

# GETTING STARTED WITH UNIX

## ROADMAP:

- . The Unix Shell: Working With A Command Line Interface
- . Standard Output
- . Command-Line Arguments
- . Redirecting Standard Output To A File
- . Displaying Files: the cat command
- . Working With Files In The Unix Shell
- . Writing C Programs On Unix Using pico And gcc
- . Redirecting Standard Input From A File

## TRANSFIGURATION 101:

Initially learning to work with Unix is like learning **magic** at **Hogwarts**-- just a matter of knowing which incantation:

- . wingardia leviosum
- . petrificus totalus
- . stupefy
- . sectum sempra  
etc.

The good news is:

- . you don't need a wand
- . the words are usually much shorter and don't sound like latin
- . you can be a squib or a muggle and it still will work
- . if you do it wrong usually nothing happens (e.g. you won't be puking slugs)-- you just get an error message

Like magic though, the only way to learn it is to **DO IT**. So as you read the material that follows, keep your keyboard and screen close by and try out everything as you go along. There are special practice exercises listed, but that is a bare minimum-- you should try more!

## THE UNIX SHELL: WORKING WITH A COMMAND-LINE-INTERFACE

The Unix shell is a COMMAND-LINE-INTERFACE (CLI):

- . The shell gives you a **prompt**
- . You type a **command** and hit RETURN
- . The command does something
- . The shell give you a prompt (and the cycle repeats)

### EXAMPLE:

```
atrium46:arnow>
```

That's my prompt: "atrium46:arnow>".

Now I'll type in a command:

```
atrium46:arnow> date
```

I typed in the "date" command. Now I'll hit the RETURN KEY:

```
atrium46:arnow> date
Mon Sep  5 11:26:34 EDT 2005
atrium46:arnow>
```

With blinding speed, the date is displayed, along with a new prompt.

I'll try another command, "who" to see who is logged in on this computer.

```
atrium46:arnow> date
Mon Sep  5 11:26:34 EDT 2005
atrium46:arnow> who
arnow pts/1 Sep  5 11:25 (ny325.east.verizon.net)
atrium46:arnow>
```

Just me! How boring.

### EDITING YOUR COMMAND LINES:

- . DO NOT USE ARROW KEYS!
- . TO ERASE THE LAST CHARACTER YOU TYPED: either BACKSPACE or DELETE (which-ever works-- varies from machine to machine)
- . TO ERASE THE ENTIRE LINE: control-U (hold the control key down and type U)
- . ONCE YOU HIT RETURN, YOU CAN'T EDIT YOUR LINE ANY MORE.

### PRACTICE:

- . Check the date on your computer
- . See who's logged in on your computer
- . Type in garbage (like "asdf") and hit RETURN
- . Type in garbage, but don't hit return-- hit "control-U".

## STANDARD OUTPUT:

In Unix, most commands are **programs** and most programs are commands. Programs typically produce some information (**output**). Every program in Unix has at least one output pathway: **standard output**. Standard output by default goes to the **screen** (where you see it).

### EXAMPLE:

```
atrium46:arnow> whoami
arnow
```

The **standard output** of the `whoami` command is your username. Mine is "arnow". So here I typed in "whoami": that started the `whoami` program, and its output ("arnow") went to the screen-- you can see it above.

### EXAMPLE:

```
atrium46:arnow> w
12:46pm up 3 day(s), 21:08, 1 user, load average: 0.01, 0.02, 0.02
User      tty          login@    idle   JCPU   PCPU   what
arnow    pts/2        11:45am             w
atrium46:arnow>
```

The **standard output** of the `w` command consists of several lines, indicating current time, how long the system has been up, who's logged in and what they're doing.

### PRACTICE:

- . Run a command that displays the date to standard output.
- . Run a command that displays a list of who's logged on your computer to standard output.

## COMMAND-LINE ARGUMENTS:

Every command-line so far has been a single "word": "date", "who", "whoami". Most commands accept additional words of information on the command-line itself. These "additional words information" are called **arguments**.

### EXAMPLE:

```
atrium46:arnow> date
Mon Sep  5 12:24:13 EDT 2005
atrium46:arnow> date -u
Mon Sep  5 16:24:15 GMT 2005
atrium46:arnow>
```

When I typed "date" by itself I got the date and time in my timezone. When I typed "date -u" I got the date and time in "universal time" (GMT). The "-u" is called an **argument** to the date command.

### EXAMPLE:

```
atrium46:arnow> echo hey
hey
atrium46:arnow> echo hey you
hey you
atrium46:arnow>
```

The echo command just displays (writes) its **arguments** to **standard output**.

### PRACTICE:

- Run a command that displays the words "hello world" to standard output.
- Run the echo command with no arguments: what happens?

## REDIRECTING STANDARD OUTPUT:

Sometimes you want to save the output of a program (a command) into a file. To do this, you can **redirect standard output to a file**.

Here's how you do this:

- . type the command (but don't hit RETURN!)
- . type the greater-than symbol: **>**
- . type the name of the file you want the output to go into

### EXAMPLE:

```
atrium46:arnow> date
Mon Sep  5 12:24:13 EDT 2005
atrium46:arnow> date > rightNow
atrium46:arnow>
```

When I typed "date" by itself, standard output was not redirected, so I can see the date and time on my screen.

When I typed "date" with ">**rightNow**", standard output was redirected to a file called "rightNow":

- . If the file did not exist, the system created it automatically
- . If the file did exist, it's old contents were ERASED
- . No output appears on the screen-- it all went into the file **rightNow**

### EXAMPLE:

```
atrium46:arnow> date > aBitLater
atrium46:arnow>
```

Again I redirected the standard output, this time to another file of my choosing. No data, no output appeared on the screen: it all went to the file **aBitLater**.

### PRACTICE:

- . Run a command that puts the current date and time into a file called "currentTime".
- . Run a command that puts the words "hello world" into a file called "greeting".

## DISPLAYING FILES: the cat command

What's the point of redirecting standard output to a file if you can't see the contents later?

You can display a file's contents using the **cat** command.

### EXAMPLE:

```
atrium46:arnow> cat rightNow
Mon Sep  5 12:24:18 EDT 2005
atrium46:arnow> cat aBitLater
Mon Sep  5 12:24:27 EDT 2005
atrium46:arnow>
```

When you give the **cat** command an argument, it expects that the argument is a filename and it displays the file.

If you give the **cat** command two arguments, it treats each as a name of a file and displays one file after another. If you mis-type the name of a file, **cat** complains.

### EXAMPLE:

```
atrium46:arnow> cat rightNow aBitLater distantFuture rightNow
Mon Sep  5 12:24:18 EDT 2005
Mon Sep  5 12:24:27 EDT 2005
cat: cannot open distantFuture
Mon Sep  5 12:24:18 EDT 2005
atrium46:arnow>
```

### PRACTICE:

- . Redirect the standard output of "who" to "who.out"
- . Redirect the standard output of "date" to "date.out"
- . Use **echo** to create a file named "last" containing the words "that is all"
- . Use the **cat** command to display the contents of **who.out**, **date.out** and **last**.
- . Redirect the standard output of "date" to "date1"
- . Redirect the standard output of "date" to "date2"
- . Use the **cat** command to display both **date1** and **date2** together (using just one **cat** command)
- . Now run the same **cat** command and redirect the standard output to a file called "twoDates".
- . Display the content of **twoDates** using the **cat** command.

## WORKING WITH FILES IN THE UNIX SHELL:

In the Unix shell, files are examined, destroyed, renamed and copied using these commands:

- . `ls` (that's "ell ess"): **list** the names of your files
- . `rm` remove one or more files (silent: produces no output)
- . `cp` copy a file (silent: produces no output)
- . `mv` rename a file (silent: produces no output)

### EXAMPLE:

```
atrium46:arnow> ls
atrium46:arnow>
```

This is what you get when you have no files: nothing!

But now I'll create a few files by redirecting standard output:

```
atrium46:arnow> date >datefile
atrium46:arnow> who >whofile
atrium46:arnow>
```

So I've created `datefile` and `whofile`. Now watch when I run `ls`:

```
atrium46:arnow> ls
datefile whofile
atrium46:arnow>
```

And there they are (see above). Now I'll create another file, `remark`:

```
atrium46:arnow> echo now I have two files >remark
atrium46:arnow> ls
datefile remark whofile
atrium46:arnow>
```

The `cp` command requires two arguments: the original file and the name of the copy. I'll use it now to make two copies of my "`datefile`". Then I'll list my file names:

```
atrium46:arnow> cp datefile datefile2
atrium46:arnow> cp datefile datefile3
atrium46:arnow> ls
datefile datefile2 datefile3 remark whofile
atrium46:arnow>
```

The `rm` command requires one or more arguments: each argument names a file to be removed. I'll use it now remove "`datefile2`". Then I'll list my file names:

```
atrium46:arnow> rm datefile2
atrium46:arnow> ls
datefile datefile3 remark whofile
atrium46:arnow>
```

The **mv** command renames a file and requires two arguments: the original filename and the new name for the file. I'll use it now to rename "datefile3":

```
atrium46:arnow> ls
datefile  datefile3  remark      whofile
atrium46:arnow> mv datefile3 datefile.copy
atrium46:arnow> ls
datefile  datefile.copy  remark      whofile
atrium46:arnow>
```

### **PRACTICE:**

- List the files you already have, using the **ls** command.
- Create six new files, calling them 111 222 333 444 555 666 by redirecting the output of **who** six times.
- List your files now.
- Copy each of your odd-numbered files (111 333 555) to (respectively) 111.backup 333.backup 555.backup by using the **cp** command three times.
- List your files.
- Rename your even-numbered files (222 444 666) to aaa bbb ccc by using the **mv** command three times.
- List your files.
- Using the **rm** command, remove the original odd-numbered files, and list what you have.
- Can you think of another way of making a copy besides using the **cp** command?
- List your files with the **ls** command but this time add a "-l" (dash ell) argument. What is all that stuff that gets displayed?
- Run "**ls -l**" and redirect the standard output to a file called "**myfiles**". Display the file with the **cat** command. Then use the same command but stick in a "-n" as the first argument. What happens? What does "-n" do in the **cat** command?



## COMMAND SUMMARY:

### *list files:*

```
ls  
ls -l
```

### *copy file:*

```
cp oldfile newfile
```

### *rename a file:*

```
mv oldname newname
```

### *display a file:*

```
cat filename  
cat -n filename
```

### *compile a file:*

```
gcc filename  
                    (must end in ".c"!)
```

### *remove a file:*

```
rm filename  
            (no takebacks!)
```

### *redirect output of a program to a file:*

```
command > outputfile
```

```
command arguments > outputfile
```

### *logout:*

```
exit
```

## WRITING C PROGRAMS ON UNIX USING pico AND gcc:

```
atrium46:arnow> ls -l
```

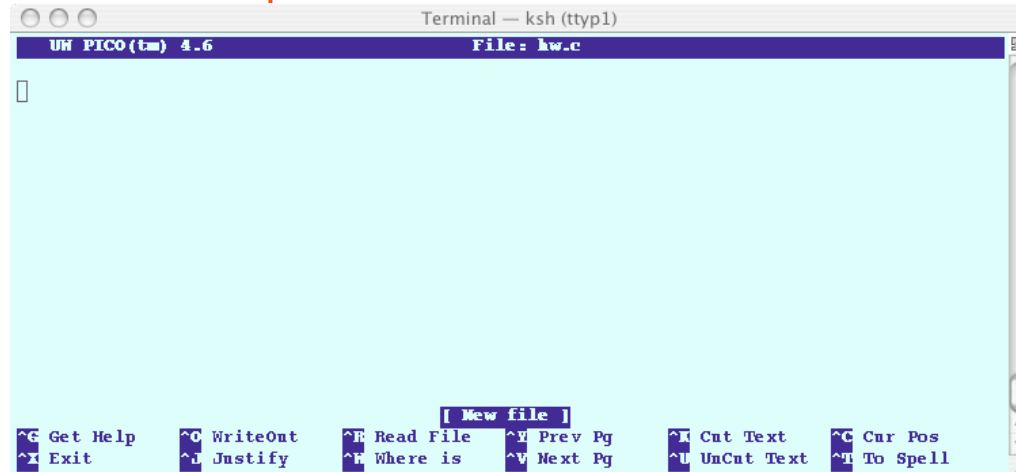
```
total 0
```

```
atrium46:arnow> pico hw.c
```

Sorry, I don't know anything about your "xterm-color" terminal.

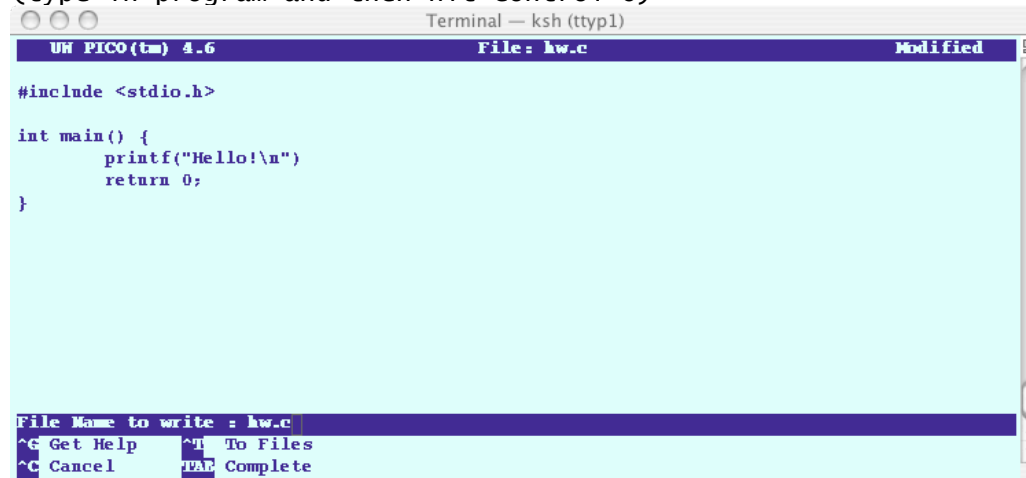
```
atrium46:arnow> export TERM=vt100
```

```
atrium46:arnow> pico hw.c
```



The screenshot shows the pico editor interface. The title bar reads "Terminal - ksh (tty1)". The editor window title is "UH PICO (tm) 4.6" and the file name is "File: hw.c". The editor area is empty, with a cursor at the top left. The status bar at the bottom contains the text "[ New file ]" and a list of keyboard shortcuts: ^G Get Help, ^O WriteOut, ^R Read File, ^Y Prev Pg, ^K Cut Text, ^C Cur Pos, ^X Exit, ^J Justify, ^W Where is, ^V Next Pg, ^U UnCut Text, and ^T To Spell.

(type in program and then hit control-0)



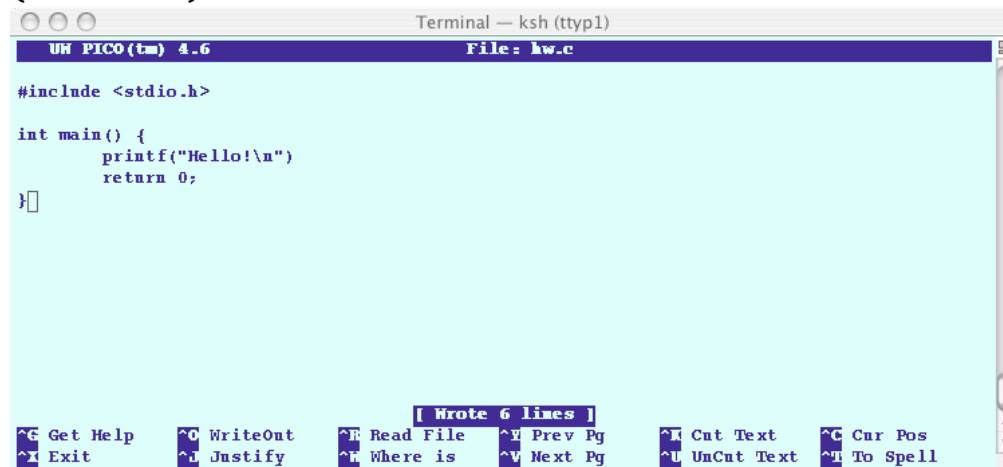
The screenshot shows the pico editor with the C program code being typed. The title bar and window title are the same as in the previous screenshot. The editor area contains the following code:

```
#include <stdio.h>

int main() {
    printf("Hello!\n");
    return 0;
}
```

The status bar at the bottom shows "File Name to write = hw.c" and "Modified". The keyboard shortcuts are the same as in the previous screenshot.

(hit return)



The screenshot shows the pico editor with the C program code saved. The title bar and window title are the same. The editor area contains the same code as in the previous screenshot. The status bar at the bottom shows "[ Wrote 6 lines ]" and the same keyboard shortcuts as in the previous screenshot.

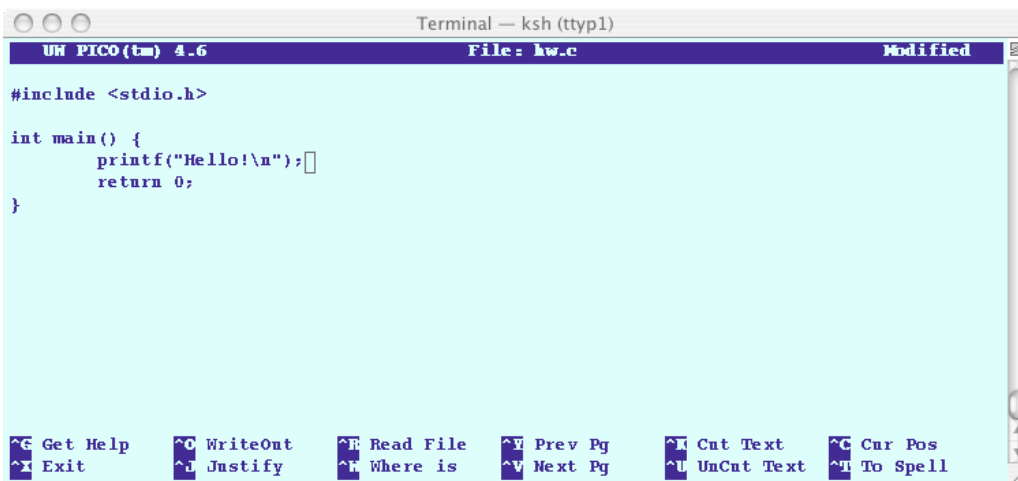
(type control-X to exit)

```

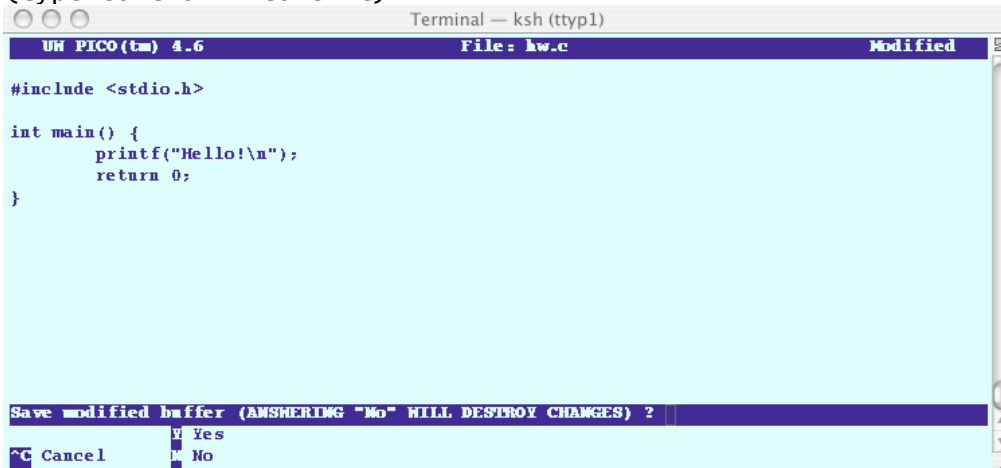
atrium46:arnow> ls -l
total 2
-rw----- 1 arnow faculty 66 Feb 2 07:53 hw.c
atrium46:arnow> cat hw.c
#include <stdio.h>

int main() {
    printf("Hello!\n")
    return 0;
}
atrium46:arnow> gcc hw.c
hw.c: In function `main':
hw.c:5: error: parse error before "return"
atrium46:arnow> cat -n hw.c
 1 #include <stdio.h>
 2
 3 int main() {
 4     printf("Hello!\n")
 5     return 0;
 6 }
atrium46:arnow> pico hw.c

```



(type control-X to exit)



(type Y to save changes and hit return)

```

atrium46:arnow> cat -n hw.c
 1 #include <stdio.h>
 2

```

```

3 int main() {
4     printf("Hello!\n");
5     return 0;
6 }
atrium46:arnow> gcc hw.c
atrium46:arnow> ls -l
total 16
-rwx--x--x  1 arnow    faculty    6476 Feb  2 08:05 a.out
-rw-----  1 arnow    faculty    67 Feb  2 08:04 hw.c
atrium46:arnow> a.out
Hello!
atrium46:arnow> mv a.out hw
atrium46:arnow> ls -l
total 16
-rwx--x--x  1 arnow    faculty    6476 Feb  2 08:05 hw
-rw-----  1 arnow    faculty    67 Feb  2 08:04 hw.c
atrium46:arnow> hw
Hello!
atrium46:arnow> hw >output
atrium46:arnow> ls -l
total 18
-rwx--x--x  1 arnow    faculty    6476 Feb  2 08:05 hw
-rw-----  1 arnow    faculty    67 Feb  2 08:04 hw.c
-rw-----  1 arnow    faculty    7 Feb  2 08:08 output
atrium46:arnow> cat output
Hello!
atrium46:arnow> cat hw.c | mail -s "CIS 1.5 HW" arnow@acm.org
/users1/arnow/mbox/outbox: No such file or directory
atrium46:arnow> cp hw.c hw2.c
atrium46:arnow> ls -l
total 20
-rwx--x--x  1 arnow    faculty    6476 Feb  2 08:05 hw
-rw-----  1 arnow    faculty    67 Feb  2 08:04 hw.c
-rw-----  1 arnow    faculty    67 Feb  2 08:12 hw2.c
-rw-----  1 arnow    faculty    7 Feb  2 08:08 output
atrium46:arnow>

```

### PRACTICE:

- Use **pico** to write C source file called **bye.c** -- the program should just print the message "bye bye!". Save the file and exit **pico** and list your files with **ls** to make sure that **bye.c** is there. Then compile **bye.c** using the **gcc** command. If there are any errors, go back into **pico** and fix them. Use **ls** again to make sure you have an **a.out** file. Then rename the **a.out** file to "**bye**" and execute the program. Execute it again and redirect the standard output to a file called "**bye.out**".
- Follow the same steps to create a C source file called **tenfold.c** -- (use **pico** to create the file, **ls** to see that it's there, **gcc** to compile it, etc.). This program (**tenfold**) should read in an integer using **scanf** and print out its value multiplied by ten. NOTE: when you run the program it will be waiting for you to type in a number-- don't make it wait too long!

## NEW COMMAND SUMMARY:

*compile a file:*

```
gcc filename  
          (must end in ".c"!)
```

*edit a file or create a file with the editor:*

```
pico filename
```

*mail a file to someone:*

```
cat filename | mail -s "whatever subject" someone@somewhere.xxx
```

## REDIRECTING STANDARD INPUT:

Programs typically require some information from the outside (**input**). Every program in Unix has at least one input pathway: **standard input**. Standard input by default comes from the **keyboard** (you type it).

### EXAMPLE:

```
atrium46:arnow> ls -l
total 4
-rw----- 1 arnow  faculty  115 Sep  5 15:48 add.c
-rw----- 1 arnow  faculty   6 Sep  5 15:52 data1
atrium46:arnow> cat -n add.c
 1 #include <stdio.h>
 2
 3 int main() {
 4     int    x, y;
 5     scanf("%d", &x);
 6     scanf("%d", &y);
 7     printf("%d\n", x+y);
 8     return 0;
 9 }
atrium46:arnow> gcc add.c
atrium46:arnow> mv a.out add
atrium46:arnow> ls -l
total 18
-rwx--x--x 1 arnow  faculty  6620 Sep  5 15:53 add
-rw----- 1 arnow  faculty  115 Sep  5 15:48 add.c
-rw----- 1 arnow  faculty   6 Sep  5 15:52 data1
atrium46:arnow> add
34
12
46
atrium46:arnow>
```

Here, I typed "add" and the program waited first for me to type in an integer for the scanf in line 5 (**34**) and then for me to type in an integer for the scanf in line 6 (**12**). Once I did so, it gave me the sum (**46**) as output.

But look: I have a file, data1, which I created using pico. And that file contains 34 and 12:

```
atrium46:arnow> cat data1
34
12
atrium46:arnow>
```

I want to use this file's data for my add program by REDIRECTING STANDARD INPUT:

```
atrium46:arnow> add < data1
46
atrium46:arnow>
```

See? I didn't type anything for the program-- it got its input from **data1**.