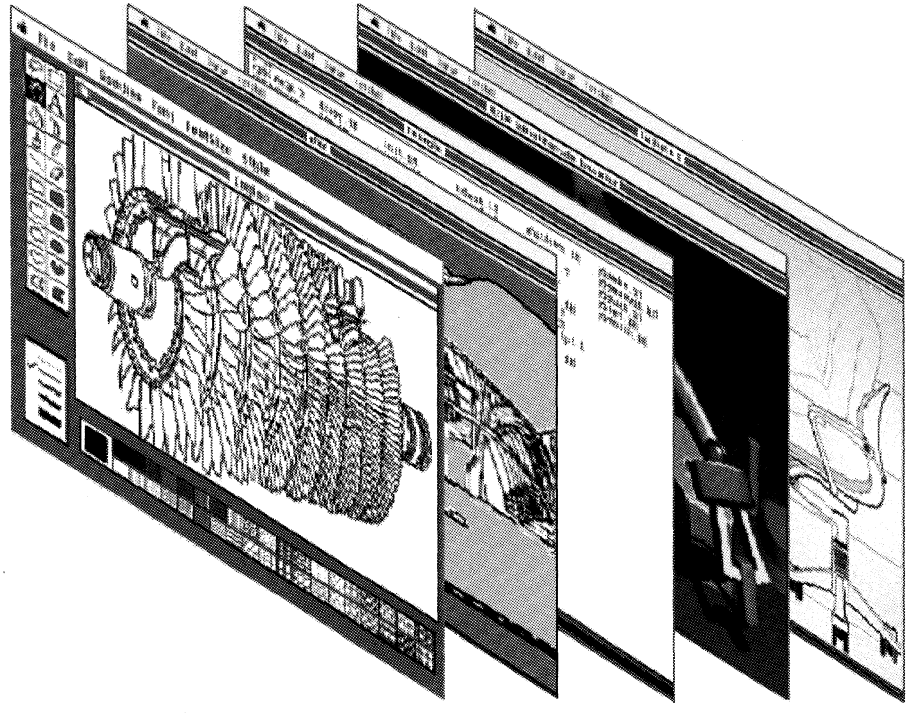
Apple® Getting Started With A/UX™



Copyright

Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE AND ITS SOFTWARE SUPPLIER MAKE NO WARRANTIES OR REPRESENTATIONS, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY,**

PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD AS IS, AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.

IN NO EVENT WILL APPLE OR ITS SOFTWARE SUPPLIER BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple and its software supplier shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Getting Started With A/UX

Contents

Preface

Conventions Used in This Manual

Throughout the A/UX manuals, words that must be typed exactly as shown or that would actually appear on the screen are in `Courier` type. Words that you must replace with actual values appear in *italics* (for example, *user-name* might have an actual value of `joe`). Key names appear in CAPS (for example, RETURN). Special terms are in **bold** type when they are introduced; many of these terms are also defined in the glossary in the *A/UX System Overview*.

Syntax notation

All A/UX manuals use the following conventions to represent command syntax. A typical A/UX command has the form

`command` [*flag-option*] [*argument*]...

where:

`command`	Command name (the name of an executable file).
flag-option	One or more flag options. Historically, flag options have the form `-`[*opt*...] where *opt* is a letter representing an option. The form of flag options varies from program to program. Note that with respect to flag options, the notation `[-a][-b][-c]` means you can select one or more letters from the list enclosed in brackets. If you select more than one letter you use only one hyphen, for example, `-ab`.
argument	Represents an argument to the command, in this context usually a filename or symbols representing one or more filenames.

[]	Surround an optional item.
...	Follows an argument that may be repeated any number of times.
`Courier type`	anywhere in the syntax diagram indicates that characters must be typed literally as shown.
italics	for an argument name indicates that a value must be supplied for that argument.

Other conventions used in this manual are:

<CR>	indicates that the RETURN key must be pressed.
^*x*	An abbreviation for CONTROL-*x*, where *x* may be any key.
cmd(*sect*)	A cross-reference to an A/UX reference manual. *cmd* is the name of a command, program, or other facility, and *sect* is the section number where the entry resides. For example, `cat`(1).

Note to the System Administrator

This note is for the system administrator and/or the network administrator. (If you are the only user on your system, and are not part of a local area network, you will need to learn about system administration; you should go through this tutorial and then read *A/UX Local System Administration.*)

The rest of this manual is written for the first-time A/UX user. It doesn't assume any other users on the system, although that possibility is acknowledged where it's appropriate. Chapter 7 assumes that `/users` has not been mounted remotely using NFS. If you distribute this manual to new users, please note that all chapters and exercises in this manual assume that they are using the `start` account.

Using this manual

The A/UX standard distribution includes a login account (`start`) and a set of files intended to accompany this tutorial.

The `start` account's home directory is `/users/start`. `/users/start` contains

```
.
..
.profile
animals
letter1
letter2
names.land
names.water
sample
setup
```

Note that the subdirectory `animals` is empty.

These files must be restored before each new user begins the tutorial. The shell script `/users/start.bak/setup` copies the contents of

`/users/start.bak` to `/users/start`, thus restoring the files and directories of the `start` account and ensuring that each new user gets a clean copy.

To restore the files, type

```
sh /users/start.bak/setup
```

We have incorporated this into the beginning of the tutorial for each user to do him or herself. (See "Getting Reading To Use This Tutorial" in Chapter 2.)

Environment

To make this tutorial as precise as possible, the following assumptions have been made as to environment:

Hardware — It's assumed you're using the Macintosh II console device, a Macintosh running MacTerminal in VT100 mode, or a VT100 terminal. Like other UNIX® systems, A/UX supports a variety of terminals, but the results of running the tutorial on them can't be predicted.

Shell — The tutorial assumes you're using the Bourne shell. Once you've completed the tutorial, you may wish to give your users the C shell or the Korn shell.

Mail — The `mail` tutorial in this manual uses `mailx`, not `/bin/mail`. `mail` is aliased to `mailx` in the `/etc/profile` file. For more information about `mail`, please see `mail`(1) and `mailx`(1) in *A/UX Command Reference*.

You may also wish to review Table 1-1 in "Before You Begin Using This Tutorial" and advise users of any changes specific to your installation.

Setting up user accounts

After a new user has worked through this tutorial using the `start` account, the user should receive his or her own account.

Chapter 7 of this manual tells you how to set up a new user's account. This chapter, like the other chapters in this manual, applies to the novice user who is his or her own system administrator. We recommend that you remove Chapter 7 before distributing this manual to other novice users on your system.

Chapter 1
Before You Begin Using This Tutorial

Contents

Tables

Chapter 1

Before You Begin Using This Tutorial

1. Your equipment

This manual, *Getting Started With A/UX*, is designed to give you a taste of what A/UX has to offer. It assumes that you have already assembled your machine and have it running. At the minimum, you should have a standard Macintosh II with the Apple keyboard. This is the main terminal (that is, keyboard and screen) of your system. It is also called the **console device.** You may also have one or more additional terminals. This tutorial assumes you are using one of the following:

- the console device with Apple keyboard or Apple Extended keyboard,
- a Macintosh running MacTerminal in VT100 mode, or
- a VT100 terminal

You may use a different terminal, but this could mean you'll need to press different keys from those specified in this manual. See Table 1-1 for a description of the key names we use in this manual and their equivalent keys on the three devices named above.

Since there is no ESCAPE key on the Macintosh keyboard, MacTerminal has redefined the key at the upper-left of the keyboard (labelled with a tilde and a back quote) to be ESCAPE. When you wish to type ESCAPE, simply press the back quote key. (Note that the sequence Command-[is also equivalent to ESCAPE.) When you wish to type a back quote, you must hold down the Command key and press the back quote key. When you wish to type a tilde, you must hold down both the Command key and the Shift key while you press the back quote key.

Table 1-1. Keynames and their equivalents on your equipment

Keyname in this manual	Equivalent for your A/UX terminal		
	Console device	Macintosh	VT100
CONTROL	control	Command	CTRL
ESCAPE	esc	` (backquote) or Command-[	ESC
RETURN (or <CR>)	return	Return	RETURN
interrupt	control-c	Command-c	CTRL-c
quit	control-\	Command-\	CTRL-\
kill	control-u	Command-u	CTRL-u
erase	delete	Backspace	DELETE

2. Using this manual

This manual is designed as an introduction to A/UX for people who have not used an operating system like A/UX before.

Before you begin using this manual, you should see the following message on your screen:

```
login:
```

Getting Started With A/UX assumes that you've used a computer before, but that you may never have used a system like A/UX. The manual takes you through the major nonprogramming facilities of A/UX, and once you've read it and done the exercises, you should be ready to tackle A/UX's more advanced capabilities.

However, *Getting Started With A/UX* is intended just to get you started. It doesn't have the more in-depth information you'll need once you're more familiar with the system; that's what the other manuals (such as *A/UX Text Editing Tools* and *A/UX Command Reference*) are for.

Getting Started With A/UX has 7 chapters. Chapter 1 is this introduction.

Chapter 2 takes you through the login and logout procedures. It explains what's going on at each step and tells about the messages you may see.

Chapter 3 explains how commands are structured, how you enter commands, and how you can vary them. Its purpose is to give you the basics of using A/UX to do what you want. It doesn't, however, explain any theory, nor does it give you the complete rundown on all possible commands. You can find a complete listing of A/UX commands in *A/UX Command Reference*.

Chapter 4 gives you a brief overview of A/UX's "tree-structured" file system and tells you how to move around in it. Chapter 4 also explains some of the more helpful commands you can use to manipulate files and file permissions. Its purpose is to give you some experience in finding out about your system and how to use it.

Chapter 5 is a tutorial in the use of the `vi` text editor. The tutorial has four lessons that acquaint you with the most frequently used commands in `vi` while you create and edit a file.

Chapter 6 describes how to use the `mail` facility, including reading mail and sending messages.

Chapter 7 is for the A/UX system administrator. It explains in detail the process of setting up a user account with A/UX.

Chapter 2
Starting and Ending Your A/UX Work Session
Logging In and Logging Out

Contents

Chapter 2

Starting and Ending Your A/UX Work Session

Logging In and Logging Out

1. Starting your session

You start working with A/UX by **logging in** to the system. Logging in tells the system you're there and allows it to verify who you are. Log in by typing a name next to the **login** prompt:

```
login:
```

on your screen. You can log in whenever you see this prompt.

1.1 Logging In

To log in, type your "login name," then press RETURN. The A/UX system has an account with the files and directories you need for the exercises in this manual. The login name for this account is `start`, so type

```
start
```

next to the login prompt (followed by RETURN). (If your A/UX system has many users, your login name may be different. See your system administrator.) Note that the login name is in lowercase letters. If you use uppercase letters to log in, the A/UX system will also use uppercase letters until you log out and log in again.

After you've pressed RETURN, A/UX asks you for a password:

```
Password:
```

Type

```
my.passw
```

and press RETURN. To preserve privacy, nothing appears on the screen as you are typing `my.passw`. When you press RETURN, however, A/UX reads the password and, if it's correct, you're logged in. (If your system has many users, your password may be different).

Next the system prompts you for a terminal type

```
TERM = (mac2)
```

If you are using the console device, simply press RETURN. If you are using MacTerminal or a VT100, type `vt100`.

Then the shell prompt (a dollar sign) will appear on the next line on the screen, indicating that the system is ready to accept your commands.

> *Note:* See your system administrator if you need additional assistance.

1.2 If you make an error

If you make an error while you're typing your login name, you can type the *interrupt* character. (See Table 1-1 in "Before You Begin Using This Tutorial" for the key sequences applicable to your keyboard.) You'll get a new line (without any prompt) so you can type your login name again.

If you make an error while you're typing your password, type the *interrupt* character. You'll get the login prompt again and you can start over.

If you make a mistake entering your login name or password, press RETURN repeatedly until you see

```
Login incorrect
login:
```

You can then start over.

2. Ending your session

When you are finished working with A/UX, you can **log out** of the system by typing `exit` followed by RETURN.

After you `exit`, you again see the message

```
login:
```

which indicates you have logged out successfully.

3. Getting ready to use this tutorial

A/UX includes a set of files that accompany this manual, so that you can follow along with the examples given here. Before you can gain access to these files, you must run a special command we have supplied, which will set up these files properly for you.

To set up your files, first log in as `start` as shown above. When the dollar sign prompt appears, type

```
sh /users/start.bak/setup
```

and press RETURN. When the dollar sign prompt appears again, you are ready to go!

Chapter 3
Using A/UX Commands
The Shell

Contents

Figures

Chapter 3

Using A/UX Commands

The Shell

1. What is the shell?

The **shell** is the A/UX utility that accepts your commands, interprets them, and passes them on to the system for execution. The dollar-sign character (`$`) at the beginning of each line is called the **shell prompt.** The shell prompt is displayed on the terminal to show that the shell is waiting for input from you. You can change the prompt to be any string of characters; it is a dollar sign (`$`) by default. Whenever the shell prompt is the first character on a line, you may enter an A/UX command on that line. When you enter commands, the shell interprets them and arranges for the commands to be carried out.

When you log in to the `start` account, you are automatically using the Bourne shell. Later, when you become more familiar with your system, you can use one of the other shells if you prefer. See *A/UX User Interface* for more information about the shells.

2. How to enter a command

You can enter an A/UX command any time you see the shell prompt:

```
$
```

Enter the command by typing its name (and any other information required), then press the RETURN key. For example,

```
$ date
```

After you press RETURN, the shell interprets your command and sends all the required information to the A/UX system. The system executes the command and, when execution is complete, prints another shell prompt on the next line. When you see the next shell prompt, the system is ready for you to enter another command.

The command you enter may send some output to your screen. For example, you might see

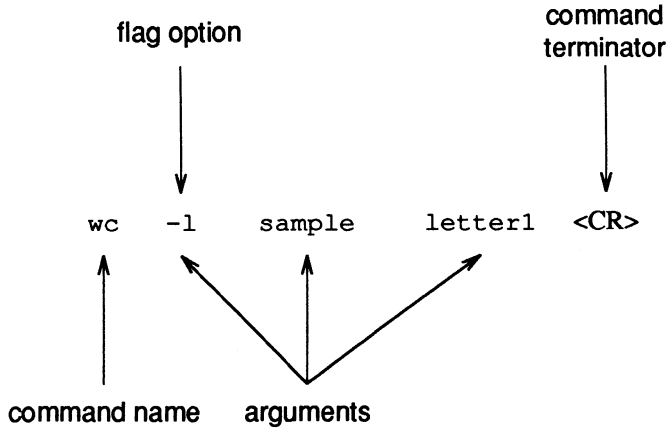
```
$ date
Wed Sep  10 12:05:09 PDT 1986
```

There are also A/UX commands that are interactive programs, such as `vi` and `mail` (see the `vi` and `mail` tutorials, Chapters 5 and 6 in this manual). **Interactive programs** suspend the shell for as long as you use them. When you exit the interactive program, you see the shell prompt again.

2.1 The parts of a command

An A/UX command may be a single word, such as the `date` command. Other commands have **flag options** that you may type after the command name. A flag option is an argument, included on the command line and preceded by a hyphen (-), that instructs a program to execute a particular deviation of the command. For example, the `-l` flag option to the `ls` command makes this utility print extra information, such as the date a file was last saved. Some commands also require some other type of **argument,** such as a filename, next to the command. All commands are sent to the shell when you press RETURN.

Figure 3-1. Parts of a command



The first word of the command line is usually the command name. This is followed by the flag options, if any, then the other arguments, if any, and then a RETURN.

2.1.1 Flag options

For many commands, the command name may be followed by flag options. A flag option is usually a hyphen followed by a single character. For example, if you use the `wc` ("word count") command to list the line, word, and character count of the files `sample` and `letter1`:

```
wc sample letter1
```

you see the counts listed (the number of lines, words, and characters in the files and their total, respectively):

```
64     406     2089 sample
27      79      494 letter1
91     485     2583 total
```

You can add the `-l` (for "line") flag option as in Figure 3-1,

```
wc -l sample letter1
```

to get just the line count for these files:

```
64     sample
27     letter1
91     total
```

Thus, flag options modify the way a command works. However, a given flag option (for example, `-l`) may have a different meaning when used with different commands. For example, while `-l` directs the `wc` command to display the line count for files, the same option used with the `ls` command (`ls -l`) changes the `ls` output to display not just the files in the directory but the files' permissions, owners, sizes, and most recent modification times.

Many commands allow you to specify more than one flag option at a time for a command (for example, `ls -la`). The command's output will reflect both options.

Most A/UX commands have flag options available. For a complete list of all available flag options for every A/UX command, see *A/UX Command Reference*.

2.1.2 Command arguments

Many commands may also include other arguments, which tell A/UX what to work on; some commands *require* an argument. For example, you can use the `wc` command to count the lines, words, and characters in a file that you name. Here, the filename is the argument.

If you enter the command

```
wc names.land
```

you see the output

```
10 11  87  names.land
```

These are the number of lines in the file named `names.land`, followed by the number of words and characters it contains. You can often supply more than one argument to a command, in which case each argument is separated by a blank space. For example,

```
wc names.land names.water
```

displays output for each file and a total count:

```
10 11  87 names.land
10 10  79 names.water
20 21 166 total
```

You should be aware that, for some commands, arguments alter the way flag options are specified on the command line. This is something you can read about on the manual page in *A/UX Command Reference* for the specific command you want to use.

2.2 Uppercase and lowercase characters

The A/UX system treats uppercase and lowercase characters differently. For example, `Q` is different from `q` in A/UX. When you enter commands, you must make sure that you use uppercase and lowercase letters correctly. In general,

- Command names are normally lowercase, but it is possible for them to contain uppercase characters.

- Flag options are either uppercase or lowercase, as appropriate to the command.
- Arguments (such as the names of files) may include uppercase and lowercase characters. Note that a file named `letters` is not the same as a file named `Letters`.

When in doubt, keep in mind that in A/UX manuals, commands in `Courier` type show literally what should be entered on the keyboard (see the Preface for our other notation conventions). The applicable page in the *A/UX Command Reference* also describes the command and shows uppercase characters as necessary.

2.3 Correcting typing errors in a command

Before you press RETURN to end a command, you can correct typing errors as follows:

- To change a character you have typed incorrectly, press the *erase* character (usually DELETE) to erase the last character you typed. Pressing this key twice erases the two previous characters, and so on. Once you have erased the characters, simply enter the correct ones.
- To erase an entire line, type the *kill* character. This starts a new line, although you do not get the `$` shell prompt, and you can begin the command over.

The A/UX system uses the DELETE key and the CONTROL-u key sequence as defaults for the system *erase* and *kill* characters. (See Table 1-1 in "Before You Begin Using This Tutorial" for the key sequences applicable to your keyboard.) You can also use the `stty` command to redefine these functions to other keys. For more information, see "Canceling Commands" in the "Bourne Shell Reference" in *A/UX User Interface* and `stty`(1) in the *A/UX Command Reference*.

2.4 Commands that are longer than one line

Until you press the RETURN key, everything you enter after the prompt is considered one command line. If the command is too long to fit on a single line, just keep typing. The shell automatically wraps the line around to a new line when necessary. It doesn't matter if command

words or filenames appear to be broken in the middle. When you do press RETURN, the shell will interpret the whole command.

If you want to control exactly where your command is broken, you may type a backslash (\) immediately followed by RETURN. The backslash tells the shell that the RETURN that follows is *not* the end of the command; it is just a way of getting to the next line. The Bourne shell will put a secondary prompt (>) at the beginning of each new line.

When you want to end the command, simply press RETURN *without* a preceding backslash. A/UX then executes your entire command.

Figure 3-2. A long command

```
cat names.land names.water | tee names.all \
| grep yellow | more
```

2.5 When you enter an impossible command

If you enter a command that cannot be executed (for example, because you made a typing mistake or gave the name of a file that does not exist), you see an error message. This message lets you know why A/UX was unable to do what you specified.

If a command is recognizable, but slightly incorrect (for example, the arguments are out of order), you may be shown the correct syntax.

If there is some other problem (for example, the file you specified is missing), the system prints a message telling you the problem. After the error message, the shell prompt ($) reappears. At this point, you can try to enter the command again or enter another command.

3. Stopping a command or process

You can stop a command that is executing by sending an *interrupt* signal to the system (please refer to Table 1-1 in "Before You Begin This Tutorial").

A few seconds after you type the *interrupt*, you'll get the $ prompt. For example, if you give the command

```
$ troff
```

and then type your *interrupt* character

CONTROL-c

you'll get the following:

```
$
```

The return of the shell prompt means that whatever was executing has been terminated, and the shell is ready to accept your next command.

The A/UX system uses CONTROL-c as the default sequence for the system *interrupt* character. You can use the `stty` command to define a different key or control sequence as the *interrupt* signal to the system. For more information, see "Canceling Commands" in the "Bourne Shell Reference" in *A/UX User Interface* and `stty`(1) in the *A/UX Command Reference*.

4. Redirecting command input and output

Every A/UX command receives input and delivers some output. Command input and output (and error messages) can be thought of as separate streams of data, and these streams are known as the **standard input,** the **standard output,** and the **standard error output.**

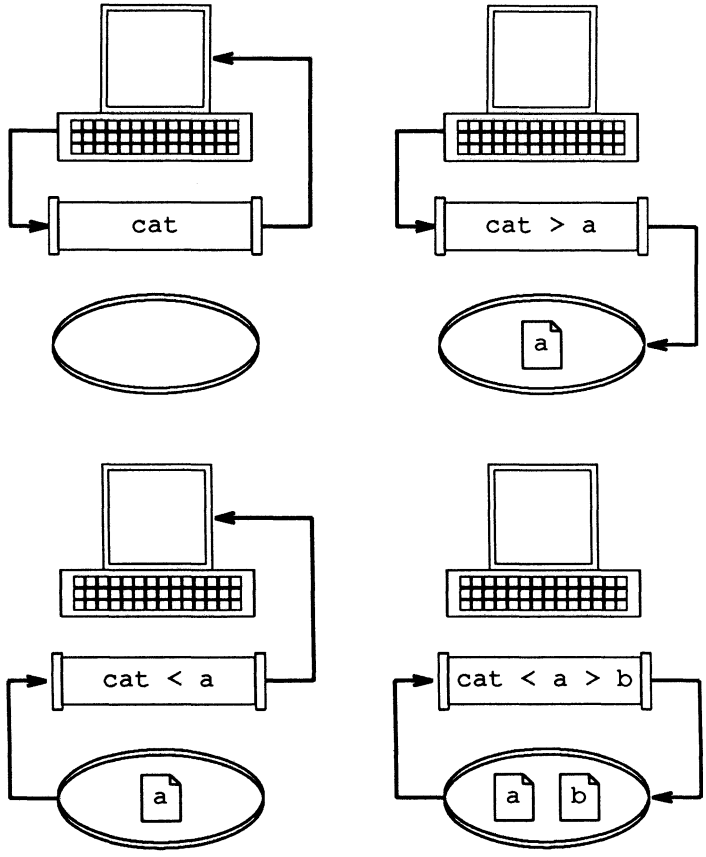
Standard input is the input to a command. By default, the shell accepts as input the characters you type from your keyboard.

Standard output is the output from a command. By default, the shell directs this to the screen.

Standard error output consists of the error messages returned from a program. By default, the shell directs error output to your screen.

You can change these default assignments using **I/O redirection,** which the following sections describe. For more on I/O redirection, see *A/UX User Interface*.

Figure 3-3. I/O redirection



4.1 Read/write a file

The `cat` command is one of the most frequently used commands in the A/UX system. In the `/users/start` directory, invoke the `cat` command as follows:

```
cat names.water
```

This copies the unformatted and unchanged contents of `names.water` to the standard output (by default, your screen).

If the file (or files) you specify does not exist, you see the message:

```
cat: cannot open filename
```

When `cat` displays information, it does not stop at the end of one screenful of text, but displays the entire file continuously and rapidly to the end of file. To stop it midstream, press CONTROL-s; to start the flow of information again, press CONTROL-q. You can do this as often as you wish.

4.2 Sending command output to a file

In many cases, command output is much more useful when you save it in a file. For example, if you are using the `cat` command to view two or more files, you may want to save the contents in a file rather than just view it on your screen (which is the default).

For example, the command

```
cat names.land names.water
```

produces the following output to the screen:

```
aardvark
baboon
cougar
dingo
egret
wallaby
walrus
weasel
yellow-bellied sapsucker
zebu
anemone
anglerfish
barnacle
bass
conch
squid
turtle
urchin
weakfish
yellowtail
```

You can redirect the output of a command into a new file with the symbol

```
>
```

followed by the name of the file that will store the new information. A/UX creates the new file and directs the output of the command into it. Using the `cat` command, if you type

```
cat names.land names.water > names.all
```

(followed by RETURN), a copy of the two files will will be in the file `names.all`. To view the contents of `names.all`, type

```
cat names.all
```

Note: When you use the > symbol to direct output to a file, make sure that a file by that name doesn't already exist. Otherwise, the contents of the existing file will be replaced by

the command output.

You can prevent overwriting an existing file by directing the output to be appended to the *end* of a file instead. The symbol for this is:

```
>>
```

A/UX looks for the file you specified after this symbol and adds the new information from your command to the end of what is already there. If no file by this name exists, it creates one.

For example, if a file containing concatenated lists already exists (`names.all`), and you want to add another file to the bottom of it, you could use the command:

```
cat letter1 >> names.all
```

When you use >> to direct output to a file, the output is appended to the file; it won't overwrite any existing information in the file. If the filename you give doesn't exist, the file is created and the output is placed in the file.

For more information on output redirection, see *A/UX User Interface*.

4.3 Taking command input from a file

Most A/UX commands take their input from the keyboard. For example, when you are sending a `mail` message to someone, you would normally supply the input (the message) by typing it at the keyboard. Sometimes, however, it is more convenient to supply a file as input to a command. To do this, use the symbol

```
<
```

followed by the the name of the file containing the information.

For example, you could send the file `letter1` to yourself (`start`) by giving the following `mail` command:

```
mail start < letter1
```

The < symbol means that `letter1` becomes the input to the `mail` program. You can direct most A/UX commands to take input from a file. For more details, see *A/UX User Interface*.

4.4 Redirecting command output to another command

In the A/UX system, the vertical bar character (|) can be used to combine two or more commands. Combined commands are called **pipelines.** The output of the first command becomes the input to the second. If there are more than two commands, the output of the second becomes the input to the third, and so on.

Pipelines save time and eliminate the need for an intermediate file. For example, suppose you want to list the number of files in your directory. Without a pipeline, you would need to use a file to save the output of the `ls -a` command (i.e., all names in a directory) For example, you could give this command:

```
ls -a > list
```

Then you use the `wc` command as follows to list the number of files:

```
wc -w list
```

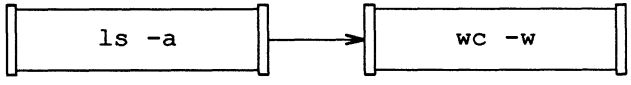
The vertical bar (|) lets you combine these two processes into one, using a pipeline. The command

```
ls -a | wc -w
```

displays how many files are in the current directory. The output of the first command is used as input to the second command, so there is no need for an intermediate file. When the second command has finished executing, the final output is directed to your screen (or into a file, if you wish).

For more information about pipelines, see *A/UX User Interface.*

Figure 3-4. Piping commands



5. Background execution

In the A/UX system, by default, a command starts in the foreground, and the shell waits until the command has finished executing before asking for the next command. However, you can start a command executing and get the shell prompt back immediately so you can continue working. The command that is executing behind the scenes is called a **background** job. Any job that isn't a background job is a **foreground** job. Foreground jobs suspend the shell until they have finished executing.

Background jobs should not produce any output to your screen. Otherwise, the output would interfere with your foreground job. If the command is supposed to return some output to you, you can direct that output into a file. Then you can look at the output file's contents later.

It is very simple to put a job in the background. Type the command, and before you press the RETURN key, simply type an ampersand character (`&`). For example,

```
wc names.land > word.count &
```

places its output into a new file named `word.count`. The system displays the process ID number (pid) of the background job after you press RETURN. Use `cat` to show the file's contents:

```
cat word.count
```

The word count is

```
10 11  87 names.land
```

Be careful not to use the resulting file until the command is complete.

To learn more about putting processes in the background (and stopping those processes, if need be), please refer to *A/UX User Interface*.

Chapter 4
The A/UX File System

Contents

Figures

Tables

Chapter 4

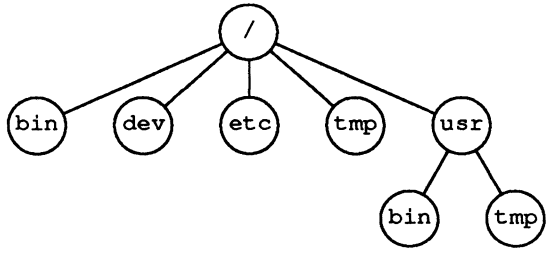
The A/UX File System

1. Overview

The A/UX file system is a collection of files. Each file has a name. A/UX files are organized in **directories,** which are special system files that act as file "cabinets" to store files. Each directory also has a name.

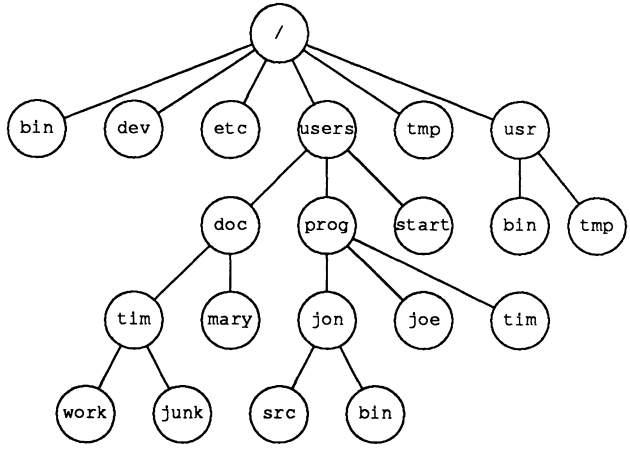
Directories are organized in a **tree structure.** Figure 4-1 shows part of this organization. Organized like an inverted tree, the file system begins with the root directory (/) at the top. Branching downward from the root are the rest of the directories and files in the system. The root directory contains a number of directories that store the system files and programs described in this chapter.

Figure 4-1. Part of a tree structure



Any directory can contain more directories (called **subdirectories**) to contain more files, to help keep your files organized. A subdirectory is called the "child" of the directory that contains it. When using this terminology, a directory is the called the "parent" of any subdirectories it contains. In Figure 4-2, `users` is the parent of `doc`, and `mary` is a child of `doc`.

Figure 4-2. A larger tree structure



Because of its tree-structured organization, A/UX uses pathnames to refer to files. A **pathname** is a filename prefixed by its directory location, for example,

```
/users/start/names.land
```

Whatever directory you are in at any point is your **current directory** or **working directory.** When you change directories, the directory you move to becomes your current directory.

Pathnames may be **absolute,** stating the complete name of the file by listing all of the directories leading down from the root directory and concluding with the filename itself, for example,

```
/users/doc/tim/work
```

Pathnames may also be **relative,** given by listing the directories leading to that file in relation to the current working directory. For example, if the current directory is `/users/doc`, then

```
tim/work
```

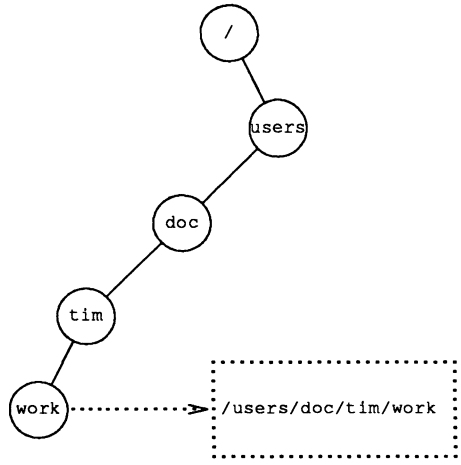
is the relative pathname for `/users/doc/tim/work`.

A **filename** is the last component of a pathname, for example,

```
work
```

(note that directories are a special class of file).

Figure 4-3. Absolute pathname



The absolute pathname is always the same, no matter what your current location in the file system. The leading slash character (/) represents the root directory: absolute pathnames always begin with / (see Figure 4-3), while relative pathnames never do.

You can use the following conventions in relative pathnames:

filename
: You don't have to include any directory names when you specify a file that's in the same directory you are. To specify a file named `names.land` from your working directory `/users/start`, the relative pathname is simply `names.land`.

directory-name

You can specify the pathname to a directory or file that's one level down from your current directory with the directory or filename. Thus, the relative pathname to the `/users/start/names.land` directory from the `/users` directory is simply `start/names.land`.

`..`

Two dots (`..`) are an abbreviation for "the parent of the current directory." You can use the relative pathname `..` to specify the `/users` directory from the working directory `/users/start`.

2. Commands for using directories

When you log in to your A/UX system, you are in your **home directory.** The system knows which directory is your home directory by looking at your `$HOME` environment variable. To find out the absolute path name of your home directory, type

```
echo $HOME
```

2.1 Finding out your current location

To find out where you are in the A/UX file system, type

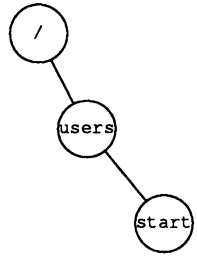
```
pwd
```

The `pwd` command stands for "print working directory." `pwd` prints the absolute pathname of your current working directory. For example, if you've just logged in as `start`, it prints

```
/users/start
```

The home directory for `start` is two levels removed from the root directory; `start` is a subdirectory of `users`, which is a subdirectory of the root directory (`/`).

Figure 4-4. `/users/start`



2.2 List directory contents

To see what is contained in your current directory (in this case, your home directory) use the command

```
ls
```

If you specify a directory to the `ls` command, you get a list of that directory's contents. For example,

```
ls /users
```

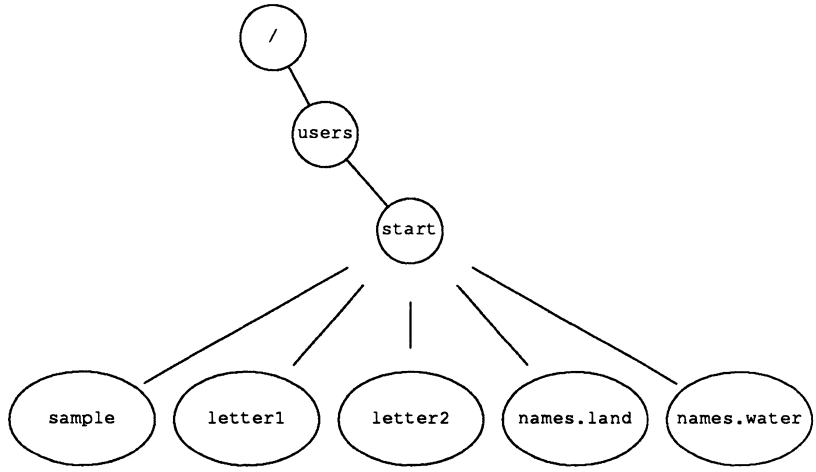
lists the contents of the `/users` directory. If you don't specify a directory name, you get a list of the contents of your current directory.

The `ls` command stands for "list." This command lists all the files and directories contained in your current directory but does not show the files that begin with a "dot" (a period character), such as the `.profile` file. In the sample login directory for your system, `/users/start`, the `ls` command prints something like:

```
animals
letter1
letter2
names.all
names.land
names.water
sample
```

So the directory hierarchy from the root directory to the current directory looks something like this:

Figure 4-5. Some files under `/users/start`



You can list all the files in a directory, *including* those that begin with a dot (`.`), by using the `-a` option as follows

```
ls -a
```

This command gives you a list something like:

```
.
..
.profile
animals
letter1
letter2
names.land
names.water
sample
```

In the first two lines of this output, "`.`" is a synonym for the current directory and "`..`" is a synonym for the parent of the current directory. If you type

```
ls .
```

you get a list of the contents of the current directory.

You can get even more information about the contents of a directory, such as the access permissions of files and subdirectories, length of files, when the files were last changed, and so forth, using the `-l` option

```
ls -l
```

This command produces a listing that looks something like this:

```
total 10
drwxrwxr-x  2  start project   32  Feb 14  11:04  animals
-rw-rw----  1  start project  494  Feb 14  08:16  letter1
-rw-rw----  1  start project  448  Mar 23  09:10  letter2
-rw-rw----  1  start project   79  Feb 14  15:15  names.land
-rw-rw----  1  start project   79  Apr 01  13:23  names.water
-rw-rw----  1  start project 2089  Aug 26  17:07  sample
```

The first line of this output (`total 10`) tells you that this directory uses ten blocks of memory. The next several lines (beginning with `drwx`) tell you the following information about the files contained in that directory:

- File type. The first character of each line tells you that the file is either

  `d` for "directory"

  `-` for "ordinary file"

  There are other possibilities, but they aren't covered here (see `ls`(1) in *A/UX Command Reference*).

- File mode or "permissions." The next nine characters in each line are either letters or hyphens that describe who can read and use the file or directory (see "File Security" in this chapter).

- Link count. The **link count** (`2` in the first line) is the number of alternate names the file has. A full discussion of the link count is beyond the scope of this manual (see *A/UX User Interface* and/or

`ln`(1) in *A/UX Command Reference*).

- Owner name. The login name of the file's owner, which is `start` for everything in the sample directory, is then listed.
- Group name. The group name is `project` for every file and directory listed in the sample. You can define a **group** to include any users on the system. The group members are defined along with the group name in the `/etc/group` file. For more information, please refer to *A/UX Local System Administration.*
- Number of bytes. The amount of storage (in bytes) used by the file or directory.
- Modification time. The month, day, and time that the file was last modified.
- File name. The file or directory name is the last item listed.

2.3 Moving to a different directory

From your login directory you can move into lower directories (which you create to organize your files) or up to system directories (which contain the A/UX system commands, files, and directories).

When you move to a new directory, it becomes your current directory. The `cd` command stands for "change directory."

For example, type the command

```
cd /users
```

to move to the `/users` directory. This command uses the absolute pathname for `/users`. Then type

```
cd start
```

to return to the `start` directory. If you don't specify a directory name, you move to your home directory, which then becomes your current directory. If you use `cd` alone while you're in your home directory, you just stay there.

2.4 Making a new directory

To create a new directory to store files, you must have write and execute permission in its parent directory. See "File Security" later in

this chapter for information about permissions.

Check to see if you're in the `/users/start` directory by typing

```
pwd
```

If `pwd` returns a different directory name, move to `/users/start` by typing

```
cd /users/start
```

Once you're in the `/users/start` directory, you can create a new directory with the command

```
mkdir biology
```

The `mkdir` command stands for "make directory." The new directory will be named `/users/start/biology`. You can create new files in the `biology` directory, or you can copy or move files into it from other directories.

You can also specify an absolute pathname to the `mkdir` command to create a new directory elsewhere in the file system, as long as you have write permission for the parent directory.

2.5 Removing a directory

If you want to remove an empty directory, use the `rmdir` command, for example:

```
rmdir biology
```

The `rmdir` command stands for "remove directory." If the directory has files or subdirectories, it can't be removed, and you'll get the message

`rmdir:` *`directory-name`*`: Directory not empty`

You can remove a directory elsewhere in the file system by specifying its absolute or relative pathname.

3. Commands for using files

Files are always located in a directory. You can create files; copy or move them to a new location; view, edit, or print them; analyze, compare, or sort their contents, and so on.

The commands described in this section are a small subset of the commands available to you. See "Command Summary by Function" in *A/UX Command Reference* for a complete list.

3.1 Creating files

You can create new files in several ways. For example, you can invoke one of the A/UX text editors with a filename or pathname (see Chapter 5, "`vi` Tutorial"). You can copy an existing file to a new filename, in which case you get a new file with contents identical to the old file.

You can also rename an existing file or move it to another directory, in which case you'll get a file with a new pathname.

3.2 Copying files

If you have an existing file named `names.land`, you can create a new file with the same contents using the command

```
cp names.land my.names
```

The `cp` command stands for "copy." Before you use this command, make sure that you don't already have a file in the current directory named `my.names`. (You can check this using the `ls` command.)

> *Note:* If you already have a file named `my.names`, the `cp` command will replace it with a copy of `names.land`. The original version of `my.names` will be lost. To prevent this, use the `-i` flag option
>
> ```
> cp -i names.land my.names
> ```
>
> This flag option causes the `cp` command to print the message
>
> `overwrite` *filename*`?`
>
> before it overwrites an existing file. A response of `y` (followed by RETURN) allows `cp` to overwrite the file; any other response aborts the copy operation. For the purposes of this example, type `n` and then RETURN.

If you try to copy a file to itself, for example,

```
cp names.land names.land
```

you'll see the message

```
cp: Cannot copy file to itself.
```

However, you may have files with identical names in different directories. For example, from `/users/start` you can use the command

```
cp names.land /users/start/animals
```

or, because `animals` is a subdirectory of `start`, just

```
cp names.land animals
```

This way, the file you list is copied into the directory you specify. First, use `ls` to check the destination directory to be sure there isn't already a file with that name, or the file will be replaced. The new file will have the same filename as the original. Either way you use `cp`, the original file (`names.land` in this example), is left untouched.

3.3 Renaming and moving files

You can use the `mv` command to change the name of a file or to move a file into another directory. For example,

```
mv my.names creatures
```

renames the file `my.names` to the new name `creatures`.

> *Note:* Before you use the `mv` command, use `ls` to check that there is not already a file by that name in the destination directory. Otherwise `mv` will replace the existing file with the file you are moving, and the original will be lost. To prevent this, use the `-i` flag option
>
> ```
> mv -i creatures names.land
> ```
>
> This flag option causes the `mv` command to print the message
>
> `remove` *filename*`?`
>
> before it overwrites the existing file. A response of `y` (followed by RETURN) allows `mv` to overwrite the file; any other response cancels the move operation. For the purposes of this example, type `n` and then RETURN.

If you use a directory instead of a new filename with the mv command

```
mv creatures /users/start/animals
```

the file named creatures is moved into the animals directory (and is renamed /users/start/animals/creatures).

After you've used mv, the file can no longer be found under the old name.

3.4 Removing files

You can use the rm command to remove files permanently. For example, in the directory /users/start/animals (get there using the cd command),

```
rm creatures
```

removes a file named creatures. If you decide you didn't want to remove creatures after all, you must restore it from your most recent backup, so use this command with caution. If you want to protect yourself from removing files by mistake, use the -i option on the command line. cd back to /users/start and type:

```
rm -i names.land
```

This tells the system to ask this question before removing the file:

rm: remove *filename*?

A response of y (followed by RETURN) allows rm to delete the file; any other response cancels the delete operation. For the purposes of this example, type n and then RETURN.

3.5 Looking at a file's contents

The pg (for "page") command shows the contents of a file one screenful at a time. After it displays the first screenful of text, pg waits for you to give a command to display another screenful of text, search for a pattern of characters, exit from pg, or perform any of the actions shown below. You can also use flag options next to the pg command, for example, to display part of a file starting at a specific line or at a line containing a certain sequence or pattern.

You can use pg to display a single file with the command

```
pg sample
```

This prints the first 23 lines of a file, followed by a colon (:). A smaller file would be printed in its entirety. You can give any of the following instructions at the colon. End each instruction (except RETURN) by pressing RETURN.

Table 4-1. List of `pg` commands

Command	Meaning
`h`	Help; display list of available `pg` commands.
`q` or `Q`	Quit `pg`.
RETURN	Display next page of text.
`l`	Display next line of text.
`d` or CONTROL-d	Display additional half page of text.
`.` or CONTROL-l	Redisplay current page of text.
`f`	Skip next page of text, and display following one.
`n`	Begin displaying next file you specified on command line.
`p`	Display previous file specified on command line.
`$`	Display last page of text in file currently displayed.
`w` or `x`	Set window size and display next page of text in file currently displayed.
`s` *savefile*	Save current file in *savefile*.
/pattern/	Search forward in file for specified character pattern.
`?`*pattern*`?` or `^`*pattern*`^`	Search backward in file for specified character pattern.
`!`*command*	Execute *command*.

Most `pg` commands can be preceded by a number. For example,

`+1` Display the next page.

`-1` Display the previous page.

`1` Display the first page of text.

When `pg` reaches the end of a file, it prints the end-of-file symbol

```
(EOF)
```

and another colon. At this point, you can return to the shell by typing `q` or `Q`, or you can give one of the other commands listed above.

3.6 Looking at top or bottom lines

Use the `head` command to see the first few lines of a file. For example, the command

```
head /users/start/animals/names.land
```

returns

```
aardvark
baboon
cougar
dingo
egret
walrus
weasel
yellow-bellied sapsucker
zebu
```

Use the `tail` command to see the last few lines of a file. For example, the command

```
tail /users/start/names.water
```

returns

```
anemone
anglerfish
barnacle
bass
conch
squid
turtle
urchin
weakfish
yellowtail
```

Because these files are relatively short, they are printed out in their entirety. You can specify how many lines you want to see by including the number of lines as an argument on the command line. For example,

```
tail -3 names.land
```

shows you the last three lines of the file `names.land`.

3.7 Format a file into pages

You can use `pr` to format a file into pages. Note that `pr` displays continuously and rapidly to the end of the file. If the formatted file is already scrolling past on your terminal, you can press CONTROL-s to stop scrolling temporarily, and CONTROL-q to resume it.

You can use the `pr` command as follows:

`pr` *filename*...

This formats the title and headings, paginates, and prints the file(s) on your screen. For example,

```
pr letter1 letter2
```

does this for the files `letter1` and `letter2`.

You can use the `pr` command with the `lp` command when you need a paper copy of a file. If you pipe the output to `lp` (see "Printing a File"), you can use `pr` to print formatted output on output devices such as a line printer. For an introduction to pipes, see Chapter 3, "Using A/UX Commands: The Shell," in this manual.

If the files you specify don't exist, you'll see the message

```
pr: can't open filename
```

3.8 Printing a file

If you have a printer connected to your system, you can send files to be printed using the `lp` command. If you don't have a printer connected to your system, please refer to *A/UX Local System Administration* for information about how to do so (or see your system administrator).

For example, to print the file `sample`, type

```
lp sample
```

Once the job has been sent to the printer you get a message

```
request id is printer-number (1 file)
```

printer-number is the identification for the job. The `sample` file is then printed. If you share your printer and there are other jobs ahead of `sample`, they will print out first. A/UX **spools** printing jobs, placing them in a queue of first-come, first-served precedence. You can get a list of the queue by typing

```
lpstat -o
```

You get a list that looks like this:

```
lp-389    groucho     99382     Feb 14   06:03
lp-007    bond        34552     Feb 14   06:12
```

If, after you've sent a job to the printer, you want to cancel it before it's printed, you can do so by using the `cancel` command with the ID for the job. To cancel the print job for `sample`, type

```
cancel printer-number
```

The message

```
request "printer-number" canceled
```

appears and the job is canceled. For more information about `lp` and `cancel`, please refer to `lp`(1) in *A/UX Command Reference*.

3.9 Analyzing the size of a file

You can use the `wc` command to count the lines, words, and characters in a file. For example,

```
wc names.land
```

The `wc` command stands for "word count." This command produces output that shows

- the number of lines in the file
- the number of words in the file
- the number of characters in the file

For example,

```
10 11  87  names.land
```

If you want only a line, a word, or a character count, select the appropriate command from the following:

`wc`	`-l`	`names.land`	Count lines only.
`wc`	`-w`	`names.land`	Count words only.
`wc`	`-c`	`names.land`	Count characters only.

You can also specify multiple files next to the `wc` command. This prints a separate line of output for each file and a line showing the totals for the combined files.

3.10 Comparing files

You can use the `diff` command to compare two files to see which lines differ. For example,

```
diff file1 file2
```

If *file1* and *file2* are identical, the system returns the shell prompt. Otherwise, differences are reported; the lines that are different appear on your screen.

`diff` reports differences between the files as follows: `a` (`append`), `c` (`change`), or `d` (`delete`). Numbers given with `a`, `c`, or `d` indicate the modified lines. The symbol < indicates a line from *file1*, the first file named; > indicates a line from *file2*, the second file named.

For example, if you use the `diff` command to find the differences between two very similar form letters, the system gives you the numbers of the lines that differ between the files, indicates how they differ, and then shows the lines in question, first from `letter1` and then from `letter2`. For example,

```
diff letter1 letter2
```

produces the following report:

```
1,3c1,3
<Mr. Abdul Haffar
<Haffar Elephant Brokerage
<26 West Portal
---
>Mr. Omar Smith
>Omar the Tent Maker
>10 Coleman Lane
7c7
<Dear Mr. Haffar,
--
>Dear Mr. Smith,
9c9
<We are enclosing a deposit on the shipment of livestock
--
>We are enclosing a deposit on the shipment of tents
11c11
<We will be expecting two dozen elephants with attendants.
--
>We will be expecting four dozen tents.
17c17
<future dealings with the Haffar Elephant Brokerage.
--
>future dealings with Omar the Tent Maker.
```

The first line of the system response

```
1,3c1,3
```

means that lines 1 through 3 in the first file are different (designated by `c`) from lines 1 through 3 in the second file. The actual lines from each file are listed next. The first three lines are from `letter1` and the next three lines are from `letter2`.

3.11 Scanning for a pattern

You can use `grep` (for "global regular expression print") to search one or more files for a particular phrase or pattern. `grep` displays any lines that contain that phrase or pattern. If you name more than one file, the name of the file that contains the pattern is also given. The pattern can be any combination of characters (a word, phrase, equation, etc.). If it contains spaces or characters that have a special meaning to the A/UX system, such as blanks, `$`, `|`, `*`, `?`, and so on, the entire pattern must be enclosed in single quotes (`' '`). For an explanation of the special meaning for these and other characters see "Using Shell Metacharacters" in the "Bourne Shell Reference" in *A/UX User Interface*.

For example, to locate the lines containing the pattern "`sailed away`," in the file named `sample`, enter the command

```
grep 'sailed away' sample
```

The system responds

```
They sailed away in a sieve, they did,
And all night long they sailed away;
```

Since you specified only one file, the name of the file doesn't appear.

If you were unable to remember which of two files, `sample` or `names.water`, was about going to sea, you could type

```
grep 'sailed away' sample names.water
```

and the system would respond

```
sample: They sailed away in a sieve, they did,
sample: And all night long they sailed away;
```

This tells you that the pattern `sailed away` is found twice in the file `sample`, in the lines "`They sailed away in a sieve, they did,`" and "`And all night long they sailed away.`"

The A/UX system also provides variations of the basic `grep` command, called `egrep` and `fgrep`, and a number of options that enhance the searching powers of the command, such as metacharacters.

See `grep`(1) in *A/UX Command Reference* for details.

3.12 Sorting and merging files

`sort` sorts and merges information from one or more files you name and displays the result on your screen.

You can invoke the `sort` command by typing

```
sort file...
```

If you don't specify any options, lines are sorted by single characters and merged alphabetically.

For example, if you have two files, `names.land` and `names.water`, and each contains a list of names that you want sorted alphabetically and merged into one list, you can first check the contents of the two files using the `cat` command

```
cat names.land names.water
```

The files appear one after the other with no break between then.

```
aardvark
baboon
cougar
dingo
egret
walrus
weasel
yellow-bellied sapsucker
zebu
anemone
anglerfish
barnacle
bass
conch
squid
turtle
urchin
weakfish
yellowtail
```

You can now sort the contents of the two files and merge them using the `sort` command. The command

```
sort names.land names.water
```

produces this output:

```
aardvark
anemone
anglerfish
baboon
barnacle
bass
conch
cougar
dingo
egret
squid
turtle
urchin
walrus
weakfish
weasel
yellow-bellied sapsucker
yellowtail
zebu
```

Note that the output appears on the screen and then is lost, unless you redirect it to a file. For information about redirecting output, see Chapter 3, "Using A/UX Commands: The Shell," in this manual.

4. File security

Because the A/UX operating system is a multiuser system, it's possible you're not working alone in the file system. Whenever file and directory permissions allow it, you and other users can follow pathnames to various directories and use files belonging to one another.

Permissions determine who can and cannot have access to, and use, a particular file or directory. For example, if a file has read-write permission for all users on the system, anyone on the system can get

into the file, and permanently change the file. If a file has read-only permission for all users on the system, all users can read the file, but no one can make any changes to the file.

4.1 File access permissions

You can set the following permissions on files you own:

- `r` Allows designated users to read a file or to copy its contents.
- `w` Allows designated users to modify a file.
- `x` Allows designated users to execute a file (that is, to run it as a command).

4.2 Directory access permissions

You can set the following permissions on directories you own:

- `r` Allows designated users to read a directory (for example, with the `ls` command).
- `w` Allows designated users to write a directory, that is, to create or delete files and subdirectories.
- `x` Allows designated users to search a directory.

You must have execute permission on a directory to enter it and make it your current directory or to access its subdirectories.

You must have *both* execute and write permissions on a directory to create or delete files and subdirectories in it.

4.3 Checking a file's permissions

To check the current permissions on a file or directory, use the `ls -l` command. For example, the command

```
ls -l sample
```

returns something like

```
-rw-r--r-- 1 start project 2089 Aug 26 17:07 sample
```

Note: For instructions about how to read the output of the `ls -l` command in general, see "List Directory Contents,"

earlier in this chapter. This section is concerned only with the first field of the output, which records the permissions for each file and directory listed.

The first field is made up of ten characters, each with a particular meaning. The first character tells you what kind of entity you are dealing with: – denotes a file; `d`, a directory. The remaining nine characters represent the permissions. They're divided into three groups of three characters each. These groups refer to the **access class.** Each access class corresponds to a category of user. There aren't any spaces between the groups.

In Figure 4-6 the typical set of permissions for a file and the access classes within them would appear on the screen as

```
-rwxrw-r--
```

Figure 4-6. Permissions and access classes

–	rwx	rw-	r--
type	*user*	*group*	*other*

In this example, *type* is file type. The hyhen (–) means that the file is a regular file. If a `d` were in this field, it would mean that the file is a directory. (See "Chapter 3, "User Administration" in *A/UX Local System Administration* for a complete discussion of file types.)

Within each access class, the order of characters is `r`, `w`, `x`. They indicate the permission granted to that category of user. If a hyphen appears instead of an `r`, `w`, or `x`, permission to perform that action is denied to that category of user.

The first access class lists the file owner's permissions (also called "user's permissions"). For `sample`, the owner permissions are `rw`. The owner has permission to read or write the file. Permission to execute the file is left as a hyphen, meaning that the file is not executable.

The second access class lists the group permissions. In the case of `sample`, the group permissions are `r`. The group only has permission to read the file.

The last access class lists the permissions for all other system users. In the case of `sample`, the others' permissions are `r`. Other system users may only read the file.

As the owner of a file or directory, you keep permissions as you would like them, regardless of how they are set up when a file is created.

4.4 Changing existing permissions

When you create a file or a directory, the system automatically gives it certain pre-set permissions for you, your group, and other system users. You can alter this automatic action to some extent by modifying your environment (see *A/UX User Interface*).

The `chmod` command allows you to change the permissions for a particular file or directory.

You give instructions to `chmod` in either symbolic or numeric terms. Only symbolic terms, which are easier to use, are covered here.

You use `chmod` to give (+) or remove (–) read, write, and execute permissions on directories and individual files for three classifications of users, called access classes.

To change the permissions of a file or directory, you can invoke the `chmod` command as follows:

`chmod` *access-class operator permission filename-list*

(No spaces should separate the access-class, operator, and permission.)

where:

access-class is one or more of the three user groups: `u`, `g` or `o`.

- `u` the user (that is, you): the owner of the files and directories in question. In Figure 4-6, the user has read, write, and execute permission on the file.
- `g` group: members of the group to which you belong (a group can consist of team members currently on

a project, members of a department, or a group arbitrarily designated by the system administrator). In Figure 4-6, any user in the specified group has read and write permission on the file.

- o others: all the other users on the system. In Figure 4-6, all other users on the system have only read permission on the file.

operator is either + to grant, or – to deny permission. You can't both grant and deny permissions in a single command. You must grant permissions in one command, then deny others in a second command.

permissions is `r`, `w`, or `x` for the permission(s) to be granted or denied. You can grant (or deny) more than one type of permission at the same time, but you can't grant *and* deny permission at the same time.

filename-list is the file(s) (or one or more directories) whose permissions are to be changed. You may use absolute or relative pathnames. For example,

```
chmod g+w sample
```

allows other users in the `project` group to change the file `sample`.

You can use `chmod` to grant or deny permission for directories simply by giving a directory name instead of a filename on the command line. However, you should be careful when you're granting or denying directory permissions.

For example, if all access classes for a directory have read, write, and execute permission, anyone on the system can enter the directory and read and modify any files that also have read and write permission. However, they will not be able to read or modify any files that do not have read and write permission.

Conversely, if the directory doesn't have execute permission, no one can read or copy a file in the directory, regardless of the individual file's permissions. If the directory doesn't have write permission, no

one can create or delete files (or subdirectories) in it. If the directory doesn't have read permission, no one can find out the names of the files (or subdirectories) in it (that is, `ls` gives an error message).

4.5 Another way of changing permissions

The method just described uses letters as symbols (`r`, `w`, `x` `u`, `g`, and `o`) to specify user categories and access classes to `chmod`. That's why it is called the "symbolic method." Besides this method, there's another way you can change permissions. The **numeric method** uses up to three octal digits (that constitute one octal number) to assign permissions to various types of users. For a discussion of the numeric method, see `chmod`(1) in *A/UX Command Reference*.

4.6 Setting default protections

The `umask` command sets default file protections for all newly created files. `umask` uses the numeric method for specifying file protections. You can use `umask` to change the default file protections for the files you create. This command is commonly stored in the `.profile` file. For more information, see Chapter 2, "Bourne Shell Reference," in *A/UX User Interface* or `sh`(1) in *A/UX Command Reference*.

5. Commands for learning about your system

5.1 Your system's name

You can learn the name and version number of your operating system using the `uname` command. Simply type

```
uname
```

You see

```
A/UX
```

To learn the version of your A/UX system, enter the command

```
uname -r
```

You see the release number of the system, for example

```
5.2
```

For further information about the `uname` command, see `uname`(1) in the *A/UX Command Reference*.

5.2 Your login name

To get your login name, enter the command

```
whoami
```

A/UX displays your login name. For example, your login name while you're using this manual is

```
start
```

(If you have a system administrator, this login name may have been changed.)

This can be useful if you share your terminal or system, and you are unsure who is logged in.

5.3 Who else is logged in

To get a list of everyone logged in on your A/UX system, type

```
who
```

to produce a list like the following:

```
groucho     ttyd0      Sep 4 07:12
harpo       ttyd1      Sep 4 08:37
chico       ttyp2      Sep 4 08:45
zeppo       ttyp3      Sep 4 09:53
gummo       ttyp5      Sep 4 09:11
larry       ttyh1      Sep 4 07:56
moe         ttyh2      Sep 4 08:37
curly       ttyh4      Sep4 07:31
```

The first column lists the users' names, the second column lists their terminal numbers, and the remaining columns list the date and time they logged in.

5.4 The status of your running processes

You can find out the process ID and status of each process you're executing with the command

```
ps
```

This command generates a list of processes similar to this list:

```
PID TTY  TIME COMMAND
631  p0  0:02 sh
658  p0  0:04 ps
```

Column 1, labeled `PID`, is the process ID. Column 2, labeled `TTY`, is your terminal number. Column 3, labeled `TIME`, is the total execution time for the process in minutes and seconds. Column 4, labeled `COMMAND`, is the command name.

Whenever you use `ps`, you see the commands `sh`, for the shell with which you're interacting, and `ps` for your current `ps` command. You also see any background commands you've entered that are still executing. Therefore, you can use the `ps` command to learn the process ID of a process you're running in the background.

5.5 Disk usage

You can determine the amount of disk space you are using with the command

```
du
```

A simple `du` command lists the number of blocks required for the current directory and each of its subdirectories

```
2     ./animals
22    .
```

Directory names are entered with the relative pathname `.` (current directory; here, the user's home directory), followed by *directory-name* (for example, `./animals`). The last line lists the total number of blocks required for the current directory `.` and includes its subdirectories.

For more precise information, the command

```
du -a
```

lists the disk use of files in the current directory and its subdirectory and displays something like the following:

```
1 ./animals/names.land
1 ./animals
1 ./.profile
1 ./setup
1 ./letter1
1 ./letter2
1 ./names.water
5 ./sample
18
```

Note here that the file in the subdirectory `animals` (`/animals/names.land`) is listed with its disk use, followed by the total for the directory itself.

Chapter 5
`vi` Tutorial
Creating and Editing a File

Contents

Figures

Chapter 5

vi Tutorial

Creating and Editing a File

Introduction

vi is one of the A/UX text editors. This chapter takes you through the basics of vi in four lessons. The first lesson concentrates on starting vi and moving around in a file. The second lesson takes you through the process of creating a file and entering text. The third lesson concentrates on editing existing text and correcting mistakes. The fourth lesson takes you through more advanced editing procedures.

While these four lessons don't, by any means, teach you all that vi has to offer, you should be well on your way to creating and editing files. Once you've finished the tutorial, you can learn more about vi by referring to *A/UX Text Editing Tools*.

Lesson 1: Starting vi

To begin the tutorial, log in to A/UX as start. When A/UX admits you to the system, you'll be in start's home directory. You can now begin the tutorial.

To create a new file or to enter an existing file, you need to start (or **invoke**) vi. When you invoke vi to open an existing file, include the file's name at the same time.

To open the file /users/start/sample, first invoke vi by typing

```
vi sample
```

vi opens the file sample. The first two stanzas are shown in Figure 5-1.

Figure 5-1. The file **sample**

```
They sailed away in a seive, they did,
        In a sieve they sailed so fast,
With only a beautiful pea-green veil,
Tied with a ribbon, by way of a sail,
        To a small tobacco-pipe mast.
And everyone said who saw them go,
"Oh! won't they be soon upset, you know,
For the sky is dark, and the voyage is long;
And, happen what may, it's extremely wrong
        In a sieve to sail so fast."


The water it soon came in, it did;
        The water it soon came in.
So, to keep them dry, they wrapped their feet
In a pinky paper all folded neat;
        And they fastened it down with a pin.
And they passed the night in a crockery jar;
And each of them said, "How wise we are!
Though the sky be dark and the voyage be long,
Yet we never can think we were rash or wrong,
        While round in our sieve we spin."
```

By default, you start with the cursor at the upper-left corner of the screen. The **cursor** is a mark on the screen that indicates your current position on the command line or inside a file. The cursor is usually a small box or an underscore, and it usually blinks. In an editor such as `vi`, the cursor marks the current position within the file being edited. When you move the cursor, it marks your new position in the file. You can move the cursor around your file a lot or little at a time.

Moving the cursor up and down a line

To move the cursor up and down in a file, one line at a time, you can use these keys:

+	Moves down one line.
j	Moves down one line.
↓	Moves down one line.
-	Moves up one line.
k	Moves up one line.
↑	Moves up one line.

Try moving up and down in the file using these keys. You can move the cursor up or down more than one line at a time using these commands if you add a number before the command.

Type any of the following:

```
5+
5j
5↓
```

The cursor moves five lines down in the file.

Now type any of the following:

```
5-
5k
5↑
```

The cursor moves up five lines.

Moving the cursor horizontally

Move the cursor left and right, one character at a time, using these keys.

`l` Moves right one character.

→ Moves right one character.

`h` Moves left one character.

← Moves left one character.

Larger motions

To move the cursor left or right more than one character at a time using the commands just described, add a number before the command.

Type either

```
8l
8→
```

to move the cursor right eight characters.

Now type either

```
3h
3←
```

to move the cursor three characters to the left in the file.

Scrolling through the file

You can scroll through a long file using still other commands. Move the cursor a specific amount using these keys.

CONTROL-d Moves down one-half screen or the number of lines you specify.

CONTROL-u Moves up one-half screen or the number of lines you specify.

CONTROL-f Moves the cursor forward (down) to the next page or the number of pages you specify.

CONTROL-b Moves back (up) one page or or the number of pages you specify.

Try typing

```
10CONTROL-d
```

You move the cursor down 10 lines in the file. This sets the extent of scrolling you get with CONTROL-d to 10 lines. You can type CONTROL-d alone and the cursor will move down 10 lines in the file. You can reset the extent of scrolling by typing a number in front of CONTROL-d.

Type

```
14CONTROL-u
```

You move the cursor up 14 lines in the file. This sets the extent of scrolling you get with CONTROL-u to 14 lines. You can type CONTROL-u alone and the cursor will move up 14 lines in the file. You can reset the extent of scrolling by typing a number in front of CONTROL-u.

If you were to type

```
2CONTROL-f
```

you'd move the cursor forward two screens in the file. If you were to type

```
5CONTROL-f
```

`vi` would flash the screen because there are fewer than five screens in the file `sample`.

If you were to type

```
3CONTROL-b
```

you would move the cursor back three screens in the file.

Quitting the `sample` file

Now type

```
:q
```

and press RETURN to quit the `sample` file without keeping any changes you've made to it.

You probably haven't made any changes to the file because you've just been moving about in it. If, however, you did make any changes, the message

```
No write since last change (:quit! overrides)
```

appears at the bottom of the screen. `vi` warns you that you haven't "written" the file (see Lesson 2) since you last changed it. If that message appears, type

```
:q!
```

and press RETURN to tell the system to go ahead and close the file without keeping the changes.

If you ever make a change that's so disastrous it could ruin the file, you can always exit the file using `:q!`. You'll get the file that you had before you made any changes at all (unless you have written changes to the file along the way). See Lesson 2 for details.

Lesson 2: Creating a new file: `jumblies`

When you invoke `vi` to create and open a new file, you provide the new file's name at the same time.

To create and open the file `jumblies`, invoke `vi` from the command line by typing

```
vi jumblies
```

`vi` opens the new, empty file `jumblies`, shown in Figure 5-2.

Figure 5-2. The new file **`jumblies`**

```
~
~
~
~
~
"jumblies" [New file]
```

When you first enter a new file, you're in **command mode.** In command mode, anything you type is taken to be a command, if possible. If you enter a command that `vi` doesn't recognize or cannot perform, the screen flashes.

The tildes (~) are placeholders that indicate an empty line. Since at this time there are only tildes, the file is currently empty.

The next step is to enter some text.

Inserting text: insert mode

Type `i`.

When you type `i`, you enter **insert mode,** which lets you insert text into the file. Whatever you type after `i` appears on the screen.

Now try typing the following:

```
They went to sea in a sieve they did;
    In a sieve they went to sea.
In spite of all their friends could say,
    In a sieve they went to sea.
And when the sieve turned round
And everyone cried "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button, we don't care a fig,
    we'll go to sea."
    Far and few
        Are the lands where the jumblies live;
    Their heads are green, and their hands are blue,
        And they went to sea in a sieve.
```

To make your text look like the text above, you need to end each line by pressing RETURN. You can match the appearance of the indented lines by using spaces or tabs. If you make a mistake, backspace over it, and retype the text.

Returning to command mode

When you've finished entering text, press ESCAPE. When you press ESCAPE, you leave insert mode and are back in command mode.

Writing your file

Now you can **write** the file, saving what you've entered. When you're editing, it's always a good idea to save any changes you've made periodically (but don't save changes in the tutorial unless you're instructed to!). This saves extra work in the unfortunate case of a machine failure or other problem.

To write the file, type:

```
:w
```

and press RETURN.

The filename, number of lines in the file, and the total number of characters in the file appears on the bottom of the screen while the system writes the changes to the file, as shown in Figure 5-4.

Figure 5-4. Writing the file

```
They went to sea in a sieve they did;
    In a sieve they went to sea.
In spite of all their friends could say,
    In a sieve they went to sea.
And when the sieve turned round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button, we don't care a fig,
    we'll go to sea."
    Far and few
        Are the lands where the jumblies live;
    Their heads are green, and their hands are blue,
        And they went to sea in a sieve.
~
~
"jumblies" [New file] 13 lines, 475 characters
```

The number of lines and characters listed on your screen may vary from the example, depending on whether you indented the lines using tabs or spaces.

After `vi` has written the file, the cursor reappears in the text. You're still in command mode.

Other ways of inserting text

When you use `I`, you move the cursor to the front of the line you're on, and you enter insert mode. Whatever text you type next appears at the beginning of that line.

In your file `jumblies`, go to the line "`we'll go to sea.`"

The screen looks this:

```
They went to sea in a sieve they did;
    In a sieve they went to sea.
In spite of all their friends could say,
    In a sieve they went to sea.
And when the sieve turned round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button, we don't care a fig,
    we'll go to sea."
    Far and few
        Are the lands where the jumblies live;
    Their heads are green, and their hands are blue,
        And they went to sea in a sieve.
```

The boldfaced **w** shows you where your cursor will be after you type `I`. Now type

```
In a sieve
```

When you're finished, press ESCAPE. Now the line you are on looks like this:

```
In a sieve we'll go to sea."
```

When you use the `a` command, you enter insert mode and whatever text you type is inserted to the right of the cursor. In this way, the `a` command actually lets you append text.

In the `jumblies` file, go to the line, "`And when the sieve turned round`".

Put the cursor on the space between `turned` and `round`. Type `a`, then type

```
round and
```

When you've added the text, press ESCAPE. The line now looks like this:

```
And when the sieve turned round and round
```

When you use the `A` command, you go to the end of the line you're on and enter insert mode. Any text you type is appended to the right of

the cursor, so you are adding text to the end of the line.

In `jumblies`, go to the first line "`Far and few`".

Type A and then type:

```
, far and few
```

When you've added the new text, press ESCAPE.

The new line looks like this:

```
Far and few, far and few
```

When you use o or O, you open a new line in the file and automatically enter insert mode. Anything you type after the o or O appears on that new line. o opens a new line *below* the line you're on; O opens a new line *above* the line you're on.

In the `jumblies` file, go to the third line in the first stanza (ending with "`say`".

Type o.

Now the screen looks like Figure 5-5.

Figure 5-5. Opening a line using o

```
They went to sea in a sieve they did;
    In a sieve they went to sea.
In spite of all their friends could say,

    In a sieve they went to sea.
And when the sieve turned round
And everyone cried "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button, we don't care a fig,
    we'll go to sea."
    Far and few
        Are the lands where the jumblies live;
    Their heads are green, and their hands are blue,
        And they went to sea in a sieve.
```

Now type

```
On a winter's morn,
```

and press ESCAPE.

The text looks like Figure 5-6.

Figure 5-6. After using o

```
They went to sea in a sieve they did;
    In a sieve they went to sea.
In spite of all their friends could say,
On a winter's morn,
    In a sieve they went to sea.
And when the sieve turned round
And everyone cried "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button, we don't care a fig,
    we'll go to sea."
    Far and few
        Are the lands where the jumblies live;
    Their heads are green, and their hands are blue,
        And they went to sea in a sieve.
```

Now go to the fifth line, "`In a sieve they went to sea`".

Type `O`, and type

```
on a stormy day,
```

and press ESCAPE.

Now the completed poem reads as follows:

Figure 5-7. The changed **jumblies** file

```
They went to sea in a sieve they did;
    In a sieve they went to sea;
In spite of all their friends could say,
On a winter's morn,
    on a stormy day,
    In a sieve they went to sea.
And when the sieve turned round and round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button; we don't care a fig,
    In a sieve we'll go to sea."
        Far and few, far and few
            Are the lands where the jumblies live;
        Their heads are green, and their hands are blue,
            And they went to sea in a sieve.
```

Correcting mistakes using the backspace key

If you make a mistake while you're entering text (in insert mode), backspace over it and retype.

If, for instance, you typed `seive` instead of `sieve`, you could backspace to the `s`, and retype the word. When you backspace over the word, it remains on the screen until you've typed over it or pressed ESCAPE.

Once you've backspaced over the word, it's actually gone from the file, even though it still appears on the screen. If you don't type over it but instead press ESCAPE, the word disappears.

If you discover an error later in the file, there are other ways to go about correcting it. These are discussed in Lesson 3 and Lesson 4.

Make sure you've pressed ESCAPE to get back to command mode before continuing with the tutorial.

Saving your text and quitting the `jumblies` file

Now you're ready to quit the `jumblies` file, writing all your changes.

Type

```
:wq
```

and press RETURN.

When the file is written, you return to the shell once again. Then, if you list your files using `ls`, you'll see `jumblies` along with the rest of your files.

Lesson 3: Editing a file: `jumblies`

Now that you've created `jumblies`, you can edit it. When you exited and saved `jumblies`, you exited `vi` as well, so to edit `jumblies`, you must invoke `vi` and specify the filename.

Type

```
vi jumblies
```

`jumblies` appears just as you left it.

Deleting text

In command mode, there are several ways to delete text. Here is a list of text deletion commands.

```
x
X
d
D
dd
```

The `x` command deletes whatever character the cursor is on. If you type a number and then `x`, that number of characters is deleted.

Put the cursor on the `i` of `Their` in "`Their heads are green...`".

The line looks like this:

```
Their heads are green, and their hands are blue;
```

Type 2x.

The line now looks like this:

```
The heads are green, and their hands are blue;
```

X deletes the character to the left of the cursor. If you type a number, followed by X, that number of characters is deleted.

In jumblies, leave the cursor on the blank after The, which is now the first word of the line.

Type 3X.

The line now looks like this:

```
heads are green, and their hands are blue;
```

The d command deletes letters and words in a number of ways. For instance, if you type dl, the letter the cursor is resting on is deleted.

In the jumblies file, put the cursor on the T of They in the first line.

Type dl.

The line now looks like this:

```
hey went to sea in a sieve they did;
```

dw deletes the word the cursor is resting on.

Leaving the cursor on the h of hey in the first line, type dw.

The line now looks like this:

```
went to sea in a sieve they did;
```

db deletes the part of the word to the left of the cursor. If you type db when the cursor is between words, the word to the left of the cursor is deleted.

Move the cursor to the space between went and to in the first line.

Type db.

The line now looks like this:

```
to sea in a sieve they did;
```

If you type a number followed by `dl` (or `dw` or `db`), it specifies how many characters, words, or parts of words to delete (for example, `3dw` deletes three words).

`D` deletes text from the cursor position to the end of the line.

Move the cursor to the beginning of the word `sieve`. Type `D`.

Now the line looks like this:

```
to sea in a
```

The rest of the line is deleted. Placing a number in front of `D` has no effect.

You can use the `dd` command to delete the line the cursor is on. If you place a number in front of `dd`, that number of lines is deleted.

Leave the cursor on the line "`to sea in a ...`" and type `dd`.

The screen looks like Figure 5-8.

Figure 5-8. Deleting lines: **dd**

```
  In a sieve they went to sea;
In spite of all their friends could say,
On a winter's morn,
    on a stormy day,
    In a sieve they went to sea.
And when the sieve turned round and round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button; we don't care a fig,
    In a sieve we'll go to sea."
        Far and few, far and few
            Are the lands where the jumblies live;
            And they went to sea in a sieve.
```

Copying and moving text

You can copy text from one place in your document and put it elsewhere in the same file using the `y` and `p` commands.

When you use `y` to "yank" and copy text, the copied text is stored in a buffer, leaving the original text untouched. After you've yanked and copied text with the `y` command, you can use `p` to put the text where you want in the file.

You can use the `y` command to copy letters and words in a number of ways. If you type `yl`, the letter the cursor is on is copied. `yw` copies the word the cursor is on. `yb` copies the part of the word to the left of the cursor. If you type `yb` when the cursor is between words, the word to the left of the cursor is copied.

If you type a number followed by `yl`, `yw`, or `yb`, that number of characters, words, or parts of words is copied (for example, `3yw` copies 3 words).

`yy` copies an entire line. If you type a number followed by `yy`, that number of lines is copied and put into the buffer. `Y` works the same way as `yy`.

After you've yanked a copy of the text, you can use `p` to put the copy where you want it in the file.

When you type `p`, the text is inserted into the file to the right of the cursor. Alternatively, you can use `P`, which puts the text to the left of the cursor. If you've yanked an entire line, `p` inserts the line below the line where the cursor rests; `P` inserts the line above the current line.

Place the cursor on the `F` of the first "`Far and few`".

Type `3yy`.

Although you won't see any change, those lines *are* copied and waiting in the buffer.

Now place the cursor on the last line.

Type `p`.

The screen now looks like Figure 5-9.

Figure 5-9. After using `y` and `p`

```
They went to sea in a sieve they did;
    In a sieve they went to sea;
In spite of all their friends could say,
On a winter's morn,
    on a stormy day,
    In a sieve they went to sea.
And when the sieve turned round and round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button; we don't care a fig,
    In a sieve we'll go to sea."
        Far and few, far and few
            Are the lands where the jumblies live;
            And they went to sea in a sieve.
        Far and few, far and few
            Are the lands where the jumblies live;
            And they went to sea in a sieve.
```

Fixing transposed characters

An easy way to transpose two characters is to place the cursor on the first letter and type `xp`. The `x` command deletes the first letter, and the `p` command puts it behind the second letter.

For example, put the cursor on the `i` of `sieve` anywhere it appears in the poem.

Type `xp`.

`sieve` becomes `seive`. Now place the cursor on the first `e` in `seive` and type `xp`.

`seive` returns to `sieve`.

This works because when you delete part of a file using `x` or `d`, the deleted text actually stays in a buffer until you use another command.

Therefore, if you're careful, you can use `x` and `d` along with `p` to move text around in a file. You must use caution if you try this, because it's

easy to make a mistake and lose the text you're trying to move. However, if you make such a mistake, you can retrieve your text before it's too late. See "Recovering Lost Text" in Chapter 4, "Using `vi`," in *A/UX Text Editing Tools.*

Undoing your changes

You can easily undo many of the mistakes you make when you're editing by using the `u` or `U` command (you can use `U` as long as you haven't left the line where you made the mistake).

Typing `u` will undo the last operation you performed.

For example, even if you've moved ahead in the file using CONTROL-d, the last operation *and* the CONTROL-d will be undone. You'll end up back where you were before. Put the cursor on the `I` of "`In spite`" on the third line.

Type `dl`.

The line currently looks like this:

```
n spite of all their friends could say,
```

Now type `u`.

The line looks like this:

```
In spite of all their friends could say,
```

Type `u` again.

Once again, the line begins, "`n spite of...`".

Type `u` once more, and it's back to normal.

You can undo all the changes you've made to a line if you type `U` *before* you've left the line. Once you've left the line, however, it's too late for `U` to have any effect whatsoever.

> *Note:* You can use `u` to undo the very last change you made in the line *only* if you haven't tried `U` first. This is because `u` undoes the most recent command which, in that case, would be the unsuccessful `U`.

Undoing all changes since you last wrote the file

If, for some reason, you want to discard all the changes you've made to a file in your session and just get the old version of the file back, you can quit the file by typing a colon followed by `q!`, then pressing RETURN.

All the changes you've made are discarded and the version of the file that remains is the version you had before you started changing things.

Since you haven't written any changes to `jumblies` in this session, type

```
:q!
```

and press RETURN.

All the changes and deletions you've practiced in this lesson are discarded; you have the original `jumblies` back, safe and sound.

Lesson 4: Advanced editing

Replacing text

The `r` command replaces a character with whatever you type. You can use the `r` command to replace letters in a number of ways.

Open the `jumblies` file using

```
vi jumblies
```

Place the cursor on the letter `j` in the word `jumblies`.

Type `rx`.

Now the line looks like this:

```
Are the lands where the xumblies live;
```

`j` is replaced by the letter `x`. Now type `u` to change `xumblies` back to `jumblies`.

If you type a number followed by `r`, that number of characters is replaced by whatever you type next. For example, `3rx` replaces the three letters to the right of the cursor with `xxx`.

R lets you replace an arbitrary number of characters within the current line. When you type R, you overwrite the existing text until you press ESCAPE.

Place the cursor on the letter A of the line

```
Are the lands where the jumblies live;
```

Type R and then type "Far, far off" followed by ESCAPE. The line looks like this:

```
Far, far offs where the jumblies live;
```

Type u to return the line to normal.

Place the cursor on the j of jumblies.

Type R and then type the following, ending each line by pressing RETURN:

```
deer and the antelope play,
where seldom is heard a discouraging word
and the skies are not cloudy all day.
```

Now press ESCAPE.

The line looks like this:

```
Are the lands where the deer and the antelope play
where seldom is heard a discouraging word
and the skies are not cloudy all day.
```

Type u to return the lines to normal.

Changing text

The c command lets you change the text. You can use the c command to change letters and words in a number of ways. A $ appears at the end of the text to be changed.

If you type cl, you can change the letter the cursor is on. cw lets you change the part of the word to the right of the cursor. cb lets you change the part of the word to the left of the cursor. If you type cb when the cursor is between words, $ will appear before the word to the left of the cursor.

If you type a number followed by `cl`, `cw`, or `cb`, that number of characters, words, or parts of words can be changed.

For example, go to the first letter in the phrase beginning "`In spite of all...`".

Type `3cw`.

It looks like this:

```
In spite o$ all their friends could say,
```

Type

```
Despite
```

and press ESCAPE.

The changed line appears.

```
Despite all their friends could say,
```

Now place the cursor on the `D` of `Despite`.

Type `cw` and change it back to the original "`In spite of.`"

`cc` lets you change an entire line. The line disappears from the display, leaving an empty line. Whatever you type next will replace the old text. If you type a number followed by `cc`, that number of lines is removed for you to type over.

`C` lets you change a line from the cursor position to the end of the line. Instead of removing the line, `vi` simply marks the end of the line with a dollar-sign character (`$`) and replaces the text of that line with the text you type in. You can type beyond the `$`. If you do, you continue to insert text until you press ESCAPE.

Joining lines

You can join lines in a file using the `J` command. You can break lines in a file by inserting a RETURN.

Put the cursor on the line

```
On a winter's morn,
```

Type J.

These combined lines now look like this:

```
On a winter's morn, on a stormy day,
    In a sieve they went to sea.
```

Any overlap is wrapped around to the next line.

To break a line, place the cursor where you'd like the first line to end. If you want to break the line to the left of the cursor, type i to enter insert mode, and press RETURN. The line breaks in two. Press ESCAPE. If you want to break the line to the right of the cursor, type a to enter insert mode, and press RETURN. The line breaks in two. Press ESCAPE.

Finding out your current line number in the file

To find out what line number you're on, press CONTROL-g.

In jumblies, go to the line "With only a beautiful pea-green veil,".

Press CONTROL-g.

The last line on the screen now looks like this:

```
"jumblies" line 17 of 79 --22%--
```

The command line

vi and ex are two aspects, or modes, of the same text editing program.

- vi is a screen editor that displays the contents of a file a full screen at a time.
- ex is a line editor that operates on a specified set of lines. With ex, you don't usually see the text you're working with.

However, several ex commands are invaluable when you're editing with vi. When you are in vi command mode you can get the ex command line by typing the colon character (:). This moves the cursor to the bottom line of the screen, preceded by the colon (the ex prompt). You can give any ex command at this prompt. You return to vi automatically after the command is executed.

The following sections explain how to use a few of these `ex` commands. For more information about `ex`, please refer to "Using `ex`" in *A/UX Text Editing Tools*.

Reading in the contents of another file

Sometimes it can be helpful to copy text from another file into the file you're working with. The `:r` command does this.

Place the cursor on the last line. Type `o` to open the new line, then press ESCAPE to return to command mode without inserting any text.

Type

```
:r sample
```

and press RETURN.

The file `sample` is inserted and the screen looks like Figure 5-10.

Figure 5-10. Copying text from another file

```
And when the sieve turned round and round
And everyone cried, "You'll all be drowned!"
They called aloud, "Our sieve ain't big,
But we don't care a button; we don't care a fig,
    In a sieve we'll go to sea."
        Far and few, far and few
            Are the lands where the jumblies live;
            And they went to sea in a sieve.

They sailed away in a sieve, they did,
        In a sieve they sailed so fast,
With only a beautiful pea-green veil,
Tied with a ribbon, by way of a sail,
        To a small tobacco-pipe mast.
And everyone said who saw them go,
"Oh! won't they be soon upset, you know,
For the sky is dark, and the voyage is long;
And, happen what may, it's extremely wrong
        In a sieve to sail so fast."
```

Scroll down through the file using CONTROL-d and back up using CONTROL-u.

For further information about the `r` command, please refer to *A/UX Text Editing Tools.*

Displaying line numbers

You can display the line numbers for each line of your file by typing

```
:set nu
```

The line numbers appear along the left margin of the screen.

If you want to get rid of the line numbers, type

```
:set nonu
```

and the numbers disappear.

Saving part of your file in another file

You can save part of your file in another file if you wish. This can be really useful if you're making a "boilerplate" document or if you plan to use a segment of text in many other documents.

To save part of a file in another file, you can specify one line or a range of lines plus the name of the file that is to contain them.

For example, to write the fourth line of the file you're using in this lesson to the file named `saveit` you'd type

```
:4w saveit
```

If you wanted to write lines 4 through 400 to `saveit` you'd type

```
:4,400w saveit
```

Note that the comma means 4 *through* 400, not 4 *and* 400.

If you try to write lines 4 through 400 now, because you don't have that many lines in your file, you get the message

```
Not that many lines in buffer
```

If you want to write the last portion of a file, for example, from line 11 through the end of the file, you'd type

```
:11,$w saveit
```

because `$` means the end of the file.

If you try to do that now, you get the message

```
File exists - use "w! saveit" to overwrite
```

You need to overwrite `saveit` because you've already saved line 4 in it. Type

```
:11,$w! saveit
```

and lines 11 through the end of the file are saved in `saveit`, overwriting what was previously in `saveit`.

Other ways to quit `vi`

There are three ways to quit `vi`, depending on what you want to do. One way is to quit the file and discard any changes (`:q!`). There are two ways to quit the file while writing the changes (`:wq` and `ZZ`). Although `q!` and `wq` were used in other lessons, they're reviewed briefly here as well.

You can quit a file without writing the changes made to it with the `q!` command. Type

```
:q!
```

and press RETURN.

This can be risky because all the changes you've made in the file will be lost, unless you've used the `w` command to write changes along the way. However, `q!` can be very helpful as well, especially if you've made an error, such as accidentally erasing a large part of your file, and you want to quit without saving your error.

You can write the file you're in, changes and all, using the `wq` command, for "write/quit." You type

```
:wq
```

and press RETURN.

As the system writes the file, the filename, number of lines, and total number of characters appear on the bottom line. After the file is written, you exit to the shell.

Another way to leave a file and write the changes you've made is to type

```
ZZ
```

(*without* a colon first). This command also writes the file and returns you to the shell.

To leave the file `jumblies` and write the changes you've made in this tutorial, type

```
ZZ
```

without a colon and without pressing RETURN.

`:wq` always writes the file, even if there were no changes; `ZZ` writes the file only if there were changes.

Once you've become familiar with these basic commands for moving around in a file, you may be interested in the more specialized `vi` text movement commands. To find out about these commands, please refer to Chapter 4, "Using `vi`," in *A/UX Text Editing Tools.*

Printing a file

If you have a printer and you want to print a file, see "Printing a File" in Chapter 4 of this manual. If you need to set up a printer and A/UX print scheduler program, please consult your system administrator or refer to *A/UX Local System Administration.*

Chapter 6
`mail` Tutorial

Contents

Figures

Chapter 6

`mail` Tutorial

Communicating with other users: `mail`

You can use the `mail` program to send messages to, and receive messages from, other users on your machine. If your machine is connected to other machines using either the UUCP or B-NET network facility, you can send mail to users in other locations. When you receive mail, the `mail` program notifies you.

The `mail` program offers a wide range of options, but what you can do with `mail`, and what it can do for you, depends a lot on your situation. Here are a few of the things `mail` can do for you:

- If you share your computer, you can send `mail` messages to the other users, and they can send them to you.
- If you belong to a network, you can send mail to, and receive mail from, the other people on the network. When you're sending mail to people on a network, you have to address the mail differently than if you're sending mail to people on your machine. To learn how to address mail to people on a network, please see "Sending Mail Over UUCP" or "Sending Mail Over B-NET" later in this chapter.
- If you are the only user on your machine and you don't have network access, you can still send mail to yourself.

This chapter describes the `/usr/bin/mailx` program, not `/bin/mail`. We have defined the name `mail` in `/etc/profile` so that it runs the program `mailx`. For more information, please see "Note to the System Administrator" in this manual, "Using `mail`" in *A/UX Communications User's Guide*, and `mail`(1) and `mailx`(1) in *A/UX Command Reference*.

Sample session: trying out `mail`

Try this sample session with `mail` to get an idea of how it works. If you are in `mail` now, you will see the `?` `mail` prompt. Type `q`, and press RETURN to return to the shell prompt `$`.

At the shell prompt, type

```
mail your-name
```

using your login name (in this case, `start`). Next, `mail` asks you the subject:

```
$ mail start
Subject:
```

At the subject prompt, type

```
sample session
```

and press RETURN. The cursor waits on the next line, ready for you to insert text.

Now type

```
How much wood would a woodchuck chuck
If a woodchuck could chuck wood?

He'd chuck all the wood
That a woodchuck could,
If a woodchuck could chuck wood.
```

Now the screen looks like Figure 6-1.

Figure 6-1. `mail`: sample session message

```
$ mail start
Subject: sample session
How much wood would a woodchuck chuck
If a woodchuck could chuck wood?

He'd chuck all the wood
That a woodchuck could,
If a woodchuck could chuck wood.
```

Press RETURN to start a new line, then type CONTROL-d to send the message. The system inserts EOT signifying you've sent the message successfully (Figure 6-2).

Figure 6-2. `mail`: sending the message

```
$ mail start
Subject: sample session
How much wood would a woodchuck chuck
If a woodchuck could chuck wood?

He'd chuck all the wood
That a woodchuck could,
If a woodchuck could chuck wood.
EOT
$
```

Wait a couple of minutes for the mail to arrive. Press RETURN a few times. When the mail has arrived, you get the shell prompt and the message

```
You have mail.
```

Type

```
mail
```

to read your mail. `mail` lists your messages. You should have at least 1 message and the screen will appear similar to Figure 6-3.

Figure 6-3. `mail`: message list

```
A/UX Release 1.0 Type ? for help.
"usr/mail/start": 1 message 1 new
>N 1 start  Fri Apr 10 12:05   12/220   sample session

?
```

Type `1` (or the number of the message you just sent yourself, if it's different) and press RETURN to read the message. The screen looks like Figure 6-4.

Figure 6-4. `mail`: reading the message

```
Message 1:
From start Fri Apr 10 12:19 PDT 1987
To: start
Subject: sample session
Status: R

How much wood would a woodchuck chuck
If a woodchuck could chuck wood?

He'd chuck all the wood
That a woodchuck could
If a woodchuck could chuck wood.


?
```

Save the message in `/users/start/mbox` by typing CONTROL-d to exit `mail`. The message is automatically saved. While it's saving the message, the system prints

```
Held n message in /usr/mail/start
```

where *n* is the number of messages it's saved.

You can also exit `mail` using `q` or `quit`; the effect is the same.

When you send mail

Sending mail

To send a message to someone, type `mail` *name*, where *name* is the addressee's login name.

For example, to send mail to someone whose login name is `fred`, type

```
mail fred
```

Then type your message. At the end of your message, type CONTROL-d. `mail` prints `EOT` (for "end of transmission") and returns you to the shell. When `fred` logs in the next time, he receives the message

```
You have mail.
```

His mail is the message you typed, preceded by a line telling him you sent the message (your login name), and the date and time it was sent.

Canceling a message you don't want to send

If, while you're writing a message, you decide not to send it, you can cancel the message by sending the *interrupt* signal to the system. When you type *interrupt*, `mail` prints

```
(Interrupt -- one more to kill letter)
```

Type *interrupt* a second time to cancel the letter. Cancelled letters are stored in the file `dead.letter` in your home directory.

Sending mail to someone on your computer

To send a message to someone on your computer, follow the same steps as in the preceding example. Just start the message with

```
mail name
```

where *name* is the login name of the person you're addressing.

Sending mail over B-NET

Sending mail to someone over B-NET is just like sending mail to anyone else except the form of address is different.

If, for example, you're sending mail to the user `iggy` on a remote machine over B-NET, you start by typing

```
mail iggy@rhost
```

(where *rhost* is the name of the remote host). The rest of the process works the same way as you learned in the sample session.

Sending mail over UUCP

Sending mail to someone over UUCP is just like sending mail to anyone else except the form of the address is different.

If, for example, you're sending mail to the user `huey` on a remote machine over UUCP, you start by typing

```
mail rhost!huey
```

(where *rhost* is the name of the remote host). The rest of the process works the same way as you learned in the sample session.

UUCP can use other machines that receive and forward mail to send messages all over the world. In this case, you need to use a "path" that specifies each machine along the way, for example,

```
mail rhost1!rhost2!rhost3!huey
```

For more information, refer to *A/UX Communications User's Guide*.

Mailing to more than one person

To send the same message to more than one person, list each addressee's login name on the command line when you invoke `mail`.

For example, to send the same message to `john`, `paul`, `george`, and `ringo`, you'd type

```
mail john paul george ringo
```

and proceed as with any `mail` message.

If you want to send mail to more than one person on the same remote machine over B-NET, you must type each person's address separately. For example, if the users `john`, `paul`, `george`, and `ringo` were all on a remote machine over B-NET, you'd type

`mail john@`*rhost* `paul@`*rhost* `george@`*rhost* `ringo@`*rhost*

(where *rhost* is the name of the remote host) and proceed as with any `mail` message.

If you want to mail to more than one person on the same remote machine over UUCP, you must type each person's address separately. For example, if the users `john`, `paul`, `george`, and `ringo` were all on the machine *rhost* over UUCP, you'd type

`mail` *rhost*`!john` *rhost*`!paul` *rhost*`!george` *rhost*`!ringo`

(where *rhost* is the name of the remote host) and proceed as with any `mail` message.

Finding out if you've received mail

The `mail` system collects your incoming messages in a file called the "system mailbox." If you receive a message while you're working, you'll see the message

```
You have mail.
```

the next time you're in the shell. The A/UX system has a default (set in the `/etc/profile` system file) that checks every 60 seconds to see if new mail has arrived for you.

When you read mail

Reading mail

When you want to read your mail, exit any interactive program you might be running (such as `vi`) so that you see the shell prompt. Then type

```
mail
```

If you don't have any mail, the message

`no mail for` *name*

appears, where *name* is your login name.

When you have mail, each message is summarized in a numbered list that shows the author's login name, the subject, and the date it was

sent. To display this summary, type `h`.

To read the first message, just press RETURN. The first message is displayed. A sample message appears in Figure 6-5.

Figure 6-5. A sample `mail` message

```
Message 1:
From start Fri Apr 17 09:53 PST 1987
To: start
Status: RO

Mr. Abdul Haffar
Haffar Elephant Brokerage
26 West Portal Ave.
Marrakesh, Morocco

Dear Mr. Haffar,

We are enclosing a deposit on the shipment of livestock
which we agreed you would send to us next Tuesday.
We will be expecting two dozen elephants with attendants.
We will send you the balance upon receipt of the
goods.

If this shipment works out well, we look forward to
future dealings with the Haffar Elephant Brokerage.

Thank you very much.

                          Sincerely yours,


                          Hannibal


?
```

The header at the top of the message tells you who sent the message, when it was sent, and what it's about.

After you read the message, press RETURN, and the next message is displayed, and so on.

If you want to read a specific message from the list, you can do so by specifying the message number. For example, if you have 11 messages and you want to read message 7, type `7`, and press RETURN. Message 7 is displayed. If you press RETURN again, message 8 is displayed.

Deleting a message

If, after you've read a message, you want to delete it, you can type `d` and press RETURN. The message is deleted when you leave `mail`.

If you have finished reading all your messages but want to delete one of them, you can specify which message you want to delete.

For example,

```
d 5
```

deletes message 5.

Saving a message in a particular file

If you want to save a message in a file, you tell `mail` to save it and give it a filename. If the filename you give doesn't exist, the system creates the file and puts the message in it.

For example, if you type

```
s 4 first.class
```

`mail` saves the fourth message on the list in the file named `first.class`.

Exiting `mail`

To quit `mail` and leave your messages as if you hadn't touched them, type

```
x
```

The `x` (for "exit") command undeletes any deleted messages, places any messages you read back on the "new message" list, and

terminates the `mail` program.

To quit `mail` the usual way, type

```
q
```

The `q` (for "quit") command removes messages you marked for deletion, places any unread messages on the "unread" list, and terminates the `mail` program.

If you change your mind and wish to restore a deleted message before you have exited the `mail` program, use the `x` command to leave `mail`. If you exit `mail` using `q` you cannot restore a message you have deleted.

Additional `mail` capabilities

`mail` has several other capabilities that you will find helpful once you've become accustomed to working with it, such as editing messages, replying to messages, and so on. For more information about these and other `mail` functions, please refer to the *A/UX Communications User's Guide*.

Chapter 7
Setting Up Your Own Account

Contents

Chapter 7

Setting Up Your Own Account

Your system administrator can follow these steps to convert the `start` account into a regular user account with whatever name you'd like. If you are your machine's system administrator, you can use this procedure; otherwise, see your system administrator.

When you set up your own account, you must:

- Become superuser. (The superuser or `root` login account has unrestricted permissions.)
- Choose a login name. Your login name is usually your real name or your initials.
- Copy and modify the `/etc/passwd` file. `/etc/passwd` is a system file that contains information about users. It is important to copy it before modifying it so you have an immediate backup if you accidentally delete required information.
- Create your home directory and copy the `.profile` startup file from the `start` directory into it. See "Using Different Shells" if you intend to use the Korn shell or C shell.
- Change ownership of your home directory. Your new login name should own all of your files. You may also want to create a new group name and change the group ownership of your files. See "Permissions" in *A/UX Local System Administration* for information on group ownership.
- Change ownership of `.profile`. Because `.profile` begins with a period, it needs to be specified explicitly when you are changing ownership. See "Using Different Shells" if you intend to use the Korn shell or C shell.
- Give yourself a password.

- Log out and log in using your new login and password.

Perform these steps using the directions in the next sections.

Become superuser (`root`)

Enter the command

```
su
```

After you've pressed RETURN, A/UX asks you for a password

```
Password:
```

You specified the root password during the installation procedure. Type the root password and press RETURN. When you type the root password, nothing appears on the screen; this is to preserve privacy. When you press RETURN however, A/UX reads the password and, if it's correct, you're logged in (note that the number sign (#) is the `root` user's prompt). If it's not correct, you get the message

```
su: Sorry
```

In this case, you are still logged in as `start`, and you need to try again to log in as superuser.

Choose a login name

Once you are logged in as superuser, you're ready to choose your own login name. The name should consist of lowercase alphabetic characters, for example, `joel` or `jrb`.

First, you need to make sure that the name you want to use isn't already being used. You can do this by checking the `/etc/passwd` file. (Only superuser can change this file, which contains user information that is essential to the system.)

To check, enter the command

```
grep name /etc/passwd
```

(where *name* is a login name using lowercase characters.) This command searches the `/etc/passwd` file to see if there is already a

user with the name you have chosen.

Only lines starting with *name* indicate that the name is already in use. If no other user has your name as a login name, the command returns the shell prompt. If your name is already being used, the command displays a line of text and code. Pick another name and try again.

Copy and modify the `/etc/passwd` file

Now that you have your new login name, you need to modify the `/etc/passwd` file to include an entry for your login name. You can do this using `vipw` (see `vipw`(1M) in *A/UX System Administrator's Reference*). It makes sure no one else is editing the password file when you are, but otherwise works the same as `vi`. However, you should make a copy of `/etc/passwd` before you modify it, in case you make an incorrect change. Copy `/etc/passwd` by typing

```
cp /etc/passwd /etc/passwd.copy
```

This way, if anything goes wrong, you still have a copy in safekeeping.

After you have made the copy, enter the command

```
vipw /etc/passwd
```

`vipw` uses a temporary file named `/etc/ptmp` to do its editing. When `vipw` opens this file, your screen looks something like this:

```
root:b92ajisaG@k6d:0:0::/:
rootcsh:b92ajisaG@k6d:0:0::/:/bin/csh
rootksh:b92ajisaG@k6d:0:0::/:/bin/ksh
daemon:*:1:1:Daemon:/:
bin:*:2:2:Bin:/bin:
sys:*:3:3:System:/sys:
adm:*:4:4:Administration:/usr/adm:
uucp:*:5:5:UUCP.admin:/usr/lib/uucp:
nuucp:*:5:5:UUCP admin:/usr/spool/uucppublic:/usr/lib/uucp/uushell
lp:*:7:7:lp:/usr/spool/lp:
ftp:*:8:2:ftp:/usr/spool/ftp:
who::22:0:who command:/bin:/bin/who
nobody::60001:60001:NFS generic user:/tmp:/bin/noshell
start:PG/qLJaYo/6mo:100:100:initial login:/users/start:/bin/sh
```

In `vipw`, move the cursor to the beginning of the line that reads

```
start:PG/qLJaYo/6mo:100:100:initial login:/users/start:/bin/sh
```

Copy the line by pressing ESCAPE (to make sure you are in command mode) and typing

```
yy
```

Place the copied line below the current line by typing

```
p
```

Then give the command

```
:s/start/name/g
```

where *name* is your new login name. The line should now look like this:

```
name:PG/qLJaYo/6mo:100:100:initial login:/users/name:/bin/sh
```

You should also change the `initial login` field in this line to contain your real name.

```
:s/initial login/name/g
```

Note: You do not need to change the two fields containing the number `100` if you are the only user on your system. If you are not the only user on your system, see your system administrator or *A/UX Local System Administration.*

Now you can write your changes and quit the file. First press ESCAPE, then type

```
:wq
```

and press RETURN.

Note that you can only modify `/etc/passwd` when you are logged in as `root`. This is one of the special privileges of the `root` account.

Create your home directory

When you have entered a suitable login name, type

```
mkdir /users/name
```

to create your new home directory.

Copy startup files into your home directory

If you intend to use the Bourne shell as your login shell (the shell that is invoked by the system whenever you log in), you need to copy the `.profile` file from `/users/start` into your new home directory. See "Using Different Shells" if you intend to use the Korn shell or C shell instead of the Bourne shell.

First, type

```
pwd
```

to find out which directory you're in. If you're in any directory other than `/users/start`, type

```
cd /users/start
```

Then type

```
cp .profile /users/name
```

to copy the `.profile` file into your new home directory. After this is completed, move to your home directory using this command:

```
cd /users/name
```

and type

```
ls -la
```

Change ownerships of your home directory

Now you need to change the user ownership and group ownership of your home directory using the `chown` (for "change ownership") and `chgrp` (for "change group") commands.

Enter the commands

```
chown name /users/name
chgrp project /users/name
```

(where *name* is your new login name). For more information about these commands, please refer to "File Permissions" in Chapter 3 of *A/UX Local System Administration.*

> *Note:* The group name `project` is an arbitrary name assigned to the `start` account. You may use this group name for your account or create a new one. If you want to assign a different group name, you need to modify the `/etc/group` file. See *A/UX Local System Administration* for more information on modifying `/etc/group`.

Change ownership of `.profile`

Now you need to change the ownership of `.profile` so that your new account owns it instead of the `root` account.

Enter the commands

```
chown name .profile
chgrp  project .profile
```

(where *name* is your new login name).

For more information about these commands, please refer to "File Permissions" in Chapter 3 of *A/UX Local System Administration.*

Give yourself a password

Now that you have your own account set up, you should give yourself a password. Remember that you are still logged in as `root`, so you must specify your new login name to the `passwd` command:

```
passwd name
```

You'll be asked to enter a new secret password

```
new passwd: your-password
```

where *your-password* is any character string that

- is at least six characters long (the first eight are the significant characters)
- has at least two alphabetic characters (uppercase or lowercase)
- has at least one numeric or special character
- is different from your login name

The prompt asks you to enter the new password twice. If you don't type the same word both times, it asks you to try again. If the password is too short (fewer than five characters), it asks you for a longer name.

Now you are ready to log out as `root` and to log in as yourself. You do this by typing

```
exit
```

Log out and log in using your new login and password

It is a good idea to log out of the system now and log in again with your new login name. The system reads all the changes you have made. When you see the shell prompt again, type

```
exit
```

When you see the login prompt

```
login:
```

enter your new login name. You are asked for a password. Enter your new password. When you are logged in, you are in your new home directory.

Using different shells

This manual assumes that you're using the Bourne shell. If you want to use the Korn shell or C shell instead, you can set your account up to automatically use one of these other shells.

- To use the Korn shell, type

```
cp .profile .sh.profile
```

to preserve the Bourne shell `.profile` file. Now type

```
cp /usr/adm/ksh.profile $HOME/.profile
```

then

```
cp /usr/adm/ksh.kshrc $HOME/.kshrc
```

Now you're ready to use the `chsh` or "change shell" command to change the shell you're using. Type

```
chsh name /bin/ksh
```

This changes your login shell from the Bourne shell to the Korn shell and you're ready to go.

- To use the C shell, type

```
cp /usr/adm/csh.login $HOME/.login
```

then

```
cp /usr/adm/csh.cshrc $HOME/.cshrc
```

Now you're ready to use the `chsh` or "change shell" command to change the shell you're using. Type

```
chsh name /bin/csh
```

This changes the shell from the Bourne shell to the C shell and you're ready to go.

Congratulations! You've finished *Getting Started With A/UX* and are ready to work with A/UX!