the selected shell for a user, the login program will start a csh after login and read in .cshrc. This file contains terminal type definitions, the prompt characterization, the variable which defines the standard editor, and aliases.

Shell Scripts

Shell scripts are files containing series of shell commands which can run both interactively and non-interactively. The file .cshrc is a good example of a simple csh script. The file /etc/rc.local is a more complex shell script. Many files in the directory /etc are shell scripts. Consult a few of them and the manual pages to get a sense of scripting.

IV. Basic Commands

After logging in to the Unix system (check with the ACC is you have questions about establishing your Unix account, logging in, or terminal emulation) and depending on your needs, a variety of commands are available for your use.

Shell Commands

`passwd`	Sets or resets your login password.
`logout, exit`	Either one logs you out of the Unix system.
`who, w`	Either one displays all users logged on the local host. **`w`** provides more information than **`who`**.

<^z>	Suspends a job (see “Suspended Jobs” above).
file *filename*	Tells you what kind of file *filename* is (text, C, etc.). It is useful to do this before working with a file.
quota -v	Shows you your available disk space quotas on the system.
!*	This repeats the last command that began with the letter *****.
!!	Repeats the last command.
fg	Resumes a suspended job in the foreground (as the job controlling the terminal).
bg	Resumes a suspended job in the background (you will not see it running).
alias *arg1 arg2*	Sets shortcuts in your work. After entering this command, typing *arg1* will effectively type *arg2*, thus allowing you to shorten frequently used commands.
unalias *arg*	Removes the alias that had been associated with *arg*.
jobs	Shows jobs suspended or running in the background.
–*l*	Gives the Process Identity (PID) of running jobs.
kill *process-id*	Cancels the job with the given Process ID#.
–9	Definite kill. Use only if a job changes process numbers.
which *file*	Finds the path for an executable file. This can be useful when looking for software on the machine.
software	Lists the available software (packages and programming) on the system. Not all products are supported by the Academic Computing Center.
ps	Shows the processes that you are running on the system.
–r	Shows only processes currently running.
–a	Shows all user run processes (even those that are not yours).
–u	Shows a detailed output of what processes are running.

File Management

pwd	Shows the current working directory that you are in.
ls *filename*	Lists all files in the current directory. If a filename is supplied, this only will be listed. If a directory is specified, its contents are listed.
–l	Shows the size, date, owner, and access privileges of the files in the current directory.
–a	Show hidden (system) files (those beginning with a period).

cp *file1 file2*	Copies *file1* to *file2* (where *file1* and *file2* are valid filenames). If a file named *file2* exists, it is overwritten.
mv *file1 file2*	Moves/renames a file (*file1*) to another location/name (*file2*). This will change the directory of the file if a path is specified for *file2*. This generally takes the same flags as **rm**.
rm *filename*	Removes a file.
–f	Silent removal. Will remove the file without warning, or in case the file cannot be removed, does not issue a warning.
–r	Recursive removal. If a file is a directory, it is removed together with all its subdirectories (this is a potentially dangerous command).
–i	Careful removal; this will always prompt you to confirm that you mean to delete the specified file (this is usually the default on our systems).
cd *dirname*	Change into a directory. With parameter, changes into that directory; without a parameter it changes into the login directory of the user.
..	Changes the current directory by navigating one folder up in the hierarchy.
mkdir *dirname*	Makes a new directory in the present directory or along the specified path.
rmdir *dirname*	Removes a directory in the present directory or along the specified path. The directory must be empty first.
chmod *arg filename*	This changes the mode of a file. You can allow a file to be executed, read, or written to by yourself or others. This is useful in various situations; for more information on syntax and argument lists, consult the manual pages.
a= *filename*	Removes all permissions of a file.

File Manipulation

cat *filename*	Very similar to **type** in DOS; **cat** scrolls a file out onto the screen. If the file is long and you need to stop it, use **<^c>**.
more *filename*	This works similarly to **cat**, except that rather than listing the entire document, it scrolls out a screen at a time. To advance to the next screen, type <**space**>.
head *filename*	Shows the first 10 lines of a file.
tail *filename*	Shows the last 10 lines of a file.
wc *filename*	Counts the lines, words, and characters of a file.
grep *pattern file*	A pattern searching utility. Searches *file* (where *file* represents a filename) for *pattern* (where *pattern* is the string of characters you're looking for). Useful for checking for differences in files, similarities in files,

etc.. For example: to look for instances of the word “chicken” in the file *food.doc*, at the Unix prompt you would type **`grep chicken food.doc`**.

`-i`	Ignores case when searching.
`gcc` *`filename`*	Complies a file. See the manual pages for more information.

Wildcard Resolution

An important feature of the shell is to take user input and process it into usable information. One aspect of this is so-called wildcards. The wildcards **`*`** and **`?`** differ in how they handle truncation:

`rm` `food.*`	Deletes any file whose name begins with *file.* such as **food.dat**, **food**., **food.hello.there**.
`rm` `food.?`	Deletes any file whose name begins with *file.* and has only a single character following the period (**food.d** and **food.h**, but not **food.doc)**.

Input/Output Redirection

The shell has a *standard input*, which is your keyboard, and a *standard output*, which is your monitor screen; instructions and text are usually entered through standard input (you type them out on the keyboard), and results of instructions and error messages are usually printed on standard output (you can see them on the computer screen). However, these functions can be redirected into files or into other commands.

`command1` **`|`** *`command2`*	Pipes the output of *command1* into *command2* , so that the output of the first becomes the input to the second, and the output of the second can be piped into the third, and so on. The eventual result is usually printed to standard output. If a **`more`** command is used, it is paginated.
`<` *`filename`*	Uses a file as input.
`>` *`filename`*	Writes output a file that does not already exist (set **`noclobber`** to prevent output redirection from overwriting existing files).
`>>` *`filename`*	Writes output to a file to which the output should be appended.

Both **`>>`** and **`>`** can be used to redirect the output from a pipe into a file for storage. For example; suppose you are trying to find all lines in the files **pm.list1**, **pm.list2**, etc. that contain the word “Health” and you want to save that in a file called **health.txt**. To do this, you would enter the following command:

```
cat pm.list* | grep "Health" > health.txt.
```

V. Editors

The Haverford College Unix system has three main programs for editing files:

- **Pico** is the simple, easy-to-learn editor that is supported by Haverford College. To learn more about using Pico, please consult the ACC guide ***Using the Pico Editor***.
- **Emacs** is a very powerful and complex editor, which is widely used on Unix platforms.

- **vi** – the Unix VIsual editor – is the default editor on most Unix machines, and many functions in Unix use vi commands for inputs.

`pico` *`filename`*	Starts the Pico editor. If a file name is specified after this, Pico will open that file to be edited. Otherwise, a new file will be opened.
`-t`	Tool mode; automatically saves file on exiting.
`-w`	Disable word wrapping; this prevents Pico from inserting line breaks into your document.
`-v`	View only; prevents you from accidentally altering important files.
	The following **commands** can be used within **Pico**:
`<^g>`	Displays a help document.
`<^t>`	Checks spelling.
`<^k>`	Deletes a line of text (cut).
`<^u>`	Undeletes the previous cut line or segment (paste).
`<^q>`	Quits Pico, prompting for a save.
`emacs` *`filename`*	Starts the Emacs editor. If a file is specified, this file will be opened for editing. After Emacs has edited a file, it leaves a backup file with the same name, followed by a ~. These backups should be periodically deleted.
	The following commands can be used within Emacs:
`<^h> t`	Gives a tutorial on using Emacs.
`<^x> <^c>`	Exits Emacs.
`<^k>`	Kills (deletes) to the end of the line that the cursor is on.
`<^g>`	Cancels any previous command.
`vi` *`filename`*	This will start the visual editor. This editor is not intuitive and you should consult the manual pages before using it.

VI. Mail

The supported Unix mail program at Haverford College is **Elm**. This package is much easier to use than the standard Mail that comes with the Unix system. To learn more about using Elm, consult the ACC guide ***Using Elm for Unix Mail***.

`elm`	Starts the Elm program; you can then use the inverse-video bar to select a message, and type **<Enter>** to read it.
	The following commands work within Elm:

`m`	Send a mail message.
`d`	Delete the selected mail message.
`r`	Reply to the selected mail message.
`f`	Forward the selected mail message.
`mail`	Starts mail and lists the first 20 unread messages. The standard mail program works on all Unix systems, though it is not as user-friendly as Elm. Since it can be used on any Unix system, it is useful to know how to get in and out of it, and some basic commands.
	The following commands work within mail:
`m`	Sends mail. Identical with **`mail`** *`username`* (where *username* is a person's username).
`?`	Gives a list of commands that can be used.
`number`	Read the numbered message (where *number* is the number of a message).
`d`	Deletes the message just read. A parameter can be supplied to delete any numbered message. If a message is deleted, all remaining messages are immediately renumbered.
`s`	Saves the message just read into a file. A parameter can be supplied to save any numbered message.
`q`	Quits, saving read messages to the file mbox
`x`	Exits, keeping your mail in the incoming mail file.
`pine`	Starts the Pine mail program. Pine is a popular Unix mail program. Pine is currently not supported by the ACC.

VII. Networking

`ftp` *`node`*	Standing for **File Transfer Protocol**, this is used to transfer files from one system to another by opening a connection to a *node* (a remote host). The node address can be entered either as an IP address (165.82.1.31) or a computer name (venus.haverford.edu). Opening a connection can also be done from within ftp by typing **`open`** *`node`*. For file transfers from systems that you do not have an account on, the remote username is usually "anonymous," with your email address as the password.
	The following commands work once a connection has been established:
`cd` *`dirname`*	Changes the current directory on the remote host.
`lcd` *`dirname`*	Changes the current directory on the local host.
`binary`	Sets the transmission protocol to binary. This mode is required if transferring non-ascii programs, archives, or compressed files.

`get` *`filename`*	Transfers the specified file from the remote host to the local machine.
`mget` *`wildcard`*	Gets all files matching the given file pattern (*wildcard* represents a filename with a wildcard). FTP will prompt for each file name.
`put` *`filename`*	Sends a file to the remote host. Certain restrictions and permissions apply.
`mput` *`wildcard`*	Sends multiple files (akin to **`mget`**).
`quit`, **`<^d>`**	Either one exits FTP.

Note: Files and software packages are often compressed to make them easier to be transported. For example, Tar and GZip (*.tar and *.gz files) are used to pack many smaller files into one large file, including subdirectories. You should always check how large files are before you transfer them.

`telnet` *`host`*	Opens a connection to a remote host. The addressing scheme is equivalent to that of FTP. This protocol will allow full capacity logins, but not file transfers. The host name can be either an Internet address or a name. You should only telnet to machines that you have an account on, or that permit anonymous logins (such as Tripod).
`<^]>`	Returns control to the local console.
`finger`	Looks at logins to a computer as specified. Without any options, finger shows who is presently logged into the system. With an option, finger gives login information about a user. It also tells when mail was last checked, and contains certain information that the user can place in his or her .plan file.
`name`	Gives information concerning the person with that username on the system where finger is run (due to privacy concerns, this feature is mostly disabled on the Haverford system).
`@hostname`	Gives information about the users presently logged into the indicated host.
`name``@hostname`	Gives information about the person with that username on the indicated host.
`–l`	Long output; gives all information available.
`–s`	Short output; gives only last login information.
`whois` *`hostname`*	Displays information about the specified host.

VIII. For More Information

For more information or help using the Unix system, refer to the online help available via the **`man`** command. The Website <***http://unix.about.com/***> also contains a wealth of Unix resources, including tutorials, FAQs, and reference guides.

If you require Unix assistance, stop by the Academic Computer Center Offices. There are several books on the subject which can be recommended to you.