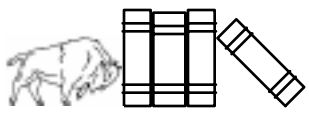
Computing and Information Technology

University at Buffalo
The State University of New York

Overtures!

A starter series for smarter computing



Introduction to UNIX

UNX-OVT-001
Academic Services
Computing and Information Technology
Computing Center
University at Buffalo
Buffalo, New York 14260 USA

This document was prepared using FrameMaker 5.5.6 on Sun OS 5.6 on the UBUnix sunCluster.

The following fonts are used in this document:

Helvetica Bold 10 points - Typeface used in the headers of each section.

`Courier 10 points` - Typeface used to indicate output on the terminal's screen.

`Courier Oblique 10 points` - Typeface used to indicate user's typed input.

`Courier Bold Oblique 10 points` - Typeface used to indicate commands and is used inline in the document.

Times Roman 10 points - Typeface for the body of the document.

Times Italic 10 points - Typeface to indicate a name or title.

Helvetica Narrow Oblique 8 points - Typeface used for margin notes.

All commands demonstrated in this document should be followed with a carriage return key. Some keys are denoted by name, including ***`<Esc>`*** for the Escape key, ***`<Delete>`*** for the Delete key, ***`<Return>`*** for the Return key, and ***`<Space>`*** for the spacebar. When using these keys, you need to press them only ***once*** to obtain the desired effect.

Revised: November 2000

Contents

Introduction

This document is intended to introduce you to the *UNIX* operating system. It will provide a basic understanding of the UNIX operating system, including its file and directory structure. This document also contains some basic UNIX commands and their functions, as well as instructions on getting online help.

What is UNIX?

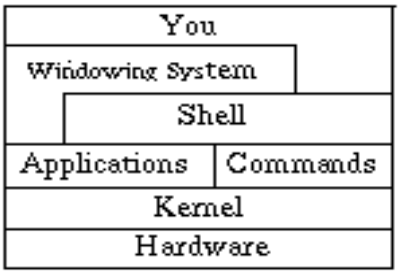
UNIX is a powerful computer operating system originally developed by AT&T Bell Laboratories. Over the years, two major forms of UNIX have evolved. The first of these is the AT&T UNIX System V. The second, developed at the University of California at Berkeley, is Berkeley Software Distribution (BSD).

This document is based on Sun's Solaris OS 5.6, which is an implementation of System V. Even through the variant of UNIX available at UB comes from Sun Microsystems, the information presented is general enough to apply to other versions of UNIX.

UNIX Layers

UNIX usage involves several *layers* of interaction between the computer hardware and the user. The first layer is the *kernel*, which runs on the actual machine hardware and manages all interaction with it. All *applications* and *commands* in UNIX interact with the kernel rather than the hardware, and these applications and commands make up the second layer. On top of the applications and commands lies the *shell*. The shell interprets commands, managing the interaction between you, the applications, and the available UNIX commands.

A final layer that may be present on your system is a *windowing system*. A windowing system usually interacts with the shell, but it can also interact directly with applications. The final layer is the *user*. The user interacts with the entire operating system through either the shell or a combination of the shell and the windowing system. The figure below gives a visual representation of the layers of UNIX:



Logging In

Before you can start using the system, you must first log in to it. In order to log in, you ***must*** have an account on the system you are trying to access. Remember that different domains require different accounts. You cannot log in to an engineering machine with a UBUnix account, even though they may share a common userid.

Entering your UB IT Name and password

Every individual associated with the University at Buffalo has been assigned a *UB IT Name* and *password*. This combination of information allows access to your computing account.

To log in to UBUnix through one of the machines in a Public IT Computing Area, use a program like *telnet* or *CRT*. Enter your UB IT Name and press the **`<Return>`** key. It is very important that you use lower-case letters for your UB IT Name, as the DCE server used to authenticate logins is case-sensitive.

After you have entered your UB IT Name, the system will prompt you for a password. Enter your password and press the ***<Return>*** key. Notice that the system does not echo your password to the screen as you type it. This prevents other users from learning your password by looking at your screen.

NOTE: If you receive a message similar to `Login failed`, you may have mistyped something. Try the procedure again. If the problem persists, see the consultant on duty.

Once you have successfully logged on, the system will display a few lines telling you when and from which machine you last logged in, in addition to any system messages of the day. The system will then present you with a command prompt. This prompt indicates that the system is ready and waiting for a command. By default, the command prompt will appear as a single `>`.

Changing your password

Each account on the system uses the same algorithm to determine the default password. As such, this password is very insecure. You should change it the first time you log in, and it is recommended that you change it on a regular basis. To change your UB IT password through UBUnix, use the ***passwd*** command. This command will prompt you for your old password, your new password, and a verification of your new password to ensure that you entered it properly the first time.

Your password must meet the following conditions:

- It must be 6 or more characters in length.
- The sequence of characters in the password must not match anything in your UNIX account information, such as your UB IT Name or an item from your account information data entry (your telephone number, address, etc.).
- The password should be a mixture of letters, numbers, and special characters (*`*`*, *`?`*, *`@`*, etc.). It must be not be found in the system dictionary.
- The password may not have three or more consecutively repeated characters or words in the dictionary contained within it.

Due to the scheduling of the password-changing process, it can take up to an hour for the system to process the new password. As a result, you should be prepared to use your old password to log in again shortly after changing it.

If you should ever forget your password, you can go to the CIT Help Desk, 216 Computing Center, and request that a new password be generated for you. You will need to bring your UB Card as proof of identity.

Logging in remotely

On occasion, you may need to log in to another work station, file server, or UNIX system while you are logged in to a workstation on a different server. For example, you may wish to connect to *armstrong.cse.buffalo.edu* from an open UBUnix connection on *lucia.acsu.buffalo.edu*. The command ***rlogin*** allows you to connect remotely to such a system, provided that you have an account on it. The syntax for ***rlogin*** is:

```
rlogin newSystem
```

where *newSystem* is the name of the remote system to which you wish to connect. You will be asked to supply a password for the account. After logging in, you will see the message of the day that is used on the new system.

If your userid on *newSystem* differs from that used on the current system , you will have to use the following form of the ***rlogin*** command:

```
rlogin newSystem -l userid
```

where *userid* is your userid on *newSystem*.

Basic UNIX Elements

There are six basic elements of UNIX:

- *Commands* are the instructions given to the system. These instructions tell the system what to do.
- *Files* are named collections of data. A file is analogous to a container in which you can store documents, raw data, or programs. A single file might contain the text of a research project, statistical data, or an equation-processing formula. Files are stored in directories.
- A *directory* is similar to a file cabinet drawer that contains many files. A directory can also contain other directories. Like any file, a directory has a name.
- The *environment* is a collection of items that describe or modify how the computing session will be carried out. It contains such items as the location of commands, the identity of the printer to which your output is sent, etc.
- A *process* is a command or application running on the system.
- A *job* is a sequence of instructions given to the system from the time a particular task begins until the time it ends. A job may have one or more processes associated with it.

UNIX Commands

UNIX has a wide range of commands. These commands allow you to manipulate not only files and data, but the environment as well. This section will provide an overview of the general syntax of UNIX commands.

Basic syntax

A UNIX command line consists of the name of the UNIX command followed by its *arguments*. Arguments include command options, filenames, and other expressions. You must press the ***<Return>*** key to execute any UNIX command. The general syntax for a UNIX command is:

```
command [-flag options] file/expression
```

where *command* indicates an action the system will take, *flag options* modify the command action, and *file/expression* is the name of the object on which the command acts.

The brackets around the flags and options are a shorthand way to indicate that they are often optional, and should be invoked only if you wish to use a particular option. You may string together multiple flags as well. Examples provided later in this document will illustrate this concept.

You should follow several rules when using UNIX commands:

- UNIX commands are case-sensitive, but most are lowercase.
- UNIX commands can only be entered at the shell prompt.
- UNIX command lines must end with a ***<Return>***.
- UNIX options often begin with a **-** (minus sign).
- Many commands can accept more than one option.

The *man* command

Perhaps the most important command on the UNIX system is the ***man*** command, which gives you access to the standard online help facility available within UNIX: the electronic reference manuals, known as the *man pages*. You can access these pages by typing the following at your UNIX prompt:

```
man command-name
```

where *command-name* is the name of the command for which you need help. The man pages provide an in-depth description of the chosen command, including an explanation of its options, examples, and further references. The information is an electronic duplicate of the paper reference manual pages available to the system administrators.

If you are unsure of the exact command name you require, you can use the ***-k*** option to search for a *keyword* among the one-line descriptions in the help files:

```
man -k keyword
```

You can read more about the ***man*** command by typing `man man`.

Other applications that reside on your system may have *man* pages. These pages can often be called up in the same manner as the *man* pages for the operating system.

Redirecting input and output

A number of conventions govern sources of input to a program or command, as well as destinations of output from that program or command. The standard input in UNIX is normally the keyboard, and the standard output is normally the screen. UNIX is very flexible, however, and it allows you to change or redirect the input source and output destination. For example, any command that would normally display results on the screen can be directed instead to send the output to a file with the output redirection symbol **>**:

```
command > outputfile
```

This command directs the system to store the output from *command* into the file named *outputfile*, rather than print it to your screen.

Each successive redirection to a particular file will overwrite all of the previously existing data in that file. To append to the end of a file, use **>>**:

```
command >> outputfile
```

Another redirection is **<**, which tells the command to take its input from a file rather than from the keyboard. For example, if you have a program that requires data input from the keyboard, you may find that you have to type the same data a number of times in the debugging stage of program development. If you put the data in a file and direct the command to read it from that file, you need to enter the data only once when you make the data file.

The syntax for **<** is as follows:

```
program < inputfile
```

where *inputfile* is the name of the file containing the data to be used by *program*. Using *inputfile* will give the same results as typing the data on the keyboard.

It is also possible to combine both kinds of redirections as follows:

```
program < inputfile > outputfile
```

The data in the file *inputfile* will then be used as input for *program*, and all output will be stored in *outputfile*. If you want to store output from multiple files to a single file, you can use **>>** to append output to the end of a file rather than replacing the previous contents, which **>** will do.

A final I/O (input/output) redirection is the pipe symbol (|). This command tells the computer to take the output created by the command to the left of the pipe and use that as the input for the command on the right. An example of the pipe command is:

```
date | program
```

In this example, the output of the **date** command would serve as input to *program*.

NOTE: Not all interactive programs accept input from a file.

Special keys and control characters

UNIX recognizes special keys and control-character keystrokes and assigns them special functions. You can invoke a control-character keystroke such as **<Ctrl-C>** by holding down the key labeled **<Ctrl>** and pressing the **c** key (much like you can hold down the **<Shift>** key and then press the **c** key to generate a capital C). The notation for control characters is usually **^C** or **<Ctrl-C>**.

Some standard special keys and control characters are summarized below:

Special Key	*Description*
<Delete>	Acts as an erase key. Pressing **<Delete>** once will back up and erase one character, allowing you to correct mistakes.
<Backspace>	This key is sometimes used to delete characters. Otherwise, it is mapped as a backspace key, which generates a ^H on the display.
<Ctrl-U>	Erases the entire command line. It is also called the *line kill character*.
<Ctrl-W>	Erases the last word on the command line.
<Ctrl-S>	Stops the flow of output on the display.
<Ctrl-Q>	Resumes the flow of output stopped by **<Ctrl-S>**.
<Ctrl-C>	Interrupts a command or process in progress and returns to the command line. If this does not return you to your command line prompt, try typing the command several times in succession.
<Ctrl-Z>	Suspends a command or process in progress.
<Ctrl-D>	Generates an end-of-file character. It can be used to terminate input to a program, or to end a shell session.

A selected command list

The next few pages summarize many of the basic UNIX commands you need to get started.

Setup and status commands

Command	*Description*
`logout`	End your UNIX session
`stty`	Set terminal options
`date`	Display or set the date
`finger`	Display information about users
`ps`	Display information about processes
`env`	Display or change current evironment settings
`set`	Set shell variables
`alias`	Define command abbreviations
`history`	Display recent commands

File and directory commands

Command	*Description*
`cat`	Concatenate and display file(s)
`more`	Paginator that allows you to browse through a text file
`less`	More versatile paginator than *more*
`mv`	Move or rename files
`cp`	Copy files
`rm`	Remove files
`ls`	List directory contents
`mkdir`	Create a directory
`rmdir`	Remove a directory
`cd`	Change working directory
`pwd`	Display name of *present working directory*
`du`	Summarize disk usage
`chmod`	Change mode (access permissions) for a file or directory
`file`	Determine the type of a file
`quota -v`	Display current disk usage for this account
`mailquota`	Display current mail usage for this account

Editing tools

Command	*Description*
pico	Simple text editor
emacs	Advanced text editor
vi	Screen-oriented (visual) text editor
grep	Search a file for a pattern
sort	Sort and collate lines of one file
diff	Show differences between the contents of two files

Formatting and printing commands

Command	*Description*
lpq	View printer queue
lpr	Send file to printer for printing
lprm	Remove job from printer spooling queue
printers	List available printers

Program controls, pipes, and filters

Command	*Description*
<Ctrl-C>	Cancel current process or command
<Ctrl-D>	Generate end-of-file character
<Ctrl-S>	Stop flow of output to screen
<Ctrl-Q>	Resume flow of output to screen
<Ctrl-Z>	Suspend current process or command
jobs	List of background jobs
kill	Terminate a process
&	Run process in background (placed at end of command line)
>	Redirect output of a command to a file
>>	Redirect and append output of a command to the end of a file
>&	Redirect standard output and standard error of a command to a file
<	Redirect a file to the input of a command
|	Pipe the output of one command into another

Other tools and applications

Command	*Description*
pine	Electronic mail program
man	Display UNIX help pages to screen
trn	*USENET* reader

Files and Directories

Information on the UNIX system is stored in *files* and *directories*. All files and directories are stored on the system in a hierarchical fashion. You can envision the system structure as an upside-down tree. At the top of the tree is the *root directory.* Its name is / (a slash character). Below the root directory lies a set of major subdirectories that usually include *bin*, *dev*, *etc*, *lib*, *pub*, *temp*, and *usr.* For example, the *bin* directory is a *subdirectory* (or *child*) of / (the root directory). The root directory, in this case, is also the *parent* directory of the *bin* directory. Each path leading down away from the root ends in a file or directory. Paths can branch out from directories, but not from files.

A large grouping of files and directories is known as a *file system*. File systems are related to the disk size, disk structure, and internal structure of UNIX. User files and directories are usually hosted on one file system, while system files and directories are hosted on another. If the number of users is large, the user files and directories may be hosted on more than one file system.

Naming conventions

Each UNIX file has a *filename* unique to its current directory. As UNIX imposes very few restrictions on filenames, it is possible to name your files in such a way that the contents can be recognized easily. For instance, if your program calculates compund interest on an amount, you might call it *interest*. If your file is a research paper on Albert Einstein, you might call it *einstein*.

A filename can be up to 256 characters long, and may consist of any sequence of alphanumeric characters on the keyboard except the / symbol (the name of the root directory). Filenames may also include periods. In a file, the part after the period is known as an *extension*. Extensions usually indicate the type of information stored in the file. UNIX does not require extensions, but they can be used to help identify similar types of files. You may use any extension desired: a text file might have the extension *.txt* or *.text*, a graphic image may have the extension *.gif*, and so forth. Some UNIX programs (especially compilers) look for certain standard extensions, so it is common practice to use the following conventions:

Extension	***Description***
.h	Header files
.c	*C* source files
.f	*FORTRAN* files
.s	Assembler source files

For example, the file *einstein.txt* indicates a text file, whereas the file *interest.cc* indicates a *C++* program called *interest*.

Some UNIX files begin with a period. These files are called *hidden files* or *initialization files*. The *.cshrc* and *.login* files are good examples. These files will not appear in a normal directory listing. In general, they serve to set up UNIX environments and applications.

NOTE: Do not include blanks in your filenames, as they will make it difficult for you to work with the file. If you do wish to separate words in a filename, use an underscore (*einstein_paper*) or a hyphen (*einstein-paper*). Remember that UNIX is case-sensitive, which means that it recognizes the difference between upper-case and lower-case letters (*einstein* and *Einstein*, for example, would refer to two different files).

UNIX directories follow the same naming conventions that UNIX files do.

Absolute and relative pathnames

Each directory and file on the system has a path by which it is accessed. This path is called a *pathname*. There are two types of pathnames. The first of these (the *full* or *absolute* pathname) traces the absolute position of a file or directory back to the root directory. The second (the *partial* or *relative* pathname) traces the position of a file or directory relative to the current directory (also called the *present working directory*, or *pwd*). Pathnames of either sort use slashes (`/`) to show the breaks between directories.

UNIX provides certain abbreviations for a few special directories. The tilde character (~) is shorthand for your home directory. The home directory of any user can thus be abbreviated from `/parent-directories/userid` to `~userid`. Likewise, you can abbreviate `/parent-directories/youruserid/file` to `~/file`. The current directory has the abbreviation `.` (a period). The parent of the current directory uses `..` (two consecutive periods) as its abbreviation.

Permissions

It is important to protect your UNIX files and directories against accidental or intentional removal or alteration by yourself or other users. UNIX maintains *permissions* for every file and directory on the system. Permissions dictate which users do and do not have access to a particular file or directory. This section describes how to inspect and change these permissions.

Every file or directory in a UNIX file system has three types of permissions (or *protections*) that define whether certain actions can be carried out. The allowances made by the permissions change, depending on whether the item is a file or a directory. The three permissions are:

- *read* (*r*): A user who has **`read`** permission for a file may look at its contents or make a copy of it. For a directory, **`read`** permission enables a user to find out what files are in that directory.
- *write* (*w*): A user who has **`write`** permission for a file can alter or remove the contents of that file. For a directory, **`write`** permission enables a user to create and delete files in that directory.
- *execute* (*x*): A user who has **`execute`** permission for a file can execute the contents of that file (provided that it is an executable file, like a program). For a directory, **`execute`** permission allows a user to change directories to that directory using the **`cd`** command (discussed later).

For each file and directory, you may set the **`read`**, **`write`**, and **`execute`** permissions separately for each of the following classes of users:

- *user* (*u*): The user who owns the file or directory.
- *group* (*g*): Several users purposely lumped together so that they can share access to each other's files. Groups are defined by the system administrator. You can view those groups to which you belong by using the **`groups`** command.
- *other* (*o*): The remainder of the authorized system users.

NOTE: If you wish to change permissions for any files or directories located on your UBFS space, you must do so using *WebACL* or *DCECP*. This includes any files or directories located in your *public_html* directory, as they are also located on UBFS. For more information, see the CIT documents *Insights: Control Access to UBFS Files and Directories via WebACL* and *Insights: Control Access to UBFS Files and Directories via DCECP*.

The **ls** command is the primary command for displaying information about files and directories. The **-l** option will display the information in *long format*. You can get information about a single UNIX file by typing the following:

```
ls -l filename
```

Each file or subdirectory displayed with the **ls -lg** (*long*, *groups*) command consists of seven fields: *permission mode*, *link count*, *owner name*, *group name*, *file size* (in bytes), *time* (of last modification), and the *filename*. Each of these fields is discussed below.

The first 10 characters make up the *mode field*. If the first character is a `d`, then the item listed is a directory. If it is a dash (`-`), then the item is a file. If it is an `l`, then it is a link to another file. Characters 2-4 indicate the user permissions, characters 5-7 indicate the group permissions, and characters 6-9 detail the other permissions. If a particular permission is set, the appropriate letter appears in the corresponding position. Otherwise, a dash indicates that the permission is not given.

The second field, the *link count*, indicates the number of links to the file. In most cases it is one, but other users may make links to your files, thus increasing the link count. The third field gives the owner's UB IT Name, while the fourth reveals the group id. The next two fields give the size of the file (in bytes) and the date and time of the last modification to the item. The last field gives the item's name. A typical display result of the **ls** command looks like this:

```
-rw-r--r-- 1 owner 588 Jul 15 14:39 myfile
```

In this example, *myfile* has **read** permissions for every class of system user and **write** permission for the owner. Note that no one has **execute** permissions for *myfile*.

The owner can change any or all of the permissions with the **chmod** (*change mode*) command. The arguments supplied to **chmod** are a symbolic specification of the changes required, followed by one or more filenames. Specification consists of the following:

- The set of users for which permissions are to be changed: **u** for user, **g** for group, or **o** for others. You may also use some combination thereof, or **a** for all.
- The way in which the permissions should be changed: **+** to add a permission, **-** to remove a permission, and **=** to set the specified permissions while removing the current ones.
- The permissions to add or remove: **r** for **read**, **w** for **write**, and **x** for **execute**.

To remove all the permissions from *myfile*, for example, you would type the following:

```
chmod a-rwx myfile
```

To allow **read** and **write** permissions for all users:

```
chmod ugo+rw myfile
```

Finally, to allow only the **read** permission to all users:

```
chmod a=r myfile
```

The file is now well protected, as it can only be read. No one can execute or write to the file. Protecting a file against writing by its owner is a safeguard against accidental overwriting, although it is no safeguard against accidental deletion.

As with files, directories have permissions associated with them. The following is an example of permissions assigned to a directory:

```
drwxrwxr-x 1 owner abcdue 588 Jul 15 9:45 home
```

The directory *home* and the files and directories contained therein may be read and executed by anyone, but only the owner and users in the group *abcdue* may write to it. Assuming you are the owner of this directory, you may decide to change the permissions to allow only yourself and members of the *abcdue* group to read and execute files in the home directory. You would set the permissions accordingly:

```
chmod o-rx home
```

If you decide that only you should be able to alter the contents of the director, then you must remove the write permission for the group:

```
chmod g-w home
```

Wildcard characters

Wildcard characters allow you to copy, list, move, remove, and otherwise manipulate items with similar names. This a great help in manipulating files and directories, though it should be noted that careless use of wildcard symbols may be do more harm than good. UNIX supports the following wildcards:

- The symbol *?* will match any single character in that position in the file name.
- The symbol *** will match zero or more characters in that position in the file name.
- Characters enclosed in left and right brackets - *[* and *]* - will match any one of the given characters in the given position in the name.

Examples of each follow:

- *?ab2* would match any name that starts with a single character and ends with *ab2*.
- *?ab?* would match all names that begin and end with a single character and have *ab* in between.
- *ab** would match all names that start with *ab*, including *ab* itself.
- *a*b* would match all names that start with *a* and end with *b*, including *ab*.
- *s[aqz]* would match *sa*, *sq*, and *sz*. *s[2-7]* would match *s2*, *s3*, *s4*, *s5*, *s6*, and *s7*.

These wildcard symbols help in dealing with groups of files, but you should remember that the command ***rm ** would erase all files in the current directory (although by default, you would be prompted to verify each deletion). Use the wildcard character * very carefully!

Files

UNIX files contain information. This information can be text, programming code, etc. UNIX files are self-contained - that is, they do not contain other files or directories, as directories do. This section explains how to create, display, list, copy, rename, delete, and print files.

Creating files

Many files are created using a *text editor.* A text editor can manipulate saved text by allowing corrections, deletions, or insertions to a file. The main text editors on UBUnix are *vi*, *emacs*, and *pico*. For more information on these editors, see the CIT documents *Reference Cards: vi*, *Insights: Introduction to Emacs*, *Reference Cards: GNU Emacs*, and *JumpStart: Pico*.

Displaying files

You can display a file in one of several ways. For example, you could use the ***cat*** command:

```
cat filename
```

where *filename* is the name of the file you would like to view. This will cause the text contained in *filename* to scroll very quickly down the screen. While it is possible to control the flow of text by using ***<Ctrl-S>*** to stop the flow of text to the screen and ***<Ctrl-Q>*** to restart it, this can be very inconvenient. The ***more*** command displays one screen of information at a time before waiting for a keyboard prompt to continue:

```
more filename
```

The first page of text contained in *filename* will echo to your screen. You can hit ***<Space>*** to advance through each page of text until you reach the end of the file. Pressing the ***?*** character while viewing the file will show the help for the ***more*** command.

A more powerful command called ***less*** is available on many systems. Among other enhancements, the ***less*** command allows you to scroll backwards through a file. The syntax for the ***less*** command is the same as that for ***more***. For more information, type `man less`.

Listing files

The ***ls*** command lists files in the current directory. This command uses many important flag options to obtain specifics about the files in the directory:

Command	*Description*
ls -a	Lists all the contents of the current directory, including files with initial periods, which are not usually listed.
ls -l	Lists the contents of the current directory in long format, including file permissions, size, and date information.
ls -s	Lists contents and file sizes in kilobytes of the current directory.

If you have many files, your directory list might be longer than one screen. You can use the pipe symbol (|) in conjunction with the commands ***more*** or ***less*** to display the output from ***ls*** one page at a time:

```
ls | more
```

Copying and renaming files

To make a copy of a file, use the ***cp*** command as follows:

```
cp filename newfilename
```

where *filename* is the file you wish to copy and *newfilename* is the file you are creating.

To rename one of your files, use the ***mv*** command as follows:

```
mv oldfilename newfilename
```

where *oldfilename* is the name of the file you wish to move, and *newfilename* is the name of the new file.

Creating links between files

It is possible to create links in different directories that point back to one particular file. The ***ln*** command creates a link that points to the file. Note that links are simply alternative names for a single file. The ***ln*** command neither renames nor copies the file.

Since only one copy of the file actually exists, any changes that you make to the file will be reflected when you access it through another of its links. If you delete the link, however, you will not delete the file to which it points.

Links are useful for cross-referencing files. This method allows multiple users to access the same file from different directories. Creating links is a better alternative than making a copy of the file for each directory, as each user can access the up-to-date information when the original is altered. It is also more convenient than using the file's absolute pathname every time you need to access it.

One special type of link is the *symbolic link*. A symbolic link contains the pathname of the source file to which you wish to link. Symbolic links can connect to any file in the file structure, and they may even call a file on another file system. Symbolic links may also refer to directories in addition to individual files.

To create a symbolic link to a file within the same directory, type:

```
ln -s originalFile linkName
```

where *originalFile* is the file to which you want to link and *linkName* is the link to that file.

To create a link in a directory other than that of the original file, type:

```
ln -s originalFile differentDirectoryName/linkName
```

If you create a link within the same directory as the original file, you cannot give it the same name as the original file. There is no restriction on a file's additional names outside of its own directory. Links do not change anything about a file. If a link is made to a file, and that file is deleted, the link will be rendered ineffective.

Removing files

To delete files, use the ***rm*** command as follows:

```
rm filename
```

IMPORTANT: ***rm*** can be very dangerous. Once a file has been removed, it is very difficult (if not impossible) to retrieve it. It may take the system administrators several days to recover your deleted file, so use a great deal of caution when deleting files. If you want to use the ***rm*** command so that it does not remove the contents of a file without permission, use the ***-i*** option (***rm -i***) to receive a confirmation prompt prior to each file deletion.

Printing files

To print a file from UBUnix, use the ***lpr*** command as follows:

```
lpr [-Pprintername] filename
```
(to send to a specific printer)

where *printername* is the name of the printer to which you want to print the file.

In the absence of a *printername* specified with the **-P** option, **lpr** will print the file to the default printer. If you have not set a default printer, then all printed files will be sent to *bell*, the Docuprint 65 printer located in 101 Bell Hall. To get a list of the printers available at your machine, type:

```
printers
```

This command echoes to your screen a list of all named printers on the system, along with their types and locations. To get status information on the printers, use the command **prstat**.

NOTE: You need access to a laser printer to handle graphics or PostScript files. PostScript is a page-description language developed by Adobe Systems, Inc., and was specially designed for creating graphics and typography on a printed page. For more information, see the CIT document *JumpStart: UNIX Printing*.

Directories

UNIX directories are similar to regular files: both have names and contain information. Unlike files, however, directories contain other files and directories. Many of the same rules and commands that apply to files also apply to directories.

Many UNIX directories have traditional names and traditional contents that do not vary from system to system. For example, directories named *bin* contain binary files, which are executable command and application files. A *lib* directory contains library files, which are collections of routines that a compiler can include in a program. The *dev* directory contains device files, which are the software components of terminals, printers, disks, etc. All *temp* directories are used to store temporary files created by certain programs. The *etc* directory is used to store miscellaneous administrative files and commands, while *pub* is for public files (files that anyone can use), and *usr* has traditionally been reserved for user directories. On large systems, however, the *usr* directory may contain other *bin*, *temp*, and *lib* directories.

Creating directories

To create a new directory, use the **mkdir** command as follows:

```
mkdir directory
```

where *directory* is the name of the directory you wish to create. The **mkdir** command will create a subdirectory beneath the current directory if a relative pathname is given. It will create a subdirectory in a different location if an absolute pathname is specified.

Displaying directories

When you initially log in to the system, UNIX places you in your home directory. The **pwd** command will display the full pathname of the current directory, in a fashion similar to the following:

```
/home/userid
```

By typing **ls -a**, you can see every file and directory in the current directory as long as you have read permissions on the directory. To display the contents of your home directory when it is not your current directory, enter the **ls** command followed by the full pathname of your home directory:

```
ls /home/userid
```

Instead of typing the full pathname for your directory, you can also use the tilde symbol (**~**) with the ***ls*** command to display the contents of your home directory:

```
ls ~
```

To help you distinguish between files and directories in a listing, the ***ls*** command has a ***-F*** option. This option appends to each object in the directory a distinguishing mark that shows the kind of data it contains: no mark for regular files, **/** for directories, **@** for links, and ***** for executable programs.

Changing directories

To change your current directory to another directory in the directory tree, use the ***cd*** command in any of the following manners:

```
cd directory (relative pathname from home directory)
cd ~/directory (full pathname using ~)
cd /home/userid/directory (full pathname)
```

To get back to the parent directory of projects, you can use the special **..** directory abbreviation:

```
cd ..
```

If you become lost navigating directories, you can issue the ***cd*** command without any arguments. This will place you in your home directory. It is equivalent to typing `cd ~`.

Moving files between directories

It is possible to move a file into another directory by using the ***mv*** command as follows:

```
mv filename directory
```

For example, the following command:

```
mv sample.txt ~/projects
```

moves the file *sample.txt* into the *projects* directory. Since the ***mv*** command is capable of overwriting files, it would be prudent to use the ***-i*** option (confirmation prompt).

You can also move a file into another directory and rename it at the same time by specifying the new name after the directory path, as follows:

```
mv sample.txt ~/projects/newsample.txt
```

Copying files to other directories

It is possible to copy files to other directories without moving or changing the original file. The syntax for the ***cp*** command is very similar to that used for ***mv***:

```
cp sample.txt ~/projects
```

The new file will have the same name as the old one unless you change it while copying it:

```
cp sample.txt ~/projects/newsample.txt
```

Copying and renaming directories

To make a copy of a directory and its contents, use the ***cp*** command as follows:

```
cp -R oldDirectory newDirectory
```

where *oldDirectory* is the directory you wish to copy and *newDirectory* is the directory you are creating.

To rename one of your directories, use the ***mv*** command as follows:

```
mv oldDirectory newDirectory
```

where *oldDirectory* is the name of the file you wish to move, and *newDirectory* is the name of the new directory or location.

NOTE: Whether you are copying a directory or renaming it, the new directory name must not be in existence prior to the execution of the command. The new directory need not be in the current directory; you can copy and move a directory anywhere within a file system, just as you can move and copy files.

Removing directories

To remove a directory, make sure that you are in that directory's parent directory, and then type the following:

```
rmdir directory
```

where *directory* is the name of the directory you wish to delete. The directory cannot be removed unless all files and subdirectories contained in it have been erased. This prevents you from accidentally erasing important subdirectories.

You can erase all the files in a directory by first going to that directory (using ***cd***) and then using ***rm*** to remove all the files in that directory. The quickest way to remove a directory and all of its files and subdirectories (and their contents) is to use the ***rm -r*** (*recursive*) command along with the directory's name. For example, to empty and remove your *projects* directory, move to that directory's parent, then type the following:

```
rm -r projects
```

IMPORTANT: ***rmdir*** can be very dangerous. Once a directory has been removed, it is very difficult (if not impossible) to retrieve it. It may take the system administrators several days to to recover your deleted directory, so use a great deal of caution when deleting directories. If you want to use the ***rmdir*** command so that it does not remove the contents of the directory without permission, use the ***-i*** option (***rmdir -i***) to receive a confirmation prompt prior to each directory deletion.

Jobs and Processes

A *process* is a command or program running on the UNIX operating system. A sequence of related processes is called a *job.* Your applications, your programs - even your shell itself - are all processes. The UNIX kernel manages these processes on the system, usually without distinguishing among them. Because UNIX is a multi-tasking system, you can run one or more jobs in the background (unseen by the user) while continuing to work on another process in the foreground (seen on the user's screen).

Viewing your processes

The ***ps*** command shows you the status of your processes in a format similar to the following:

```
PID   TT   STAT   TIME    COMMAND
4804  p3    S     0:00    -sh (csh)
1352  p3    R     0:00    ps
3874  p7    IW    0:25    xclock -g 90x90-0+0
3875  p7    S     0:48    xbiff -g 90x90-95+0
```

The ***ps*** command displays the process ID (*PID*); the control terminal (*TT*); the process state (*STAT*); the cpu time used by the process so far, including both user and system time (*TIME*); and finally, an indication of the command (*COMMAND*) that is running the process.

Foreground and background jobs

If you put a program into an unattended state in which it continues to execute, you have placed it in the *background*. (Running a program on one machine and displaying its output on another via a windowing system like X-Windows is not considered backgrounding the job.)

You can run a job in the background by adding *&* (an ampersand) to the end of the command:

```
jobname &
```

You will receive a response like this:

```
[1] 5432
```

This response informs the user of three things: first, that there is one job running in the background; second, that its job number is 1; and third, that its process identification number (*PID*) is 5432.

Managing jobs and processes

The PID is necessary if you want to abort (*kill*) a job. To kill the job in the above example, you would use the ***kill*** command as follows:

```
kill 5432
```

You could also use *kill %1* to kill the process by its job number, as indicated by the number in brackets.

You can use the job number to switch a job between foreground and background. To switch a job from the foreground to the background, press ***<Ctrl-Z>*** to suspend the job and then use the ***bg*** (background) command as follows:

```
bg %jobNumber
```

where *jobNumber* is the number of the job you wish to switch to the background. To switch a job from the background to the foreground, use the ***fg*** (foreground) command:

```
fg %jobNumber
```

where *jobNumber* is the number of the job you wish to bring to the foreground. The ***jobs*** command will display a list of jobs and job numbers running in the background.

Logging Out

To end a work session, you must explicitly log out of UNIX. To do this, type ***logout*** at the command prompt. Once you have logged out, you will no longer be connected to the system.

If you experience difficulty logging out, check to see if you are having one of the following problems:

Problem: *The system replies "*`There are suspended jobs`*" when you try to log out.*

Type ***jobs*** to see the jobs that are running under your userid, type ***kill %%*** to terminate all stopped background jobs, then try logging out again.

Problem: *The system replies, "*`Not in login shell.`*"*

Type ***exit*** and press ***<Return>***. This should place you back in your login shell. After that, type ***logout*** at the prompt, if necessary.

Problem: *Nothing seems to work and you can't log out.*

You might be stuck in a program or shell other than the login shell. Press ***<Return>*** to clear any previous commands, then try typing ***<Ctrl-C>***, ***q***, ***:q***, ***:q!***, ***exit***, ***<Ctrl-D>***, ***<Ctrl-Q>***, or ***<Ctrl-Z>*** to get back into the login shell. After this, try the ***logout*** command again.

NOTE: You should never turn a workstation off, as this will not necessarily log you out.

Additional Help

This document is only a brief introduction to UNIX and does not include information on how to use all its capabilities. If you want to know more, you will need to seek out more detailed introductory information.

A wide variety of books cover more of the operating system than does this document. If you are looking for one of these books, you should first check the UB Libraries. You can also purchase inductory UNIX books at most local bookstores with computing sections.

IT Workshops are available to all those affiliated with the University. For information regarding registration, see the CIT document *Resource Guide: IT Workshops*, or register online at the following URL:

`http://wings.buffalo.edu/itworkshops`

CIT also offers a number of tutorial and reference documents free of charge at the CIT Help Desk and other Public IT Computing Areas. Documents are also available online at the following URL:

`http://wings.buffalo.edu/computing/documentation`

For additional help, contact the CIT Help Desk, 216 Computer Center, at (716) 645-3542, or send e-mail to *cit-helpdesk@buffalo.edu*.