|  |  |  |
| --- | --- | --- |
| Bourne shell  #!/bin/sh  if [ $days -gt 365 ]  then  echo This is over a year.  fi |  | C shell  #!/bin/csh  if ( $days > 365 ) then  echo This is over a year.  endif |

## Bash shell là tiến hóa từ Bourne shell

## Shell Variables and Environment Variables

Every UNIX process runs in a specific environment. An environment consists of a table of *environment variables*, each with an assigned value. When you log in certain *login files* are executed. They initialize the table holding the environment variables for the process. When this file passes the process to the shell, the table becomes accessible to the shell. When a (parent) process starts up a child process, the child process is given a copy of the parent process' table. Environment variable names are generally given in upper case by convention.

The shell also maintains a set of internal variables known as *shell variables*. These variables cause the shell to work in a particular way. Shell variables are local to the shell in which they are defined; they are not available to the parent or child shells. By convention, shell variable names are generally given in lower case in the C shell family and upper case in the Bourne shell family.

### C Shell Family

The C shell family explicitly distinguishes between *shell variables* and *environment variables*.

#### Shell Variables

A shell variable is defined by the set command and deleted by the unset command. The main purpose of your .cshrc file (discussed later in this chapter) is to define such variables for each process. To define a new variable or change the value of one that is already defined, enter:

% set name=value

where name is the variable name, and value is a character string that is the value of the variable. If value is a list of text strings, use parentheses around the list when defining the variable, e.g.,

% set name=(value1 value2 value3)

The set command issued without arguments will display all your shell variables. You cannot check the value of a particular variable by using set name, omitting =value in the command; this will effectively assign an empty value to the variable.

To delete, or unset, a shell variable, enter:

% unset name

To use a shell variable in a command, preface it with a dollar sign ($), for example $name. This tells the command interpreter that you want the variable's value, not its name, to be used. You can also use ${name}, which avoids confusion when concatenated with text.

To see the value of a single variable, use the echo command:

% echo $name

If the value is a list, to see the value of the nth string in the list enter:

% echo $name*[n]*

The square brackets are required, and there is no space between the name and the opening bracket.

To prepend or append a value to an existing shell variable, use the following syntax:

% set name=prepend\_value${name}

or

% set name=${name}append\_value

Note that when a shell is started up, four important shell variables are automatically initialized to contain the same values as the corresponding environment variables. These are *user*, *term*, *home* and *path*. If any of these are changed, the corresponding environment variables will also be changed.

#### Environment Variables

*Environment variables* are set by the setenv command, and displayed by the printenv or env commands, or by the echo command as individual shell variables are. Some environment variables are set by default (e.g., *HOME*, *PATH*).

The formats of the commands are (note the difference between set and setenv):

% setenv [NAME value]

% unsetenv NAME

where value is interpreted as a character string. If the string includes blanks (i.e. if it encompasses multiple values), enclose the string in double quotes ("), e.g.,

% setenv NAME "value1 value2 ..."

The current environment variable settings can be displayed using the setenv command with no arguments.

To use an environment variable in a command, preface it with a dollar sign ($), for example $NAME. This tells the command interpreter that you want the variable's value, not its name, to be used. You can also use ${NAME}, which avoids confusion when concatenated with text.

To prepend or append a value to an existing environment variable, use the following syntax:

% setenv NAME "prepend\_value${NAME}"

or

% setenv NAME "${NAME}append\_value"

If the pre- or appended value is the value of a preexisting environment variable, enclose the variable name in braces, too, e.g.,

% setenv NAME "${NAME*}${XYZ\_VAR}"*

Appending and prepending is commonly used with the *PATH* variable, and a colon is used as a separator, e.g.,

% setenv PATH "${PATH}:${XYZ\_DIR}"

### Bourne Shell Family

**You may create or set a shell variable by typing a name followed immediately by an equal sign (=)**

Environment variables

When you are running in a bash shell, many things constitute your environment, such as the form of your prompt, your home directory, your working directory, the name of your shell, files that you have opened, functions that you have defined, and so on. Your environment includes many variables that may have been set by bash or by you. The bash shell also allows you to have shell variables, which you may export to your environment for use by other processes running in the shell or by other shells that you may spawn from the current shell.

Both environment variables and shell variables have a name. You reference the value of a variable by prefixing its name with '$'. Some of the common bash environment variables that you will encounter are shown in Table 2.

Table 2. Some common bash environment variables

Name Function

USER The name of the logged-in user

UID The numeric user id of the logged-in user

HOME The user's home directory

PWD The current working directory

SHELL The name of the shell

$ The process id (or PIDof the running bash shell (or other) process

PPID The process id of the process that started this process (that is, the id of the parent process)

? The exit code of the last command

Listing 7. Environment and shell variables

[ian@atticf20 ~]$ echo $USER $UID

ian 1000

[ian@atticf20 ~]$ echo $SHELL $HOME $PWD

/bin/bash /home/ian /home/ian

[ian@atticf20 ~]$ (exit 0);echo $?;(exit 4);echo $?

0

4

[ian@atticf20 ~]$ echo $$ $PPID

3175 2457

When a shell starts up, it reads the information in the table of environment variables, defines itself a shell variable for each one, using the same name (also uppercase by convention), and copies the values. From that point on, the shell only refers to its shell variables. If a change is made to a shell variable, it must be explicitly "exported" to the corresponding environment variable in order for any forked subprocesses to see the change. Recall that shell variables are local to the shell in which they were defined.

Shell variables are defined by assignment statements and are unset by the unset command. The format of the assignment statement is:

$ NAME=value; export NAME

Cái này NAME=value là set

where there are no spaces around the equal sign (=). In most cases you will want to include the optional export portion of the command (following the semicolon), but this is optional. The unset command format is:

$ unset NAME

where NAME is the variable name, and value is a character string that is the value of the variable. If the string includes blanks (i.e. if it encompasses multiple values), enclose the string in double quotes, e.g.,

$ NAME="value1 value2 ..."

The values of all the current variables may be displayed with the set command.

To use a variable in a command, preface it with a dollar sign ($). This tells the command interpreter that you want the variable's value, not its name, to be used. For example, to see the value of a single variable, enter:

$ echo $NAME

You can also use ${NAME}, which avoids confusion when concatenated with text.

To prepend or append a value to an existing environment variable, use the following syntax:

$ NAME=prepend\_value$NAME

or

$ NAME=$NAMEappend\_value

Appending and prepending is commonly used with the *PATH* variable, and a colon is used as a separator, e.g.,

$ PATH=${PATH}:${XYZ\_DIR}