shell part II: variables

1. Overview

A Unix shell is a program people use to run programs. Most people use it as a 'command interpreter.' The shell prints a prompt, the user types the name of a program, the shell locates, loads, and runs the program, the shell waits for the program to exit.

The shell does three other major things. (1) It has string variables, (2) it does input/output redirection and piping, (3) it has control flow. In this discussion, we shall look at the shell's system of variables.

There are several shells available for Unix. We shall study and emulate the Bourne Shell. To run this shell, type **sh** . To exit this shell, press Ctrl-D.

2. Assigning Values to String Variables

The shell is a programming language. Like most programming languages, it has variables and an assignment operator. The variables in the shell can store only strings. For example:

```
$ NAME=Jane
$ Day=Monday
$ LOCATION="Cambridge, MA"
```

are three assignment statements. The format of an assignment statement is

```
variablename=string_of_chars
```

There are *no spaces* around the = operator. The variable name may be any combination of letters, digits, and the underscore character. The first character may not be a digit. Case matters. The string of characters may be any string of characters. If the string includes any spaces or special characters (such as * ? # [ ]) the string must be enclosed in quotation marks.

3. Retrieving Values Stored in String Variables

Retrieve the string stored in a variable with the **$** operator. For example:

```
$ echo Hello, $NAME, welcome to $LOCATION
$ vi $Day.c
```

The shell replaces **$varname** with the string stored in that variable. Once these replacements are done, the shell continues with its processing of the command line.

List all shell variables and their values with the **set** command.

```
$ set
Day=Monday
EDITOR=/bin/vi
HOME=/usr/bem
IFS=

LOCATION=Cambridge, MA
NAME=Jane
PATH=.:/bin:/usr/bin:/usr/local
PS1=$
PS2=>
```

4. Passing String Variables to Other Programs

By default, shell variables are local to the shell. Other programs may not access or modify these variables. If you like, you can tell the shell to pass copies of these variable to programs the shell calls. These 'global' variables are made part of the 'environment' of the shell. The following example shows the shell passing a variable to a program called 'printNAME':

```
$ NAME=Jane
$ export NAME
$ echo $NAME
Jane
$ printNAME
The variable called NAME contains Jane
```

The program printNAME is simply:

```
main()
{
    char *getenv();
    printf("The variable called NAME contains %s\n", getenv("NAME"));
}
```

The shell command **export** marks a shell variable as one that must be passed to programs the shell calls. Such a variable is said to be 'marked for export.'

The library function `getenv(char *varname)` looks up the variable 'varname' in the environment. If there is such a variable, getenv() returns a pointer to its value. If there is no such variable in the environment, getenv() returns the NULL pointer.

5. How Do Local Variables Work?

We have examined two aspects of shell variables. The first thing is the storage and retrieval of strings within the shell. This filing system may be implemented in several ways. The system has to keep a list of variables, each with a string name and a string value. A linked list, a hash table, a simple array, a binary tree are all possible structures to hold these variables. A simple pair of functions `install(varname,varval)` and `lookup(varname)` and a data structure will do the trick.

6. How do Environment Variables Work?

The second thing is the passing of some of these variables to other programs. A mechanism for this action is not so obvious. This tranfer involves a global variable called ‘environ’ which execvp() uses to transfer strings.

The `extern char **environ` variable is an array of pointers to strings of the form `var-name=value` . This array of pointers is NULL-terminated. A process may examine this list with a simple loop like

```
extern char **environ;

main()
{
     int i;
     for( i = 0 ; environ[i] ; i++ )
          printf("%s\n", environ[i] );
}
```

This little program shows how a process reads its environment, but does not show how it can modify it.

To modify the environment, the program builds a new array of environment strings and sets the global variable ‘environ’ to point to that new array. The execvp() call does the rest. If the program wants to add new strings to the environment, it must allocate a new array of pointers and fill it with the strings it wants to pass to other programs.

7. Yes, But How is this Done?

A shell inherits a set of variables from its caller. These variables are stored in the environment. The shell copies these strings into its table of local variables and marks them as from the environment.

When a user defines a new variable, the shell adds that variable to the table of variables. If the user marks a variable for export, the shell must record that mark with the variable.

Before the shell exec’s a new program, it must create a new environment array and set ‘environ’ to point to that array.

That is all there is to it.