Unix Environment Variables

Learn about environment variables, a powerful way to configure applications and processes in Unix-like operating systems, and how to manage them.

You will need

- A Unix CLI

Recommended reading

- Unix Processes

Table of contents

What is an environment variable?

An environment variable is a **named value that can affect the way running processes will behave** on a computer.

When a process runs on a Unix system, it may query variables such as:

- `HOME` - The home directory of the user running the process.
- `LANG` - The default locale.
- `TMP` - The directory in which to store temporary files.
- And more.

Another common example is the `PATH`, an environment variable that indicates in which directories to look for binaries to execute when typing commands in a shell.

What are they for?

Environment variables can **affect the behavior of programs without modifying them**.

If a program bases some of its behavior on an environment variable, you can simply change the value of the variable before running it, allowing you to customize it without changing one line of code.

Environment variables can be used as a dynamic means of configuration, an alternative to configuration files or hardcoded values.

Managing environment variables

Getting them, listing them, setting them, deleting them.

Getting an environment variable

To display the current value of an environment variable, use the `echo` command. A variable can be referenced by its name prefixed with a dollar sign (`$`):

```
$> echo $USER
ubuntu

$> echo $HOME
/home/ubuntu

$> echo $SHELL
/bin/bash

$> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

If a variable is not set, nothing will be displayed:

```
$> echo $FOO
```

Listing all environment variables

The `env` command prints all environment variables currently set in your shell, and their values:

```
$> env
LC_ALL=en_US.utf-8
LS_COLORS=rs=0:di=01;34:...
LANG=C.UTF-8
USER=ubuntu
PWD=/home/ubuntu
HOME=/home/ubuntu
```

```
LC_CTYPE=UTF-8
SSH_TTY=/dev/pts/0
MAIL=/var/mail/ubuntu
TERM=xterm-256color
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Setting an environment variable

There are multiple ways to set an environment variable. The lifetime of the variable depends on how you set it:

- You can set it for one command.
- You can set it for the current shell session.
- You can set it in your shell configuration file (e.g. `.bashrc`).

In order to test these techniques, download this simple script which prints the value of an environment variable if set:

```
$> curl -sL https://git.io/fpdar > print-env-var.sh

$> chmod 755 print-env-var.sh

$> ./print-env-var.sh PATH
The value of $PATH is /usr/local/sbin:/usr/local/bin:...

$> ./print-env-var.sh FOO
$FOO is not set
```

Setting a variable for one command

You can prefix a command by an environment variable assigment:

```
$> `FOO=bar` ./print-env-var.sh FOO
The value of $FOO is bar
```

This only sets the variable **for the process executed by that command**. As you can see, the variable is still not set if we check later, even in the same shell session:

```
$> ./print-env-var.sh FOO
$FOO is not set
```

Setting a variable for a shell session

The `export` command exports an environment variable to all the child processes running in the current shell session:

```
$> `export FOO=bar`

$> ./print-env-var.sh FOO
The value of $FOO is bar

$> ./print-env-var.sh FOO
The value of $FOO is bar
```

As you can see, the variable remains set.

However, if you close the shell and reopen a new one, the variable is no longer set:

```
$> ./print-env-var.sh FOO
$FOO is not set
```

Setting a variable in the shell configuration file

If you add the `export` command to your shell configuration file (`.bash_profile` for Bash), it will be run every time you start a new shell:

```
$> `echo 'export FOO=bar' >> ~/.bash_profile`

$> cat ~/.bash_profile
export FOO=bar
```

This will not immediately take effect in the current shell, as the configuration file is only evaluated when the shell starts. But you can evaluate it with the `source` command:

```
$> source ~/.bash_profile

$> ./print-env-var.sh FOO
The value of $FOO is bar
```

The variable will still be set if you close this shell and launch another one:

```
$> ./print-env-var.sh FOO
The value of $FOO is bar
```

Removing an environment variable

The `unset` command removes a variable from the environment:

```
$> ./print-env-var.sh FOO
The value of $FOO is bar

$> unset FOO

$> ./print-env-var.sh FOO
$FOO is not set
```

> 💎 **Tip**

> Of course, if the variable is exported in your shell configuration file, this will only remove it for the current shell session. You must remove the export from the configuration file to remove the variable from future shells.

Getting environment variables from code

Every programming language has a simple way of retrieving the value of environment variables:

Language	Code
C	`getenv("PATH")`
Elixir	`System.get_env("FOO")`
Erlang	`os.get_env("FOO")`
Go	`os.Getenv("FOO")`
Java	`System.getenv("FOO")`
Node.js	`process.env.FOO`
PHP	`getenv("FOO")`
Python	`os.getenv("FOO")`
Ruby	`ENV["FOO"]`
Rust	`env::var("FOO")`

Environment variables are always strings

You may put whatever kind of value you want into an environment variable:

```
export MEANING_OF_LIFE=42  # A number
export PERSON='{"name":"John Doe","age":24}'  # Serialized JSON
```

In your programming language of choice, however, the value will **always** be a character string. It's up to you to parse it if you want to use it as another type, for example in Node.js:

```
> console.log(process.env.MEANING_OF_LIFE);
42
*> process.env.MEANING_OF_LIFE + 2
*'422'
> typeof process.env.MEANING_OF_LIFE
string
> parseInt(process.env.MEANING_OF_LIFE, 10) + 2
44

> process.env.PERSON.name
undefined
> JSON.parse(process.env.PERSON).name
'John Doe'
```

References

- Environment Variable
- Linux/Unix list of common environment variables
- All you need to know about Unix environment variables

↑ Back to top