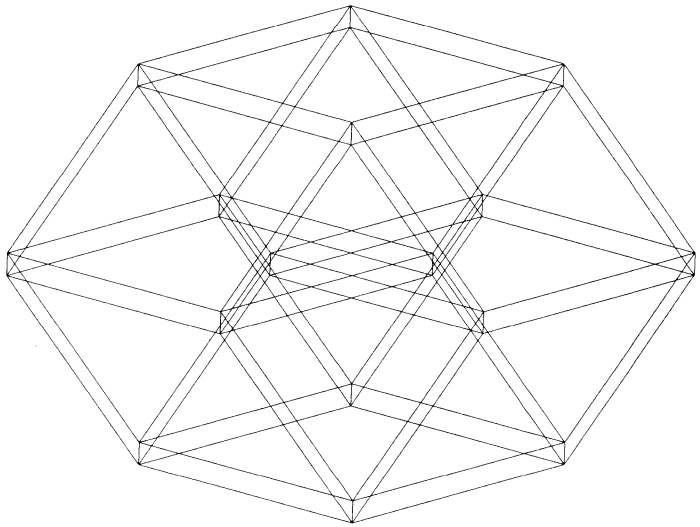
Getting Started with Your IRIS Workstation



Silicon Graphics, Inc.
2011 Stierlin Road
Mountain View, CA 94043

Document Number 007-7001-010

Technical Publications:

Marcia Allen
Lori Blankenbecler
Robin E. Florentine
Gail Kesner
Steven A. Locke
Susan Luttner
Amy B. W. Smith
Celia Szente
Diane M. Wilford

Getting Started with
Your IRIS Worsktation
Document number: 007-7001-010

To the Reader

Getting Started with Your IRIS Workstation is designed for programmers with little or no UNIX experience. By investing only one hour in this book, you will learn how to:

- issue some basic UNIX commands
- use the text editor, vi
- create and run two simple graphics programs

You will also learn where you can find more detailed sources of information about UNIX and your IRIS Workstation.

Table of Contents

1. Trying Out Your IRIS

To get started with your IRIS, you need access to a working system. If your IRIS is not installed yet, see chapters 2-4 of the *IRIS Workstation Guide, Series 2000* or *IRIS Series 3000 Owner's Guide* to learn how to install, boot, and back up your system. You will also learn how to run the flight simulator demo.

This chapter takes you through a typical session with the IRIS, after it has been installed and booted. It shows you how to:

- log in to the guest account
- issue commands to the UNIX operating system
- create simple C and FORTRAN programs using the vi text editor
- compile and run those programs
- log out

We use the following conventions in this book:

1. All text that you will see on your IRIS screen, whether you type it or the IRIS displays it, is printed in this **typewriter font**.
2. UNIX file names and commands are printed in *italics*.
3. Special keys you press, such as **RETURN**, are printed in a box.

Your IRIS is case-sensitive; that is, it distinguishes between upper and lower case letters. If we show you a command that consists of all lower case letters, or one that consists of a combination of lower and upper case letters, be sure to type in the command exactly as it is printed on the page.

Logging In

After you boot the IRIS, it is running the UNIX operating system and is waiting for you to log in. This is what you see on the screen:

```
IRIS login:
```

To log in, type:

```
guest RETURN
```

This is what you see:

```
IRIS login: guest

Silicon Graphics, Inc.
IRIS Workstation

%
```

Chapter 4 of your *IRIS Workstation Guide, Series 2000* or *IRIS Series 3000 Owner's Guide* tells you how to set up user accounts and passwords so you can log in using your own login name. For now, you are logged in to the "guest" account.

Issuing Commands

The percent sign is the system prompt; it means the IRIS is ready to accept your commands. After you type in the name of a command, press RETURN and the command is executed. Erase characters by pressing the BACK SPACE key. Abort a command and start over on a new line by pressing the BREAK key, or by pressing CNTRL-U.

Try issuing this command:

```
date RETURN
```

The system responds with the time and date:

```
% date
Tue Dec 10 10:25:14 PST 1985
%
```

After the IRIS executes the command, it displays a percent sign to prompt you for more commands.

Creating Graphics Programs Using vi

Now that you are logged in and you know how to issue commands, you can do what you were meant to do with the IRIS — write, compile, and execute a graphics program. In the next sections, you actually create two programs: one in C and one in FORTRAN. (Note: FORTRAN is an option on the IRIS, so you can compile and run the FORTRAN program only if you have this option.)

You write programs (i.e. create source files) on the IRIS by using the vi text editor. The C and FORTRAN programs you write in this chapter produce the same effect on the screen, but the algorithms are slightly different, as you will see. We recommend that you write both programs (if your IRIS has the FORTRAN option), even if you intend to use only one of the languages.

After you have created a source file, you need to compile it. There are two compiler commands: *cc* and *f77*. *cc* uses your C source file to create an executable file, and *f77* uses your FORTRAN source file the same way. After compilation, the original source file is still available to be changed and compiled again.

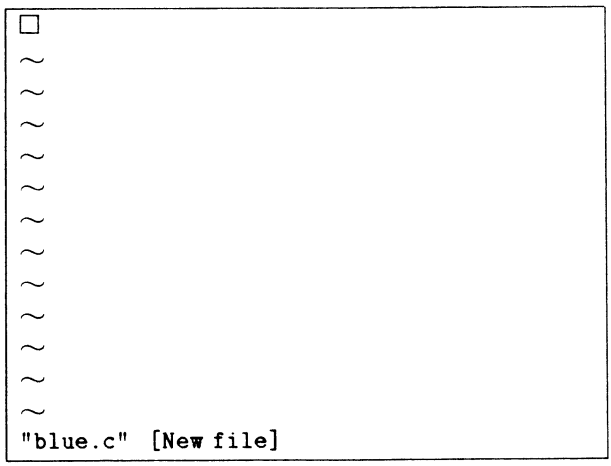
To execute the compiled file, type the name of the file as you would type the name of a UNIX command (such as **date**).

Starting the vi Text Editor

First, create the source file for the C program by typing:

```
vi blue.c RETURN
```

This starts the vi (vee-eye) text-editing program. *blue.c* is the name of the file. The ".c" is a required convention that means the file is a source file in the C programming language. Your screen looks like this:



The cursor box at the top of the screen points to the place in the file where you can insert text. The arrow keys at the upper right corner of the main keyboard move the cursor around in the file. Since *blue.c* is empty, there is only one place for the cursor to be. If you try to move it, the IRIS will beep. The tildes (~) fill up the part of the screen that does not yet contain text.

Inserting Text

vi has two modes: **command mode** and **insert mode**. Right now it is in command mode — the next character you type will be interpreted as a command. The first command you need to know is `i`. When you type `i`, you put vi into insert mode, and vi inserts any new text **before** the cursor position. Everything you type after `i` (including carriage returns) is inserted into your working buffer. Try typing `i`, followed by any text you want. Include some RETURNs. Press BACK SPACE to correct or erase characters

on a line (BACK SPACE will not move to a previous line). When you have inserted a few lines of text, press ESC to return to command mode.

Moving the Cursor

When you press ESC, vi goes back into command mode and the next character you type is interpreted as a command. When you press ESC just once, you do not get any feedback telling you that you have successfully changed modes. If you are not sure which mode you are in, press ESC twice and you'll hear a beep — this assures you that you are in command mode. Now you can move the cursor with the keyboard arrows and insert text somewhere else in the file. Try moving the cursor with the arrows. Pick a place in your text and insert some more text by typing `i` followed by the new text. When you are done, type ESC. You can now move the cursor again and insert more text somewhere else.

Deleting Text

There are two basic commands for deleting text: `x` and `dd`. `x` deletes the character within the cursor box. You can use `x` whenever you're in command mode and it is executed immediately. You don't need to type RETURN or ESC. Try deleting some of your text with the `x` command. Move the cursor to the character you want to delete and press `x`.

`dd` deletes the entire line of text in which the cursor is located. Move the cursor to a line you want to delete and press `dd` (no RETURN or ESC is needed).

Writing Your C Program

Now type your C program into the file *blue.c*. First, delete all the text from the screen using `dd`. When everything is gone, the cursor is in the upper left corner of the screen and the system beeps if you press `dd` or any of the keyboard arrows. Type in the following program by first typing `i` to put vi into the insert mode.

Type the following text, using a carriage return after each line:

```
#include "gl.h"

main()
{
   ginit();
   color(BLUE);
   clear();

   sleep(5);
   gexit();
}
```

Your text should look exactly like this. Make sure that you press RETURN after the final }. Use BACK SPACE to correct typing errors before you move on to the next line. If you need to change a line after you've moved on to another, use the commands you learned in the previous sections:

- Press ESC to complete your insertion and return to command mode.
- Use the keyboard arrows to move the cursor to a new position.
- Use `x` and `dd` to delete text.
- Use `i` to insert more text.

Another text-insertion command you should know is `a`, which inserts text **after** the cursor position. (Remember that `i` inserts text **before** the cursor position.)

Saving Your Edits

None of the text on the screen is saved in the file *blue.c* until you tell the system to save it. Instead, it's held in the working buffer, which is displayed on the screen. When you've typed in the correct text (and pressed ESC to complete the last insert command), you need to press the colon key (`:`).

This makes a colon appear at the bottom of the screen. To write the text in the working buffer into the file *blue.c* (i.e., to save the text as *blue.c*), type `w`, which also appears at the bottom of the screen, followed by RETURN.

This is what your screen looks like after you save your edits:

```
#include "gl.h"

main()
{
   ginit();
   color(BLUE);
   clear();

   sleep(5);
   gexit();
}
□
~
~
"blue.c" [New file] 12 lines, 80 characters
```

As soon as *blue.c* is saved, you are back in command mode. You can continue to edit *blue.c* by moving the cursor and using the insertion and deletion commands. Any time you want to save your changes, press ESC and tell vi to "write" the file by typing:

```
:w RETURN
```

Note that each time you save *blue.c* in this way, you overwrite the previous version.

Exiting vi

To exit vi, type:

```
:q RETURN
```

If you get a message that says you can't exit vi, it means you haven't saved your edits with the `:w` command. Try typing:

```
:wq RETURN
```

which saves your edits and exits vi all at once.

If you ever want to exit vi without saving the changes you have made, you can do so by typing `:q!` RETURN. Do this only if you're sure you don't want to save your changes.

Listing the Files in Your Directory

After exiting the vi text editor, you can prove your file exists by typing the "list" command, `ls`. Type:

```
ls RETURN
```

This is what appears on the screen:

```
% ls
blue.c
%
```

You may find files other than *blue.c* in the guest directory if someone else has been using it. `ls` is very useful; you can use several options with it to get more informative listings.

Summary of vi Commands

→	move cursor one character to the right
←	move cursor one character to the left
↓	move cursor down one line
↑	move cursor up one line
i*text*...ESC	insert text before the cursor
a*text*...ESC	insert text after the cursor
x	delete character
dd	delete line
:w RETURN	write to file
:q RETURN	exit vi
:q! RETURN	exit vi without saving edits

Compiling and Running Your Graphics Programs

To compile a C source file into a binary file that can be executed by the IRIS, use the `cc` command. `cc` is a key command that you should read about in the *UNIX Programmer's Manual, Vol. IA*. To compile *blue.c*, type:

```
cc blue.c -Zg RETURN
```

Wait for a few seconds until the percent sign appears again.

Many UNIX commands have "options" that provide additional functionality. An option is usually specified by a hyphen and one or more characters. In this case, `cc` takes the option "`-Zg`", which tells the compiler that there are graphics routines in the program.

After the program has been compiled, type `ls` to see that an executable file called *a.out* has been created in your directory.

```
% ls
a.out  blue.c
%
```

Every program you compile this way is called "*a.out*". Next time, you'll see how to use an option that lets you name the executable file whatever you want. If you get any error messages from the compiler, check to make sure *blue.c* contains exactly the same text as is listed here, edit it if necessary, and recompile it.

Running a Program

Now that you have compiled your program, you are ready for the easiest and most enjoyable part of the procedure — running the program. All you have to do is type:

```
a.out RETURN
```

The screen clears to blue and stays that way for five seconds. (The cursor might appear in red on the screen, but don't worry about that.) When the program is done, your textport appears on the screen and you can communicate with UNIX again. You have just completed your first graphics program. Now that you've compiled the program, you can run it any time you want.

Changing Your Program

Suppose you want to change *blue.c*. You need to edit it, compile it and run it again. This is the typical process for developing a graphics program. You write something, run it, and see what happens; then you change it and run it again until you get it the way you want it.

To edit *blue.c*, you need to go into vi again. Type:

```
vi blue.c RETURN
```

This is what you see on the screen:

```
#include "gl.h"

main()
{
   ginit();
   color(BLUE);
   clear();

   sleep(5);
   gexit();
}

~
~
~
~
"blue.c" 12 lines, 80 characters
```

Let's briefly discuss what some of this code means.

`#include "gl.h"`	Included in all graphics programs. Defines type definitions, useful constants, and external definitions for all commands.
`main()`	The name of the main program segment in all C programs. The brackets ({,}) contain the contents of `main`.
`ginit();`	Begins the graphical part of every graphics program. It initializes the hardware and software so that you can write simpler programs. Note that this command, like all the commands in this program, ends with a semi-colon.
`color(BLUE);`	Tells the IRIS to use blue for all following drawing commands (until the color is changed by another `color` command).
`clear();`	The only drawing command in this program. It paints the whole screen in the current color, which is blue.
`sleep(5);`	A UNIX routine that tells the IRIS to do something else for five seconds. This command allows you to see what you've drawn for a while before the textport pops up when the program is finished.
`gexit();`	The final command in all graphics programs.

The logical place to add commands to this program is after the `clear()` command and before the `sleep(5)` command. Move the cursor down to the blank line after `clear()` using the down arrow on the keyboard. Then type `i` followed by

```
color(RED);
rectfi(10, 10, 300, 300);
```

and ESC.

This is what the new lines mean:

`color(RED);`	Changes the drawing color so that any new drawing will be red.
`rectfi(10, 10, 300, 300);`	Draws rectangles. This is a variation of the `rect` command. The `f` in `rectfi` means the rectangle is filled, rather than outlined. The `i` means the coordinates that follow are integers, rather than floating point numbers. The two `10`s are the x and y coordinates of the lower left corner of the rectangle, and the two `300`s are the coordinates of the upper right corner.

Note that the IRIS screen goes from 0 to 1023 along the x axis (left-to-right) and 0 to 767 along the y axis (bottom-to-top).

This is what your new program should look like:

```
#include "gl.h"

main()
{
   ginit();
   color(BLUE);
   clear();
   color(RED);
   rectfi(10, 10, 300, 300)
   sleep(5);
   gexit();
}
~
~
~
~
~
```

If the program is correct, save it and exit vi by pressing ESC and typing:

```
:wq RETURN
```

If you list your files with `ls`, you'll see that there is still just one *blue.c*. The contents of the old *blue.c* were replaced by your new program when you saved it. If we had wanted to save the old *blue.c* we could have named the new file something else when we saved it; or, we could have copied the old one and renamed it before we saved the new one. Both of these tasks are discussed in the next chapter of this document.

Compile the program, but this time add an option that names the executable file "*bluec*":

```
cc blue.c -Zg -o bluec RETURN
```

The "`-o`" (minus-oh, not minus-zero) stands for output. The name that immediately follows the `-o` is the name of the executable file.

When you see the percent sign prompt, run your program by typing:

```
bluec RETURN
```

A red box appears in the lower left corner over a blue background. You have now written two successful graphics programs.

Writing a FORTRAN Program

Now you can write a FORTRAN version of the same program. This is what it looks like:

```
$               INCLUDE /usr/include/fgl.h
C
                INTEGER I
C
                CALL GINIT
                CALL COLOR(BLUE)
                CALL CLEAR
                CALL COLOR(RED)
                CALL RECTFI(10, 10, 300, 300)
                DO 10  I=0,999999
10              CONTINUE
                CALL GEXIT
                STOP
                END
```

Use vi to create the FORTRAN source file called *"blue.f"*:

```
vi blue.f RETURN
```

Use TAB s to line up the columns of routine calls.

Let's look at the differences between this version and the C version.

- Traditionally, most of the letters are upper case.
- The "INCLUDE" line has a dollar sign in the first position, and the entire pathname of the include file is specified.
- The lines that begin with "C" are comment lines and sometimes contain text that describes parts of the program. In this program they are used to distinguish the different sections of code.
- The "INTEGER I" line declares the variable "I" as an integer that we can use later in the program.
- All the FORTRAN routines are preceded by the word "CALL".
- The `sleep()` routine we used in the C version isn't available for a FORTRAN program, so we use a simple iterative loop to create a pause before the program stops.

Once you have created *blue.f*, save it, quit working on the file, then compile it with the **f77** command, which operates just like **cc**:

```
f77 blue.f -Zg -o bluef RETURN
```

Note that we call the executable file "*bluef*" to distinguish it from the C program. The compiler will print information on the screen that looks like this:

```
% f77 blue.f -Zg -o bluef

% MC68000  FORTRAN 77 Compiler  V2.4
(C) Copyright 1981,  1985  Silicon Valley Software, Inc.

0 errors.  122 lines.  File /usr/people/guest/ctm005644
% MC68000  Code Generator  V2.4
(C) Copyright 1980,  1985  Silicon Valley Software, Inc.

% MC68000  UNIX Object Code Formatter V2.4
(C) Copyright 1982,  1985  Silicon Valley Software, Inc.
%
```

If you see any error messages, check your source file for mistakes, edit it, and recompile it. To run the program, type:

```
bluef RETURN
```

The same red box on a blue background appears on the screen for a few seconds. Then the textport reappears.

```
% bluef

    Programmed STOP
%
```

You have now successfully written and executed a FORTRAN graphics program.

Logging out

When you are ready to log out, type:

```
logout RETURN
```

This is what you see:

```
IRIS login:
```

Now the system is ready for you or someone else to log in.

2. A Basic Lesson on UNIX

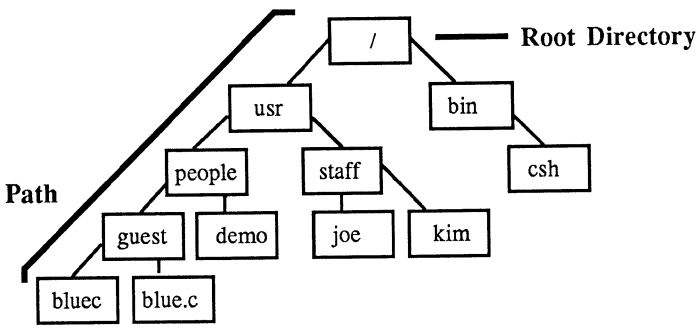
The IRIS uses the UNIX operating system; so, to use the IRIS, you need to know some of the basics about UNIX. The previous chapter, *Trying Out Your IRIS*, took you through a sample programming session; this chapter gives you a further introduction to UNIX. Specifically, it:

- describes the UNIX directory system
- explains how to manipulate directories and files
- provides a summary list of useful commands

Log in as "guest" so that you can try out commands as they are discussed in this chapter. (See page 2 if you've forgotten how to log in.)

The UNIX Directory System

To the user, UNIX appears as a hierarchy of directories and files. At the top of the hierarchy is the "root" directory, which is written as / (slash). The root directory contains various directories and files, which in turn contain other directories and files, and so on.



Pathnames

A "pathname" is the list of directories that you must go through in order to get to a particular directory or file. For example, in the diagram on the previous page, the path that you must follow to get to *bluec* is */usr/people/guest*. Therefore, *bluec*'s full pathname is */usr/people/guest/bluec*.

Current Working Directory

Whenever you use UNIX, you are "located" somewhere in the file system, in one of the directories. Your location at a given time is called your "current working directory". Rather than always referring to a file or a directory by its entire pathname, you can refer to the file in terms relative to your current working directory. For example, if your current working directory is */usr/staff/joe*, you can refer to the file */usr/staff/joe/test* as *test*, rather than using its full pathname. Whenever you type a name without a preceding "/", the IRIS assumes that it is in your current working directory. Note that if you do put a "/" at the beginning of a name, the system will look in the / (root) directory for the name that follows the slash.

To find out where you are right now, type:

```
pwd RETURN
```

pwd stands for "print working directory" and displays where you are located in the file system. Since you are logged in as "guest", this is what you see:

```
% pwd
/usr/people/guest
%
```

/usr/people/guest is the "home directory" for guest. The home directory is the one in which you are automatically placed each time you log in. Chapter 4 of the *IRIS Series 3000 Owner's Guide* and the *IRIS Workstation Guide, Series 2000* tells you how to specify a user's home directory.

Using Directories and Files

Making New Directories

As you create more and more files, you will find it convenient to organize them into subdirectories. Create a subdirectory called *"sub"* in the *guest* directory by typing:

```
mkdir sub RETURN
```

Now use the **ls** command to look for your new subdirectory. Type:

```
ls RETURN
```

Here is what you see:

```
% mkdir sub
% ls
a.out bluec bluef blue.c blue.f sub
%
```

Understanding the Contents of a Directory

At this point your *guest* directory has several files in it, and you may not remember what each different name means. If, for example, you had named the new subdirectory *"blues"*, you might not know whether it was a subdirectory or a file the next time you logged in.

Your IRIS can give you a more informative listing about the contents of your directory when you use the **-F** option with the **ls** command. Try it now by typing:

```
ls -F RETURN
```

This is what you see:

```
% ls -F
a.out* bluec* bluef* blue.c blue.f sub/
%
```

When you use the `-F` option (`F` stands for flag), `ls`:

- puts an asterisk next to all executable (or binary) files, such as *a.out*, *bluec*, and *bluef*
- puts a trailing "/" after all subdirectories, such as *sub*
- does not add any marks to text (e.g., source) files, such as *blue.c* and *blue.f*

Since this directory is fairly full, let's start fresh and put some files into the new subdirectory. Just as you were "moved" into the *guest* directory when you logged in so it would be convenient to work with the files in that directory, you can now move into the *sub* directory to make it convenient to work with new files.

Changing to a New Working Directory

`cd`, which stands for "change directory", is the command you use to change to a new working directory. To move into the *sub* directory, type:

```
cd sub RETURN
```

Note that you could also move into *sub* by typing the full pathname:

```
cd /usr/people/guest/sub RETURN
```

Use **pwd** to make sure that you know where you are:

```
% pwd
/usr/people/guest/sub
%
```

An **ls** shows that there are no files or directories in *sub*:

```
% ls
%
```

You can create files in *sub*, just as you did in *guest*, using vi. You can also create subdirectories in *sub* using **mkdir**.

Copying Files

A third way to create files in *sub* is by copying them from another directory with the **cp** command. To use **cp**, you need to include:

- the pathname of the file you want to copy
- the pathname of the new file

As usual, these pathnames can be full pathnames (from / on down) or pathnames relative to the current working directory. Let's copy one of your files from *guest* into *sub*.

First, change directories (**cd**) to the *guest* directory. Type:

cd /usr/people/guest RETURN

There is also an easier way to do it. ".." (two periods) is always equivalent to the name of the "parent" of your current working directory. The parent of your current directory is the directory which contains your current directory. For example, if you are located in */usr/people/guest*, ".." is */usr/people*. If your current directory is */usr/people*, ".." is */usr*. So, another way you could have moved from */usr/people/guest/sub* to */usr/people/guest*, would have been to type: **cd ..** RETURN.

Check where you are now by using **pwd**.

```
% pwd
/usr/people/guest
%
```

Now do an `ls -F` to see what's in *guest*.

```
% ls -F
a.out* bluec* bluef* blue.c blue.c sub/
%
```

To copy *blue.c* into *sub*, type:

```
cp blue.c sub/blue.c RETURN
```

To make sure *blue.c* was copied, move into *sub* and do an **ls**:

```
% cd sub
% ls
blue.c
%
```

You could compile *blue.c* if you wanted and create an executable file in *sub*. But instead, let's try copying *bluec* from *guest* into *sub*. To do this, use ".." to specify *guest* instead of writing out the entire pathname. Type:

```
cp ../bluec bluec RETURN
```

Now do an **ls** to see what's in *sub*.

```
% cp ../bluec bluec
% ls
bluec blue.c
%
```

Looking at the Contents of a File

If you want to see what's in a text file without entering vi, you can use the **cat** command or the **more** command. **cat**, which stands for concatenate, displays the contents of a text file on the screen. (The name comes from one of the functions of the command, which is to join files together.) If the file is long, it keeps scrolling off the top of the screen, and you have a chance to read only the final screenful. Try displaying *blue.c* in the *guest* directory by typing:

```
cat blue.c RETURN
```

This shows up on the screen:

```
#include "gl.h"

main()
{
ginit();
color(BLUE);
clear();

sleep(5);
gexit();
}
%
```

The **more** command also displays text files, but it only shows one screenful at a time. **more** displays the first screenful of the file with a message at the bottom of the screen indicating what

percentage of the file has been displayed. When you want to see more of the file, you can either press **RETURN** to see one more line, or you can press **SPACEBAR** to see the next screenful.

Try the **more** command with *blue.c*.

```
%more blue.c

#include "gl.h"

main()
{
ginit();
color(BLUE);
clear();

sleep(5);
gexit();
}
%
```

It looks the same as **cat** because the file is less than one screenful long. Later, when you need to look at a longer file, try **more** again.

Note that if you use **cat** or **more** on an executable file, strange characters will be displayed on the screen. That's because executable files are binary files, not ASCII files, and they contain some non-printable characters.

Moving (or Renaming) Files

The **mv** (move) command changes the pathname of a file. This has the effect of renaming it. You could rename a file using the commands you already know by copying the file and then removing the original. An easier way is to type **mv** followed by the name of the file you want to rename and the new name. For example, you can change *blue.c* to *red.c* by typing:

```
mv blue.c red.c RETURN
```

An **ls** shows the new name:

```
% ls
bluec  bluef  blue.f  red.c
%
```

Since you don't want to be confused about the contents of the file, change it back to *blue.c*.

```
% mv red.c blue.c
% ls
bluec  bluef  blue.c  blue.f
%
```

Removing Files

Occasionally you will need to remove (or delete) files from a directory. This is done with the **rm** command. Be careful when you remove a file because it can be recovered only from a backup tape which you have made.

Let's try removing *blue.c* from the *sub* directory. Remember you still have a copy in *guest*. First make sure you are in *sub*:

```
% pwd
/usr/people/guest/sub
%
```

Then type the following to remove *blue.c*:

```
% rm blue.c
%
```

Do an **ls**, and see that *blue.c* is gone.

```
% ls
bluec
%
```

Removing Directories

The **rmdir** command removes subdirectories from a directory. To remove *sub* from *guest*, you must first empty *sub* of all its files and subdirectories. The only thing in *sub* is *bluec*. Remove it by typing:

```
rm bluec RETURN
```

An **ls** shows that the *sub* directory is empty.

```
% rm bluec
% ls
%
```

Now move up to *guest* by typing:

```
cd .. RETURN
```

Remove *sub* by typing:

```
rmdir sub RETURN
```

To make sure *sub* is really gone, do an **ls**.

```
% ls
bluec bluef blue.c blue.f
%
```

The *man* Command

To find out more about system commands, type **man** (which stands for "manual page") followed by the name of the command. Your IRIS displays the following information:

- a brief description of what the command does;
- a "synopsis" that shows the syntax of the command;
- a longer description of the command and its flags;
- a "see also" section that lists related commands.

There may also be sections on files, diagnostics, and bugs.

Try typing:

```
man cd RETURN
```

After a few seconds, the "manual" page for the **cd** command appears on your screen. The documentation is displayed one screenful at a time. Press SPACEBAR to display the next screenful.

Summary of Important Commands

The previous sections have introduced you to the fundamental UNIX commands. This section summarizes their basic functions. You should read about these commands in the *UNIX Programmer's Manual, Vol. IA*, since they have more functionality than has been discussed here.

Most Useful Commands

login *user-name*	log in to the system
pwd	print current working directory
cd *directory*	change current working directory
ls	list contents of directory
cat *textfile*	display contents of text file
more *textfile*	display contents of text file (by screenfuls)
cp *oldfile newfile*	copy a file
rm *file*	remove or delete a file
mv *oldname newname*	move or rename a file
mkdir *directory*	make a subdirectory
rmdir *directory*	remove or delete a subdirectory
man *command-name*	display a manual page
logout	log out of the system

Other Useful Commands

This section lists commands you may want to read about in the *UNIX Programmer's Manual, Vol. I.* This should give you an idea about what commands are available to you.

Manipulating and Displaying Files and Directories

`awk`	pattern scanning and processing language
`cat`	display contents of a file
`cd`	change current working directory
`cmp`	compare two files
`comm`	select or reject lines common to two sorted files
`cp`	copy a file
`diff`	display the differences between two files
`find`	find files
`grep`	search a file for a pattern
`head`	give first few lines of a file
`ls`	list contents of a directory
`mkdir`	make a new subdirectory
`more`	display contents of a file (by screenful)
`mv`	move (rename) a file
`pr`	format a file for printing
`pwd`	print current working directory
`rm`	remove (delete) a file
`rmdir`	remove (delete) a directory
`sort`	sort and/or merge files
`spell`	find spelling errors
`split`	split a file into pieces
`uniq`	report repeated lines in a file
`wc`	count lines, words, and characters in a file
`whereis`	locate source, binary, and/or manual for a program

Communications

`mail`	send mail to users or read mail
`mesg`	permit or deny displaying of messages
`wall`	write to all users
`write`	write to another user

Programming Tools

`adb`	general-purpose debugging program
`cc`	C and FORTRAN compiler
`f77`	FORTRAN compiler
`lint`	C program checker
`make`	maintain, update, and regenerate groups of programs
`od`	octal dump
`touch`	update access and modification times of a file

System Status and Administration

`chmod`	change the access mode of a file
`chown`	change owner of a file
`chgrp`	change group of a file
`cpio`	copy file archives in and out (mostly used for tape backups)
`df`	report number of free disk blocks
`du`	summarize disk usage
`file`	determine file type
`fsck`	file consistency check and interactive repair
`kill`	terminate a process

`multi`	switch the system to multi-user mode
`ncheck`	generate file names from inode numbers
`passwd`	change login password
`ps`	report process status
`reboot`	UNIX bootstrapping procedures
`shutdown`	terminate all processing
`su`	become super-user or another user
`sync`	update the super block
`tar`	tape archiver
`umask`	set file-creation mode mask
`who`	who is on the system

Editing, Formatting, and Printing Documents

`eqn`	format mathematical text for nroff or troff
`lp`	send or cancel requests to an LP spooling system
`nroff`	format text
`sed`	stream editor
`tbl`	format tables for nroff or troff
`vi`	screen-oriented (visual) display editor

System-to-system Functions (XNS Communications)

`xcp`	copy files to or from a remote system
`xlogin`	log in to a remote system
`xx`	run a program on a remote system

System-to-system Functions (TCP/IP Communications)

`rcp`	copy files to or from a remote system
`rlogin`	log in to a remote system
`rsh`	run a program on a remote system

Miscellaneous

`at`	execute commands at a later time
`clear`	clear terminal screen
`csh`	the "c" shell, the standard interactive shell for the IRIS
`date`	print and set the date
`echo`	echo arguments
`gclear`	clear IRIS graphics screen
`help`	ask for help
`login`	log in to system
`logout`	log out of the system
`man`	display a manual page
`sleep`	suspend execution for an interval
`stty`	set options for a terminal
`tty`	display the terminal's name
`tset`	change the attribute settings on a terminal

3. Where to Find More Information

You have four good sources of more detailed information about your IRIS and UNIX. They are:

1. The IRIS documentation that you received with your IRIS
2. The UNIX documentation that you received with your IRIS
3. Silicon Graphics Geometry Hotline
4. Additional books written on UNIX and on computer graphics

IRIS Documentation

Your workstation was shipped with these three IRIS documents:

- *Getting Started with Your IRIS Workstation*
- *IRIS Series 3000 Owner's Guide* or
  IRIS Workstation Guide, Series 2000
- *IRIS User's Guide*

You are reading the first of these documents right now. The other two are described in this section.

Owner's Guide or *Workstation Guide*

These documents contain information about system installation and administration. Use them to find out how to:

- Install and boot your workstation.
- Install optional peripherals, ASCII terminals, and non-standard video monitors.
- Perform administrative tasks, such as making accounts for new users, backing up your system, and reconfiguring your disk.

IRIS User's Guide

The *IRIS User's Guide* describes the Graphics Library, the set of graphics routines developed for the IRIS. The graphics routines are presented in two forms:

- The *Programming Guide* is a narrative description of the routines, divided into chapters according to subject matter: objects, coordinate transformations, and curves and surfaces, for example.
- The *Reference Manual* is a collection of alphabetized manual pages, similar in structure to the UNIX manual pages.

The *IRIS User's Guide* contains two other useful chapters:

- *The IRIS Window Manager* describes *mex*, Silicon Graphics' Multiple Exposure window manager.
- *Programming Examples* provides useful, commented examples of IRIS graphics programming.

UNIX Documentation

UNIX Programmer's Manual

This manual contains two volumes of two books each.

- *Volume I* contains *manual pages*, hard copies of what you see on your screen when you issue the **man** command.
- *Volume II* contains papers that describe how to use the UNIX system. These papers cover the following topics:
  - General Works
  - Editors
  - Document Preparation
  - Programming
  - Support Tools

If you're not familiar with the UNIX system, you'll probably want to browse through these four books rather than try to read them cover to cover. You'll soon discover which parts are worth spending more time on.

Silicon Graphics Geometry Hotline

Silicon Graphics provides a comprehensive support and maintenance program for the IRIS 2000 and 3000 series products. For further information, contact Customer Service through the Geometry Hotline.

Silicon Graphics Geometry Hotline	
(800) 252-0222	U.S. except California (toll-free)
(800) 345-0222	California (toll-free)
(415) 962-0606	Worldwide (collect)

Additional Reading

The following books may help you with UNIX and computer graphics.

UNIX:

- *A Practical Guide to the UNIX Operating System*, Mark G. Sobell, The Benjamin/Cummings Publishing Company, Inc., 1984
- *The UNIX System*, S.R. Bourne, Addison-Wesley Publishing Company, 1983
- *The UNIX Programming Environment*, Brian W. Kernighan and Rob Pike, Prentice-Hall, Inc., 1984

Graphics:

- *Principles of Interactive Graphics*, William M. Newman and Robert F. Sproull, McGraw-Hill Book Company, 1979
- *Fundamentals of Interactive Graphics*, James D. Foley and Andries Van Dam, Addison-Wesley Publishing Company, 1983