An Ultimate Guide of Bash Environment Variables

Written by
Mohammad Shah Miran

Reviewed by
Md. Ashakul Islam Sowad

Bash scripting has become an indispensable skill for developers and system administrators, enabling them to automate tasks with ease. At the heart of this power lies the concept of **environment variables'** dynamic placeholders that store crucial system information. In this article, I will discuss **the environmental variables in Bash Script**. So let's start!

Table of Contents [Expand]

Key Takeaways

- Understanding the concept of Environmental Bash Variable.
- Getting the List of Environmental Variables.
- Learning about the setting of temporary and permanent environment variables.

Free Downloads

Download the Practice Files

What is the Environment Variable?

In **Linux**, an **environment variable** is a dynamic named value that holds information about the system's **configuration**, **preferences**, and **behavior**, which can be accessed by processes and programs running on the system. Environment Variables offer a **simple** and **efficient** way to communicate essential information about the current operating environment to the executing program.

When a program or command runs, it receives an array of strings known as the "**environment**." This **array** consists of **key/value** pairs in the form of **key=value**. Each **key/value** pair acts as an **environment variable**, making it accessible to the executed command or program. The shell provides different methods to mark a variable for export to the **environment variables**, with the preferred approach in **Bash** being the use of the **declare -x** command.

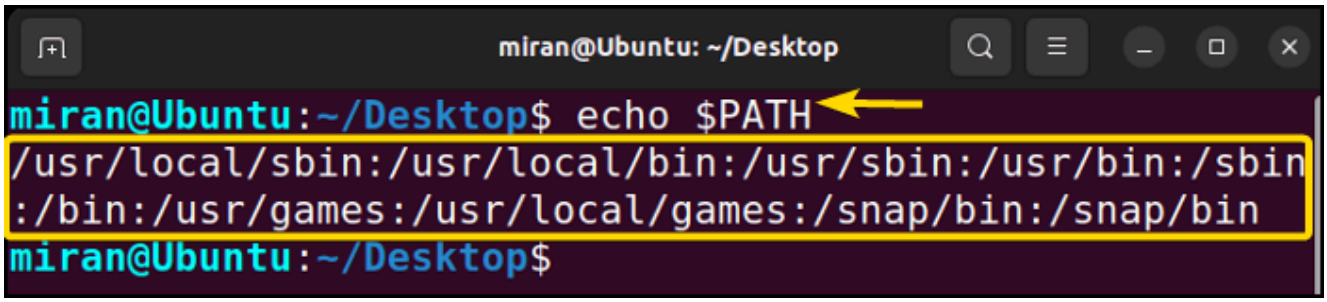
Accessing Environment Variables

Accessing environment variables in a shell allows you to retrieve the values stored in these variables, which can be crucial for configuring the behavior of various programs and scripts.

A. Using Environment Variables in Command Line

In **Bash**, you can access the value of an **environment variable** by using the **dollar sign $** followed by the variable name. For example, to access the **PATH** environment variable, which stores a list of directories where the shell looks for executable files, you would use **$PATH**. The syntax is as follows:

```
echo $PATH
```



Here, this command would display the value of the **PATH** environment variable.

B. Using Environment Variables in Bash Scripts

You can use **environment variables** within scripts to customize their behavior based on the **current environment**. Follow the **Bash Script** described below:

Steps to Follow >

❶ At first, launch an **Ubuntu** Terminal.

❷ Write the following command to open a file in Nano:

```
nano setvar.sh
```

EXPLANATION

- **nano:** Opens a file in the Nano text editor.
- **setvar.sh**: Name of the file.

❸ Copy the script mentioned below:

```
#!/bin/bash

# Store the value of the HOME environment variable in a variable
user_home=$HOME

# Use the HOME environment variable to change the working directory
cd $HOME
```

```
# Utilise an environment variable in a command
echo "Hello, $USER! The current time is: $(date)"
```

EXPLANATION

It begins by capturing the value of the **HOME** environment variable, which holds the path to the **user's home directory**, and stores it in a variable named **user_home**. Next, the script employs this stored value to change the current working directory to the user's home directory using the **cd command**. Finally, the script employs environment variables to construct a message that's displayed using the **echo command**. This message greets the user with their **username**, is accessed through the **USER environment variable**, and provides the current time through the date command's output, which is achieved using command substitution **($(date))**.

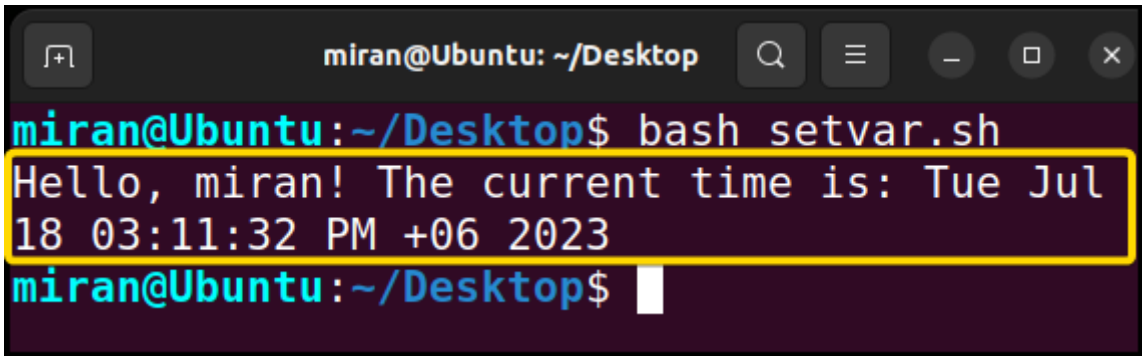
❹ Press **CTRL+O** and **ENTER** to save the file; **CTRL+X** to exit.

❺ Now run the file using the following command.

```
bash setvar.sh
```

EXPLANATION

- **bash:** Executes the shell file.
- **setvar.sh**: Name of the file.



Here the Bash code print **username** and **current date** using the **$USER** and **$date** variable. Where **$USER** is an environmental variable.

List of Environment Variables in Bash

So far, I have introduced some **Bash** variables like **PATH** and **HOME** variables. Now the question is how many environmental variables there in Bash Script are? In **Bash**, there are **several environment variables** that have predefined meanings and are used to configure the behavior of the shell and other programs. Below is a list of some common **environment variables** in **Bash** along with their descriptions:

Variable	Description
HOME	Represents the current user's home directory.
PATH	Contains a colon-separated list of directories where the shell looks for executable files.
USER	Stores the **username** of the current user.
SHELL	Specifies the path to the current user's default shell.
PWD	Represents the **present** working directory.
OLDPWD	Stores the **previous** working directory.
LANG	Defines the default language and character encoding for interpreting text and data.
TERM	Specifies the terminal type, helping programs determine how to interact with the terminal.
PS1	Defines the primary prompt **string**, displaying information like the **username**, **hostname**, and **current directory** in the shell prompt.
PS2	Specifies the secondary prompt string used when entering multiline commands.
PS3	The prompt string used for the **select** command in shell scripts.

PS4	The prompt string used for debugging with the **set -x** option.
IFS	Specifies the **Internal Field Separator**, used by the shell for word splitting.
BASH_VERSION	Stores the version number of the Bash shell.
HOSTNAME	Represents the hostname of the computer.
UID	Stores the numeric user **ID** of the current user.
EUID	Stores the effective numeric **user ID** of the **current user**.
RANDOM	Contains a random integer between **0** and **32,767**.
OSTYPE	Specifies the operating system type (e.g., **linux–gnu** or **darwin** for **macOS**).

These are just some of the commonly used environment variables in **Bash**. There are many other environment variables used by various programs and utilities for specific purposes. You can view the complete list of environment variables in your Bash shell by running the command **printenv** or **env,** just like demonstrates below.

```
miran@Ubuntu:~/Desktop$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/Ubuntu:@/tmp/.ICE-unix/16
52,unix/Ubuntu:/tmp/.ICE-unix/1652
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LC_ADDRESS=en_US.UTF-8
GNOME_SHELL_SESSION_MODE=ubuntu
LC_NAME=en_US.UTF-8
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
LC_MONETARY=en_US.UTF-8
GTK_MODULES=gail:atk-bridge
PWD=/home/miran/Desktop
LOGNAME=miran
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=wayland
SYSTEMD_EXEC_PID=1687
```

Setting an Environment Variable in Linux

Typically, the installation process **automatically updates** your **environment variables** to accommodate the new application. However, there are instances where you might need to manually handle an environment variable when installing something outside your distribution's standard tools. Alternatively, you might choose to customize an environment variable according to your preferences.

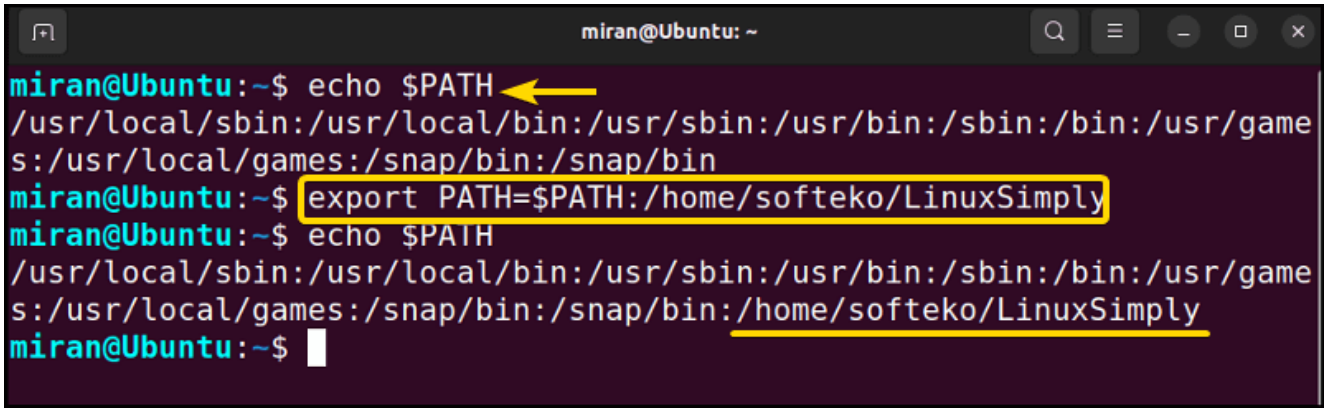
For example, if you wish to keep certain applications in a **bin folder** within your **home directory**, you'll need to add that directory to your **PATH**. By doing so, your operating system will recognize the location and look there applications to execute whenever you issue a command.

A. Set Temporary Environment Variables in Linux

You have the option to append a location to your system's **PATH**. However, this method has a limitation such as the change will only remain effective as long as the current shell session remains open. If you open a Bash shell and modify your system path using this approach:

```
export PATH=$PATH:/home/softeko/LinuxSimply
```

Now if you print the **PATH** variable using the **echo command** then you will notice that the above-mentioned path has been appended to the **PATH** variable.



However, close the current session using the [exit command] in your command line, then write `echo $PATH` to the terminal. Thus you can see the appended path is no longer in the **PATH** variable.

As you can see, the variable has returned to its original/default state because the **PATH** is not being set with every new shell session. To ensure that your variables are configured to load each time a shell is launched, you need to go for a **permanent setting** for the **Environmental Variable**.

B. Set Permanent Environment Variables in Linux

Permanent environment variables are those that are set to persist across **multiple sessions** and are available whenever a user logs in or opens a **new terminal** session. These variables remain active until explicitly **unset** or **modified**.

In Bash, the most common files used to set **permanent environment variables** in Bash are **~/.bashrc** and **~/.bash_profile** (or **~/.profile**).

Here's how to set a permanent environment variable in Bash:

Steps to Follow >

❶ At first, launch an **Ubuntu** Terminal.

❷ Open the **~/.bashrc** by running **nano ~/.bashrc** in your command line.

```
nano ~/.bashrc
```

❸ To edit the **~/.bashrc** file and append your variable to it, add the following line at the end of the file:

```
export VARIABLE_NAME=value
```

Replace **VARIABLE_NAME** with the name of your **environment variable** and value with the desired value. For instance, I want to set a permanent environmental variable named **project_name** and **LinuxSimply** as its value in the **~/.bashrc** file. To do so, I will write project_name=LinuxSimply at the end of the **~/.bashrc** file.

```
miran@Ubuntu: ~
  GNU nano 6.2      /home/miran/.bashrc
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

#Customized and  permanent environment variable.
project_name=LinuxSimply

^G Help      ^O Write Out ^W Where Is   ^K Cut
^X Exit      ^R Read File ^\ Replace    ^U Paste
```

❹ Save the file with **CTRL + O** and Exit from the editor using **CTRL + X** shortcut keys.

❺ To make the changes take effect, either restart your terminal session or run the following command to apply the changes to the current session:

```
source ~/.bashrc
```

❻ Now print the the value of **project_name** variable using the following command.

```
echo $project_name
```

As the image describes, the value of the permanent environment variable is **LinuxSimply**. However, you should keep in mind that setting permanent environment variables in a **user's shell configuration** files only affects that

current **user's sessions**. If you need to set variables for all users, you should consider using **/etc/environment** or other system-wide configuration files.

Conclusion

In conclusion, **environment variables** in Bash scripting offer a powerful and flexible way to manage configurations and preferences. They facilitate seamless communication between scripts, applications, and the operating system, enabling greater adaptability and efficiency. In this article, I have tried to give you a complete guideline about what **environmental variables** are in **Bash Script**, listing some of them and how to set up your own **environmental variable**. However, if you have any questions or queries related to this article, feel free to comment below. Thank You!

People Also Ask

− How to set environment variables in bash?

To set environment variables in **Bash**, use the **export command** followed by the **variable name** and its **value**. For example **export VARIABLE_NAME=value.**

+ What is the $SHELL environment variable?

+ How to access environment variables in shell script?

+ How to check if env variable is set in bash?

Related Articles

- [What Are Built-in Variables in Bash [2 Cases With Examples]]
- [The ".bashrc" Environment Variables [4 Cases]]
- [String Variables in Bash [Manipulation, Interpolation & Testing]]
- [What is Variable Array in Bash? [4 Cases]]
- [An Extensive Exploration of Bash Special Variables [9 Examples]]
- [What is Boolean Variable in Bash? [3 Cases With Examples]]

- What is HereDoc Variable in Bash? [5 Practical Cases]
- What is PS1 Variable in Bash? [3 Customization Examples]
- What is PS3 Variable in Bash? [3 Practical Examples]

<< Go Back to Types of Variables in Bash | Bash Variables | Bash Scripting Tutorial

Rate this post

Bash Scripting

variable types

Mohammad Shah Miran

Hey, I'm Mohammad Shah Miran, previously worked as a VBA and Excel Content Developer at SOFTEKO, and for now working as a Linux Content Developer Executive in LinuxSimply Project. I completed my graduation from Bangladesh University of Engineering and Technology (BUET). As a part of my job, i communicate with Linux operating system, without letting the GUI to intervene and try to pass it to our audience.

Leave a Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Related Articles

What is PS3 Variable in Bash? [3 Practical Examples]

What is “HereDoc” Variable in Bash? [5 Practical Cases]

What is PS1 Variable in Bash? [3 Customization Examples]

What is Boolean Variable in Bash? [3 Cases With Examples]

Special Variables in Bash [With 9 Practical Examples]

What is Variable Array in Bash? [4 Cases]

String Variables in Bash [Manipulation, Interpolation & Testing]

The ".bashrc" Environment Variables [4 Cases]

About LinuxSimply

LinuxSimply serves as an informational repository about the Linux operating system.

Get in Touch

Enter email address

Subscribe

Company

About

Contact

Contributors

Resources

Glossary

Commands

Downloads

Legal Corner

Disclaimer

Privacy Policy

Editorial Policy

Terms & Conditions