**Environment**

Each time we launch the terminal application, it creates a new session. The session immediately loads settings and preferences that make up the command line *environment*.

We can configure the environment to support the commands and programs we create. This enables us to customize greetings and command aliases, and create variables to share across commands and programs.

You can reference the filesystem for this lesson [here](https://s3.amazonaws.com/codecademy-content/courses/learn-the-command-line/img/LCL-fileTrees-04.png).

**Instructions**

**1.**

We’ll begin by learning to use a simple, command line text editor called *nano*.

In the terminal, type

nano hello.txt

This will open the nano text editor.

**2.**

In nano, at the top of the window, type

"Hello, I am nano."

Using the menu at the bottom of the terminal for reference, type Ctrl + O (the letter, not the number) to save the file. This is the letter “O”, not the number zero.

Press Enter, when prompted about the filename to write.

Then type Ctrl + X to exit nano.

Finally, type clear to clear the terminal window. The command prompt should now be at the top of the window.

# nano

Nice. You just edited a file in the nano text editor. How does it work?

$ nano hello.txt

nano is a command line text editor. It works just like a desktop text editor like TextEdit or Notepad, except that it is accessible from the command line and only accepts keyboard input.

1. The command nano hello.txt opens a new text file named **hello.txt** in the nano text editor.
2. "Hello, I am nano" is a text string entered in nano through the cursor.
3. The menu of keyboard commands at the bottom of the window allow us to save changes to **hello.txt** and exit nano. The ^ stands for the Ctrl key.

* Ctrl + O saves a file. ‘O’ stands for output.
* Ctrl + X exits the nano program. ‘X’ stands for exit.
* Ctrl + G opens a help menu.
* clear clears the terminal window, moving the command prompt to the top of the screen.

In this lesson, we’ll use nano to implement changes to the environment. You can learn more about nano [here](http://www.nano-editor.org/).

**Instructions**

**1.**

Now that you are familiar with editing text in nano, let’s create a file to store environment settings.

In the terminal, type

nano ~/.bash\_profile

This opens up a new file in nano.

(**Tip**: If you are unable to type ~, you can find help [here](http://discuss.codecademy.com/t/my-keyboard-does-not-allow-me-to-type-a-character-used-in-a-codecademy-lesson-what-should-i-do/29228).)

**2.**

In **~/.bash\_profile**, at the top of the file, type

echo "Welcome, Jane Doe"

You can use your name in place of “Jane Doe”.

Type Ctrl + O to save the file.

Press Enter to write the filename.

Type Ctrl + X to exit.

Finally, type clear to clear the terminal window.

**3.**

In the terminal, type

source ~/.bash\_profile

You should see the greeting you entered.

**Bash Profile**

You created a file in nano called **~/.bash\_profile** and added a greeting. How does this work?

$ nano ~/.bash\_profile

**~/.bash\_profile** is the name of file used to store environment settings. It is commonly called the “bash profile”. When a session starts, it will load the contents of the bash profile before executing commands.

* The ~ represents the user’s home directory.
* The . indicates a hidden file.
* The name **~/.bash\_profile** is important, since this is how the command line recognizes the bash profile.

1. The command nano ~/.bash\_profile opens up **~/.bash\_profile** in nano.
2. The text echo "Welcome, Jane Doe" creates a greeting in the bash profile, which is saved. It tells the command line to echo the string “Welcome, Jane Doe” when a terminal session begins.
3. The command source ~/.bash\_profile activates the changes in **~/.bash\_profile** for the current session. Instead of closing the terminal and needing to start a new session, source makes the changes available right away in the session we are in.

**Instructions**

**1.**

Now that we know what bash profile is, let’s continue configuring the environment by adding command aliases.

Open **~/.bash\_profile** in nano.

**2.**

In **~/.bash\_profile**, beneath the greeting you created, type

alias pd="pwd"

Save the file.

Press Enter to write the filename

Exit nano.

Clear the terminal window.

**3.**

In the command line, use the source command to activate the changes in the current session.

source ~/.bash\_profile

**4.**

Let’s try out the alias. Type

pd

You should see the same output as you would by typing the pwd command.

**Aliases I**

What happens when you store this alias in **~/.bash\_profile**?

alias pd="pwd"

The alias command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

1. Here alias pd="pwd" creates the alias pd for the pwd command, which is then saved in the bash profile. Each time you enter pd, the output will be the same as the pwd command.
2. The command source ~/.bash\_profile makes the alias pd available in the current session.

Each time we open up the terminal, we can use the pd alias.

**Instructions**

**1.**

Let’s practice aliases some more. Open **~/.bash\_profile** in nano.

**2.**

In the bash profile, beneath the previous alias, add

alias hy="history"

Save the file.

Press Enter to write the filename.

**3.**

Add another alias

alias ll="ls -la"

Save the file.

Press Enter to write the filename.

Exit nano

Clear the terminal window.

**4.**

In the command line, use source to activate the changes to the bash profile for the current session.

**5.**

Let’s try out the aliases. In the command line, type

hy

**6.**

Now type

ll

# Aliases II

What happens when you store the following aliases in **~/.bash\_profile**?

alias hy="history"

hy is set as alias for the history command in the bash profile. The alias is then made available in the current session through source. By typing hy, the command line outputs a history of commands that were entered in the current session.

alias ll="ls -la"

ll is set as an alias for ls -la and made available in the current session through source. By typing ll, the command line now outputs all contents and directories in long format, including all hidden files.

**Instructions**

**1.**

Now that you are familiar with configuring greetings and aliases, let’s move on to setting environment variables.

Open **~/.bash\_profile** in nano.

**2.**

In the bash profile, beneath the aliases, on a new line, type

export USER="Jane Doe"

Feel free to use your own name.

Save the file.

Press Enter to write the filename.

Exit nano.

Finally, clear the terminal.

**3.**

In the command line, use source to activate the changes in the bash profile for the current session.

**4.**

Type

echo $USER

This should return the value of the variable that you set.

(**Tip**: If you are unable to type $, you can find help [here](http://discuss.codecademy.com/t/my-keyboard-does-not-allow-me-to-type-a-character-used-in-a-codecademy-lesson-what-should-i-do/29228).)

**Environment Variables**

What happens when you store this in **~/.bash\_profile**?

export USER="Jane Doe"

*environment variables* are variables that can be used across commands and programs and hold information about the environment.

1. The line USER="Jane Doe" sets the environment variable USER to a name “Jane Doe”. Usually the USER variable is set to the name of the computer’s owner.
2. The line export makes the variable to be available to all child sessions initiated from the session you are in. This is a way to make the variable persist across programs.
3. At the command line, the command echo $USER returns the value of the variable. Note that $ is always used when returning a variable’s value. Here, the command echo $USER returns the name set for the variable.

**Instructions**

**1.**

Let’s learn a few more environment variables, starting with the variable for the command prompt.

Open **~/.bash\_profile** in nano.

**2.**

On a new line, beneath the last entry, type

export PS1=">> "

Save the file.

Press Enter to write the filename.

Exit nano

Finally, clear the terminal window.

**3.**

In the command line, use source to activate the changes in the bash profile for the current shell session.

**4.**

Let’s try out the new command prompt. In the terminal type

echo "hello"

**5.**

Now type

ls -alt

Did you notice that the prompt has changed?

**PS1**

What happens when this is stored in **~/.bash\_profile**?

export PS1=">> "

PS1 is a variable that defines the makeup and style of the command prompt.

1. export PS1=">> " sets the command prompt variable and exports the variable. Here we change the default command prompt from $ to >>.
2. After using the source command, the command line displays the new command prompt.

**Instructions**

**1.**

Let’s learn about two more environment variables.

In the command line, type

echo $HOME

This returns the value of the HOME variable.

# HOME

What happens when you type this command?

$ echo $HOME

The HOME variable is an environment variable that displays the path of the home directory. Here by typing echo $HOME, the terminal displays the path **/home/ccuser** as output.

You can customize the HOME variable if needed, but in most cases this is not necessary.

**Instructions**

**1.**

In the command line, type

echo $PATH

**2.**

Type

/bin/pwd

**3.**

Type

/bin/ls

**PATH**

What happens when you type this command?

$ echo $PATH

/home/ccuser/.gem/ruby/2.0.0/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin

PATH is an environment variable that stores a list of directories separated by a colon. Looking carefully, echo $PATH lists the following directories:

1. /home/ccuser/.gem/ruby/2.0.0/bin
2. /usr/local/sbin
3. /usr/local/bin
4. /usr/bin
5. /usr/sbin
6. /sbin
7. /bin

Each directory contains scripts for the command line to execute. The PATH variable simply lists which directories contain scripts.

For example, many commands we’ve learned are scripts stored in the **/bin** directory.

/bin/pwd

This is the script that is executed when you type the pwd command.

/bin/ls

This is the script that is executed when you type the ls command.

In advanced cases, you can customize the PATH variable when adding scripts of your own.

**Instructions**

**1.**

In the command line type

env

**2.**

Then type

env | grep PATH

(**Tip**: If you are unable to type |, you can find help [here](http://discuss.codecademy.com/t/my-keyboard-does-not-allow-me-to-type-a-character-used-in-a-codecademy-lesson-what-should-i-do/29228).)

# env

What happens when you type this command?

env

The env command stands for “environment”, and returns a list of the environment variables for the current user. Here, the env command returns a number of variables, including PATH, PWD, PS1, and HOME.

env | grep PATH

env | grep PATH is a command that displays the value of a single environment variable. Here the standard output of env is “piped” to the grep command. grep searches for the value of the variable PATH and outputs it to the terminal.

**Instructions**

The commands and feature we’ve covered so far are commonly used to configure the environment.

Click Next to continue.

# Generalizations

Congratulations! You learned to use the bash profile to configure the environment. What can we generalize so far?

* The environment refers to the preferences and settings of the current user.
* The nano editor is a command line text editor used to configure the environment.
* **~/.bash\_profile** is where environment settings are stored. You can edit this file with nano.
* environment variables are variables that can be used across commands and programs and hold information about the environment.
  + export VARIABLE="Value" sets and exports an environment variable.
  + USER is the name of the current user.
  + PS1 is the command prompt.
  + HOME is the home directory. It is usually not customized.
  + PATH returns a colon separated list of file paths. It is customized in advanced cases.
  + env returns a list of environment variables.