2.5.3 Environment Variable Facts

Like most variables, an environment variable is a data object that has a name and a value. In Linux, environment variables provide a way to share settings and configurations across different applications, processes, and services.

This lesson covers the following topics:

- Environment variables vs shell, user, and local variables
- Creating and working with environment variables
- Creating persistent environment variables
- Common environment variables

Environment Variables vs Shell, User, and Local Variables

When a shell session is spawned, environment information is gathered from a variety of files and settings on the system and is made available to the shell process. This environment is implemented as key-value pairs or environment variables. Each environment variable has a name or variable identifier, which is conventionally uppercase. Environment variables can be assigned one value or multiple values. If the shell session spawns a new shell or runs a program, this environment (that is, this set of environment variables) is inherited or passed on to the child process.

Shell variables may appear to be environment variables, since they also have uppercase names. However, these variables are not inherited from a parent process, but are generated by scripts as the shell is initiated. For example, HISTFILESIZE is created during a Bash shell initialization and determines the number of commands that are stored in a history file. If this Bash shell spawns a C shell as a child process, HISTFILESIZE won't be defined. It isn't inherited from the Bash shell and is not generated when the C shell is initiated. This confusion between environment variables and shell variables is amplified by the fact that users generally work with only one shell type, and the shell variables are initiated across all shell sessions in the same way. This consistency makes shell variables appear to be inherited from a parent shell when, in reality, they are generated during the shell initialization. A further reason for confusion is that many environment variables available in a shell are created and exported when the shell initializes.

All Linux variables can be classified as one of two types, environment variables or shell variables, with the main difference being that environment variables are inherited by child processes, but shell variables aren't.

Both user variables and local variables are strictly shell variables. If there is any distinction between them, it is in how they are created and used. User variables are typically defined at the shell prompt or added when scripts that are defined in a user's profile are run. Variables defined at the shell prompt are considered local variables, since they are dropped as the process exits. Variables defined in scripts and in script functions are usually defined as local variables, since they generally do not exist after the script or function is run. By convention, user and local variables are given lowercase names to avoid overwriting environment variables and shell variables.

The following table summarizes these ideas.

Variable Type	Typical Source	Naming Convention	Inherited by Child Processes
Environment	Inherited from a parent process, which may be the system process which passes environment information gathered from a variety of system files and settings.	Uppercase	Yes
Shell	Generated by shell startup scripts.	Uppercase	No
User	Created by a user at the shell prompt or added when scripts that are defined in a user's profile are run.	Lowercase	No
Local	Defined in scripts and in script functions.	Lowercase	No

Creating and Working with Environment Variables

A shell variable becomes an environment variable when it is exported. Essentially, this changes a variable's export attribute. Be aware of the following when working with environment variables:

- By convention, environment variable names (variable identifiers) are defined and referenced using uppercase characters, making them less likely to be overridden by lowercase shell variable identifiers.
- If an environment variable has multiple values, they are separated by a colon (:) character.
- The default values assigned to environment variables can be overridden to customize the user's computing environment.
  - An overridden environment variable will only apply to the current shell session and any subsequent child processes.

The following table lists command that are used when creating and working with environment variables.

Command	Function	Example
[name]=[value]	Creates a new shell variable with an assigned value or change the value of an existing variable. To append information to a variable instead of replacing it, include the current variable in the command. (See example.) Changing the value of an environment variable will change if for the current shell session any subsequent child processes.	**TRAINING="TestOut"** **PATH=$PATH:/bin/additionalpath**
export [name]=[value]	Creates a new environment variable for the current shell session and any subsequent child processes.	**export TRAINING="TestOut"**
export [name]	Exports an existing shell variable to make it an environment variable for the current shell session and any subsequent child processes.	**TRAINING="TestOut"** **export TRAINING**
declare -x [name]=[value]	Creates a new environment variable for the current shell session and any subsequent child processes (functionally equivalent to **export [name]=[value]**).	**declare -x TRAINING="TestOut"**
declare -x [name]	Exports an existing shell variable to make it an environment variable for the current shell session and any subsequent child processes (functionally equivalent to **export [name]**).	**TRAINING="TestOut"** **declare -x TRAINING**
export -n [name]	Removes the export property of an environment variable, making it a shell variable.	**export -n TRAINING**
declare +x [name]	Removes the export property of an environment variable, making it a shell variable (functionally equivalent to **export -n [name]**).	**declare +x TRAINING**
echo $[name]	Displays the contents of an environment variable (or any variable).	**echo $TRAINING**
printenv	Displays a list of the current environment variables.	**printenv**
env	Displays a list of the current environment variables. The **env** command can also be used to run a command using temporarily manipulated environment variables.	**env**
export -p	Displays a list of all exported variables and functions.	**export -p**
set	Displays a list of all environment variables, shell variables, local variables, and shell functions. The **set** command can also be used to set or unset values of shell options and positional parameters.	**set**
unset [name]	Removes the variable and its value independent of whether the variable is an environment variable or a shell variable.	**unset TRAINING**

Creating Persistent Environment Variables

The above commands can be added to the following system and bash configuration files to create environment variables that persist across system reboots and bash shell startups:

File(s)	Description
/etc/environment	The /etc/environment file is a system-wide configuration file which is used by all users. Environment variables defined in this file are available to all shells and processes. Changes to this file do not take affect until after a reboot.

/etc/profile	This file is run for all users, but only during interactive logins (when a username and password is required to log in). While environment variable can be defined in this file, the preferred method is to define them in a separate script file saved in the /etc/profile.d directory. An interactive login occurs when the user must supply a username and password. An example is when the user connects using SSH, when using Ctrl+Alt+F2 to log into a virtual terminal, or when using the **su** (switch user) command.
/etc/profile.d/*.sh	These script files are run for all users, but only during interactive logins. Environment variable definitions can be added to these files. These files are run by the /etc/profile file.
~/.bash_profile ~/.bash_login ~/.profile	These hidden files are located in a user's home directory and are only run for that specific user and only during interactive logins. A search is made for these files in the order that is listed. Only the first file found will be run. Environment variable definitions can be added to these files.
/etc/bashrc	This file is run for all users during a Bash shell initialization. It is run during both interactive logins and interactive non-logins. Environment variable definitions can be added to them. An interactive non-login occurs when the user is not required to enter a username and password to open a shell. An example is when the user opens a graphical terminal window in gnome, or opens a new shell using the **bash** command.
~/.bashrc	This hidden file is located in a users home directory and is only run for that specific user during both interactive logins and interactive non-logins. Environment variable definitions can be added to this file.

Common Environment Variables

The table below lists common environment variables.

A few of these environment variables are exported during shell initialization and are not inherited from the system process. They are still considered environment variables because they are inherited by any child processes.

Variable	Description
SHELL	The user's login shell.
DISPLAY	The location where X Windows output is displayed.
ENV	The location of the configuration file for the current shell.
HISTSIZE	The number of lines or commands that are stored in memory in a history list while your shell session is ongoing.
HOME	The absolute path of the user's home directory.
HOSTNAME	HOSTNAME is identical to HOST, but is only used on certain distributions.
LOGNAME	The username of the current user.
MAIL	The path to the current user's mailbox file.
OLDPWD	The directory the user was in prior to switching to the current directory.
PATH	The directory paths used to search for programs and files. ▪ Use a colon to separate entries in the PATH variable. ▪ Do not include a period (.) in the PATH variable. A period indicates that the working directory is in the path and poses a security risk.
PWD	The path of the current working directory.
LANG	The language the operating system uses.
TERM	The type of terminal to emulate when running the shell.

USER	The current logged in user.

The table below lists useful bash shell variables:

These shell variables are not inherited by child shells and processes. However, when the child shell is created, these shell variables will be initialized.

Variable	Description
BASH	The location of the bash executable file.
BASHOPTS	The list of options that were used when bash was executed.
BASH_VERSION	The version of bash being executed in a readable form.
BASH_VERSINFO	The version of bash in machine-readable output.
EUID	The ID number of the current user.
HISTFILE	The filename where past commands are stored.
HISTFILESIZE	The number of past commands that HISTFILE stores for the multiple sessions.
OSTYPE	The type of operating system (usually Linux).
PS1	The characters the shell uses to define what the shell prompt looks like.
COLUMNS	The number of columns being used to draw output on the screen.
LINES	The number of lines being used to draw output on the screen.
UID	The user ID of the current user.