Environment Variables

1. Shell - Linux/Mac Users

For Unix/Linux/Mac operating systems, a shell is a command-line program that accepts users' commands and executes those commands on the underlying kernel. Each command has a specific job to perform.

There are multiple shells available. The default shell for (most) Linux systems is the bash shell. Other examples are ksh, tcsh, and zsh. The default shell for macOS 10+ is .zsh.

Your default shell boots when you open a terminal, which allows you to execute commands.

2. Environment Variables - Linux/Mac Users

Assume you store the user-specific secrets, such as username, password, or private key, into a simple file. It might not be a safe approach because all the sensitive information may become public if you put that information on Github/any other Version Control System. User-specific secrets, visible publicly, are never a good thing.

Here comes the role of Environment variables in this scenario. Environment variables are pretty much like standard variables, in that they have a name and hold value. The environment variables only belong to your local system and won't be visible when you push your code to a different environment like Github.

a. The .env file

The .env file is one of the *hidden* files in which you can store your choice of environment variables. The variables stored in this file are your individual working environment variables. ***Note that the environment variables that are stored in the*** ***.env*** ***file override the variables set in the*** ***/etc/environment*** ***file, that is shared by all users of that computer.***  
You will need to follow the steps below to configure environment variables in a .env file:

1. **Install environment variables package** -

npm install dotenv --save

This will allow you to use the environment variables that you'll set in a new file.

1. **Create a new** **.env** **file** in the root of your project. Fill the .env file with your new variables, and their corresponding values. For example:

POSTGRES\_USERNAME = yourUsername

POSTGRES\_PASSWORD = yourpassword

AWS\_REGION = yourAWSRegion

AWS\_PROFILE=awsProfileName

1. **Require the package in your server** - Add the following code on top of the server.ts file

**require**('dotenv').**config**();

1. **Use your environment variables** - If you want to refer the environment variables that you just saved in the .env file, anywhere in the code, try putting a prefix process.env. in front of the variable name. For example, process.env.POSTGRES\_USERNAME will fetch you the value stored in it.
2. **Add** **.env** **to your** **.gitignore** - You wouldn't want your .env file to be available publicly in the project Github repository. For this reason, go to the .gitignore file in the project root, and add and entry .env to it. It will make sure that you don't push our environment variables to Github!

b. The process.env file

The process.env file is a default file that stores the variables for the current terminal environment. When you run the following command, it will store the POSTGRES\_USERNAME to the current terminal environment:

**export** POSTGRES\_USERNAME = yourUsername

By default, the Node is accessing the same set of variables that are defined in your process.env file.

c. Bash Profile - .profile file

You won't want to export the user-specific variables *every time* you'll log in to your system, and do not want to override the variables set in the root level /etc/environment file. The solution is to store the new variables *either* in .profile,.bashrc or .zshrc file, depending on your shell. These are the files that the shell executes even before you type your first command to it. ***Note that every user of the computer has its own*** ***.profile*** ***file.***

When you put

**export** AWS\_PROFILE=awsProfileName

inside the .profile file, it will run this command before you start firing commands in your shell.

Usually, the bash profile is found at ~/.profile, where ~ represents your current logged in user's home directory. Keep in mind the . preceding profile means this file will be hidden.

If you wish to instruct your Node to execute the .profile file anytime, you can run the following command:

**source** ~/.profile

d. Using the Manual Page - man command

Most Bash commands in the terminal give you instructions on how to use them when you type man <command> where <command> could be any CLI command. For example, typing man bash into the terminal will give you the manual page for bash.

The *INVOCATION* section of this man page will give you some hints to where bash looks for profiles when starting.

3. Environment Variables - Windows Users

Windows has the same concept of variable stored at the OS level to use within and across applications. Windows has two types of Environment Variables:

* **User Environment Variables** which are accessible only to the currently logged in user
* **System Environment Variables** which are available *all* users on the machine

**Setting Windows Environment Variables**

Environment variables are set on Windows using a GUI (Graphical User Interface). On Windows 10, this can be found by:

1. From the start menu, right-click the Computer icon
2. Select Properties
3. Select Advanced System Settings on the left
4. In the new window, click Environment Variables
5. Use the New... and Edit... buttons to set and modify your variables

You can follow [**this handy guide**](https://www.computerhope.com/issues/ch000549.htm) for your flavor of Windows.

4. Run Linux Environment on Windows

Windows OS also has a concept of the shell. The default shell in Windows is the command-line tool **Cmd.exe**. There is another shell available in Windows 7 SP1and above, [**PowerShell**](https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/install/installing-windows-powershell?view=powershell-7). PowerShell is primarily used for Windows system administration. Neither CMD nor PowerShell can run bash, ssh, git, apt, or any Linux commands by default.

The solution is to use *either* of the options below:

Option 1 - Windows Subsystem for Linux

[**Windows Subsystem for Linux**](https://docs.microsoft.com/en-us/windows/wsl/about) (WSL) - WSL allows us to run Linux environment, including most command-line tools, utilities, and applications, from the Windows Command Prompt (CMD). You can even mix the Linux and Windows commands after installing WSL. Refer to the installation instructions [**here**](https://docs.microsoft.com/en-us/windows/wsl/install-win10) to install WSL on Windows.

The next step is to install and run a Linux distribution parallelly on WSL. There are multiple choices for installing - Ubuntu, OpenSUSE, Debian, and many more. If you have no preference, you can install [**Ubuntu on Windows**](https://www.microsoft.com/en-us/p/ubuntu/9nblggh4msv6?activetab=pivot:overviewtab) App, and proceed as mentioned in the installation instructions above.

Option 2 - Git Bash on Windows

Git is an open-source distributed Version Control System (VCS). Github is a repository hosting and version control service, where you can store, share, or download the repository content in collaboration with multiple contributors. Git provides a Unix style command-line tool called [**Git for Windows**](https://git-scm.com/download/win) to help users work with Github repositories. Once you download and install Git for Windows, it can be run either in CMD or a GUI.

[**Git Bash**](https://www.atlassian.com/git/tutorials/git-bash) is a command-line tool by default included in Git for Windows. Besides running Git commands, Git Bash allows users to run Linux/Bash commands as well.