COMP311
Linux OS Laboratory
Lab5:Shell Usage and Configuration (I)

By

Alaa’ Omar

Objectives

1. Become familiar with common Linux shells.

2. Recognize and manipulate system and user defined shell variables.

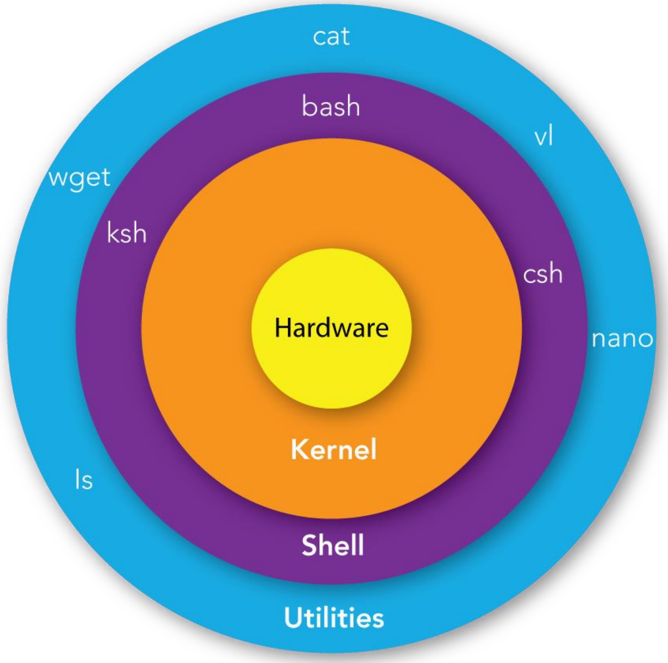
3. Identify and use shell functions like command substitution, aliasing, command line editing and file name completion.

Linux Shells

A shell is a **command interpreter**. It is the main program used by the user to **access** and use the **Linux operating system**. When the user enters a command and hits the Return key the shell checks the command and rejects or accepts it. Accepted commands are then passed on to the Kernel part of the OS for execution and the result is displayed by the shell to the user.

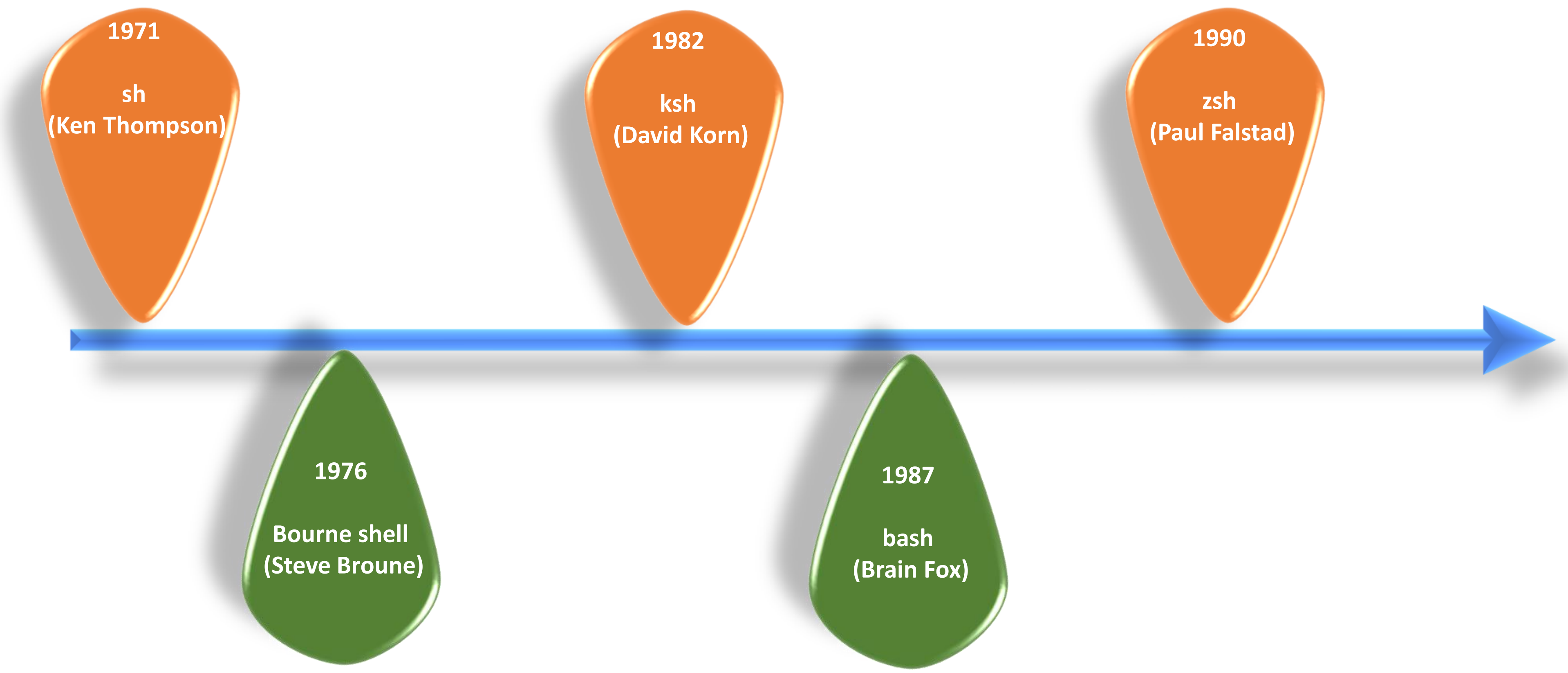
Shell examples in Linux:

- sh (bourne shell)
- csh (C shell)
- ksh (korn shell)
- bash (bourne again shell)

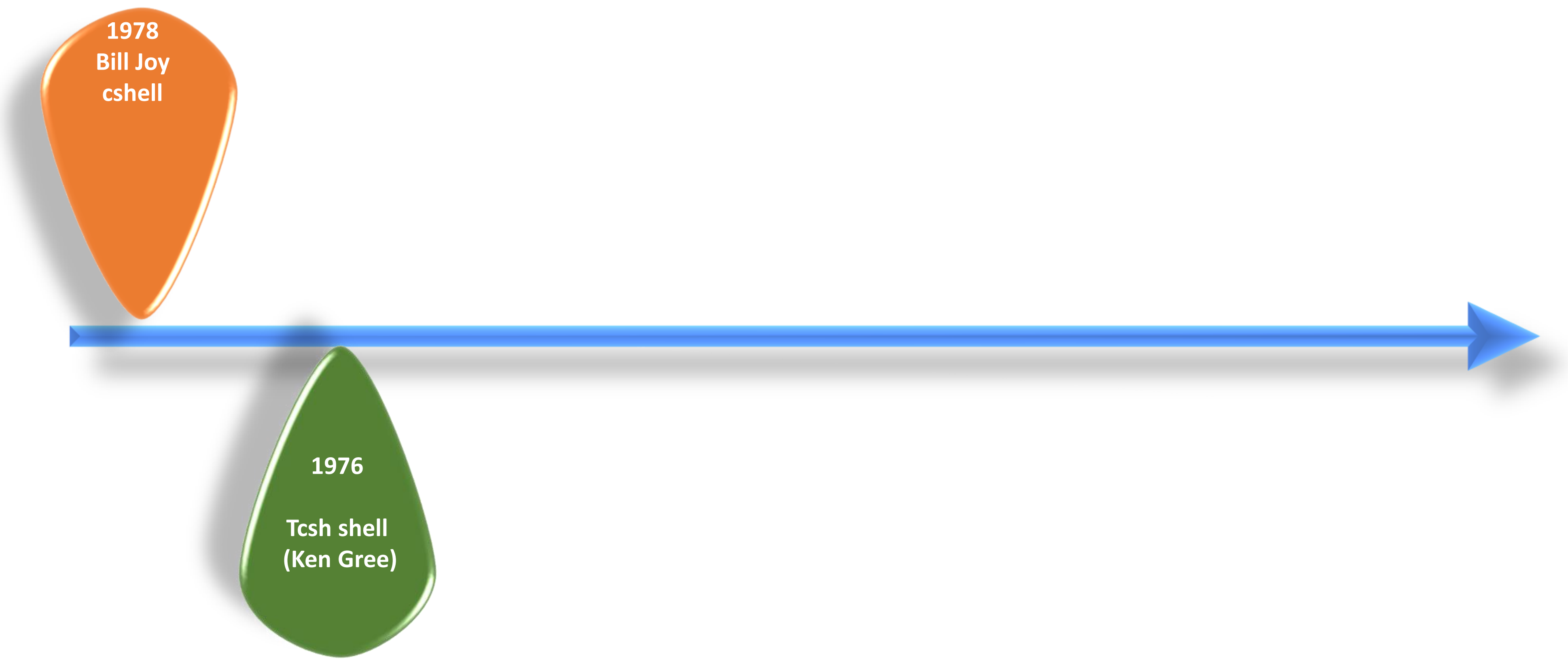


To change from one shell to another simply type the name of the shell on the command line and hit Enter

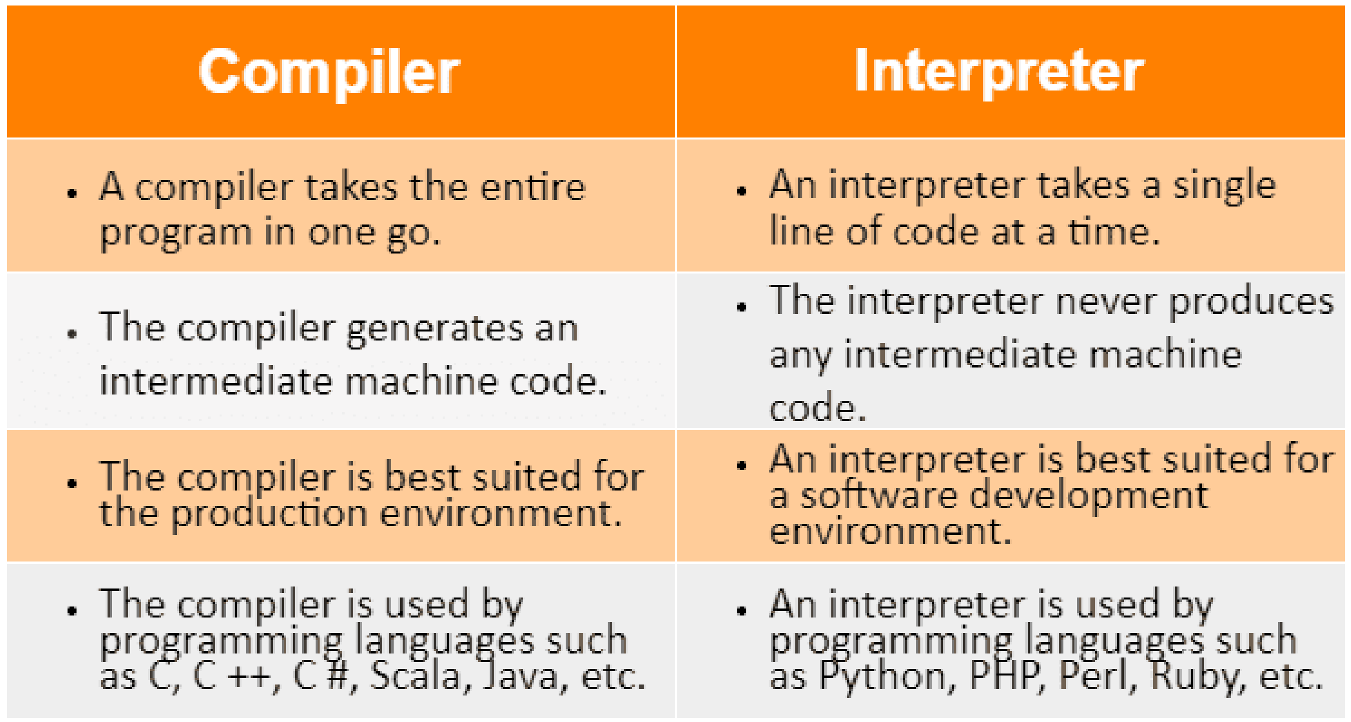
The Broune SHELL family



The C-SHELL family



Compiler vs Interpreter



Compiler	Interpreter
• A compiler takes the entire program in one go.	• An interpreter takes a single line of code at a time.
• The compiler generates an intermediate machine code.	• The interpreter never produces any intermediate machine code.
• The compiler is best suited for the production environment.	• An interpreter is best suited for a software development environment.
• The compiler is used by programming languages such as C, C ++, C #, Scala, Java, etc.	• An interpreter is used by programming languages such as Python, PHP, Perl, Ruby, etc.

SHELL Variables

- A variable: is an entity used to store data. Each variable has a name and a value. The name is an identifier we use to refer that variable. While the value is the data stored within the variable.

- Variables types:
  - Shell variables (system variables)
  - Environment variable (user defined variable)

Variables almost always store only one type of data, a character string "sequence pf plain-text characters"

Shell Features:Variable Substation

A shell is a program that has several variables that help the shell do its work. Many of those variables are **system defined variables (usually written using upper case letters)** and some may be **user defined variables**.

- To display and use the value of a variable, we use $ character followed by the name of the variable echo command. (echo $PATH)
- To set a value for the variable, we use the variable name followed by assign operator = (PATH=‘/’)

The syntax for the echo command is:

```
echo [option(s)] [string(s)]
```

Shell Features:Variable Substation

System Variables:

Let us consider some of the system defined variables to see how the shell uses them.

PATH variable:

Run the command:

echo PATH

What did you get?_______________________.

Now run the command:

echo $PATH

What did you get?____.

Use echo to Display a Linux Environment Variable: **echo $<variable_name>**, For example :

```
$ echo $PATH
```

Shell Features: Variable Substation

SAVEPATH=$PATH (saves the value of PATH in variable SAVEPATH)

$ls Did it work?

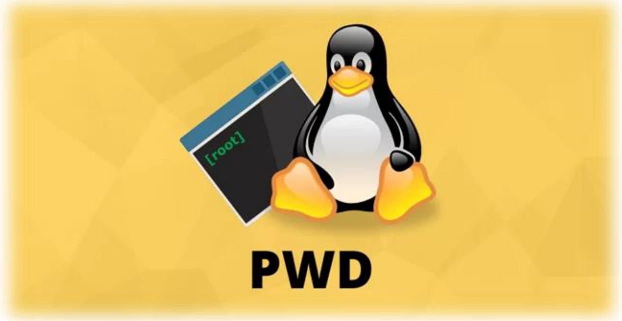
PATH=/etc

$ls Does it work now?   . Why?

Restore the original value for variable PATH. Command? PATH=$SAVEPATH.

Use echo to Display a Linux Environment Variable: echo $<variable_name>, For example :
$ echo $PATH

Shell Features:PWD and PS1 Variables:

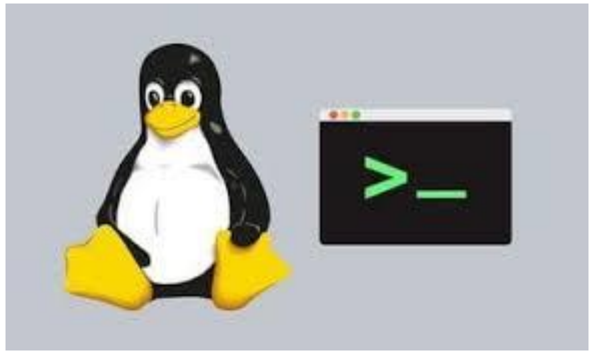


Display the value of the PWD variable. Command?
Change your directory to /etc. Command
What is the value of PWD now?
How do you think the pwd command works

The pwd command uses the value of the PWD environment variable to determine the current working directory. When the shell starts, it sets the PWD variable to the current working directory. When you run the cd command to change the directory, the shell updates the PWD variable to reflect the new working directory. When you run the pwd command, it simply prints the value of the PWD variable.

Shell Features:PWD and PS1 Variables:

Now run the following command:
PS1=”hello >”
What happened to your prompt?
Now run the command:
PS1=’$PWD >’
What happened?



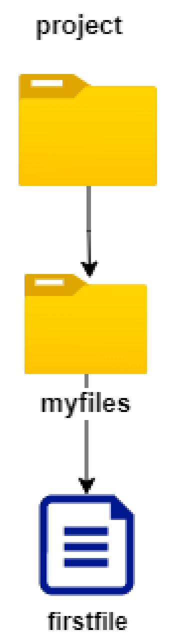
There are several other variables such as HOME, PS2, SHELL, MAIL and so forth. To display the variables in your shell run the command:
set | more
List three more variables other than the ones mentioned above and their values:

1. **USER** - This variable stores the username of the current user. For example, if the current user is john, the value of the USER variable would be john.
2. **TERM** - This variable stores the type of terminal or console being used by the user. For example, if the user is using a Linux console, the value of the TERM variable might be linux.
3. **LANG** - The language environment variable. It specifies the default language for the user.
4. **LC_ALL** - The locale environment variable. It specifies the default locale for the user.

Shell Features:User-Defined Variables

User defined Variables:

Users can define their own shell variables to simplify their work or store values for later use.
Under your home directory (*cd*) create the following structure:



Now create a new variable called **myprojfiles** as follows:
myprojfiles=$HOME/project/myfiles
Now you can use the new variable to manipulate your project directory. Try the following commands and write what each does:
vi $myprojfiles/firstfile

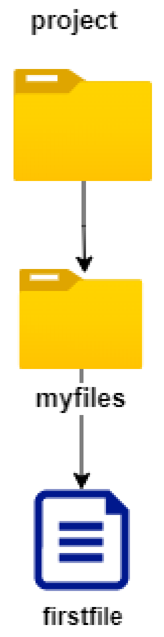
cp /etc/passwd $myprojfiles

touch good; mv $HOME/good $myprojfiles

Shell Features:User-Defined Variables

User defined Variables:

To summarize, a shell checks a command for any variables (words starting with **$)** and substitutes them with their values before executing the command. e.g. in the command **echo $PWD** the shell first substitutes the variable **PWD** for its value and then executes the echo command on that value.



Shell Features:Command Substitution

User defined Variables:

Another important shell function is command substitution where the shell substitutes commands with their results before executing the main command.

Try the command:

date

What is the result? Current date time

Now try to command:

echo $(date)

What is the result?

Shell Features:Command Substitution

User defined Variables:

Another important shell function is command substitution where the shell substitutes commands with their results before executing the main command.

Try the command:
date
What is the result? Current date time
Now try to command:
echo $(date)
What is the result?

Shell Features:Command Substitution

The result of both commands is the same, but for different reasons. In the first case, command date is executed, and the result of the command is displayed. In the second case, the shell first substitutes the result of the command date (which is indicated using the $(command) notation) and then executes the main command echo on that result. Thus, the output of the date command is used as an argument for command echo. Command substitution is very useful for saving command outputs in variables for later use.

Run the command:

```
grep yourusername /etc/passwd | cut -d: -f5 | cut -d_ -f1
```

What is the result? ____.

Shell Features:Command Substitution

To get that result again you need to run the same command each time. You can save the result of that command in a variable for later use using command substitution as follows:

```
firstname=$(grep yourusername /etc/passwd | cut -d: -f5 | cut -d_ -f1)
```

Now you can use the variable **firstname** whenever you need it. This is especially useful in shell scripts. You can run the following command for example:

```
echo how are you doing $firstname?
```

The notation ***$(command)*** is common to many shells, but not all. The **csh** shell does not use that notation. There is another older notation which is understood by most if not all shells. Instead of ***$(command)*** the notation used is ***`command`*** (The single quote used here is the one below the ESC key on the keyboard).

Try the new notation to get your last name and save it in a variable called ***lastname.***

Command:

Shell Features: (5) Aliasing

Another function of the shell is aliasing which is basically used to give new simple names to complicated or long commands. For example:

```
$alias dir=”ls -ali”
$dir
```

The new alias **dir** will now behave exactly as **“ls –ali”** when executed.
To display the aliases, you already have on your system, run the command:

```
alias
```

List three aliases that you have and their values:

- alias dir='ls -ali'
- alias egrep='egrep --color=auto'
- alias fgrep='fgrep --color=auto'
- alias grep='grep --color=auto'
- alias l='ls -CF'
- alias la='ls -A'
- alias ll='ls -alF'
- alias ls='ls --color=auto'

Shell Features: Aliasing

To cancel an alias, use the ***unalias*** command. For example:
unalias dir (cancels the ***dir*** alias)
Always be careful of aliases that have the same names as commonly used commands. An alias such as the following may be very dangerous. Do NOT try it.
alias ls=”rm -f *”

Shell Features:Command Line Editing

The commands you enter on the command line are stored by the shell in a history file called .bash_history under the bash shell. To use or modify commands you executed earlier you can use the arrow keys. The up and down keys are used to get commands and the left and right arrows are used to move and modify a command if needed.

Rename (use the mv command) the file .bash_history to .save_history.
Command:

Exit from the system and log back in.
Check the commands stored in .bash_history. What did you find?

What can you do to restore all your previous commands?

Shell Features:File name completion

Another useful shell function is file name completion where the shell completes long file names for you when you type commands as follows:

Suppose I have a file called : **abcdefghijklmnopqrstuvwxyz** and I need to copy it.

All I have to do is type:

cp abcESCESC newfilename

If there are no other files starting with abc then the shell will complete the long name for me. If there are other files that start with abc then the shell will display them and I need to specify the first different character and then press ESCESC for the shell to complete the name.

Shell Features: Making changes permanent

Many of the changes mentioned above such as creating new variables, changing existing variables, or creating aliases will disappear after exiting and logging back into the system. To make those changes permanent, they need to be added to your environment file (.bash_profile). Be very careful when modifying this file and always copy it first before making modifications.

Copy the file .bash_profile to file .save_bash_profile

Command: **cp .bash_profile .save_bash_profile**

Add the following to the end of your .bash_profile file:

1. **Add the . (current directory) to your PATH variable**
2. **Add a variable called myproj with the name of a project directory under your home directory.**

Save the file and quit.
Exit the system and then log back in.
Check to see if the changes still exist on the system. Do they? ____________________.

The End