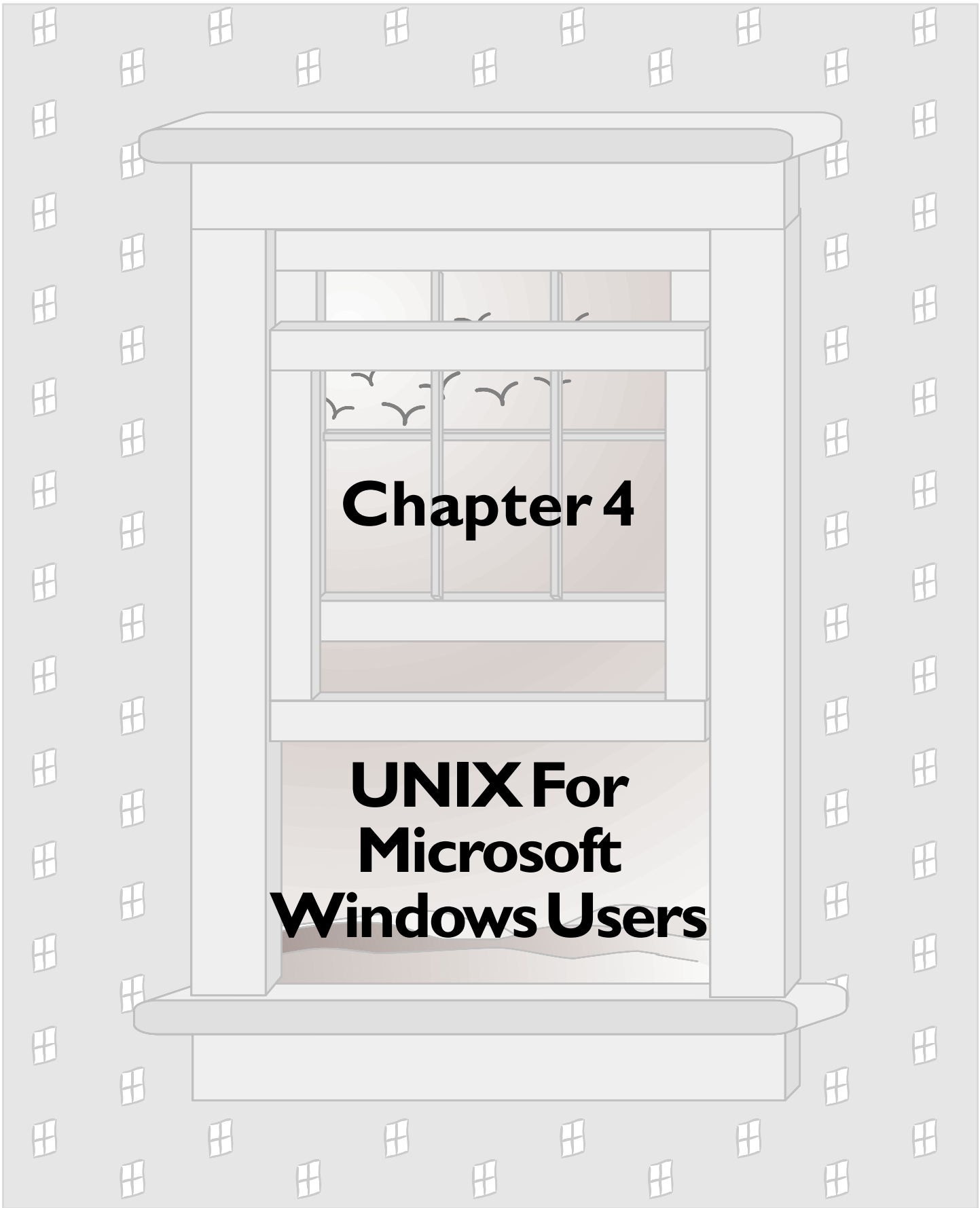

Chapter 4

UNIX For Microsoft Windows Users

UNIX gets a bad rap for being difficult to use, from the standpoint of the general user. This is unfair—after all, it's really not much more difficult than DOS....

> ***User-friendly*** – adj. *Programmer hostile. Generally used by hackers in a critical tone, to describe systems that hold the user's hand so obsessively that they make it painful for the more experienced and knowledgable to get any work done.*
>
> – Eric Raymond, *The New Hacker's Dictionary*, 1991

By implication, UNIX is a very powerful operating system, and *many* programmers (but not all: see the *Hacker's Dictionary* entry for "UNIX weenie") find UNIX to be a more technically satisfying operating system in which to program.

But truly, UNIX is not so difficult when you compare it to DOS. Most UNIX commands are no less mnemonic and no more arcane than their DOS counterparts, and in many ways the flexibility of UNIX makes things easier than DOS for the user, not more difficult. Many of the difficulties in UNIX are simply a consequence of the fact that UNIX does much, much more than DOS.

The *joy* of Wabi is that you don't need to worry much about the UNIX underpinnings—you can continue to live in your happy world of Windows "productivity applications" while still reaping the benefits of a robust multiuser operating environment that can also, by the way, run *many* powerful programs. The goal of this chapter, then, is to give you just enough information about UNIX to be dangerous (or tedious) at parties....

In This Chapter

About UNIX

UNIX is the most widely used multiuser, multitasking, general-purpose operating system in the world today. It can run on an extraordinarily wide variety of computer equipment—from PCs to microcomputers to mainframes—and is, in fact, the *lingua franca* of the world's largest computer network, the Internet.

The UNIX operating system was created in 1969 at Bell Laboratories by Ken Thompson. Rumor has it that Thomson invented UNIX so he could play games on his PDP-7 computer, which he salvaged after Bell Labs abandoned its MULTICS (the big brother of UNIX) project.

Dennis Ritchie (the C programming language deity) is generally considered to be the coauthor of UNIX. In the years 1972-74, UNIX was reimplemented almost entirely in C, which made it the first operating system that was truly portable—that is, it became relatively easy to port UNIX to different hardware platforms by recompiling its C source code. (The programmers in the audience may be cringing just now—*nothing* is that easy—but UNIX *is* a heck of a lot easier to port than most other operating systems.)

As UNIX evolved, it was embraced by the academic and scientific communities. Because of the generally experimental mind-set in those populations, and because of the wide availability of UNIX source code, UNIX was hacked and extended and tweaked and refined for different purposes, the product of which was a veritable tower of babbling operating systems. Eventually, by the mid 1980s, AT&T consolidated the various flavors of UNIX into a standard operating system. These efforts resulted first in System III, and finally in System V—the phylum from which the most popular strains of UNIX today are descended.

UNIX System V Release 4.0, released in 1989, and its ancillary System V Interface Definition (SVID), form the standard on which SunOS™, HP/UX®, IBM/AIX®, and SCO/UNIX are based. The bare-bones command-line interfaces of these systems have, in turn, been augmented by graphical, windowing environments (no, Microsoft did *not* invent such environments) like OPEN LOOK, OSF/Motif, and HP OpenVue™ (these particular environments are, in turn, implementations of the X Window System standard—see “About X Window” on page 100).

Basic UNIX Concepts

UNIX is founded on the concepts of multiple users, multiple tasks, timesharing, and interoperability. Before going any further, let's define these terms, because they are totally foreign to DOS.

- **Multiple users** – This one is easy: it means more than one user. The implications, however, are more complex. To support multiple users, an operating system must have some means of identifying those users, preventing them from getting in each other's way, and making it difficult for them to wreck each other's stuff. On the more optimistic side, it means providing tools with which multiple users can share data and programs, exchange messages, and generally apprise one another of each other's existence. Hence the concepts of logins, user names, privileges, protections, file locking, email, and so forth.
- **Multiple tasks** – Again, nominally simple: it means doing more than one thing at a time. Again, however, the implications are complex, and the details of their implementation are beyond the scope of this book. For the purposes of running Wabi, all you need to know is that, unlike Microsoft Windows, UNIX provides true *preemptive* multitasking. Program *threads* can operate independently of each other and other programs. This makes it possible, for example, for one program to update the display (using the video subsystem) while another program does a database lookup (using CPU and disk resources). By contrast, under Microsoft Windows, you would get the dreaded hourglass until one or the other activities was completed.
- **Timesharing** – The ability to allow authorized user access to programs, data, and resources (like printers, modems, and disks) on a system. Equally important is that users can gain such access *when they need it*; a user does not have to wait for other users to get off the system, within certain volume limits, before being allowed access.
- **Interoperability** – The sharing of programs, data, and resources among different machines in a relatively seamless way. UNIX's portability is also the basis for its interoperability.

At its highest level, UNIX can be divided into three primary components:

- **Kernel** – Provides core operating system functions
- **File system** – Hierarchical naming structure for files and directories
- **User shell** – The user interface through which you interact with the operating system

Of these three components, the file system and the user shell are of the greatest interest to Wabi users.

The UNIX File System

The basics of getting around in the UNIX file system are of interest to Wabi users because Wabi, Windows, and any Windows applications you install under Wabi will reside in UNIX.

Like DOS, UNIX provides a hierarchical file and directory structure, as illustrated in Figure 4-1. That is, *files* reside in *directories*, and directories reside in other directories. In Windows parlance, directories are also sometimes referred to as *folders*.

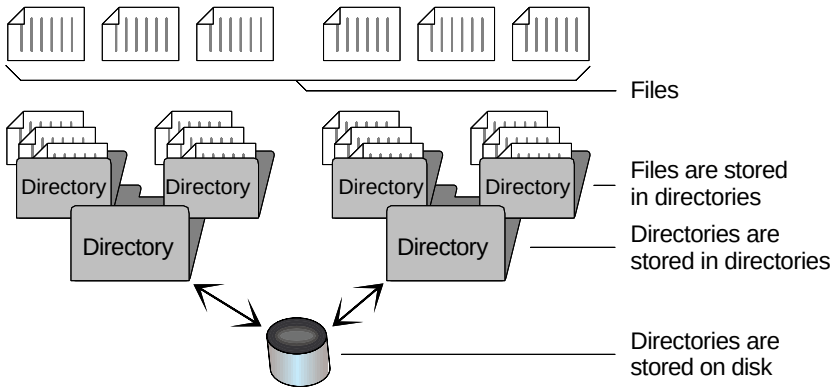

Figure 4-1 Hierarchical Relationship of Files and Directories

From a practical standpoint, there are several important differences between the DOS and UNIX file systems:

- In DOS, file names are limited to eight alphanumeric characters, plus a three character extension. For example, FILENAME.DOC. UNIX file names can be up to 255 characters.
- DOS file names are not case-sensitive—that is, forget about the SHIFT key. UNIX file names are case-sensitive; you can mix uppercase and lowercase characters.
- There are several characters you can't use in DOS file names, but which you can use in UNIX. (See Table 3-8 on page 76).
- Directory names in DOS are separated by backslashes (\); in UNIX, they are separated by forward slashes (/). For example, you could have a DOS directory named \STUFF\LETTERS. The same directory relationship in UNIX would be represented as /STUFF/LETTERS.
- The concepts of file security attributes, as provided in UNIX, are mostly not present in DOS.

These differences are explained in (excruciating) detail in Chapter 3, in the section "What's in a Name?" on page 73.

Wabi goes a long way towards smoothing out the differences between the two file systems, so you don't have to worry about them. For the most part, when running Windows applications under Wabi, you name and work with files and directories just as you would under DOS. There are two exceptions to this:

- **When you are installing Wabi** – Wabi installation requires at least some knowledge of how to get around in the UNIX file system. This requirement is minimal, however, and the installation instructions provided with Wabi software pretty much hold your hand through the process.
- **When you want to mount (connect to) a UNIX directory** – In Wabi, you mount—that is, connect to—UNIX directories so that they appear as disk drives to Windows applications. To do this, you need to understand how UNIX directories are named. Mounting directories is explained later in this chapter, and also in Chapter 5, "Using Wabi Software," and in Chapter 6, "Managing Drives."

The NFS File System

The Solaris operating environment is based on the *NFS*® distributed file system. In most cases, when using Wabi, you will be working with NFS-based files and directories. You may also, on occasion, find yourself connecting to other file systems, but NFS will probably be the most common system you will use with Wabi.

NFS is a UNIX-based distributed file system, developed by Sun Microsystems, that enables computers on a network to cooperatively access each other's files in a transparent, seamless manner. That is, from the user's standpoint, remote file systems (those that are on machines other than the user's) are indistinguishable from local file systems (those that are actually on the user's machine).

For example, instead of duplicating a set of directories on all machines on a given network, the NFS file system lets you have one common set of directories on a machine that is shared by all other systems. Each user sees that set of shared directories as local to his or her own workstation.

The file name and attribute mapping scheme used by Wabi (see "What's in a Name?" on page 73) is based on the NFS file system.

NFS Mount Points and Wabi Drives

A *mount point* is a location in an NFS directory structure at which you want to make a network connection from within Wabi. Such connections appear to Windows applications running in the Wabi environment as *virtual disk drives*—otherwise known, in Wabi parlance, as *Wabi drives*.

To Windows applications running under Wabi, Wabi drives appear to be regular PC disks—like a drive `E:` or `F:` You can treat Wabi drives pretty much the same way as regular PC disks; you can save, copy, move, rename, and delete files, and install and run applications.

For example, you could define an NFS directory named `/home/yourstuff` as a mount point, and then assign drive letter `P:` as the Wabi drive to represent that directory in Windows applications running in the Wabi environment. From Windows File Manager in Wabi, you could then click on the drive icon for `P:` to display the files in `/home/yourstuff`.

In the above example, /home/yourstuff is considered to be the root directory on the P: drive. Therefore, any subdirectories off that root are also accessible to you, provided that you have sufficient network *privileges*. For example:

Table 4-1 Sample Wabi Drive Assignments

NFS Directory Name	Wabi Drive Name
/home/yourstuff	P:\
/home/yourstuff/mail	P:\mail
/home/yourstuff/work	P:\work
/home/yourstuff/work/daysoff	P:\work\daysoff

This privileges thing is very important in the UNIX world—you may not (and probably don't) have access to all files and directories on your network. Access privileges are associated with your UNIX user account, which is described on the next page, in "Who Are You?"

Going Home

For general users, the most important, and most frequently accessed directory is your home directory. This is the directory in which you store your day-to-day data files, your email files, and many of your custom system configuration files, among other items. For example, your home directory may be named /home/president.

When viewing directory names, particularly in documentation or configuration files, you may occasionally encounter a tilde (~), and wonder what the heck it means. In UNIX, the tilde in directory paths represents your home directory. For example:

```
/home/president/letters
```

is the same as:

```
~/letters
```

Who Are You?

To support the practical implications of multiple users, UNIX provides the concept of *user names*. A user name is a unique name used to identify a user to the operating system. Every user on a UNIX system must have a *user account*, the two primary identification components of which are a user name and a unique *password* associated with that name.

You enter your user name and password in response to *login* prompts when you first try to access a system. This login process is explained in more detail later in this chapter, in "Logging In" on page 96.

After identifying yourself to the system, UNIX is then able to automatically determine or configure various operating system components for you; for example, your home directory, your PATH environment variable, default configuration files, and so forth. Your user account also has associated with it a set of *privileges* (or *permissions*), which are the degrees to which you are allowed access to files, directories, and devices on your network or your machine. UNIX privileges are explained in more detail in the next section, "May I?"

UNIX users are also usually grouped into sets of users, collectively called *user groups*. Any single user can belong to one or more user groups, the specific list of which is part of user's account information. Like individual users, each user group has associated with it a set of privileges. This makes it possible to provide and manage collective access to, say, a set of document directories.

May I?

All access to files, directories, and devices (like printers) in UNIX is closely controlled by various permissions. Of particular interest to Wabi users are file and directory permissions, which you can control and sometimes need to modify. By contrast, device permissions are usually managed by your system administrator.

File and directory permissions are based on user and group accounts, and can therefore be divided into the following three user categories:

- **User** – You
- **Group** – Anyone in any of the user groups to which you belong
- **Other (or World)** – Everyone else

You can assign the following permissions to files and directories:

- **Read (**r**)** – The file can be viewed.
- **Write (**w**)** – The file can be modified.
- **Execute (**x**)** – The file can be run (only for programs or scripts). In the case of directories, the directory can be switched to; in UNIX you can only change to directories for which you have execute permission.

You can change the permissions on your own files and directories by using the UNIX chmod command. Alternatively, from within Wabi, you can use the Microsoft Windows File Manager to change some (but not all) permissions. The mapping of permissions between UNIX and Windows is described in detail in Table 3-9 on page 77.

Briefly, to use the chmod command, enter chmod, followed by a 3-digit value (4-digit values are another story) representing user, group, and world permissions respectively, followed by a file name. Figure 4-2 illustrates how to set chmod values.

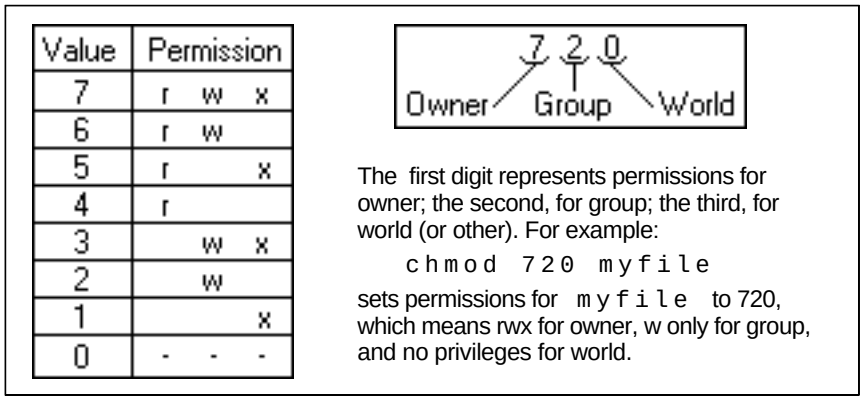

Figure 4-2 Setting chmod Values

Table 4-2 lists some common chmod values.

Table 4-2 Common chmod Values

Value	Meaning
755	Owner: rwx; Group: r-x; World: r-x
700	Owner: rwx; Group, World: no access
777	Unrestricted access to all
666	rw- access to all

Remember, in UNIX, directories are just another kind of file. You use chmod to change directory permissions the same way you change file permissions. Moreover, you cannot switch to a directory unless it has execute permissions!

umask and You Shall Receive

Your user account has associated with it default file and directory permissions. That is, every file or directory you create is assigned a default set of permissions. This default set of permissions is assigned (and can be changed) with the UNIX umask command.

The umask command is similar in many ways to chmod. The differences between the two, however, can be subtle and confusing:

- **Global versus local** – chmod is used to change permissions on an *as needed* basis; that is, you change permissions for specific files and directories. By contrast, umask is used to set *global* permissions—your umask setting is applied everywhere, to all files and directories you subsequently create. You use chmod to specifically override the umask setting on a file-by-file basis.
- **Positive versus negative** – chmod lets you specify what people *can* do to files; umask is the obverse of that concept—it specifies what you *can't* do to files. Think of umask as a filter (or a mask); the holes in the mask (values not set) are the permissions that are allowed to pass through.

Figure 4-3 illustrates how to set umask values. Compare these values to those shown for chmod in Figure 4-2 on page 88.

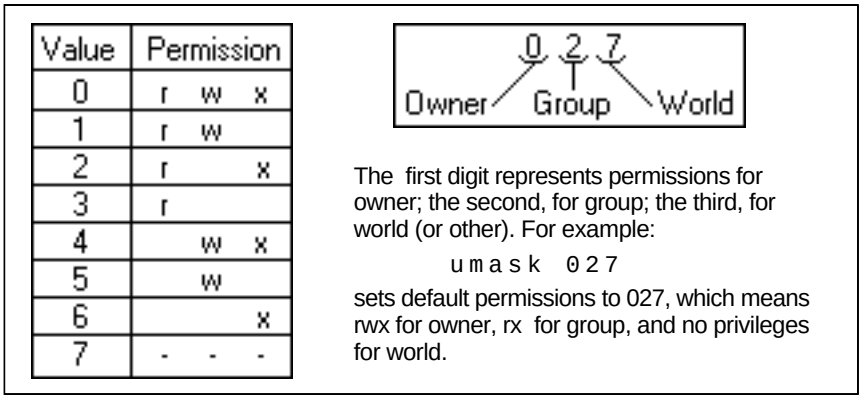

Figure 4-3 Setting umask Values

Table 4-2 lists some common umask values.

Table 4-3 Common umask Values

Value	Meaning
022	Anyone can read and execute a file; only the owner can write; for a directory, anyone can list the contents.
077	No access for anyone other than the owner.
000	Unrestricted access all.

Some Common DOS and UNIX Commands

Now that you've been sufficiently terrified by the concepts of chmod and umask, it is time to assuage your dismay by explaining some common DOS commands—with which you should already be comfortable—and their UNIX counterparts.

If you should ever find yourself stranded at a UNIX command prompt outside of Wabi, refer to Table 4-4, which lists DOS commands and their UNIX equivalents. Note that this is not an exhaustive list; just some of the basic commands are listed here.

Table 4-4 Common DOS and UNIX Commands Compared

DOS Command	UNIX Command	Comments
attrib	chmod or umask	Change the attributes of files or directories; note that many DOS attributes have no UNIX equivalents, and vice versa. Refer to Table 3-9 on page 77 for more information.
cd	cd or chdir	Change directories.
chkdsk	df -k fsck	Display disk statistics; fsck, for disk repair, is run automatically at bootup.
cls	clear	Clear the screen.
copy	cp	Copy files and directories.
del	rm	Delete files; also directories in UNIX.
dir	ls	Display the contents of a directory.
edit/edlin	textedit or vi	Edit text files; texedit is Solaris only.
find	grep	UNIX find finds files; grep finds strings.
format	format	Format diskettes and hard drives.
help	man	Get help on a specific command.
mkdir (md)	mkdir	Create new directories.
path	env or setenv	Display or set PATH environment variable.
print	lpr or lp	Print files.
rename (ren)	mv	Rename files; mv lets you rename directories or move to another directory.
rmdir (rd)	rmdir or rm -r	Remove directories.
type	more or cat	Display the contents of a file; more displays one screen at a time; cat displays continuously.
ver	uname -sr	Display operating system version number.

Getting Help

As described briefly in Table 4-4, you can display help text for most UNIX commands by using the `man` command. To use `man`, enter the `man` command followed by the command for which you want help. For example:

```
man grep
```

displays help text for the `grep` command.

You can also use `man -k`, followed by a keyword, to display all commands associated with that keyword. For example:

```
man -k copy
```

displays a list of all help topics that have anything to do with copying.

Climbing Into Your Shell

As mentioned at the beginning of this chapter, the second primary component of the UNIX operating system of interest to Wabi users, after the file system, is the *user shell*. The user shell is the face that UNIX puts on when people are around. In its raw, even *naked* state, UNIX is not all that good-looking. Similar to DOS, you get a bare, character-mode command line that doesn't say much—but will quickly do what you tell it to if you are specific, literal, and don't screw up.

When UNIX gets its clothes on, however, it can be a sartorial wonder, interface-wise. The range of interfaces for UNIX—graphical, textual, windowing and otherwise—is impressively (some would say confusingly) large and customizable. These interfaces can generally be divided into two categories:

- Character-mode command lines
- Graphical, mouse-driven windowing environments

In terms of character mode command-line interfaces, the most common are the C, Korn, and Bourne shells. Again, these are roughly similar in appearance to the common DOS command line, but are, in fact, far more powerful. For example, among other things, the C shell provides a *scripting* language with a syntax reminiscent of the C programming language. This lets you create scripts (analogous to DOS batch files) that can contain logical flow controls, nested functions, variables, and so forth—much more advanced than their DOS counterparts. Moreover, you can run one shell from within another shell of a different type; for example, you could start a Korn shell from within a C shell. When integrated with a multitasking windowing environment like OPEN LOOK, you can run different or multiple command shells simultaneously, each in its own window.

In terms of graphical interfaces, numerous environments are available. For example, OpenWindows—the graphical environment included in the Solaris package—is based on the OPEN LOOK windowing system. OPEN LOOK, in turn, is an implementation of the X Window System, which is a network-based, low-level windowing system developed at MIT in 1984 (see page 100). Similarly, Hewlett-Packard provides OpenVue, which is also derived from the X Window system.

Wabi software is an X Window application, and utilizes the X Toolkit (similar to the Microsoft Windows API). This makes Wabi software especially portable between various X-based UNIX environments (see "Wabi Operating Environments" on page 17). For example, the SunSoft version of Wabi software is compiled to run in the OpenWindows environment, using the OPEN LOOK window manager. The X Window System is explained in more detail later in this chapter, and in Chapter 2, "How Wabi Works."

Basic Networking Concepts

Like DOS, UNIX can run on standalone PCs, without being connected to any network. Unlike DOS, UNIX was designed for and shines in a networked environment. Being a multiuser operating system, UNIX provides many network utilities and resources; DOS, by contrast, is totally reliant on other programs and add-ons for its network support.

More often than not, when you use UNIX, you will be connected to a network of some sort. With this in mind, it may be useful here to explain a few basic networking concepts, as they apply to Wabi users.

What Is a Network?

In computing, a *network* refers to hardware devices, such as two or more computers, that are connected by communications hardware and software so they can share resources.

- Networking software running on each networked device handles the communications process between hardware devices, and provides the means through which the user or other application programs can work with the resources being networked.
- The software rules governing network communications are referred to as *protocols*; Solaris software (and hence Wabi software) is based on the widely available *Internet Protocol* (IP). This is often implemented in a protocol stack referred to as *Transfer Control Protocol/Internet Protocol* (TCP/IP). Wabi uses TCP/IP, but also supports the Windows Sockets (Winsock) networking interface, which allows some applications, such as Lotus Notes, to communicate through the network directly.
- The hardware connection between networked devices commonly consists of Ethernet® or TokenRing® cabling and interface *ports*, but may also include modems on telephone lines, serial cables, or wireless devices.

- Commonly shared resources on a network include printers, application programs, data, and data storage devices. For example, a person may be at a workstation that does not have a hard disk. That person can use the network facilities to store data files on a *remote* machine on the network.

Local Versus Remote

The term *local* refers to the computer at which you are physically located—that is, *your* computer—or any peripheral device, such as a printer, that is directly attached to your computer. Devices on the network, other than your local computer and directly attached peripherals, are said to be *remote* or *network* devices. Thus, a printer attached to a remote computer on the network is referred to as a *remote printer* or *network printer*; a printer attached directly to your computer is a *local printer*. A remote computer that provides data file storage and retrieval services for your local machine is called a *remote file server* or a *network file server*. Figure 4-4 illustrates the relationships between some basic network components.

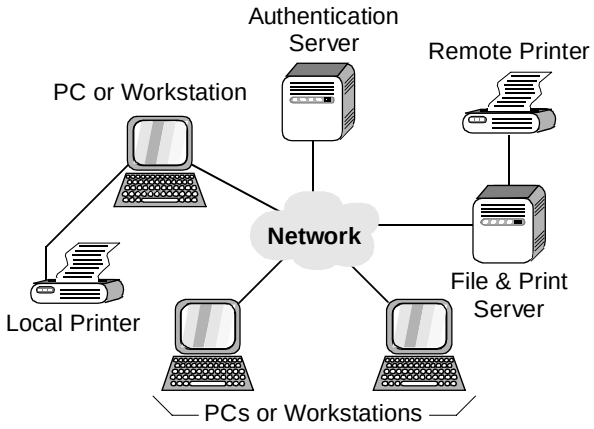

Figure 4-4 Basic Network Components

Servers and Clients

In networking, a *server* (also called a *host*) is any computer that performs services for other computers on the network. Computers that use any of these services are called *clients*. For example, a computer that provides data file storage and retrieval services is called a *file server*; a computer that accesses those files is a *client*.

Theoretically, any computer on a network can act as a server for other computers. Conversely, some computers may provide more than one kind of service. The specific design of any given network depends on the available hardware and software resources. For example, even though there may be a network-accessible printer directly attached to your computer, you may not want to have a lot of network traffic directed through your computer to access that printer.

Logging In

Logging in (also *logon* or *login*) refers to the process by which you identify yourself to a given computer or network system, usually by specifying your *user name* and *password* in response to a login *prompt* when you attempt to connect to a server. Whether your UNIX system is connected to a network or is a standalone machine, you must log in before doing anything else on the system.

For example, the first prompt you see after booting a Solaris-based workstation is:

```
login:
```

After entering your user name, you are next prompted to enter the password associated with your user name. In UNIX, you must have a specific user name—you can't just enter any name—and a unique password, usually assigned by a system administrator, before being allowed access to the system.

When you log in, the system usually runs a login script of some sort, which sets up your default environment, loads programs, sets configuration parameters, and so forth—similar in many ways to the AUTOEXEC.BAT and CONFIG.SYS files under DOS.

Authentication Servers

On most (but not all) NFS-based networks, the machine handling the login process is referred to as an *authentication* (or login) server. Authentication servers contain password databases and other user information files that describe the devices and files to which you are allowed access.

For example, an authentication server may contain a list of other servers to which you are allowed to connect. After you log in through an authentication server, the network becomes "aware" of your user name and the privileges associated with your user account; you can then connect to other authorized servers and devices, if any.

User names, passwords, and privileges are usually assigned by a system administrator, who is responsible for maintaining the operability and security of one or more machines on the network.

The root User

The root user is the user name of the *big cheese* on a UNIX-based computer—he or she has *complete* access to the machine, for better or worse, and can perform system-level maintenance or destruction. The root user is a *machine*-level, rather than network-level, concept; that is, every machine has one root user. More to the point, the root user on one computer is not the same root user on another computer or on the network as a whole.

When you log in as root on a given machine, and then enter the appropriate root user password, you gain access to every file on that machine. This is useful for system maintenance and configuration chores, but is not used for everyday work on your system. For example, installing or removing Wabi software requires that you log in as root. When you actually use Wabi software, however, you should log in using your regular user name (that is, not root).

nobody Is Somebody

In the world of computers, where counting starts with the number 0, rather than 1, it should not be surprising that in UNIX, nobody is somebody. That is, you can use the name nobody to log in to a network and, in most cases, be allowed a limited level of network access.

The user name nobody is the name to use when you do not or cannot log in to a network through an authentication server. Usually, however, you cannot get much work done when you are logged in as nobody—for example, you might not be able to view or edit your own files.

The specific privileges granted to nobody are assigned by your system administrator, and can vary widely from system to system.

Naming Network Drives

Each hardware device on a network must somehow identify itself to the network software before any type of network communications—such as logins, printing, or file transfers—can take place. Hardware devices are identified at the software level by at least two different kinds of labels:

- **Server (or host) name** – A unique alphanumeric name assigned to a network device, such as a workstation or file server. Analogous to user names, server names provide a convenient way to refer to hardware devices.
- **Internet address** – Commonly referred to as an *IP address*; a unique 32-bit, dot-separated, numeric address assigned to each hardware device on the Internet. An *internet* is a collection of networks connected in such a way that they can act as a single, virtual network; the *Internet* (with a capital "I") is the largest internet in the world.

For example, a file server on a network could be named `bigfilemachine`, and be assigned an IP address of `129.148.20.125`. Server names and IP addresses are assigned by your network administrator.

Domain Name System

Internet addresses are cryptic and unwieldy, and not at all convenient to remember. As an alternative, the Internet provides a hierarchical naming system called the *Domain Name System* (DNS), which is not only easier to use than 12-digit numbers, but offers a number of system management features that are of interest to system administrators.

From the standpoint of the user, DNS provides a mechanism for resolving —or associating—mnemonic machine names (that is, names with letters in them) with their corresponding numeric Internet addresses.

DNS is based on the concept of Internet *domains*, which for our purposes here can be loosely defined as a naming scheme for subgroupings (or *subnetworks*) within a network. For example, a university might include the following domain-name definitions in the network environment:

```
profs.history.princeton.edu
grad.history.princeton.edu
admin.bursar.princeton.edu
housing.bursar.princeton.edu
```

In this example, `princeton` is a subnetwork (or subnet) in the `edu` domain; `history` and `bursar` are subnets in the `princeton` domain, `profs` and `grad` are subnets in the `history` domain, and `admin` and `housing` are subnets in the `bursar` domain. A file server named `bigfilemachine` in the `grad` domain would have the following fully qualified DNS name:

```
bigfilemachine.grad.history.princeton.edu
```

Device Names and Wabi

When mounting network drives through the Wabi Configuration Manager (see Chapter 6, “Managing Drives”), you may sometimes need to specify network mount points (directories) that are not local to your machine, or not even local to your network. In such cases, you would use DNS names for specifying the server to which you want to connect. For example, to connect to a directory named `/public/library` on the server `bigfiles.grad.history.princeton.edu`, you would use the name:

```
bigfilemachine.grad.history.princeton.edu:/public/library
```


About X Window

As mentioned earlier, Wabi is based on the X Window System. This makes it possible to compile Wabi to run on a wide variety of platforms, under a variety of X-compliant windowing systems.

The X Window System is a network-based graphical windowing system initially developed in 1984 at the Massachusetts Institute of Technology. It has been adopted as the low-level windowing standard by a consortium (the X Consortium) of industry leaders like Sun, IBM, Hewlett-Packard, DEC, and AT&T. X11R5 (X Version 11, Revision 5), the version on which most popular windowing systems (like OPEN LOOK) for UNIX are based, was released in 1991. The most recent version, Version 11, Revision 6 (X11R6) was released in May of 1994.

X Window (*X* for short) is based on a *client-server* model, in which application programs (*clients*) communicate with a display device indirectly via a display program (the *server*). Client programs do not need to run on the same machine as the server program, which makes the X Window system particularly suited for distributed computing environments.

X Window is a *low-level* windowing standard in that it is a set of specifications defining rudimentary mechanics for window controls and behavior. Most X-derived window managers add higher-level, platform-specific features—that is, the particular "look and feel" of dialog boxes, mouse actions, and menu behavior. For example, the Solaris OPEN LOOK window manager and HP's OpenVue are both based on the X Window standard, but each has its own unique feature and design elements, and the two environments look fairly different from each other.

X Window Terminology

From the standpoint of the Wabi user, what you see from within Wabi is Microsoft Windows—not UNIX, and not X Window. Because Wabi utilizes X Window resources, however, you should be aware of some potentially confusing terminology and interface quirks you may encounter.

Clients and Servers

Perhaps the most confusing terminology twist in the X Window system is the use of the term *client* and *server*. In X, the concepts are basically flip-flopped from what you might expect in a networking context, as described on page 96. Specifically, in X:

- An X *server* is a process that provides windowing services to an application, or client process. For example, the OPEN LOOK window manager running on your local computer is an X display server. In this model, the client and the server can run on the same machine or on separate machines.

  X servers handle user input from the keyboard and mouse, and send it to a client application, which is usually running remotely on a host on the network. The client, in turn, sends the server the contents of the application window and other program responses for display on the server screen.

- An X *client* application is a program that receives its input and/or displays its output on a local X terminal or workstation (the display server in this scenario). On “dumb” X terminals, most X client applications run on a remote client host—that is, the bulk of the processing performed by the program occurs remotely—with only input/output occurring locally on the display server. On more robust workstations, many client applications run locally on the display server. In the case of Wabi, it is usually run locally on your machine.

Confused yet? Not to worry—from your standpoint, Wabi runs locally on your machine, displays locally on your machine, and accepts mouse and keyboard input locally on your machine—the concept of Wabi X clients and servers is moot. If, however, you want to get tricky, and try out some of the X remote display capabilities with Wabi, refer to the instructions in Chapter 14, "Tips and Tricks," starting on page 288.

For a nice description of the hows and whys of the X Window system, written for general users, read the *X Window System User's Guide* (Valerie Quercia and Tim O'Reilly; O'Reilly & Associates, Inc., 1990.).

Resources

X Window and the applications running under X Window get their configuration information about each other—details about the window manager being used, colormaps, fonts, and many other items—from X *resource* files. These resource files are similar to the `.INI` files used by Microsoft Windows and Windows applications.

For example, applications tell X Window how their dialog boxes should look and behave—such as what's in them, when they should be displayed or closed, their relationships to other dialog boxes—via a series of *hints* registered in one or more application resource files. Similarly, the `.Xdefaults` file sets workspace color, mouse input characteristics, and defines window borders and styles, among other things, which are used by the application.

Colormaps

In the X Window System, *colormaps* are tables of information that are loaded into memory, and which describe all the colors that a given display can handle. X-based applications can also provide their own colormaps, which can override the default colormap used by a given hardware device. Multiple colormaps can be loaded into memory, but only one colormap can be actively displayed at a time.

Similarly, Microsoft Window provides a color mapping scheme that is hardware dependent—you tell Windows what kind of display you are running, and it in turn gives to Windows applications only those colors it thinks the display hardware can handle.

Wabi must translate the Windows color mapping schemes to X colormaps. Where possible, Wabi uses the default X colormap; when necessary, it will swap in a custom *virtual colormap* as needed. This can sometimes cause excessive color flashing when switching between Windows applications running under Wabi and other non-Windows applications running on your Solaris desktop. You can mitigate this problem, however, by changing various settings in your `WIN.INI` file.

Information about these `WIN.INI` settings, as well as a more detailed explanation of colormaps and how Wabi uses them, are provided in Chapter 11, “Managing Colors,” starting on page 264.

Font Servers

In the X Window System, a *font server* is a machine on the network containing font files used by local display servers. Display servers on the network download fonts as needed from the font host. Font files can be quite large; therefore, some networks may have a machine that is dedicated to the task of font serving.

In X Window, a *font server* is also a software process that manages the distribution and display of X fonts according to the X *font service protocol*. Wabi software includes its own font server, `wabifs`, which interacts with an X server via the font service protocol.

One of the most important new features of Microsoft Windows 3.1 was the addition of TrueType font technology. TrueType is a scalable outline font display and print technology, bundled with Microsoft Windows, which provides a WYSIPMWYG (What-You-See-Is-Pretty-Much-What-You-Get) way to view fonts on screen and print on your printer.

Wabi 2.0 used X font files, and shipped with several bitmapped and TrueType fonts. Wabi 2.1, however, bypasses the X font file resources and uses TrueType and Windows system fonts exclusively.

Refer to Chapter 12, “Fonts and Wabi,” for more information about `wabifs` how Wabi works with fonts.

Don't Worry, Be Wabi

For people who are coming from the worrisome world of DOS, with its memory limitations, real mode and protected mode switching, IRQ conflicts, DMA channel selection, and so forth, there are many things about which you no longer need to concern yourself when using Wabi.

True, UNIX can be complicated in its own ways—enough to engender a classic nerd culture—but also true is that most people have a system administrator or three to take care of all that stuff(!).... This section describes those DOS sorts of things about which you no longer need worry when running your Windows applications under Wabi software.

AUTOEXEC.BAT and CONFIG.SYS

Wabi uses `AUTOEXEC.BAT` and `CONFIG.SYS` files primarily to fool Windows applications that modify these files during installation into thinking that they have done their deed and they can end the installation successfully.

The only things Wabi looks at in `AUTOEXEC.BAT` are the `PATH` statement and any other environment variables (like `SET VAR=`*value*) put there by an application—Wabi software simply ignores *everything* in `CONFIG.SYS`.

Extended and Expanded Memory

Windows under DOS goes a long way toward circumventing the problems associated with DOS extended, expanded, and conventional memory constrictions (as explained on page 70). The problem is getting out of DOS and into Windows in the first place.

Well, forget about it. In UNIX, there are no such things as extended, expanded, and conventional memory.

Memory Managers

Because extended, expanded, and conventional memory are moot points under Wabi, you don't need to even think about memory management programs like MEMMAKER, QEMM, EMM386.EXE, and the like.

SMARTDRV and Other Disk Caching Programs

Microsoft Windows relies on DOS for its input/output (I/O) services, such as access to disk read/write functions. Similarly, Windows under Wabi gets its I/O services from UNIX. Consequently, disk caching programs like SMARTDRV.EXE are unnecessary. Even better, Wabi does everything in true 32-bit mode.

Device Drivers

DOS device drivers, which are usually loaded under DOS via a CONFIG.SYS file, are not used under Wabi. As mentioned earlier in this section, all statements in CONFIG.SYS are ignored by Wabi. On the negative side, if your application requires such a driver, like a scanner driver, chances are it will not work under Wabi.

Wabi does not work at all with virtual device drivers. Programs that use virtual device drivers are not compatible with Wabi. See "Supported Applications" on page 6 for more information.

386 Enhanced Mode

Wabi software runs in Windows 386 Enhanced Mode. That's it. Forget about Standard Mode. Moreover, Solaris, which provides a true 32-bit processor environment (even on Intel platforms), does a lot of things better than Windows in 386 Enhanced Mode under DOS; for example, more robust multitasking and more DOS sessions (with an appropriate DOS emulator, like SunPC or Merge).

Wabi also has no use for the settings managed with the Windows 386 Enhanced Mode Control Panel tool—this set of Control Panel functions is not installed when you install Windows under Wabi with the Wabi Windows Install program.

What, you may ask, happens to all those nifty things like swap files and 32-bit disk access? The fate of these functions is described below.

Virtual Memory

The virtual memory setting in the 386 Enhanced Mode Control Panel tool lets you specify the type and size of a disk-based *swap file*. Such swap files are used by Windows to simulate physical RAM memory when none is really available.

UNIX provides similar swap file services for Wabi, but does it better than Windows under DOS. UNIX swap space is more flexible and performs better than Windows swap files.

32-Bit Disk Access

32-bit Disk Access is also a setting in the 386 Enhanced Mode Control Panel tool that is irrelevant under Wabi.

Solaris, on both Intel (*x*86) and RISC (SPARC) platforms provides true 32-bit disk access. Under DOS, Windows uses 32-bit disk access to avoid a couple of processor mode switches between real mode and protected mode when performing disk reads or writes. Under UNIX, such mode switching is unnecessary, because disk I/O functions are miles away from the world of DOS. Windows 32-bit disk access also does not work with all disk controllers—specifically any controller that works with disk sectors of 1024 bytes. UNIX controllers have no such problems.

Device Contention

Windows under DOS is not very clever when it comes to managing contention between serial ports. For example, if two applications are competing for the same port, Windows can sometimes freeze to the point of requiring a hard system reset—it all depends on how well behaved the Windows applications are.

Wabi, because it runs through UNIX, handles such contention more gracefully, and simply tells the offending applications to please stop talking so loudly and wait their turn. When the port becomes available, the UNIX operating system releases it to Wabi.

PIF Files

PIF files exist in the DOS/Windows environment so Windows can figure out how to futz with conventional memory and OEM-hardware-mapped, character mode displays in DOS applications. Because Wabi does not actually run DOS applications itself, but rather spawns a separate DOS emulation program like SunPC or Merge, Wabi does not have to do any such futzing, and PIF files are unnecessary. DOS applications run better under such emulation because they are running in DOS, rather than in DOS-in-Windows.