Introduction to Shell Scripting
Using Zsh

Rev20230812

Contents

The guide was written using macOS Ventura version 13.4.1 which includes Zsh version 5.9 as it's default shell. Most of the lessons in this guide will work with bash and we will call out any lessons that are Zsh specific.

What is Zsh?

Zsh is called Z Shell, which is an extension of Bash that has many new features and themes. Zsh was released in 1990 by Paul Falstad. Zsh has similarities with Korn shell as well. Zsh is built on top of bash thus it has additional features.

Starting with macOS Catalina, your Mac uses zsh as the default login shell and interactive shell. Zsh provides the user with more flexibility by providing various features such as plug-in support, better customization, theme support, spelling correction, etc.

Zsh is highly compatible with the Bourne shell (sh) and mostly compatible with bash, with some differences. For more about zsh and its comprehensive command-line completion system, enter man zsh in Terminal.

The purpose of this guide is to provide an introduction to shell scripting. No prior shell script experience is required to follow along with this guide. We recommend using a non production Mac computer when following along with the lessons in this guide.

NOTE: When following the lessons in this guide, do NOT copy and paste the code from this PDF. Doing so may cause issues when running the scripts. We recommend typing out all the commands or using the example script files for best results.

Section 1: Basic Commands

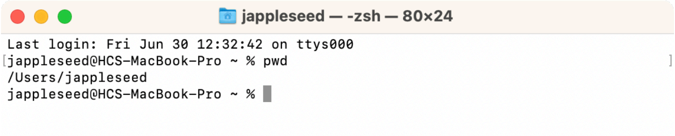
In this lesson we will briefly go over a few basic commands.

1. Launch Terminal from /Applications/Utilities.

2. The pwd command stands for Print Working Directory. This command will show you exactly where you are at when using the terminal. Enter the following command:

   `pwd`

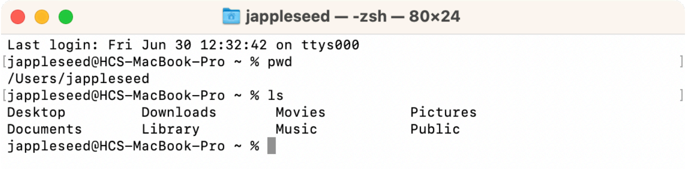
3. Press the return key. The output of the command will be listed in the terminal.



```
Last login: Fri Jun 30 12:32:42 on ttys000
jappleseed@HCS-MacBook-Pro ~ % pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ % 
```

4. The **`ls`** command is short for list. This command will list items. Enter the following command:

   `ls`

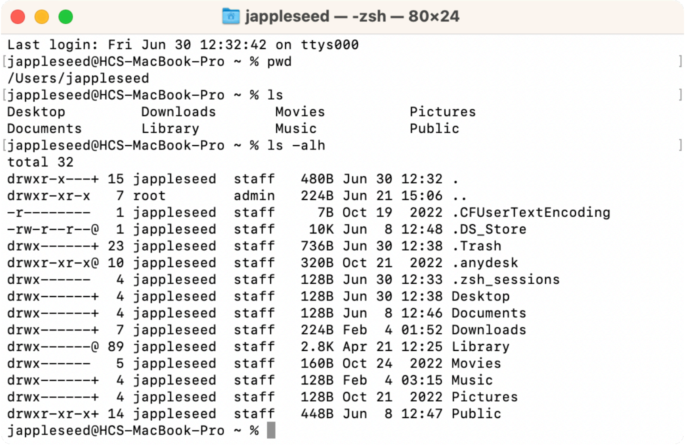


```
Last login: Fri Jun 30 12:32:42 on ttys000
jappleseed@HCS-MacBook-Pro ~ % pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ % ls
Desktop         Downloads       Movies          Pictures
Documents       Library         Music           Public
jappleseed@HCS-MacBook-Pro ~ % 
```

5. The **`ls`** command has a lot of options that can be assigned to it. In our last example, we saw a listing of the directory but not much else. Let's get a bit more info from the **`ls`** command by assigning the **`-alh`** option to it. Enter the following command:

   `ls -alh`

   The **`-alh`** option will show All Files, in Long Format and Human Readable. Notice you will have files in the list that show up with a dot in front of the name. These are hidden files.

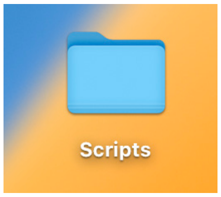


```
Last login: Fri Jun 30 12:32:42 on ttys000
jappleseed@HCS-MacBook-Pro ~ % pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ % ls
Desktop         Downloads       Movies          Pictures
Documents       Library         Music           Public
jappleseed@HCS-MacBook-Pro ~ % ls -alh
total 32
drwxr-x---+ 15 jappleseed  staff   480B Jun 30 12:32 .
drwxr-xr-x   7 root        admin   224B Jun 21 15:06 ..
-r--------   1 jappleseed  staff     7B Oct 19  2022 .CFUserTextEncoding
-rw-r--r--@  1 jappleseed  staff    10K Jun  8 12:48 .DS_Store
drwx------+ 23 jappleseed  staff   736B Jun 30 12:38 .Trash
drwxr-xr-x@ 10 jappleseed  staff   320B Oct 21  2022 .anydesk
drwx------   4 jappleseed  staff   128B Jun 30 12:33 .zsh_sessions
drwx------+  4 jappleseed  staff   128B Jun 30 12:38 Desktop
drwx------+  4 jappleseed  staff   128B Jun  8 12:46 Documents
drwx------+  7 jappleseed  staff   224B Feb  4 01:52 Downloads
drwx------@ 89 jappleseed  staff   2.8K Apr 21 12:25 Library
drwx------   5 jappleseed  staff   160B Oct 24  2022 Movies
drwx------+  4 jappleseed  staff   128B Feb  4 03:15 Music
drwx------+  4 jappleseed  staff   128B Oct 21  2022 Pictures
drwxr-xr-x+ 14 jappleseed  staff   448B Jun  8 12:47 Public
jappleseed@HCS-MacBook-Pro ~ % 
```

6. The **mkdir** command is short for make directory. This command will make a new directory. Enter the following command:

```
mkdir /Users/<yourAccount>/Desktop/Scripts
```

A folder named Scripts will appear on your Desktop.



7. The **rmdir** command is short for remove directory. This command will delete a directory. Enter the following command:

```
rmdir /Users/<yourAccount>/Desktop/Scripts
```

The folder named Scripts will be deleted from your Desktop.

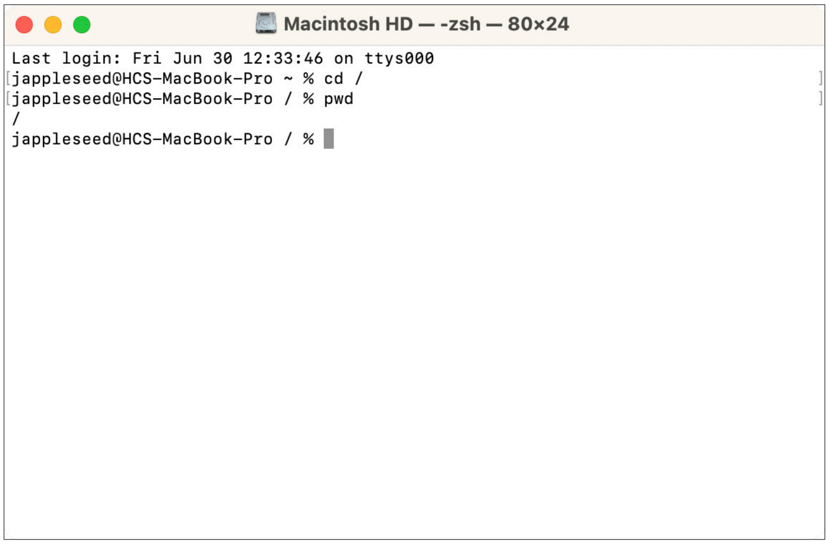
8. The **cd** command is short for Change Directory. This command will move you from one directory to another directory. Enter the following command:

```
cd /
```

Then enter:

```
pwd
```

You will be at the root of your hard drive.



```
Last login: Fri Jun 30 12:33:46 on ttys000
jappleseed@HCS-MacBook-Pro ~ % cd /
jappleseed@HCS-MacBook-Pro / % pwd
/
jappleseed@HCS-MacBook-Pro / %
```

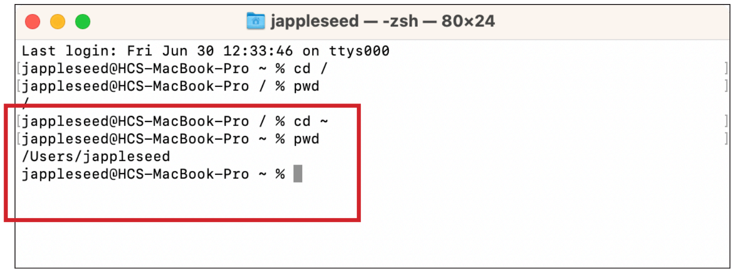
9. Enter the following command:

   ```
   cd ~
   ```

   Then Enter:

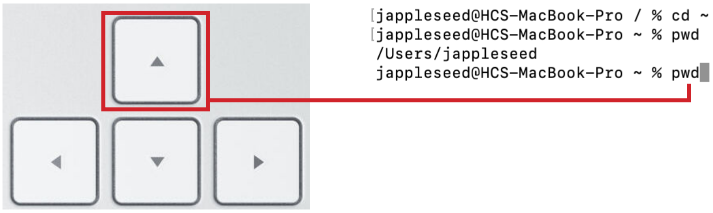
   ```
   pwd
   ```

   This will bring you to the root of your Home Directory. The Tilde (~) is used as a shortcut to a users home directory.



```
Last login: Fri Jun 30 12:33:46 on ttys000
jappleseed@HCS-MacBook-Pro ~ % cd /
jappleseed@HCS-MacBook-Pro / % pwd
/
jappleseed@HCS-MacBook-Pro / % cd ~
jappleseed@HCS-MacBook-Pro ~ % pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ %
```

10. There are a variety of ways to get to a command that was previously ran. The simplest way is to use the up arrow on the keyboard. Go ahead and press the up arrow once. This will show you the last command you ran. If you keep pressing the up arrow, you will see all your previous commands.

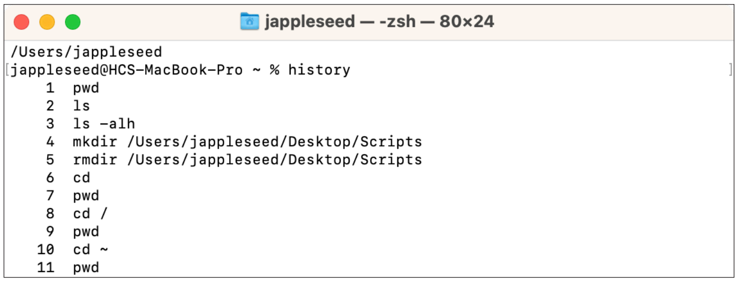


```
jappleseed@HCS-MacBook-Pro / % cd ~
jappleseed@HCS-MacBook-Pro ~ % pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ % pwd
```

11. Another way to see a command that was previously run, is to Enter the following command:

    ```
    history
    ```

    This will list out all of the commands that you have run on your computer. A number is assigned to each previous command. This number can be used to call the command as well. We will cover this in the next step.

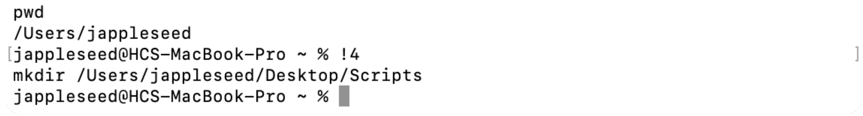


```
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ % history
    1  pwd
    2  ls
    3  ls -alh
    4  mkdir /Users/jappleseed/Desktop/Scripts
    5  rmdir /Users/jappleseed/Desktop/Scripts
    6  cd
    7  pwd
    8  cd /
    9  pwd
   10  cd ~
   11  pwd
```

12. An exclamation point (**!**), also referred to as a bang, can be used to recall commands from **history**. In our last example, the history command showed a number next to each command in your history. Enter the following command.

    !4
    (NOTE: your history number will be different. Pick a number of your choosing.)

    In our example, history item number 4 was the **mkdir** command.

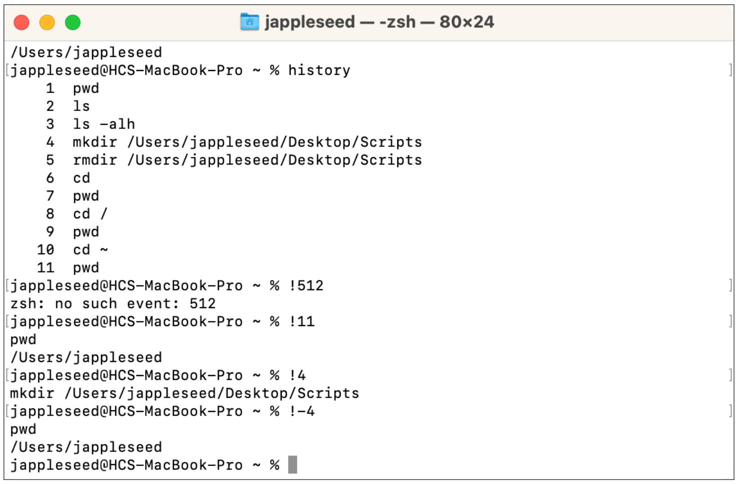


```
pwd
/Users/jappleseed
[jappleseed@HCS-MacBook-Pro ~ % !4
mkdir /Users/jappleseed/Desktop/Scripts
jappleseed@HCS-MacBook-Pro ~ %
```

13. This command will execute the command X numbers from the last command ran. Enter this:

    !-4

    This will run the 4th command from the last command you had in your history.

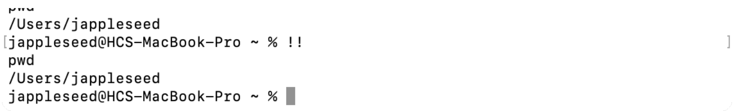


```
/Users/jappleseed
[jappleseed@HCS-MacBook-Pro ~ % history
    1  pwd
    2  ls
    3  ls -alh
    4  mkdir /Users/jappleseed/Desktop/Scripts
    5  rmdir /Users/jappleseed/Desktop/Scripts
    6  cd
    7  pwd
    8  cd /
    9  pwd
   10  cd ~
   11  pwd
[jappleseed@HCS-MacBook-Pro ~ % !512
zsh: no such event: 512
[jappleseed@HCS-MacBook-Pro ~ % !11
pwd
/Users/jappleseed
[jappleseed@HCS-MacBook-Pro ~ % !4
mkdir /Users/jappleseed/Desktop/Scripts
[jappleseed@HCS-MacBook-Pro ~ % !-4
pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ %
```

14. If you want to run the previous command, enter the following:

    !!

    A double exclamation point will run the previous command.

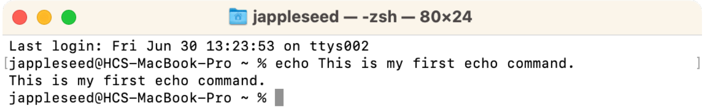


```
pwd
/Users/jappleseed
[jappleseed@HCS-MacBook-Pro ~ % !!
pwd
/Users/jappleseed
jappleseed@HCS-MacBook-Pro ~ %
```

15. The echo command is used to print items to the screen. Enter the following command:

    ```
    echo This is my first echo command.
    ```

    You will see "This is my first echo command" on the screen.



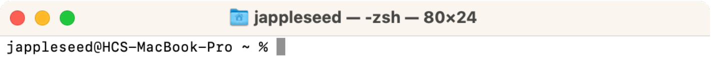
```
Last login: Fri Jun 30 13:23:53 on ttys002
[jappleseed@HCS-MacBook-Pro ~ % echo This is my first echo command.            ]
This is my first echo command.
jappleseed@HCS-MacBook-Pro ~ % █
```

16. The clear command is used to clear the screen. Enter:

    ```
    clear
    ```

    You will now have a clear terminal screen.

    NOTE: You can also clear the screen by using the keyboard shortcut: Command (⌘) +K

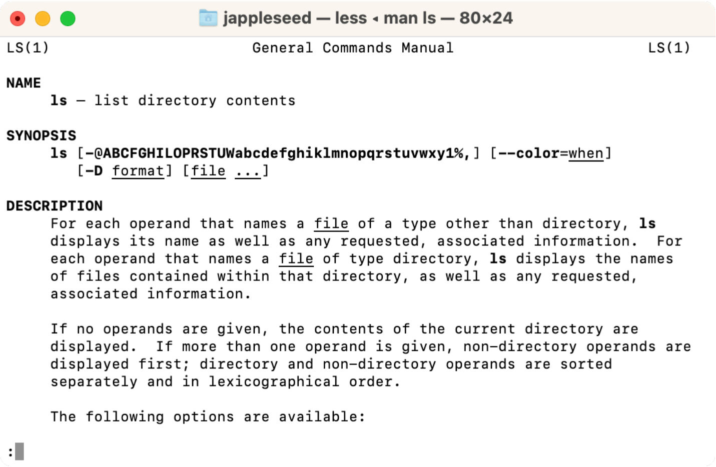


```
jappleseed@HCS-MacBook-Pro ~ % █
```

17. One of the most helpful commands is the **`man`** command. This command will show you the manual page for just about every program that you use in Terminal. Enter the following command:

    ```
    man ls
    ```

    This will show you the manual page for the **`ls`** command. I encourage you to view the man pages to get familiar with all the options for the commands you will use. Enter **`Q`** to quit the man page.



```
LS(1)                       General Commands Manual                      LS(1)

NAME
     ls – list directory contents

SYNOPSIS
     ls [-@ABCFGHILOPRSTUWabcdefghiklmnopqrstuvwxy1%,] [--color=when]
        [-D format] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names
     of files contained within that directory, as well as any requested,
     associated information.

     If no operands are given, the contents of the current directory are
     displayed.  If more than one operand is given, non-directory operands are
     displayed first; directory and non-directory operands are sorted
     separately and in lexicographical order.

     The following options are available:

:█
```

This completes this lesson.

Section 2: The Anatomy of a Shell Script

In this lesson we will go over the anatomy of creating a shell script. The following will be covered:

1. Create a folder for our scripts.
2. Choosing a text editor for writing shell scripts.
3. Shebang - The Interpreter directive.
4. Commenting your code.
5. Add code to your script
6. Saving your script.
7. Make your script executable.
8. Run your script.

1. If not already open, launch Terminal and enter the following command:

```
mkdir ~/Desktop/Zsh_Scripts
```

This will create a folder named Zsh_Scripts on your Desktop. We will store all of our scripts in this folder for the duration of this guide.

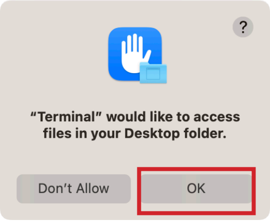
2. There are a myriad of text editors available for creating shell scripts. The most important thing to remember is to use a plain text editor and NOT a rich text editor when creating your shell scripts. Rich Text editors can cause your script to not function properly as it can interpret characters differently from plain text. We will use the nano editor built into macOS for this guide. Navigate to the Zsh_Scripts folder on the Desktop, and create your first shell script. Enter the following command:

```
cd ~/Desktop/Zsh_Scripts
```

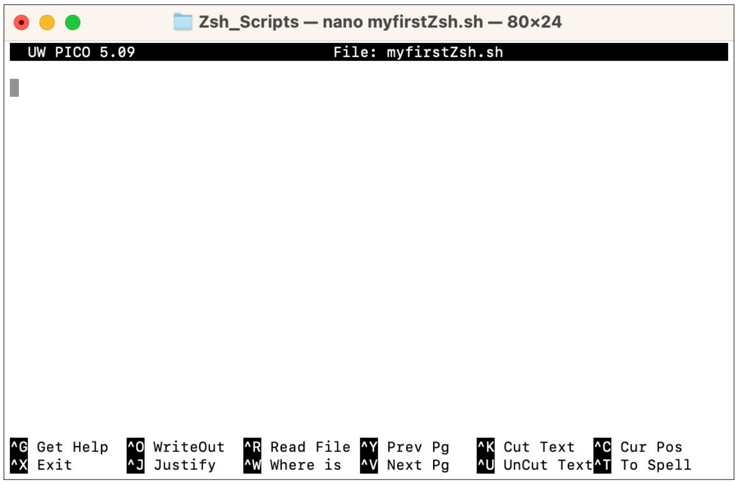
Then Enter:

```
nano myfirstZsh.sh
```

This will launch the nano editor.


“Terminal” would like to access files in your Desktop folder.

NOTE:
A prompt may appear asking you to grant Terminal to access your Desktop. Click OK.

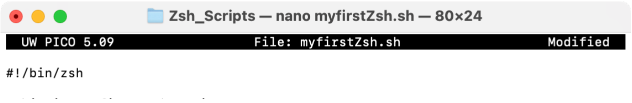


```
Zsh_Scripts — nano myfirstZsh.sh — 80×24
  UW PICO 5.09                         File: myfirstZsh.sh

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

3. The myfirstZsh.sh will be empty. A shell script requires an interpreter directive. The interpreter must be an absolute path to an executable program. Enter the following on the first line of the myfirstZsh.sh:

```
#!/bin/zsh
```

This line is known as the shebang. This will tell the interpreter to run the Zsh program located in /bin/sh.

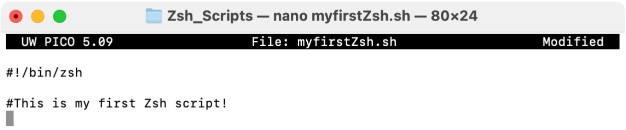


```
  UW PICO 5.09                 File: myfirstZsh.sh                  Modified

#!/bin/zsh
```

4. A very important step in writing shell scripts is to comment the scripts. This is very helpful if you need to revise the script down the road. A hashtag (#) is used for commenting in shell scripts. Hashtags are ignored when a shell script is run. Go back to the myfirstZsh.sh file and press the return key twice after the `#!/bin/zsh` and enter the following:

```
#This is my first Zsh script!
```

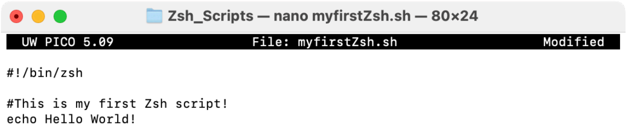
You just created your first comment.



```
  UW PICO 5.09                 File: myfirstZsh.sh                  Modified

#!/bin/zsh

#This is my first Zsh script!
```

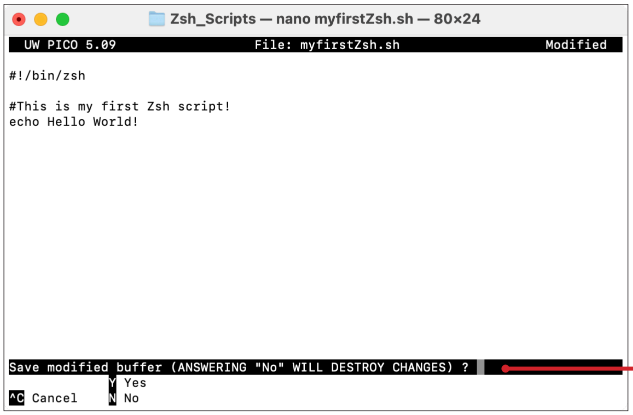
5. Go back to the myfirstZsh.sh file and press the return key once after the `#This is my first Zsh script!` and enter the following:

```
echo Hello World!
```



```
  UW PICO 5.09                 File: myfirstZsh.sh                  Modified

#!/bin/zsh

#This is my first Zsh script!
echo Hello World!
```

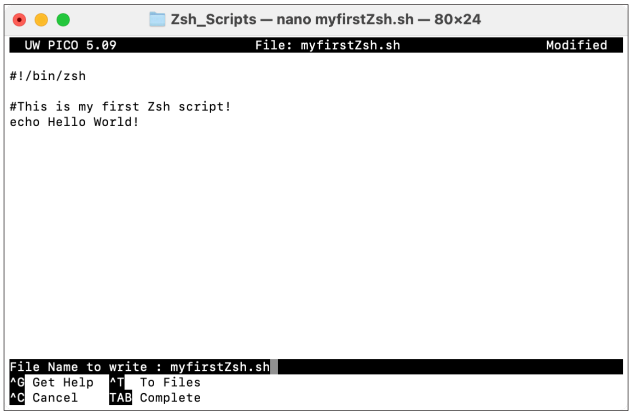
6. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                 File: myfirstZsh.sh                  Modified

#!/bin/zsh

#This is my first Zsh script!
echo Hello World!










Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

Enter Y for yes

7. To finish saving the file, press the Return key.

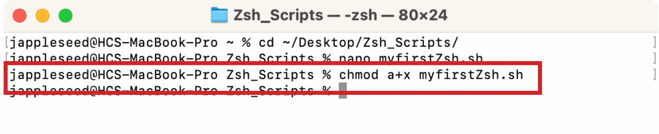


Zsh_Scripts — nano myfirstZsh.sh — 80×24

```
  UW PICO 5.09                  File: myfirstZsh.sh                   Modified

#!/bin/zsh

#This is my first Zsh script!
echo Hello World!








File Name to write : myfirstZsh.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

8. To run a shell script, it must be executable. Enter the following command to make the shell script executable for all users:

```
chmod a+x myfirstZsh.sh
```



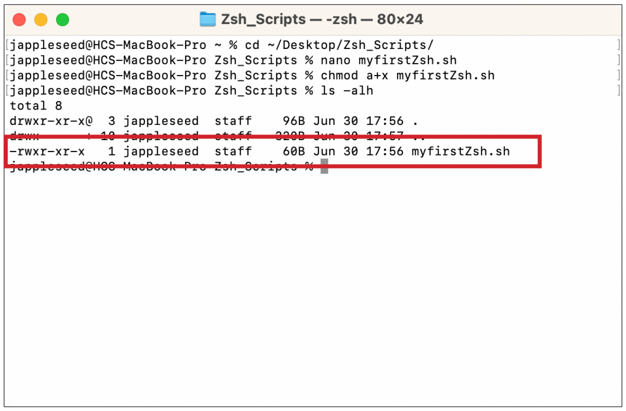
Zsh_Scripts — -zsh — 80×24

```
jappleseed@HCS-MacBook-Pro ~ % cd ~/Desktop/Zsh_Scripts/
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

9. To confirm the script is executable. Enter the following command:

```
ls -alh
```

The **x** shown is the file permissions and confirms that the file is executable.



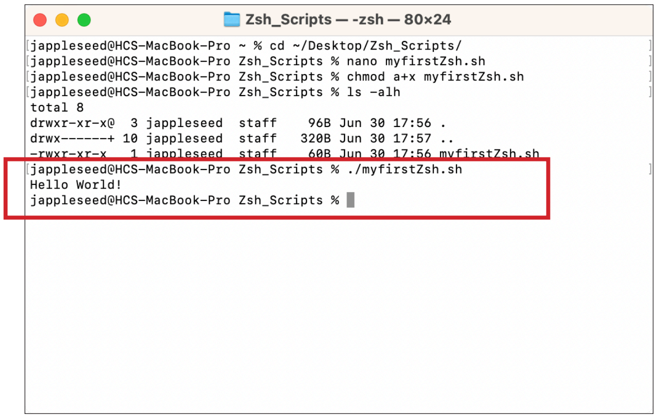
Zsh_Scripts — -zsh — 80×24

```
jappleseed@HCS-MacBook-Pro ~ % cd ~/Desktop/Zsh_Scripts/
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls -alh
total 8
drwxr-xr-x@  3 jappleseed  staff    96B Jun 30 17:56 .
drwx------+ 10 jappleseed  staff   320B Jun 30 17:57 ..
-rwxr-xr-x   1 jappleseed  staff    60B Jun 30 17:56 myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

10. There are many ways to run a script. Since we are already in the directory that the script resides in, enter the following command to run the script:

    ```
    ./myfirstZsh.sh
    ```

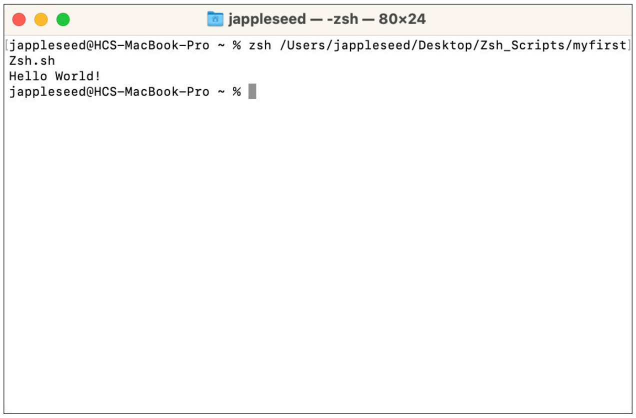
    Hello World! will be echoed to the screen. Congratulations, you just created and ran your first shell script.



```
jappleseed@HCS-MacBook-Pro ~ % cd ~/Desktop/Zsh_Scripts/
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls -alh
total 8
drwxr-xr-x@  3 jappleseed  staff    96B Jun 30 17:56 .
drwx------+ 10 jappleseed  staff   320B Jun 30 17:57 ..
-rwxr-xr-x   1 jappleseed  staff    60B Jun 30 17:56 myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./myfirstZsh.sh
Hello World!
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

NOTE: If you were not already in the location where the script resides, You would need to call the script by specifying it's absolute path with zsh prepending the command.

```
zsh /Users/< HomeFolder >/Desktop/Zsh_Scripts/myfirstZsh.sh
```



```
jappleseed@HCS-MacBook-Pro ~ % zsh /Users/jappleseed/Desktop/Zsh_Scripts/myfirst
Zsh.sh
Hello World!
jappleseed@HCS-MacBook-Pro ~ % 
```

This completes this lesson.

Section 3: Shell Expansion

In this lesson we will go over shell expansion. There are 8 expansions in total:

1. Brace Expansion.
2. Tilde Expansion.
3. Shell parameter and variable Expansion.
4. Command Substitution.
5. Arithmetic Expansion.
6. Process Substitution.
7. Word splitting.
8. File name Expansion.

For more information on the items above, go to:

https://zsh.sourceforge.io/Doc/Release/Expansion.html

We will cover the following in this lesson:

1. Brace Expansion.
2. Shell parameter and variable Expansion.
3. Command Substitution.

Brace expansion is a mechanism by which arbitrary strings may be generated. Patterns to be brace-expanded take the form of an optional PREAMBLE, followed by a series of comma-separated strings between a pair of braces, followed by an optional POSTSCRIPT. The preamble is prefixed to each string contained within the braces, and the postscript is then appended to each resulting string, expanding left to right. Brace expansions may be nested. The results of each expanded string are not sorted; left to right order is preserved.

Shell parameter and variable Expansion

The "$" character introduces parameter expansion, command substitution, or arithmetic expansion. The parameter name or symbol to be expanded may be enclosed in braces, which are optional but serve to protect the variable to be expanded from characters immediately following it which could be interpreted as part of the name.

When braces are used, the matching ending brace is the first "}" not escaped by a backslash or within a quoted string, and not within an embedded arithmetic expansion, command substitution, or parameter expansion. The basic form of parameter expansion is "${PARAMETER}". The value of "PARAMETER" is substituted. The braces are required when "PARAMETER" is a positional parameter with more than one digit, or when "PARAMETER" is followed by a character that is not to be interpreted as part of its name.

If the first character of "PARAMETER" is an exclamation point, Zsh uses the value of the variable formed from the rest of "PARAMETER" as the name of the variable; this variable is then expanded and that value is used in the rest of the substitution, rather than the value of "PARAMETER" itself. This is known as indirect expansion.

Command substitution allows the output of a command to replace the command itself. Command substitution occurs when a command is enclosed in parenthesis or back ticks. Zsh performs the expansion by executing COMMAND and replacing the command substitution with the standard output of the command, with any trailing newlines deleted. Embedded newlines are not deleted, but they may be removed during word splitting. When the old-style back quoted form of substitution is used, backslash retains its literal meaning except when followed by "$", "`", or "\". The first back ticks not preceded by a backslash terminates the command substitution. When using the "$ (COMMAND)" form, all characters between the parentheses make up the command; none are treated specially.

Command substitutions may be nested. To nest when using the back quoted form, escape the inner back ticks with backslashes. If the substitution appears within double quotes, word splitting and file name expansion are not performed on the results.

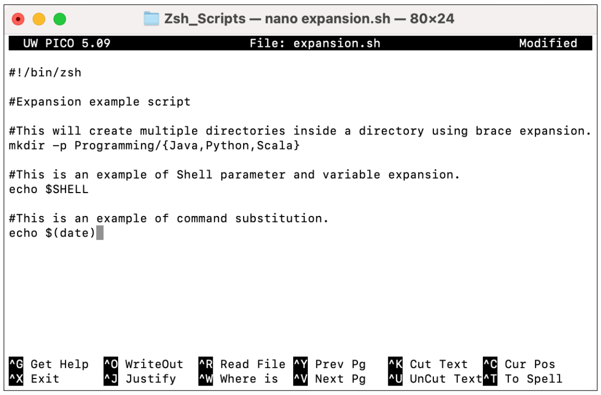
1. If not already open, launch terminal and navigate to the Zsh_Scripts folder.

   `cd ~/Desktop/Zsh_Scripts/`

2. Enter the following command:

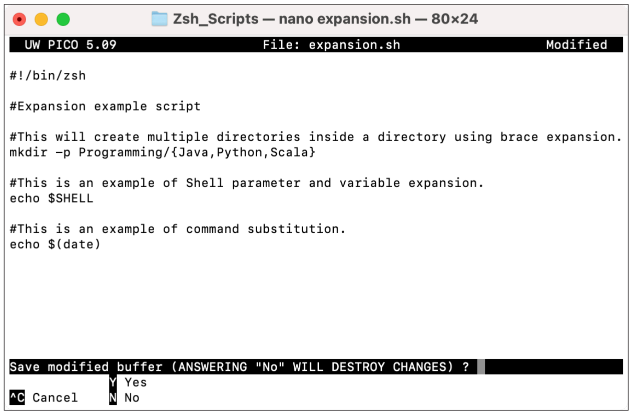
   `nano expansion.sh`

3. A blank document will open. Enter the following:
   Enter on the first line: **`#!/bin/zsh`**
   Press return key twice, enter: **`#Expansion example script`**
   Press return key twice, enter:
   `#This will create multiple directories inside a directory using brace expansion.`
   Press return key once, enter: **`mkdir -p Programming/{Java,Python,Scala}`**
   Press return key twice, enter:
   `#This is an example of Shell parameter and variable expansion.`
   Press return key once, enter: **`echo  $SHELL`**
   Press return key twice, enter: **`#This is an example of command substitution.`**
   Press return key once, enter: **`echo $(date)`**



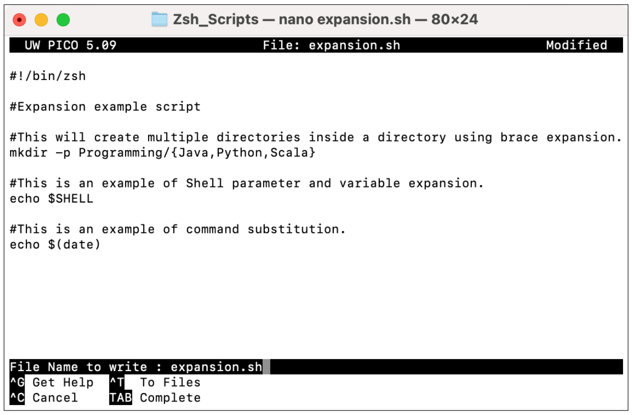
```
  UW PICO 5.09                    File: expansion.sh                    Modified

#!/bin/zsh

#Expansion example script

#This will create multiple directories inside a directory using brace expansion.
mkdir -p Programming/{Java,Python,Scala}

#This is an example of Shell parameter and variable expansion.
echo $SHELL

#This is an example of command substitution.
echo $(date)
```

4. To save the script, press the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                    File: expansion.sh                    Modified

#!/bin/zsh

#Expansion example script

#This will create multiple directories inside a directory using brace expansion.
mkdir -p Programming/{Java,Python,Scala}

#This is an example of Shell parameter and variable expansion.
echo $SHELL

#This is an example of command substitution.
echo $(date)

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
```

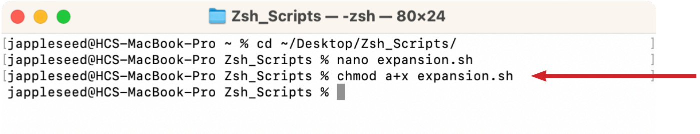
5. To finish saving the file, press the Return key.



```
  UW PICO 5.09                   File: expansion.sh                   Modified

#!/bin/zsh

#Expansion example script

#This will create multiple directories inside a directory using brace expansion.
mkdir -p Programming/{Java,Python,Scala}

#This is an example of Shell parameter and variable expansion.
echo $SHELL

#This is an example of command substitution.
echo $(date)







File Name to write : expansion.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

6. To run a shell script, it must be executable. Enter the following command to make the shell script executable for all users:

```
chmod a+x expansion.sh
```

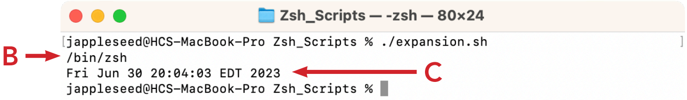


```
jappleseed@HCS-MacBook-Pro ~ % cd ~/Desktop/Zsh_Scripts/
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano expansion.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x expansion.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

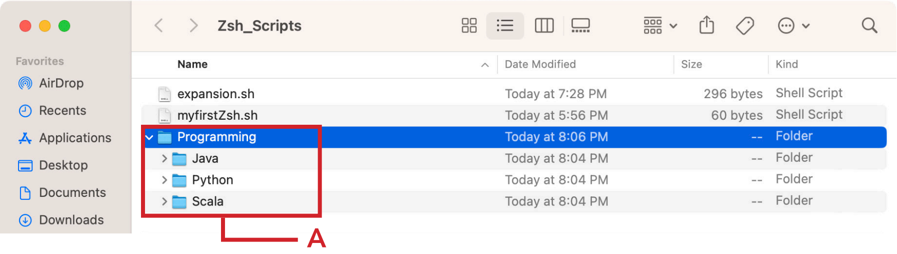
7. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./expansion.sh
```

8. Confirm the following:
   A. Inside the Zsh_Scripts folder, there is a directory named Programming. There are folders named Java, Python, and Scala inside the Programming folder. This was done with brace expansion.
   B. The terminal shows **/bin/zsh** - This was done with Shell parameter and variable Expansion.
   C. The terminal shows the current date and time. This was done with Command Substitution.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./expansion.sh
/bin/zsh
Fri Jun 30 20:04:03 EDT 2023
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```



Name	Date Modified	Size	Kind
expansion.sh	Today at 7:28 PM	296 bytes	Shell Script
myfirstZsh.sh	Today at 5:56 PM	60 bytes	Shell Script
Programming	Today at 8:06 PM	--	Folder
Java	Today at 8:04 PM	--	Folder
Python	Today at 8:04 PM	--	Folder
Scala	Today at 8:04 PM	--	Folder

9. Enter the following command to delete the Programing folder and it's contents:

```
rm -r Programming
```

This completes this lesson.

Section 4: Input/Output Redirection

In this lesson we will go over redirection.

What is redirection?

Redirection simply means capturing output from a file, command, program, script, or even code block within a script and sending it as input to another file, command, program, or script.

There are always three default files that are open.

- stdin (the keyboard)
- stdout (the screen)
- stderr (error messages output to the screen).

The files contain numbers known as file descriptors. You can call them by their file descriptor numbers when redirecting them. For example: 2>&1 can be used in a script to redirect stderr and stdout to a log file.

- stdin - File Descriptor Number 0
- stdout - File Descriptor Number 1
- stderr - File Descriptor Number 2

These, and any other open files, can be redirected. We will cover the following in this lesson:

1. Redirecting Output
2. Appending redirected output
3. Redirecting error output
4. Appending Redirected error output
5. Transporting stdout and stderr through a pipe
6. Redirecting input
7. Here Documents

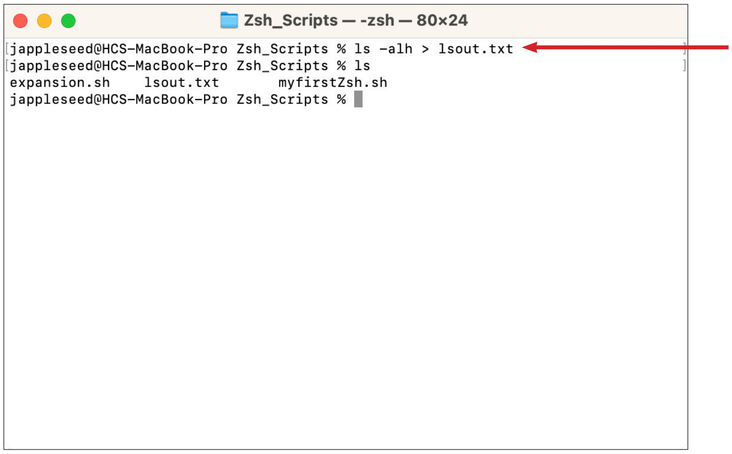
1. If not already open, launch terminal and navigate to the Zsh_Scripts folder.

```
cd ~/Desktop/Zsh_Scripts/
```

2. Let's redirect some output to a file. You use the greater than sign (>) to redirect data to a file. Enter the following command:

```
ls -alh > lsout.txt
```

3. Enter the ls command. You will see a file called lsout.txt.

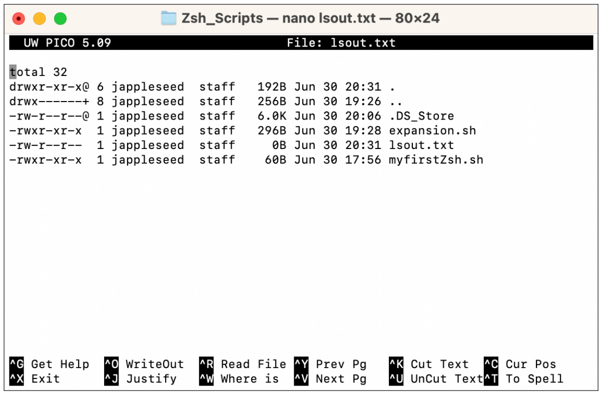


```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls -alh > lsout.txt
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls
expansion.sh    lsout.txt       myfirstZsh.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

4. To view the lsout.txt file with the nano editor. Enter the following command:

   `nano lsout.txt`

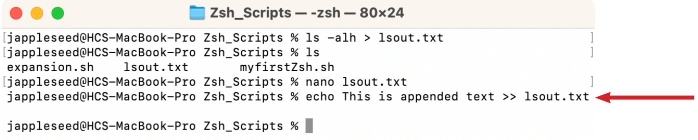
   This file contains the output of the ls -alh command. Exit out of the lsout.txt file by pressing the Control and x keys.



```
  UW PICO 5.09                          File: lsout.txt

total 32
drwxr-xr-x@ 6 jappleseed  staff   192B Jun 30 20:31 .
drwx------+ 8 jappleseed  staff   256B Jun 30 19:26 ..
-rw-r--r--@ 1 jappleseed  staff   6.0K Jun 30 20:06 .DS_Store
-rwxr-xr-x  1 jappleseed  staff   296B Jun 30 19:28 expansion.sh
-rw-r--r--  1 jappleseed  staff     0B Jun 30 20:31 lsout.txt
-rwxr-xr-x  1 jappleseed  staff    60B Jun 30 17:56 myfirstZsh.sh

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

5. Lets append some more data to the **`lsout.txt`** file. We will use the double-greater than sign which instruct the data to be appended to a file. Enter the following command.

   `echo This is appended text >> lsout.txt`

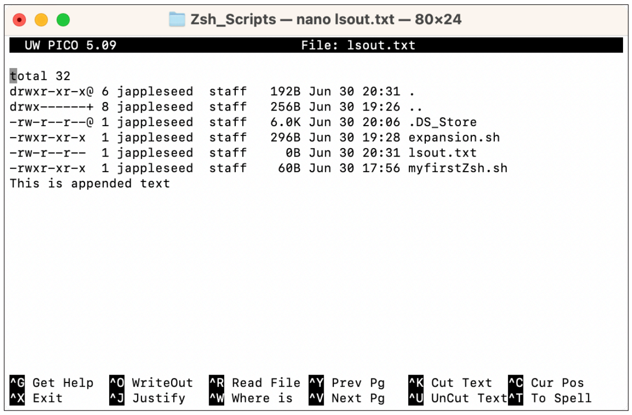


```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls -alh > lsout.txt                  ]
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ls                                   ]
expansion.sh    lsout.txt       myfirstZsh.sh
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano lsout.txt                       ]
jappleseed@HCS-MacBook-Pro Zsh_Scripts % echo This is appended text >> lsout.txt

jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

6. Lets view the lsout.txt file with the nano editor. Enter the following command:

   `nano lsout.txt`

   The appended text is now part of this file. Exit out of the lsout.txt file by pressing the Control and x keys.



```
  UW PICO 5.09                          File: lsout.txt

total 32
drwxr-xr-x@ 6 jappleseed  staff   192B Jun 30 20:31 .
drwx------+ 8 jappleseed  staff   256B Jun 30 19:26 ..
-rw-r--r--@ 1 jappleseed  staff   6.0K Jun 30 20:06 .DS_Store
-rwxr-xr-x  1 jappleseed  staff   296B Jun 30 19:28 expansion.sh
-rw-r--r--  1 jappleseed  staff     0B Jun 30 20:31 lsout.txt
-rwxr-xr-x  1 jappleseed  staff    60B Jun 30 17:56 myfirstZsh.sh
This is appended text

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

7. There are a few ways to handle stdout and stderr. The following commands will produce different results. Enter the following commands:

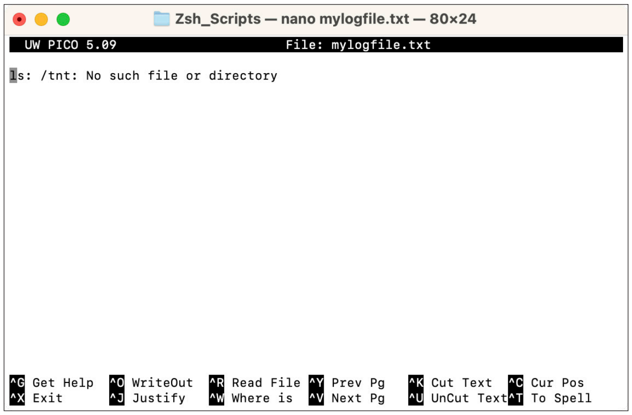
```
ls -alh /tnt > mylogfile.txt 2>&1
```

This command will send the stdout and stderr output to a file called mylogfile.txt.

8. To view the mylogfile.txt file with the nano editor. Enter the following command:

```
nano mylogfile.txt
```

An error message shows up. Exit out of the mylogfile.txt file by pressing the Control and x keys.



Zsh_Scripts — nano mylogfile.txt — 80×24

```
  UW PICO 5.09                       File: mylogfile.txt

ls: /tnt: No such file or directory
```

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit   ^J Justify   ^W Where is   ^V Next Pg   ^U UnCut Text   ^T To Spell

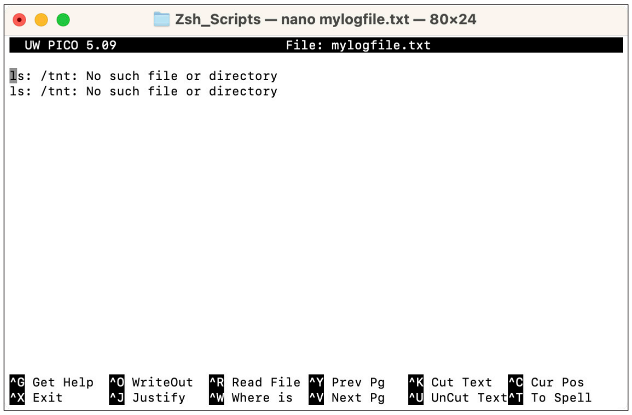
9. This command will append the stdout and stderr output to a file called mylogfile.txt.

```
ls -alh /tnt >> mylogfile.txt 2>&1
```

10. Lets view the mylogfile.txt file with the nano editor. Enter the following command:

```
nano mylogfile.txt
```

An error message shows up with the appended text. Exit out of the mylogfile.txt file by pressing the Control and x keys.



Zsh_Scripts — nano mylogfile.txt — 80×24

```
  UW PICO 5.09                       File: mylogfile.txt

ls: /tnt: No such file or directory
ls: /tnt: No such file or directory
```

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit   ^J Justify   ^W Where is   ^V Next Pg   ^U UnCut Text   ^T To Spell

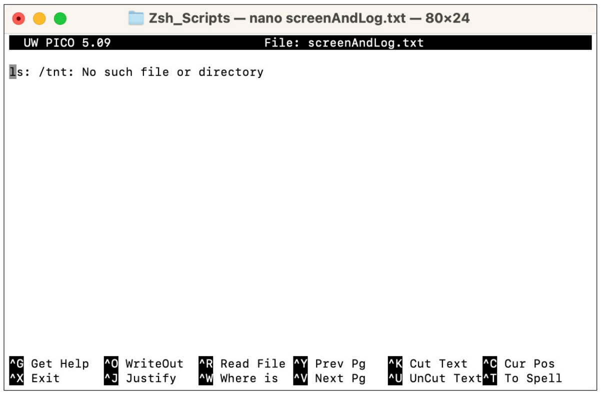
11. This command will send stdout and stderr to both the screen and to a file using a pipe for redirection and the tee command.

```
ls -alh /tnt 2>&1 | tee screenAndLog.txt
```

12. To view the screenAndLog.txt file with the nano editor. Enter the following command:

```
nano screenAndLog.txt
```

An error message shows up. Exit out of the screenAndLog.txt file by pressing the Control and x keys.



```
  UW PICO 5.09                    File: screenAndLog.txt

ls: /tnt: No such file or directory
```

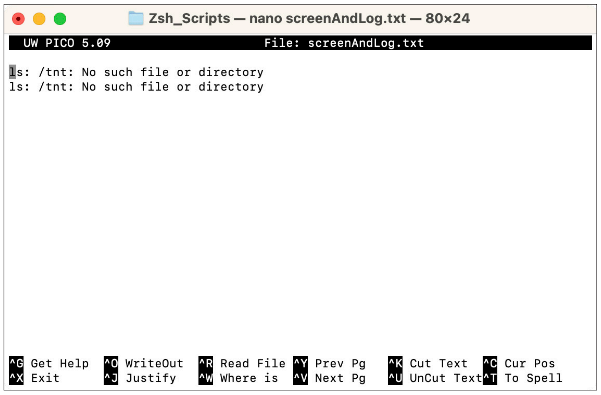
13. This command will append stdout and stderr to both the screen and to a file using a pipe for redirection and the tee command.

```
ls -alh /tnt 2>&1 | tee >> screenAndLog.txt
```

14. To view the screenAndLog.txt file with the nano editor. Enter the following command:

```
nano screenAndLog.txt
```

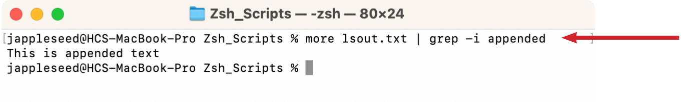
An error message shows up. Exit out of the screenAndLog.txt file by pressing the Control and x keys.



```
  UW PICO 5.09                    File: screenAndLog.txt

ls: /tnt: No such file or directory
ls: /tnt: No such file or directory
```

15. To redirect standard input, use the more command to read the lsout.txt file as input and pipe it to the grep command and search for all lines that contain the word "appended" in the document. Enter the following command: (NOTE: the -i in the grep command will make the search ignore the case)

```
more lsout.txt | grep -i appended
```



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % more lsout.txt | grep -i appended
This is appended text
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

Here Documents

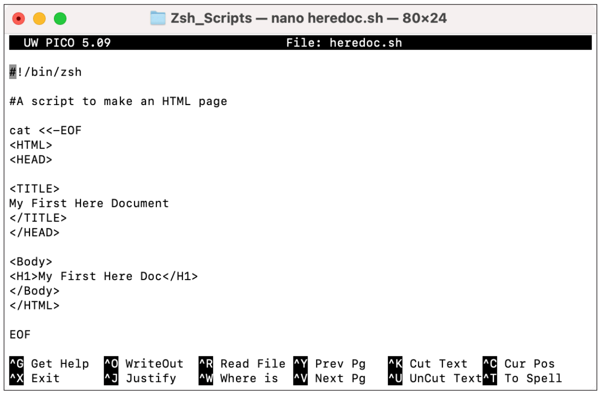
<<- EOF is a syntax used in a here-document. It uses I/O redirection to feed a list of commands to an interactive program until reaching the specified delimiter, which is EOF (end-of-file). Heredoc statements don't have to use EOF, you can use anything you want as long as it starts with <<-HCS and ends with HCS. We will use the cat command in this next step. The cat utility reads files sequentially, writing them to standard output. For more info about the cat command, read the man page on it.

16. Enter the following command:

```
nano heredoc.sh
```

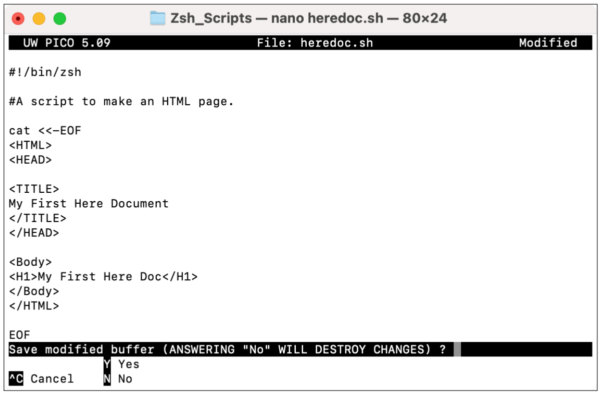
17. Enter the following info into the heredoc.sh document:

```
#!/bin/zsh

#A script to make an HTML page

cat <<-EOF
<HTML>
<HEAD>
        <TITLE>
My First Here Document
</TITLE>
</HEAD>

<Body>
<H1>My First Here Document</H1>
</Body>
</HTML>
EOF
```



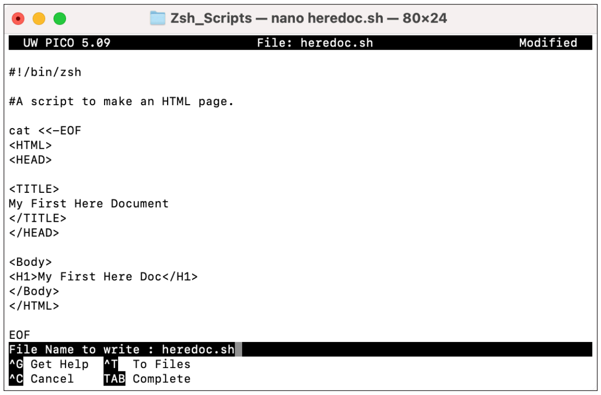
```
  UW PICO 5.09                                    File: heredoc.sh

#!/bin/zsh

#A script to make an HTML page

cat <<-EOF
<HTML>
<HEAD>

<TITLE>
My First Here Document
</TITLE>
</HEAD>

<Body>
<H1>My First Here Doc</H1>
</Body>
</HTML>

EOF

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

18. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                    File: heredoc.sh                    Modified

#!/bin/zsh

#A script to make an HTML page.

cat <<-EOF
<HTML>
<HEAD>

<TITLE>
My First Here Document
</TITLE>
</HEAD>

<Body>
<H1>My First Here Doc</H1>
</Body>
</HTML>

EOF
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

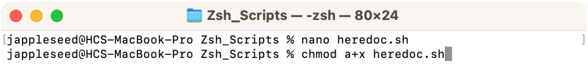
19. To finish saving the file, press the Return key.



```
  UW PICO 5.09                    File: heredoc.sh                    Modified

#!/bin/zsh

#A script to make an HTML page.

cat <<-EOF
<HTML>
<HEAD>

<TITLE>
My First Here Document
</TITLE>
</HEAD>

<Body>
<H1>My First Here Doc</H1>
</Body>
</HTML>

EOF
File Name to write : heredoc.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

20. To run a shell script, it must be executable. Enter the following command to make the shell script executable for all users:

```
chmod a+x heredoc.sh
```

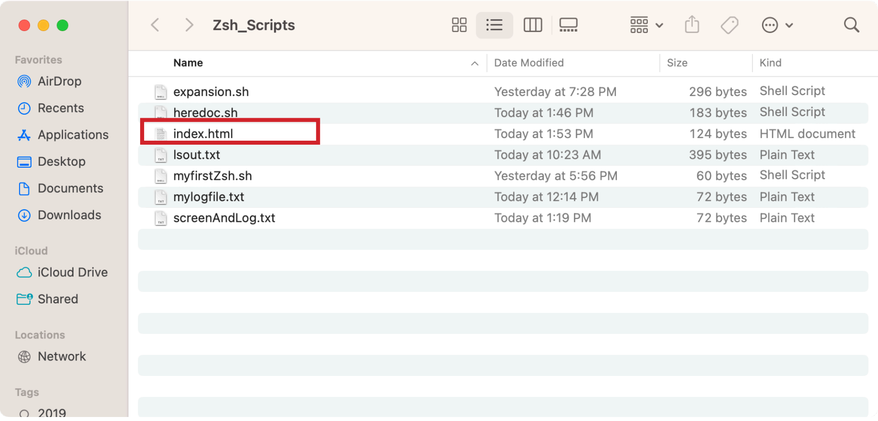


```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano heredoc.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x heredoc.sh
```

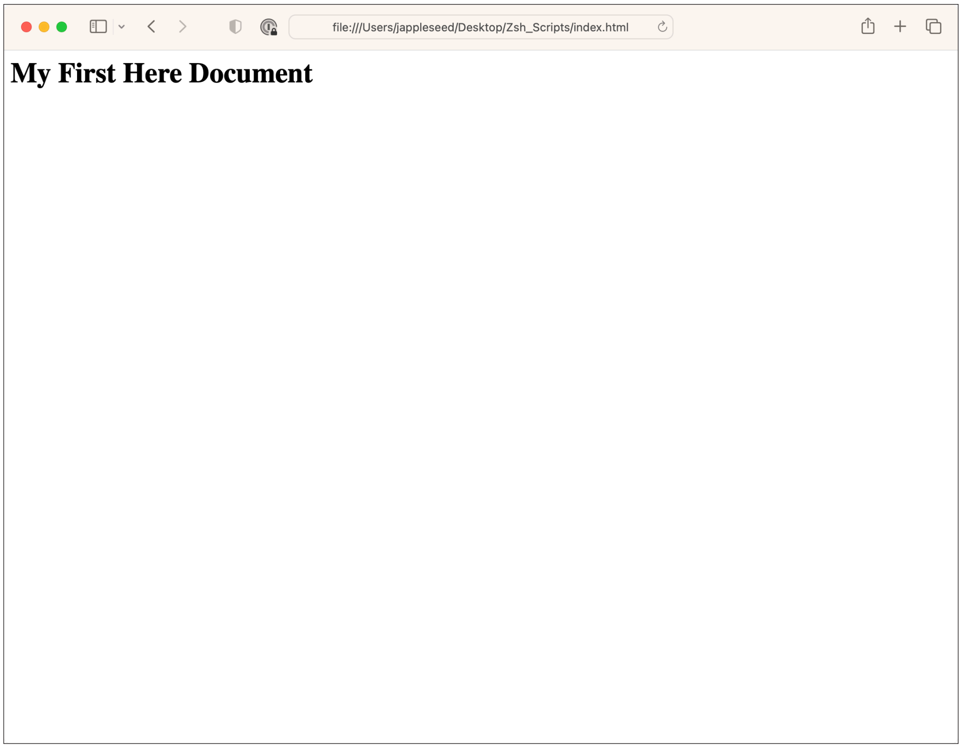
21. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./heredoc.sh > index.html
```

22. Navigate to your Zsh_Scripts folder located on your Desktop and open the index.html file. Confirm that it opens in a web browser and displays "My First Here Document".



Name	Date Modified	Size	Kind
expansion.sh	Yesterday at 7:28 PM	296 bytes	Shell Script
heredoc.sh	Today at 1:46 PM	183 bytes	Shell Script
index.html	Today at 1:53 PM	124 bytes	HTML document
lsout.txt	Today at 10:23 AM	395 bytes	Plain Text
myfirstZsh.sh	Yesterday at 5:56 PM	60 bytes	Shell Script
mylogfile.txt	Today at 12:14 PM	72 bytes	Plain Text
screenAndLog.txt	Today at 1:19 PM	72 bytes	Plain Text



My First Here Document

This completes this lesson.

Section 5: Variables

In this lesson, we will cover two types of variables. Internal variables and General Assignment variables.

Internal or Built in variables, are variables that are set and used by Zsh. Other shells may ignore these variables as they are NOT built into the shell.

General Assignment or Custom variables are variables that you can create when needed. These variables are not built into Zsh and should be respected by most shells.

Variables are case sensitive and capitalized by default. Giving local variables a lowercase name is a convention which is sometimes applied. However, you are free to use the names you want or to mix cases. Variables can also contain digits, but a name starting with a digit is not allowed.

We will cover the following in this lesson:

- Internal Variables
- General Assignment Variables

1. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
echo $HOME
```

This is an Internal variable meaning you did not set the information in it. When the echo $HOME command is run, it tells Zsh to look at it's internal variable for $HOME and display the output which is the home directory of the current logged in user.

```
/Users/< HomeFolder >
```

2. Enter the following commands one by one to see the internal variables output.

`echo $PWD` - This will print out the current working directory to the screen.

```
/Users/< HomeFolder >/Desktop/Zsh_Scripts
```

`echo $MACHTYPE` - This will output the Machine info.

```
x86_64
```

`echo $HOST` - This will output the hostname of the computer.

```
HCS-MacBook-Pro.local
```

`echo $SECONDS` – This will output the number of seconds a script has been running.

```
16824
```

`echo $ZSH_VERSION` - This will output the version of Zsh on your computer.

```
5.9
```

`echo $UID` - This will output the UID of the currently logged in user.

```
501
```

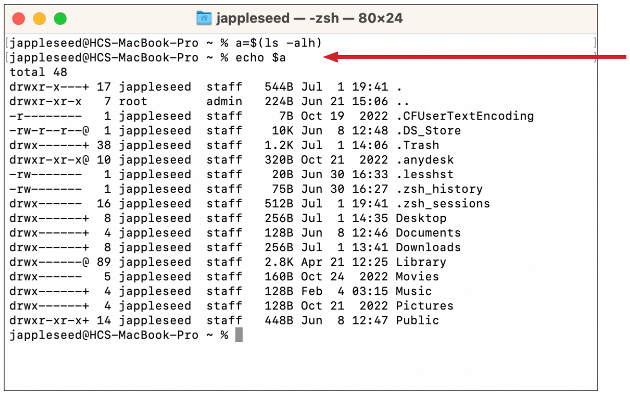
3. Now lets have a look at some General Assignment or Custom variables. Enter the following commands one by one then use echo to see the variable output. These are simple case variables:

   Enter this in Terminal and press enter: **`a=50`**
   Then enter: **`echo $a`**
   The result of the echo will be **`50`**

   Enter this in the terminal and press enter: **`b=$a`**
   Then enter: **`echo $b`**
   The result of the echo will be **`50`**

4. These are command substitution variables:

   Enter this in the terminal and press enter: **`a=$(ls -alh)`**
   Then enter: **`echo $a`**



```
jappleseed@HCS-MacBook-Pro ~ % a=$(ls -alh)
jappleseed@HCS-MacBook-Pro ~ % echo $a
total 48
drwxr-x---+ 17 jappleseed  staff   544B Jul  1 19:41 .
drwxr-xr-x   7 root        admin   224B Jun 21 15:06 ..
-r--------   1 jappleseed  staff     7B Oct 19  2022 .CFUserTextEncoding
-rw-r--r--@  1 jappleseed  staff    10K Jun  8 12:48 .DS_Store
drwx------+ 38 jappleseed  staff   1.2K Jul  1 14:06 .Trash
drwxr-xr-x@ 10 jappleseed  staff   320B Oct 21  2022 .anydesk
-rw-------   1 jappleseed  staff    20B Jun 30 16:33 .lesshst
-rw-------   1 jappleseed  staff    75B Jun 30 16:27 .zsh_history
drwx------  16 jappleseed  staff   512B Jul  1 19:41 .zsh_sessions
drwx------+  8 jappleseed  staff   256B Jul  1 14:35 Desktop
drwx------+  4 jappleseed  staff   128B Jun  8 12:46 Documents
drwx------+  8 jappleseed  staff   256B Jul  1 13:41 Downloads
drwx------@ 89 jappleseed  staff   2.8K Apr 21 12:25 Library
drwx------   5 jappleseed  staff   160B Oct 24  2022 Movies
drwx------+  4 jappleseed  staff   128B Feb  4 03:15 Music
drwx------+  4 jappleseed  staff   128B Oct 21  2022 Pictures
drwxr-xr-x+ 14 jappleseed  staff   448B Jun  8 12:47 Public
jappleseed@HCS-MacBook-Pro ~ %
```

Variables are very useful when creating shell scripts. You can setup your variables at the beginning of a script and call them by variable name later on in your script. This makes changing a variable in a script very easy. Just change it in one place and it will change it everywhere it's called in your script.

This completes this lesson.

Section 6: Working with Numbers

We will cover the following in this lesson:

- Arithmetic Evaluation
- Floating Point Numbers

Zsh has an advantage over bash as it works with floating point numbers without the need to pipe the output to another command like bc.

1. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
echo $((2+2))
```

The output will be **4**. This is called Arithmetic Expansion. Expressions must be inside double parenthesis. Bash does not support floating point numbers so the output must be sent to the bc program. bc is a language that supports arbitrary precision numbers with interactive executive statements. It's built into most *nix system.

2. Let's work with floating point numbers. Enter the following command:

```
echo $((2.1+2.1))
```

The result is: **4.2000000000000002**
Zsh handles the floating point number with no issue, however what if we wanted to trim off all of the extra numbers after the first 2.

3. We can pipe the output of the echo command to the input of the bc command to get our desired result. Enter the following command:

```
echo 2.1 + 2.1 | bc -l
```

The output will be **4.2**

This completes this lesson.

Section 7: Loops

In this lesson we will cover the following loop types:

- For Loops
- Until Loops
- While Loops

Loop Types explained

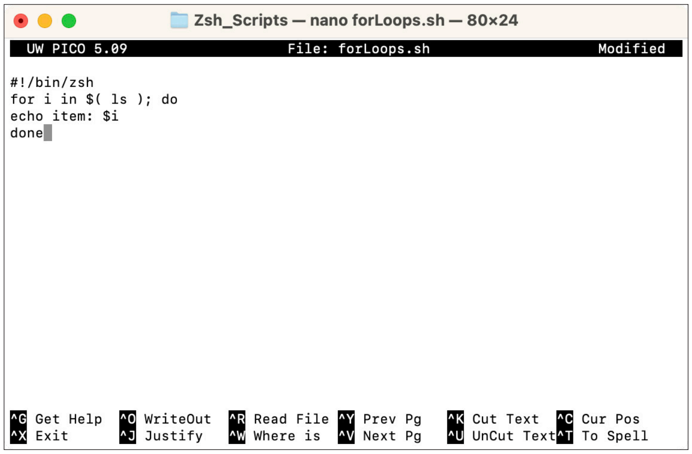
The for loop is a little bit different from other programming languages. Basically, it let's you iterate over a series of words within a string. The while executes a piece of code if the control expression is true, and only stops when it is false (or a explicit break is found within the executed code.) The until loop is almost equal to the while loop, except that the code is executed while the control expression evaluates to false. If you suspect that while and until are very similar you are right.

1. Lets create a for loop. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
nano forLoops.sh
```

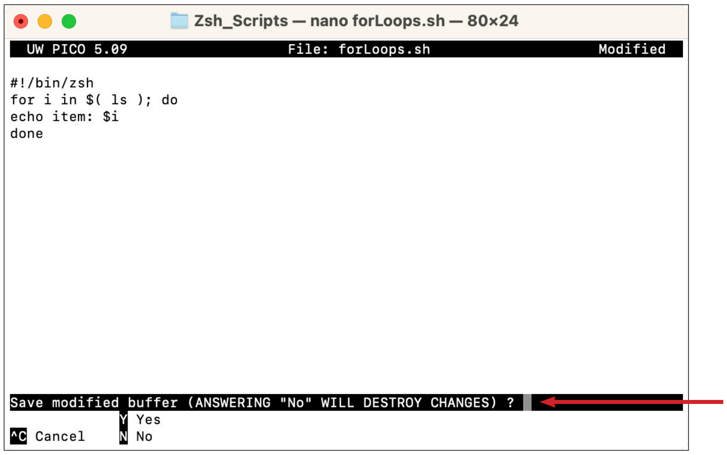
2. In the blank forLoop.sh document, enter the following:

```
#!/bin/zsh
for i in $( ls ); do
echo item: $i
done
```



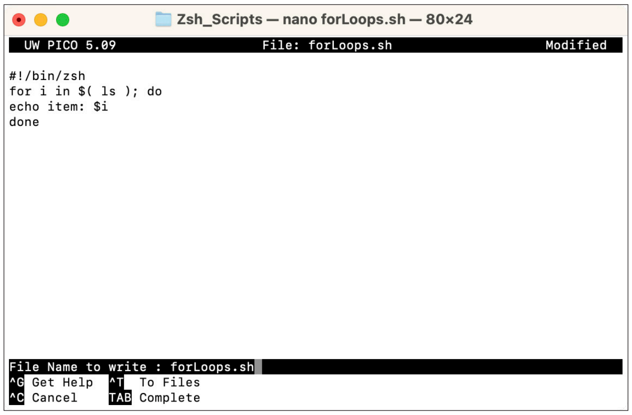
```
  UW PICO 5.09                    File: forLoops.sh                    Modified

#!/bin/zsh
for i in $( ls ); do
echo item: $i
done









^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

3. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                    File: forLoops.sh                    Modified

#!/bin/zsh
for i in $( ls ); do
echo item: $i
done









Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

4. To finish saving the file, press the Return key.



```
  UW PICO 5.09                    File: forLoops.sh                     Modified

#!/bin/zsh
for i in $( ls ); do
echo item: $i
done









File Name to write : forLoops.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

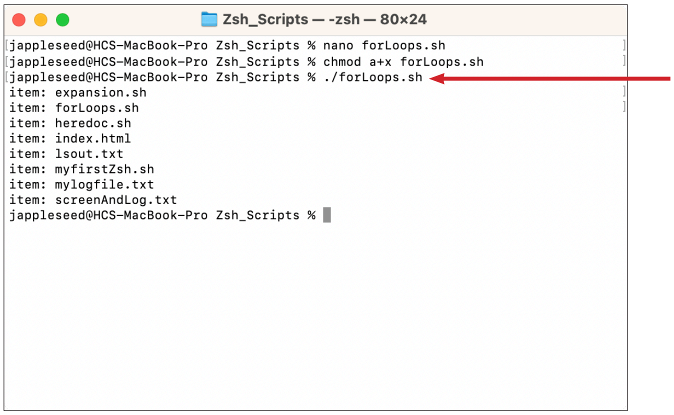
5. Enter the following command to make the script executable for all users:

```
chmod a+x forLoops.sh
```

6. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./forLoops.sh
```

You will see the output of the ls command formatted line by line. You just created your first for loop.



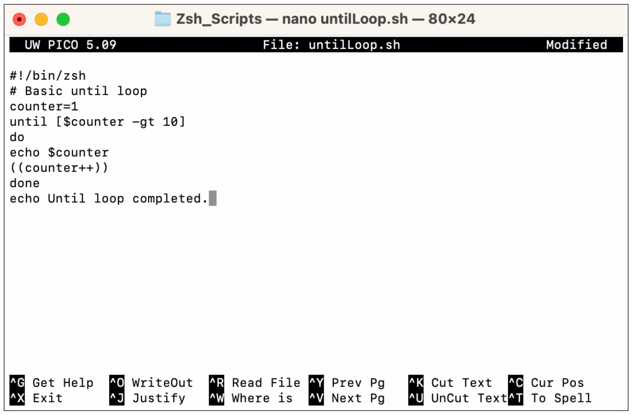
```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano forLoops.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x forLoops.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./forLoops.sh
item: expansion.sh
item: forLoops.sh
item: heredoc.sh
item: index.html
item: lsout.txt
item: myfirstZsh.sh
item: mylogfile.txt
item: screenAndLog.txt
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

7. Lets create an until loop. Enter the following command:

```
nano untilLoop.sh
```

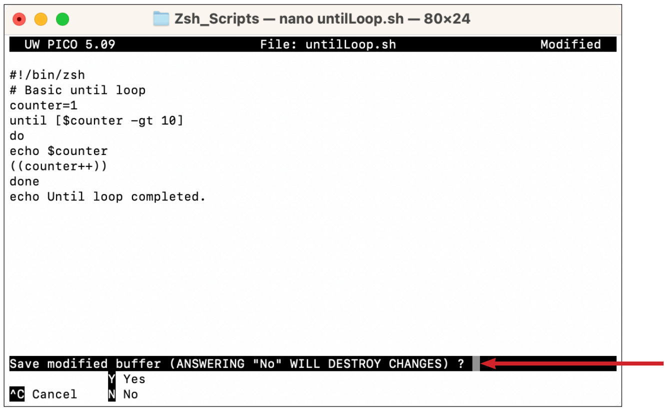
8. In the blank untilLoop.sh document, enter the following:

```
#!/bin/zsh
# Basic until loop
counter=1
until [ $counter -gt 10 ]
do
echo $counter
((counter++))
done
echo Until loop completed.
```



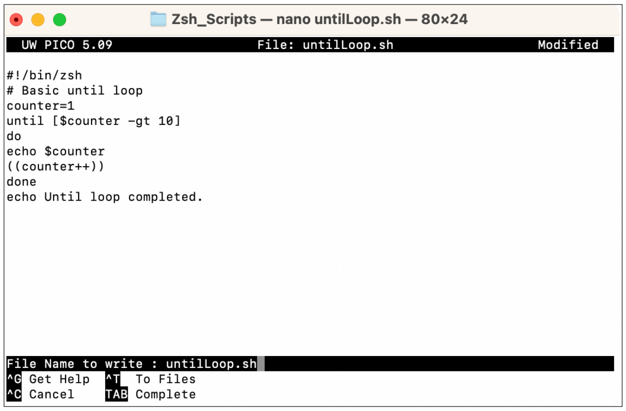
```
  UW PICO 5.09                 File: untilLoop.sh                      Modified

#!/bin/zsh
# Basic until loop
counter=1
until [$counter -gt 10]
do
echo $counter
((counter++))
done
echo Until loop completed.

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

9. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                 File: untilLoop.sh                      Modified

#!/bin/zsh
# Basic until loop
counter=1
until [$counter -gt 10]
do
echo $counter
((counter++))
done
echo Until loop completed.

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

10. To finish saving the file, press the Return key.



```
  UW PICO 5.09                 File: untilLoop.sh                      Modified

#!/bin/zsh
# Basic until loop
counter=1
until [$counter -gt 10]
do
echo $counter
((counter++))
done
echo Until loop completed.

File Name to write : untilLoop.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

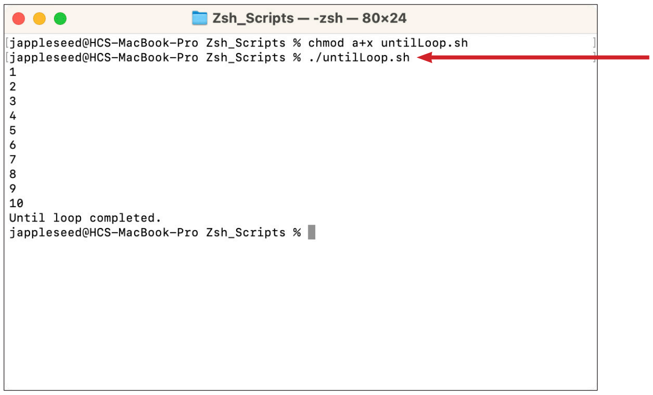
11. Enter the following command to make the script executable for all users:

```
chmod a+x untilLoop.sh
```

12. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./untilLoop.sh
```

You will see the output increment until it reaches 10 then exits. You just created your first until loop.



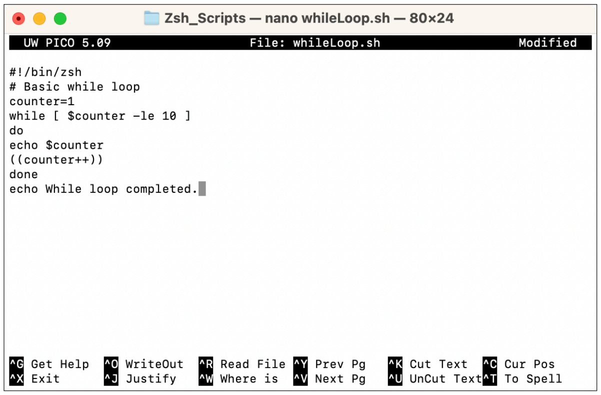
```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x untilLoop.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./untilLoop.sh
1
2
3
4
5
6
7
8
9
10
Until loop completed.
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

13. Lets create an while loop. Enter the following command:

```
nano whileLoop.sh
```

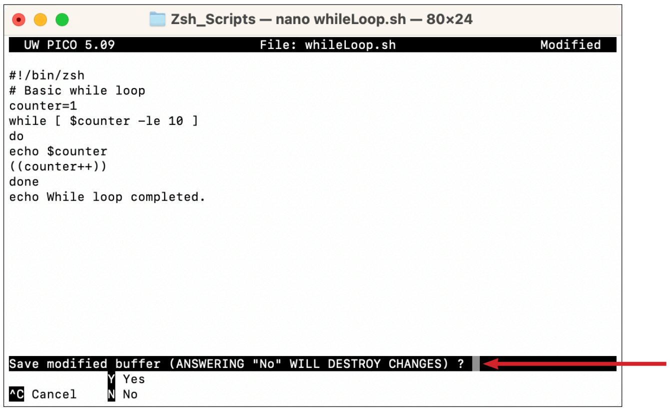
14. In the blank whileLoop.sh document, enter the following:

```
#!/bin/zsh
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo While loop completed.
```



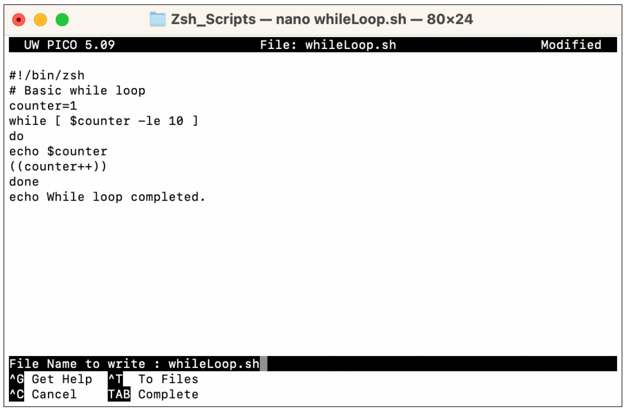
```
  UW PICO 5.09                    File: whileLoop.sh                    Modified

#!/bin/zsh
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo While loop completed.
```

15. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                     File: whileLoop.sh                    Modified

#!/bin/zsh
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo While loop completed.

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

16. To finish saving the file, press the Return key.



```
  UW PICO 5.09                     File: whileLoop.sh                    Modified

#!/bin/zsh
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo While loop completed.

File Name to write : whileLoop.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

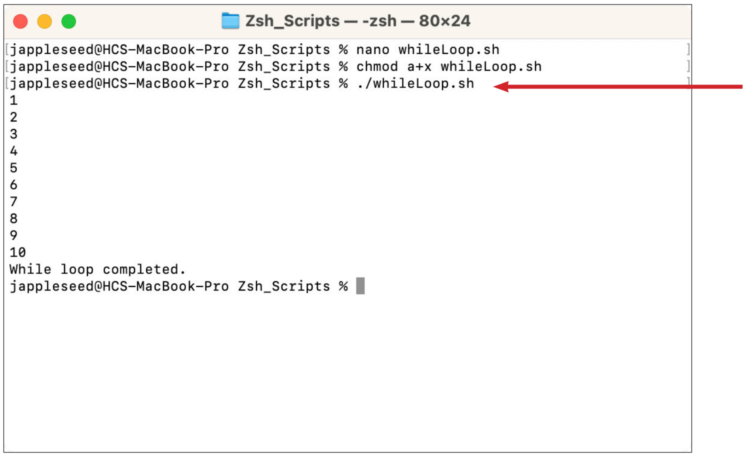
17. Enter the following command to make the script executable for all users:

```
chmod a+x whileLoop.sh
```

18. Since we are already in the directory that the script resides in, enter the following command to run the script:

    ```
    ./whileLoop.sh
    ```

    You will see the output increment until it reaches 10 then exits. This is very similar to the until loop script. You just created your first until loop.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano whileLoop.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x whileLoop.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./whileLoop.sh
1
2
3
4
5
6
7
8
9
10
While loop completed.
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

This completes this lesson.

Section 8: Conditional Statements

In this lesson we will cover the following loop types:

If Statement This statement is used when there is a need to check only conditions. If the condition founds to be true then the statement was written inside the if block will get executed.

If-Else Statement When an IF statement block gets executed and the condition is false nothing is returned or executed. If we want the program to perform certain action after the IF statement condition is false, we use the ELSE statement after the IF statement.

If-Elif-Else Statement If more than two conditions exist which can not be solved only by using IF-ELSE statement then ELIF is used. Multiple ELIF conditions can be defined inside one if-else loop.

Nested If Statement This works similar to other decision-making statements such as if, else, if..else, etc. It executes a block of code if the condition written within the if statement is true. However, in the nested-if statement, the block of code is placed inside another if block.

Case Statement The case statement simplifies complex conditions with multiple different choices. This statement is easier to maintain and more readable than nested if statements. The case statement tests the input value until it finds the corresponding pattern and executes the command linked to that input value.

Expressions used If

This is an overview of the so-called "primaries" that make up the list of commands. These primaries are put between square brackets to indicate the test of a conditional expression

Expressions	Meaning
[**-a** FILE]	True if FILE exists.
[**-b** FILE]	True if FILE exists and is a block-special file.
[**-c** FILE]	True if FILE exists and is a character-special file.
[**-d** FILE]	True if FILE exists and is a directory.
[**-e** FILE]	True if FILE exists.
[**-f** FILE]	True if FILE exists and is a regular file.
[**-g** FILE]	True if FILE exists and its SGID bit is set.
[**-h** FILE]	True if FILE exists and is a symbolic link.
[**-k** FILE]	True if FILE exists and its sticky bit is set.
[**-p** FILE]	True if FILE exists and is a named pipe (FIFO).
[**-r** FILE]	True if FILE exists and is readable.
[**-s** FILE]	True if FILE exists and has a size greater than zero.
[**-t** FD]	True if file descriptor FD is open and refers to a terminal.
[**-u** FILE]	True if FILE exists and its SUID (set user ID) bit is set.
[**-w** FILE]	True if FILE exists and is writable.
[**-x** FILE]	True if FILE exists and is executable.
[**-O** FILE]	True if FILE exists and is owned by the effective user ID.
[**-G** FILE]	True if FILE exists and is owned by the effective group ID.
[**-L** FILE]	True if FILE exists and is a symbolic link.
[**-N** FILE]	True if FILE exists and has been modified since it was last read.

Expressions	Meaning
[`-S` FILE]	True if FILE exists and is a socket.
[FILE1 `-nt` FILE2]	True if FILE1 has been changed more recently than FILE2, or if FILE1 exists and FILE2 does not.
[FILE1 `-ot` FILE2]	True if FILE1 is older than FILE2, or is FILE2 exists and FILE1 does not.
[FILE1 `-ef` FILE2]	True if FILE1 and FILE2 refer to the same device and inode numbers.
[`-o` OPTIONNAME]	True if shell option "OPTIONNAME" is enabled.
[`-z` STRING]	True of the length if "STRING" is zero.
[`-n` STRING] or [STRING]	True if the length of "STRING" is non-zero.
[STRING1 `==` STRING2]	True if the strings are equal. "=" may be used instead of "==" for strict POSIX compliance.
[STRING1 `!=` STRING2]	True if the strings are not equal.
[STRING1 `<` STRING2]	True if "STRING1" sorts before "STRING2" lexicographically in the current locale.
[STRING1 `>` STRING2]	True if "STRING1" sorts after "STRING2" lexicographically in the current locale.
[ARG1 `OP` ARG2]	"OP" is one of -eq, -ne, -lt, -le, -gt or -ge. These arithmetic binary operators return true if "ARG1" is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to "ARG2", respectively. "ARG1" and "ARG2" are integers. See Table Below.

Arithmetic operators are used for checking the arithmetic-based conditions such as less than, greater than, equals to, etc.

Operator	Description
`-eq`	Equal
`-ge`	Greater Than or Equal
`-gt`	Greater Than
`-le`	Less Than or Equal
`-lt`	Less Than
`-ne`	Not Equal

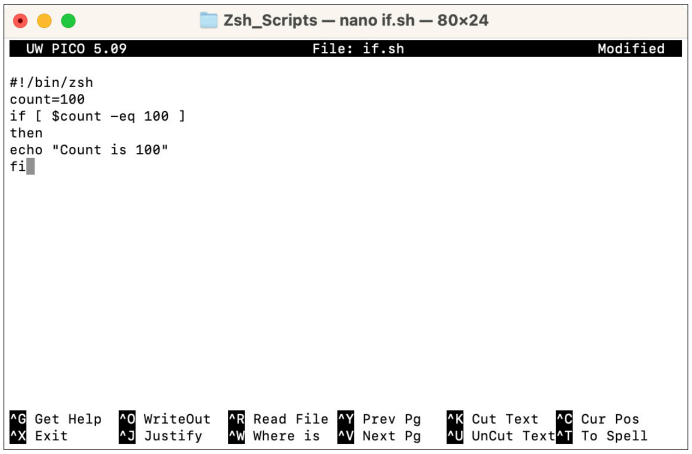
NOTE:
`[]` vs. `[[ ]]`
Contrary to `[`, `[[` prevents word splitting of variable values. So, `if VAR="var with spaces"`, you do not need to double quote `$VAR` in a test - eventhough using quotes remains a good habit. Also, `[[` prevents pathname expansion, so literal strings with wildcards do not try to expand to filenames. Using `[[`, `==` and `!=` interpret strings to the right as shell glob patterns to be matched against the value to the left, for instance: `[[ "value" == val* ]]`.

1. Lets create an If Statement. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

   nano if.sh

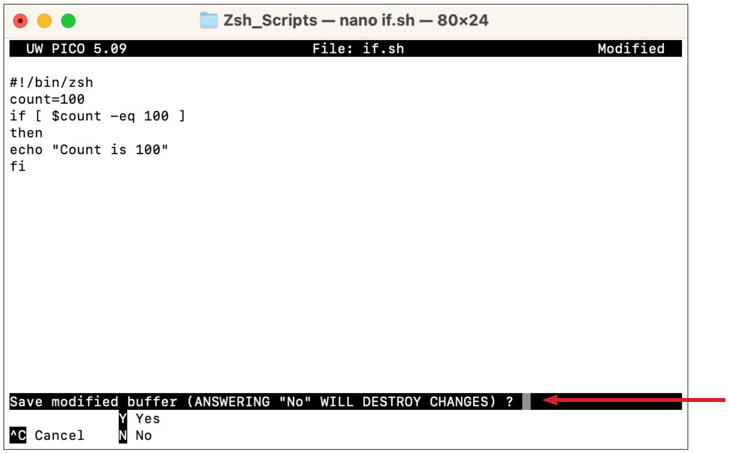
2. Enter the following:

```
#!/bin/zsh
count=100
if [ $count -eq 100 ]
then
echo "Count is 100"
fi
```



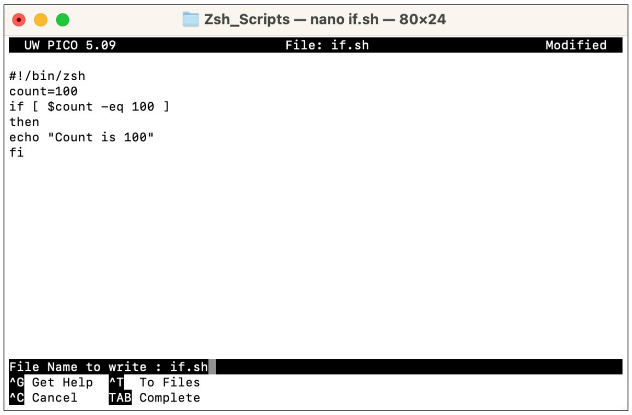
```
  UW PICO 5.09                       File: if.sh                        Modified

#!/bin/zsh
count=100
if [ $count -eq 100 ]
then
echo "Count is 100"
fi













^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

3. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                       File: if.sh                        Modified

#!/bin/zsh
count=100
if [ $count -eq 100 ]
then
echo "Count is 100"
fi














Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

4. To finish saving the file, press the Return key.



```
  UW PICO 5.09                     File: if.sh                          Modified

#!/bin/zsh
count=100
if [ $count -eq 100 ]
then
echo "Count is 100"
fi

File Name to write : if.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

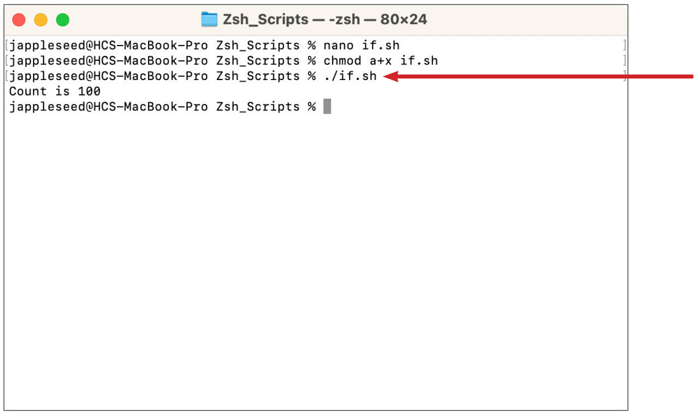
5. Enter the following command to make the script executable for all users:

```
chmod a+x if.sh
```

6. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./if.sh
```

You will see the output: Count is 100 because the count variable was set to 100. You just created your first If Statement.



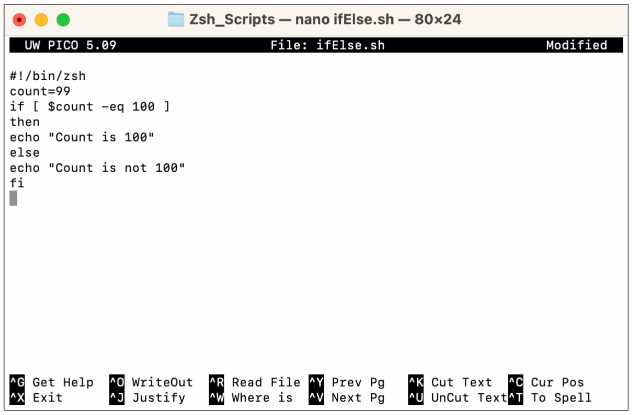
```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano if.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x if.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./if.sh
Count is 100
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

7. Lets create an If Else Statement. Enter the following command:

```
nano ifElse.sh
```

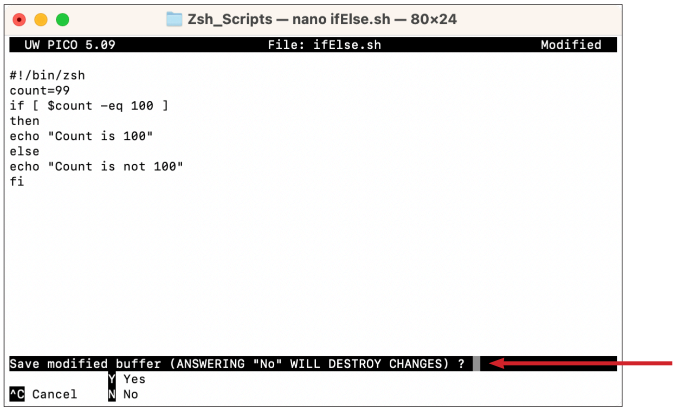
8. Enter the following:

```
#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
else
echo "Count is not 100"
fi
```



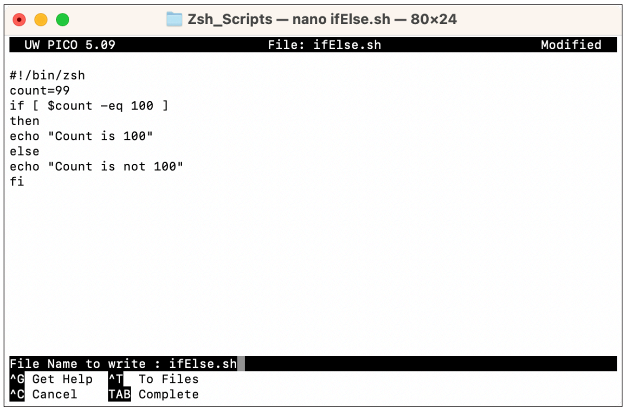
```
  UW PICO 5.09                    File: ifElse.sh                    Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
else
echo "Count is not 100"
fi
```

9. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                    File: ifElse.sh                    Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
else
echo "Count is not 100"
fi

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
```

10. To finish saving the file, press the Return key.



```
  UW PICO 5.09                    File: ifElse.sh                    Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
else
echo "Count is not 100"
fi

File Name to write : ifElse.sh
```

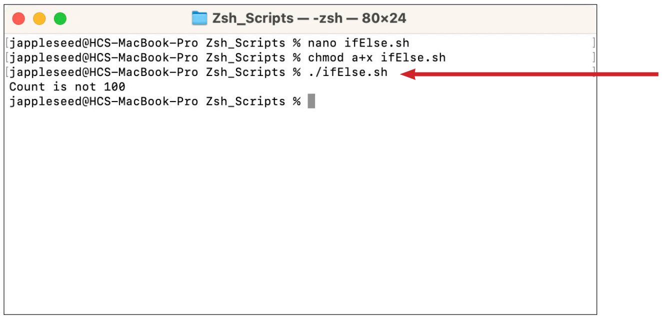
11. Enter the following command to make the script executable for all users:

    ```
    chmod a+x ifElse.sh
    ```

12. Since we are already in the directory that the script resides in, enter the following command to run the script:

    ```
    ./ifElse.sh
    ```

    You will see the output: Count is not 100 because the count variable was set to 99. You just created your first IfElse Statement.



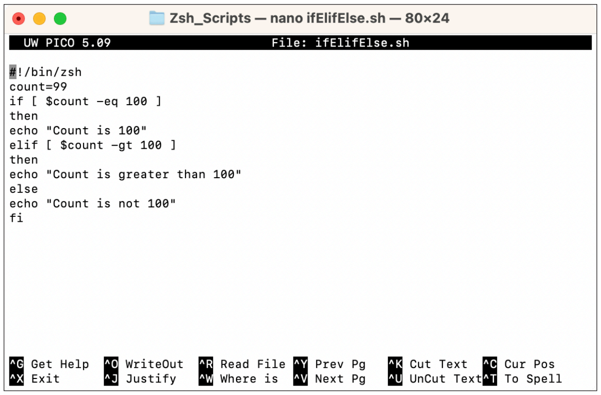
```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano ifElse.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x ifElse.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./ifElse.sh
Count is not 100
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

13. Lets create an If-Elif-Else Statement. Enter the following command:

    ```
    nano ifElifElse.sh
    ```

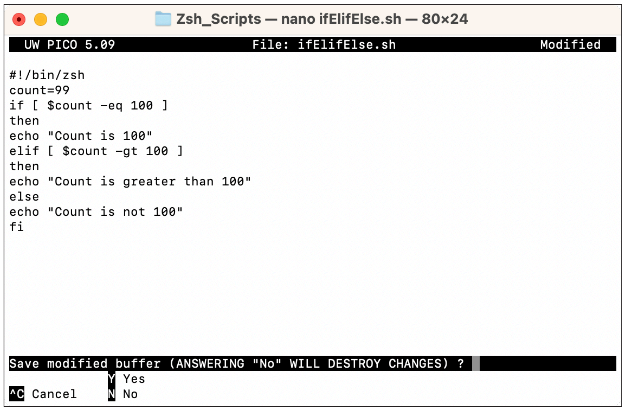
14. Enter the following:

    ```
    #!/bin/zsh
    count=99
    if [ $count -eq 100 ]
    then
    echo "Count is 100"
    elif [ $count -gt 100 ]
    then
    echo "Count is greater than 100"
    else
    echo "Count is not 100"
    fi
    ```



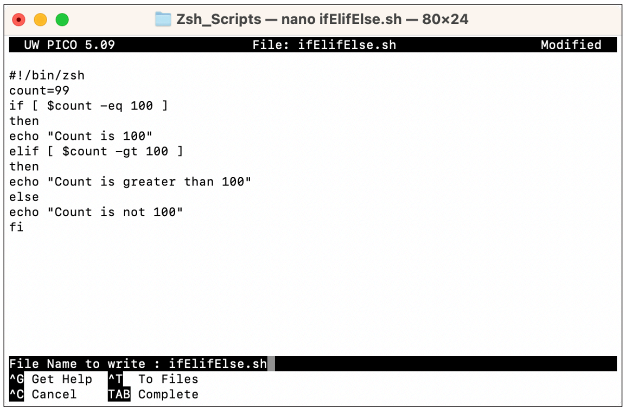
```
#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
elif [ $count -gt 100 ]
then
echo "Count is greater than 100"
else
echo "Count is not 100"
fi
```

15. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                 File: ifElifElse.sh                  Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
elif [ $count -gt 100 ]
then
echo "Count is greater than 100"
else
echo "Count is not 100"
fi







Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
              Y Yes
^C Cancel     N No
```

16. To finish saving the file, press the Return key.



```
  UW PICO 5.09                 File: ifElifElse.sh                  Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
echo "Count is 100"
elif [ $count -gt 100 ]
then
echo "Count is greater than 100"
else
echo "Count is not 100"
fi







File Name to write : ifElifElse.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

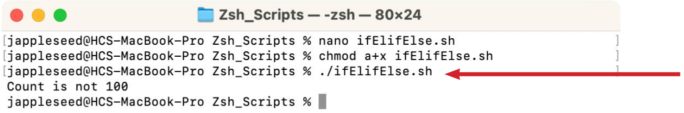
17. Enter the following command to make the script executable for all users:

```
chmod a+x ifElifElse.sh
```

18. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./ifElifElse.sh
```

You will see the output: Count is not 100 because the count variable was set to 99. You just created your first IfElse Statement.



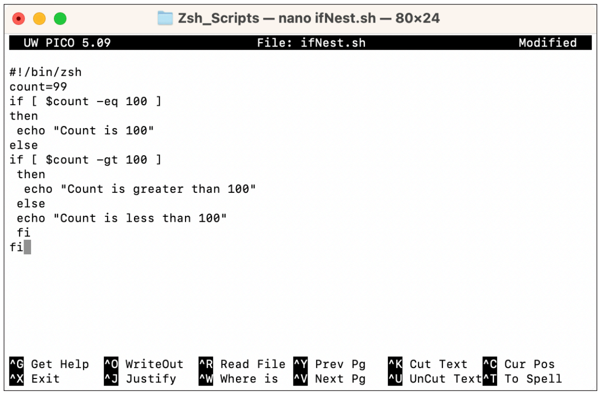
```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano ifElifElse.sh
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x ifElifElse.sh
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./ifElifElse.sh
 Count is not 100
 jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

19. Lets create a Nested If Statement. Enter the following command:

```
nano ifNest.sh
```

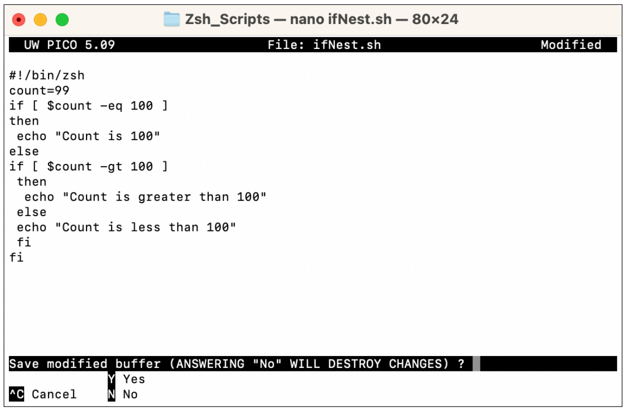
20. Enter the following:

```
#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
 echo "Count is 100"
else
 if [ $count -gt 100 ]
 then
 echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi
```



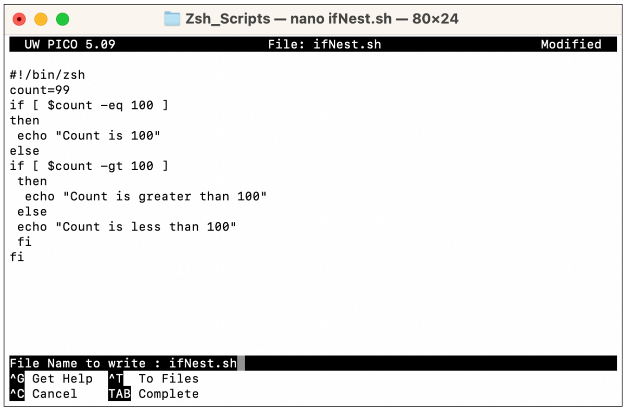
```
  UW PICO 5.09                File: ifNest.sh                     Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
 echo "Count is 100"
else
if [ $count -gt 100 ]
 then
  echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi




^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

21. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                File: ifNest.sh                     Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
 echo "Count is 100"
else
if [ $count -gt 100 ]
 then
  echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi




Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

22. To finish saving the file, press the Return key.



```
  UW PICO 5.09                     File: ifNest.sh                     Modified

#!/bin/zsh
count=99
if [ $count -eq 100 ]
then
 echo "Count is 100"
else
if [ $count -gt 100 ]
 then
  echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi

File Name to write : ifNest.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

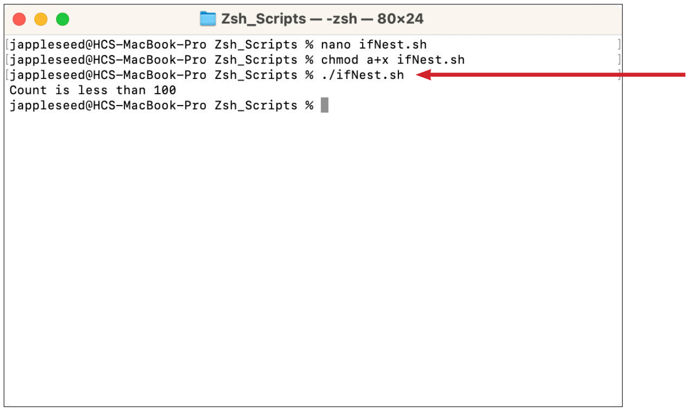
23. Enter the following command to make the script executable for all users:

```
chmod a+x ifNest.sh
```

24. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./ifNest.sh
```

You will see the output: Count is not 100 because the count variable was set to 99. You just created your first Nested If Statement.



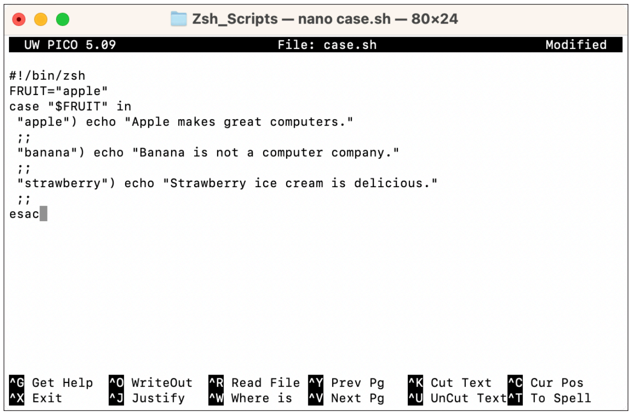
```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano ifNest.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x ifNest.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./ifNest.sh
Count is less than 100
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

25. Lets create a Case Statement. Enter the following command:

```
nano case.sh
```

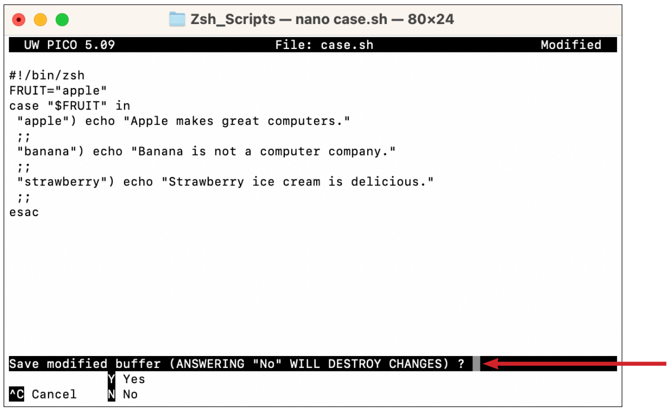
25. Enter the following:

```
#!/bin/zsh
FRUIT="apple"
case "$FRUIT" in
 "apple") echo "Apple makes great computers."
 ;;
 "banana") echo "Banana is not a computer company."
 ;;
 "strawberry") echo "Strawberry ice cream is delicious."
 ;;
esac
```



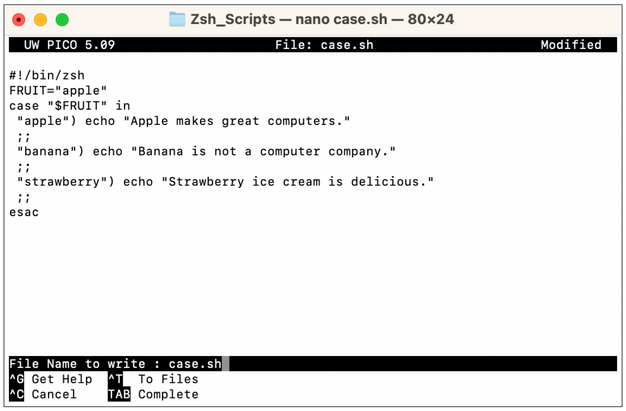
```
  UW PICO 5.09                      File: case.sh                        Modified

#!/bin/zsh
FRUIT="apple"
case "$FRUIT" in
 "apple") echo "Apple makes great computers."
 ;;
 "banana") echo "Banana is not a computer company."
 ;;
 "strawberry") echo "Strawberry ice cream is delicious."
 ;;
esac









^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

26. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
  UW PICO 5.09                      File: case.sh                        Modified

#!/bin/zsh
FRUIT="apple"
case "$FRUIT" in
 "apple") echo "Apple makes great computers."
 ;;
 "banana") echo "Banana is not a computer company."
 ;;
 "strawberry") echo "Strawberry ice cream is delicious."
 ;;
esac









Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

27. To finish saving the file, press the Return key.



```
  UW PICO 5.09                    File: case.sh                         Modified

#!/bin/zsh
FRUIT="apple"
case "$FRUIT" in
 "apple") echo "Apple makes great computers."
 ;;
 "banana") echo "Banana is not a computer company."
 ;;
 "strawberry") echo "Strawberry ice cream is delicious."
 ;;
esac

File Name to write : case.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

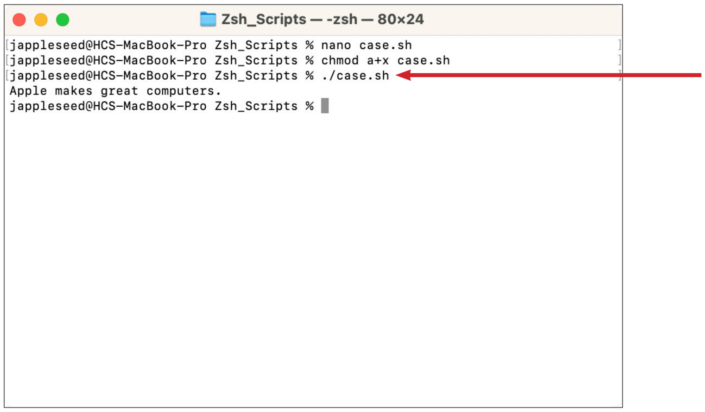
28. Enter the following command to make the script executable for all users:

```
chmod a+x case.sh
```

29. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./case.sh
```

You will see the output: Apple makes great computers because the fruit variable was set to apple. You just created your first Case Statement.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano case.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x case.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./case.sh
Apple makes great computers.
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

This completes this lesson.

Section 9: Functions

In this lesson we will cover the creations of functions with the following:

- Creating a function
- Using a function in a shell script
- Editing a function

Functions

As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion. Declaring a function is just a matter of writing function my_func () { my_code }. Calling a function is just like calling another program, you just write its name.

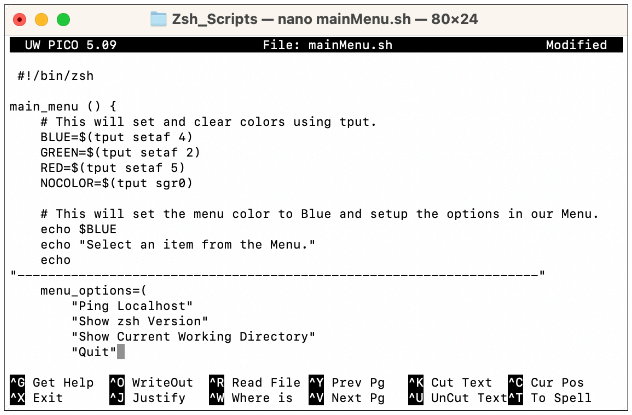
1. Lets create an If Statement. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
nano mainMenu.sh
```

2. Enter the following:

```
#!/bin/zsh
main_menu (){
  # This will set and clear colors using tput.
  BLUE=$(tput setaf 4)
  GREEN=$(tput setaf 2)
  RED=$(tput setaf 5)
  NOCOLOR=$(tput sgr0)

# This will set the menu color to Blue and setup the options in our Menu.
  echo $BLUE
  echo "Select an item from the Menu."
  echo "--------------------------------------------------------------"
  menu_options=(
      "Ping Localhost"
      "Show Zsh Version"
      "Show Current Working Directory"
      "Quit"
  )

# We will use the select command to build our menu with a case statement.
  select menu_option in "${menu_options[@]}"; do
      case $menu_option in
          "${menu_options[1]}")
              ping -c 3 127.0.0.1
              echo $GREEN
              echo "Successful ping of localhost"
              echo $NOCOLOR
              break
              ;;
          "${menu_options[2]}")
              zshVersion=$(zsh --version)
              echo $RED
              echo "Version: $zshVersion"
              echo $NOCOLOR
              break
              ;;
          "${menu_options[3]}")
              echo $GREEN
```

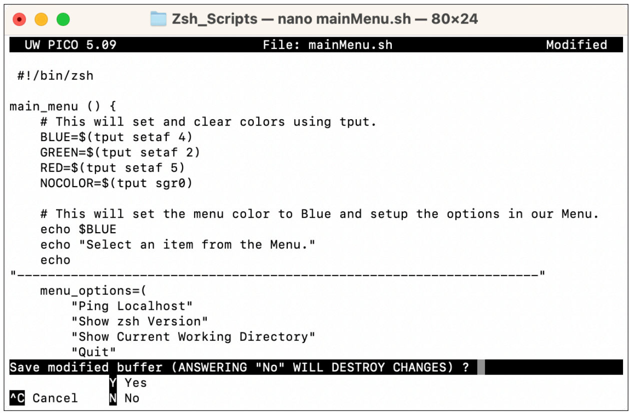
```
                echo "This is your current working directory: $PWD"
                echo $NOCOLOR
                break
                ;;
            "${menu_options[4]}")
                echo $NOCOLOR
                exit
                ;;
            *)
                echo $RED
                echo "Please enter a number between 1 and 4"
                echo $BLUE
                ;;
        esac
    done
  }
  # We will call our function to be used in this script.
  main_menu
```



Zsh_Scripts — nano mainMenu.sh — 80×24

```
  UW PICO 5.09                  File: mainMenu.sh                      Modified

 #!/bin/zsh

main_menu () {
    # This will set and clear colors using tput.
    BLUE=$(tput setaf 4)
    GREEN=$(tput setaf 2)
    RED=$(tput setaf 5)
    NOCOLOR=$(tput sgr0)

    # This will set the menu color to Blue and setup the options in our Menu.
    echo $BLUE
    echo "Select an item from the Menu."
    echo
"-------------------------------------------------------------------"
    menu_options=(
        "Ping Localhost"
        "Show zsh Version"
        "Show Current Working Directory"
        "Quit"

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

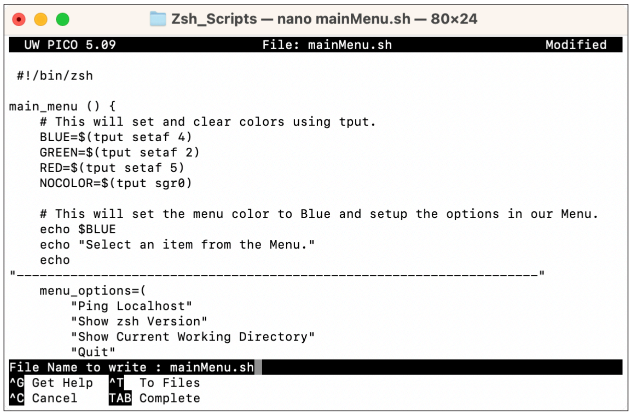
3. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



Zsh_Scripts — nano mainMenu.sh — 80×24

```
  UW PICO 5.09                  File: mainMenu.sh                      Modified

 #!/bin/zsh

main_menu () {
    # This will set and clear colors using tput.
    BLUE=$(tput setaf 4)
    GREEN=$(tput setaf 2)
    RED=$(tput setaf 5)
    NOCOLOR=$(tput sgr0)

    # This will set the menu color to Blue and setup the options in our Menu.
    echo $BLUE
    echo "Select an item from the Menu."
    echo
"-------------------------------------------------------------------"
    menu_options=(
        "Ping Localhost"
        "Show zsh Version"
        "Show Current Working Directory"
        "Quit"
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

4. To finish saving the file, press the Return key.



```
  UW PICO 5.09                     File: mainMenu.sh                    Modified

 #!/bin/zsh

main_menu () {
    # This will set and clear colors using tput.
    BLUE=$(tput setaf 4)
    GREEN=$(tput setaf 2)
    RED=$(tput setaf 5)
    NOCOLOR=$(tput sgr0)

    # This will set the menu color to Blue and setup the options in our Menu.
    echo $BLUE
    echo "Select an item from the Menu."
    echo
"--------------------------------------------------------------------"
    menu_options=(
        "Ping Localhost"
        "Show zsh Version"
        "Show Current Working Directory"
        "Quit"
File Name to write : mainMenu.sh
^G Get Help    ^T  To Files
^C Cancel      TAB Complete
```

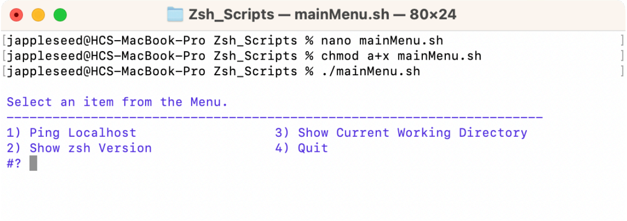
5. Enter the following command to make the script executable for all users:

```
chmod a+x mainMenu.sh
```

6. Since we are already in the directory that the script resides in, enter the following command to run the script:

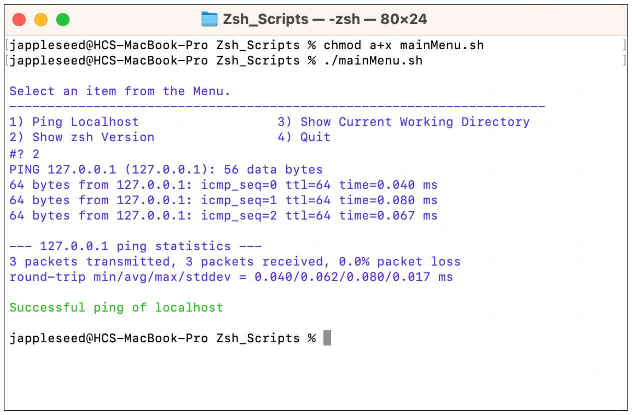
```
./mainMenu.sh
```

7. You will be presented with a menu that has 4 items in Blue.



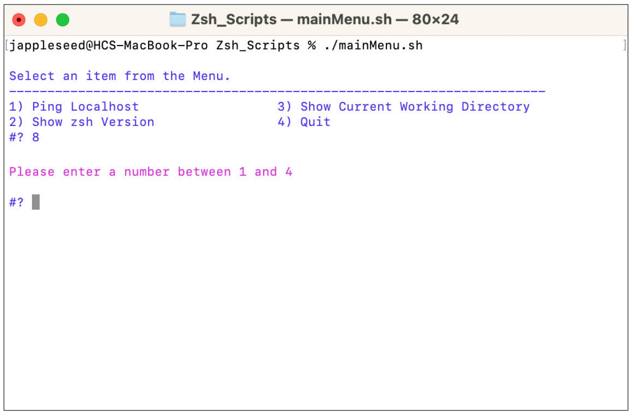
```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano mainMenu.sh                      ]
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x mainMenu.sh                 ]
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./mainMenu.sh                         ]

Select an item from the Menu.
----------------------------------------------------------------------
1) Ping Localhost                  3) Show Current Working Directory
2) Show zsh Version                4) Quit
#?
```

8. Select item 1. A ping of your localhost will begin. When completed, "Successful ping of localhost" will be in Green. This is happening because we used the tput commands to setup our color variables. Notice once the ping exists successfully, our command prompt changes back to Black. This is due to the NOCOLOR variable that we configured using tput.



```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x mainMenu.sh                 ]
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./mainMenu.sh                         ]

Select an item from the Menu.
----------------------------------------------------------------------
1) Ping Localhost                  3) Show Current Working Directory
2) Show zsh Version                4) Quit
#? 2
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.080 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.067 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.040/0.062/0.080/0.017 ms

Successful ping of localhost

jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

9. Press the up arrow on the keyboard. You should see: ./mainMenu.sh Press the enter key. At the menu, enter the number 8. Since 8 is not an item in the menu, you are greeted with a prompt to enter in a number between 1 and 4. Enter number 4 to exit the menu. You just created your first function.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./mainMenu.sh

Select an item from the Menu.
----------------------------------------------------------------------
1) Ping Localhost                  3) Show Current Working Directory
2) Show zsh Version                4) Quit
#? 8

Please enter a number between 1 and 4

#? 
```

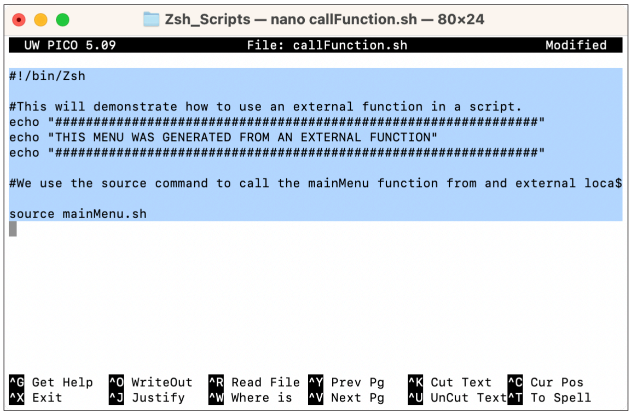
The great thing about functions is you only have to write the code once. It can be used over and over and can even be used by other scripts.

10. Lets create a new script and use our menu function in our new script. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
nano callFunction.sh
```

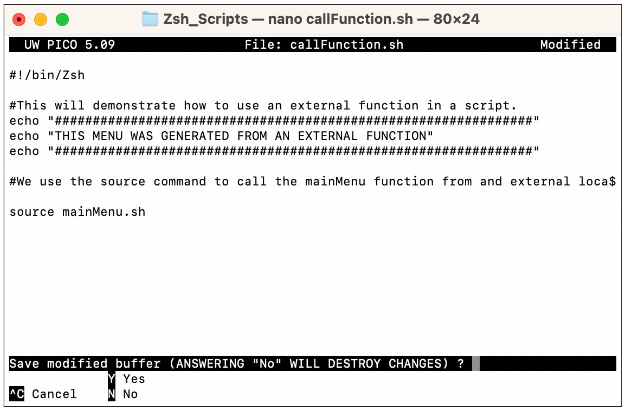
11. In the blank callFunction.sh document, enter the following:

```
#!/bin/zsh

#This will demonstrate how to use an external function in a script.
echo "#################################################################"
echo "THIS MENU WAS GENERATED FROM AN EXTERNAL FUNCTION"
echo "#################################################################"

#We use the source command to call the mainMenu function from and external
location.

source mainMenu.sh
```



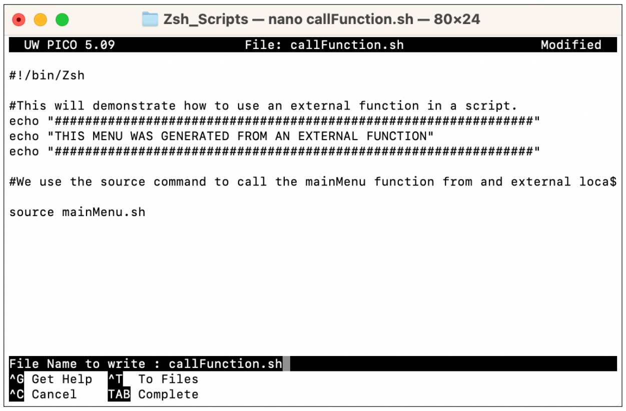
```
  UW PICO 5.09                 File: callFunction.sh                 Modified

#!/bin/Zsh

#This will demonstrate how to use an external function in a script.
echo "#################################################################"
echo "THIS MENU WAS GENERATED FROM AN EXTERNAL FUNCTION"
echo "#################################################################"

#We use the source command to call the mainMenu function from and external loca$

source mainMenu.sh


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

12. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.



```
Zsh_Scripts — nano callFunction.sh — 80×24
  UW PICO 5.09                  File: callFunction.sh                   Modified

#!/bin/Zsh

#This will demonstrate how to use an external function in a script.
echo "###############################################################"
echo "THIS MENU WAS GENERATED FROM AN EXTERNAL FUNCTION"
echo "###############################################################"

#We use the source command to call the mainMenu function from and external loca$

source mainMenu.sh









Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

13. To finish saving the file, press the Return key.



```
Zsh_Scripts — nano callFunction.sh — 80×24
  UW PICO 5.09                  File: callFunction.sh                   Modified

#!/bin/Zsh

#This will demonstrate how to use an external function in a script.
echo "###############################################################"
echo "THIS MENU WAS GENERATED FROM AN EXTERNAL FUNCTION"
echo "###############################################################"

#We use the source command to call the mainMenu function from and external loca$

source mainMenu.sh









File Name to write : callFunction.sh
^G Get Help  ^T  To Files
^C Cancel    TAB Complete
```

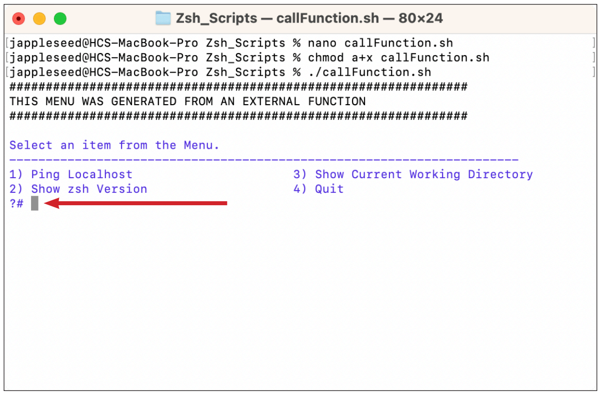
14. Enter the following command to make the script executable for all users:

```
chmod a+x callFunction.sh
```

15. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
./callFunction.sh
```

16. You will be presented with the menu function that we created earlier. Select item 4 to quit. You just created your External function script.



Zsh_Scripts — callFunction.sh — 80×24

```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano callFunction.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x callFunction.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./callFunction.sh
###############################################################
THIS MENU WAS GENERATED FROM AN EXTERNAL FUNCTION
###############################################################

Select an item from the Menu.
----------------------------------------------------------------------
1) Ping Localhost                      3) Show Current Working Directory
2) Show zsh Version                    4) Quit
?# 
```

Functions are easy to edit. In our mainMenu function, we had item one doing a Ping of our local host. This is not really useful so I'd like to add a Ping Scanner to see which of our hosts are on and offline. For this, we will use a for loop inside our case function for item 1.

17. For safe keeping, Let's copy our mainMenu.sh function to a new file so we can edit it without worry of destroying the original function. Enter the following command:

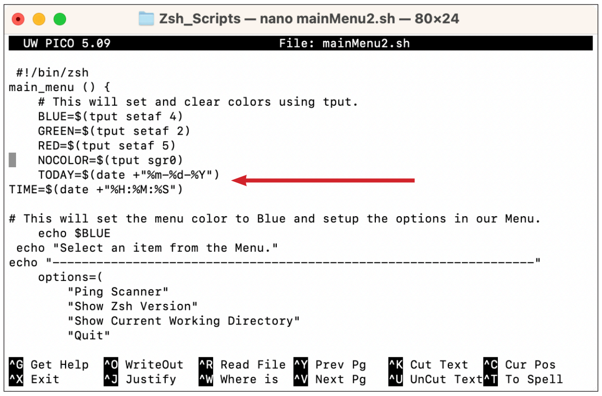
```
cp mainMenu.sh mainMenu2.sh
```

18. Open the mainMenu2.sh file with nano.

```
nano mainMenu2.sh
```

19. Let's edit the function. In the variable section for color at the top, add this:

```
TODAY=$(date +"%m-%d-%Y")
TIME=$(date +"%H:%M:%S")
```



Zsh_Scripts — nano mainMenu2.sh — 80×24

```
  UW PICO 5.09                             File: mainMenu2.sh

 #!/bin/zsh
main_menu () {
    # This will set and clear colors using tput.
    BLUE=$(tput setaf 4)
    GREEN=$(tput setaf 2)
    RED=$(tput setaf 5)
    NOCOLOR=$(tput sgr0)
    TODAY=$(date +"%m-%d-%Y")
TIME=$(date +"%H:%M:%S")

# This will set the menu color to Blue and setup the options in our Menu.
    echo $BLUE
 echo "Select an item from the Menu."
echo "------------------------------------------------------------------"
    options=(
        "Ping Scanner"
        "Show Zsh Version"
        "Show Current Working Directory"
        "Quit"

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

20. In the Options section, change the name from Ping localhost to Ping Scanner.

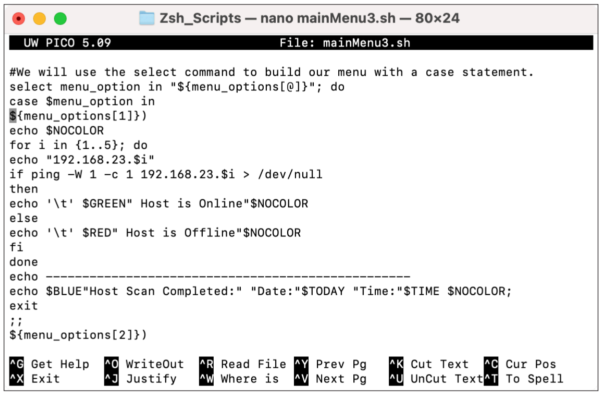
```
menu_options=(
"Ping Scanner"
"Show Zsh Version"
"Show Current Working Directory"
"Quit"
```

21. In the Case section of the function, for option 0 change it from this:
NOTE: Change the ip range below to match what you use in your environment.

```
${menu_options[1]})
ping -c 3 127.0.0.1; echo $GREEN; echo "Successful ping of localhost";
echo
$NOCOLOR
break
```

To this:

```
${menu_options[1]})
echo $NOCOLOR
for i in {1..5}; do
echo "192.168.110.$i"
if ping -W 1 -c 1 192.168.110.$i > /dev/null
then
echo '\t' $GREEN" Host is Online"$NOCOLOR
else
echo '\t' $RED" Host is Offline"$NOCOLOR
fi
done
echo --------------------------------------------------
echo $BLUE"Host Scan Completed:" "Date:"$TODAY "Time:"$TIME $NOCOLOR;
exit
```



```
  UW PICO 5.09                    File: mainMenu3.sh

#We will use the select command to build our menu with a case statement.
select menu_option in "${menu_options[@]}"; do
case $menu_option in
${menu_options[1]})
echo $NOCOLOR
for i in {1..5}; do
echo "192.168.23.$i"
if ping -W 1 -c 1 192.168.23.$i > /dev/null
then
echo '\t' $GREEN" Host is Online"$NOCOLOR
else
echo '\t' $RED" Host is Offline"$NOCOLOR
fi
done
echo --------------------------------------------------
echo $BLUE"Host Scan Completed:" "Date:"$TODAY "Time:"$TIME $NOCOLOR;
exit
;;
${menu_options[2]})

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

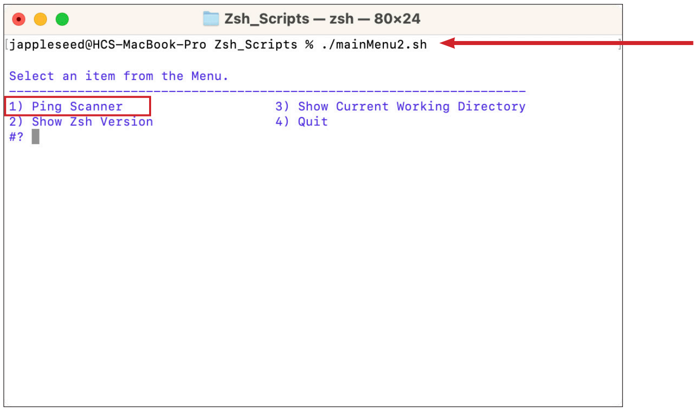
22. The final function should look like this: NOTE: IP range will be based on your environment.

```
#!/bin/zsh
main_menu (){
#This will set and clear colors using tput.
BLUE=$(tput setaf 4)
GREEN=$(tput setaf 2)
RED=$(tput setaf 5)
NOCOLOR=$(tput sgr0)
TODAY=$(date +"%m-%d-%Y")
TIME=$(date +"%H:%M:%S")

#This will set the menu color to Blue and setup the options in our Menu.
echo $BLUE
echo "Select an item from the Menu."
echo "-------------------------------------------------"
menu_options=(
"Ping Scanner"
"Show Zsh Version"
"Show Current Working Directory"
"Quit"
)
#We will use the select command to build our menu with a case statement.
select menu_option in "${menu_options[@]}"; do
case $menu_option in
${menu_options[1]})
echo $NOCOLOR
for i in {1..5}; do
echo "192.168.23.$i"
if ping -W 1 -c 1 192.168.23.$i > /dev/null
then
echo '\t' $GREEN" Host is Online"$NOCOLOR
else
echo '\t' $RED" Host is Offline"$NOCOLOR
fi
done
echo --------------------------------------------------
echo $BLUE"Host Scan Completed:" "Date:"$TODAY "Time:"$TIME $NOCOLOR;
exit
;;
${menu_options[2]})
echo $RED; echo "You are using Zsh version: $ZSH_VERSION"; echo
$NOCOLOR
break
;;
${menu_options[3]})
echo $GREEN; echo "This is your current working directory: $PWD"; echo
$NOCOLOR
break
;;
${menu_options[4]})
echo $NOCOLOR; exit
;;
*)
echo $RED; echo "Please enter between number 1 and 4"; echo $BLUE
;;
esac done
}
#We will call our function to be used in this script.
main_menu
```

23. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.
24. To finish saving the file, press the Return key.
25. Lets run our edited function. Enter the following command:

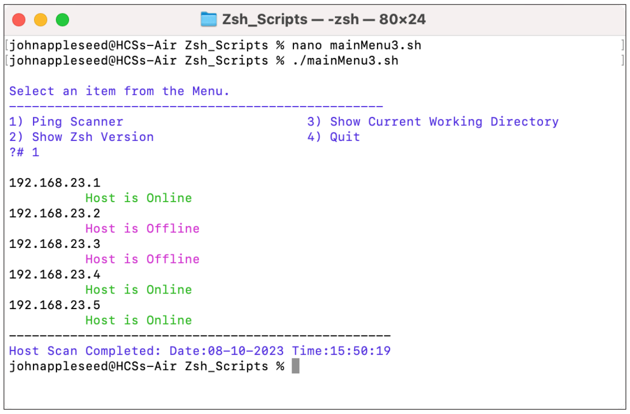
```
./mainMenu2.sh
```

Notice that item 1 is now named Ping Scanner.



```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./mainMenu2.sh

Select an item from the Menu.
---------------------------------------------------------------------
1) Ping Scanner                    3) Show Current Working Directory
2) Show Zsh Version                4) Quit
#? 
```

26. Enter 1. A ping scan of the subnet that you defined will begin. Once done, it will be date and time stamped.



```
[johnappleseed@HCSs-Air Zsh_Scripts % nano mainMenu3.sh
[johnappleseed@HCSs-Air Zsh_Scripts % ./mainMenu3.sh

Select an item from the Menu.
-------------------------------------------------
1) Ping Scanner                        3) Show Current Working Directory
2) Show Zsh Version                    4) Quit
?# 1

192.168.23.1
          Host is Online
192.168.23.2
          Host is Offline
192.168.23.3
          Host is Offline
192.168.23.4
          Host is Online
192.168.23.5
          Host is Online
--------------------------------------------------
Host Scan Completed: Date:08-10-2023 Time:15:50:19
johnappleseed@HCSs-Air Zsh_Scripts % 
```

You just edited a function.

This completes the lesson.

Section 10: Troubleshooting

In this lesson we will cover some troubleshooting basics. When Zsh runs with the x option turned on, it prints out every command it executes before executing it (to standard error). That is, after any expansions have been applied. As a result, you can see exactly what's happening as each line in your code is executed. Pay very close attention to the quoting used. Zsh uses quotes to show you exactly which strings are passed as a single argument.

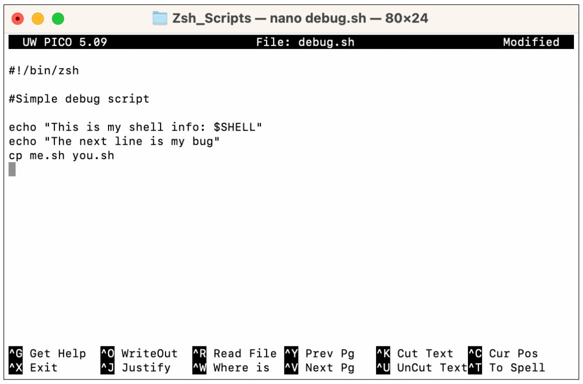
There are three ways of turning on this mode.
1. Call the script with the -x option. Example: **`zsh -x myscript`**.
2. Modify your scripts header to include -x. Example: **`#!/bin/zsh -x`**.
3. Turn on debugging in certain parts of your code. Example: **`#!/usr/bin/env zsh set - x`**.

1. If not already open, Launch Terminal and navigate to the Zsh_Scripts folder. Enter the following command:

```
nano debug.sh
```

2. In the blank debug.sh document, enter the following:

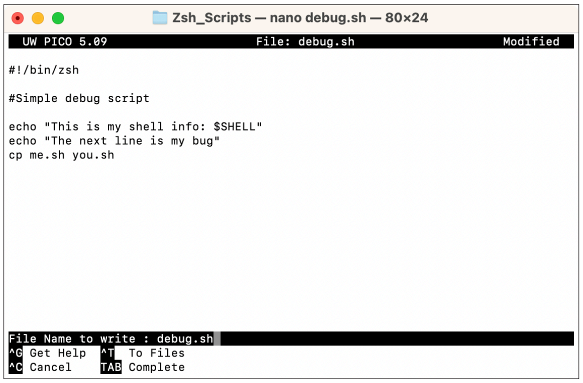
```
#!/bin/zsh

#Simple debug script
echo "This is my shell info: $SHELL"
echo "The next line is my bug"
cp me.sh you.sh
```



```
Zsh_Scripts — nano debug.sh — 80×24
  UW PICO 5.09                       File: debug.sh                     Modified

#!/bin/zsh

#Simple debug script

echo "This is my shell info: $SHELL"
echo "The next line is my bug"
cp me.sh you.sh


^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Pg    ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where is   ^V Next Pg    ^U UnCut Text ^T To Spell
```

3. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.

4. To finish saving the file, press the Return key.



```
Zsh_Scripts — nano debug.sh — 80×24
  UW PICO 5.09                       File: debug.sh                     Modified

#!/bin/zsh

#Simple debug script

echo "This is my shell info: $SHELL"
echo "The next line is my bug"
cp me.sh you.sh


File Name to write : debug.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

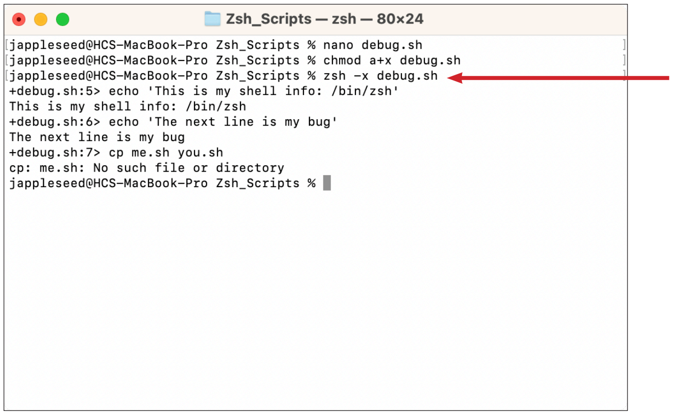
5. Enter the following command to make the script executable for all users:

```
chmod a+x debug.sh
```

6. Since we are already in the directory that the script resides in, enter the following command to run the script:

```
zsh -x debug.sh
```

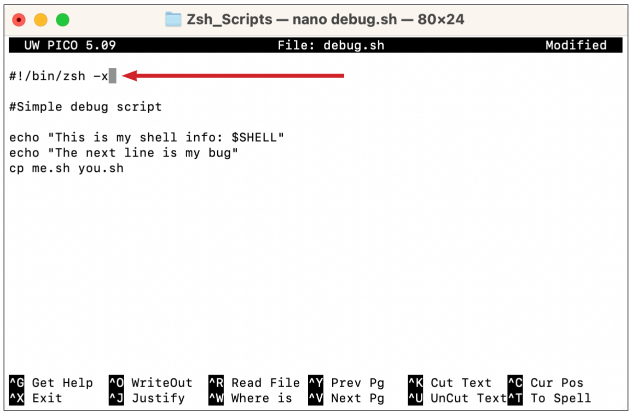
Each command in the script will be printed to the screen with a + sign next to it and the results of the command. If there was an error, it will be displayed to the screen as well.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano debug.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % chmod a+x debug.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % zsh -x debug.sh
+debug.sh:5> echo 'This is my shell info: /bin/zsh'
This is my shell info: /bin/zsh
+debug.sh:6> echo 'The next line is my bug'
The next line is my bug
+debug.sh:7> cp me.sh you.sh
cp: me.sh: No such file or directory
jappleseed@HCS-MacBook-Pro Zsh_Scripts %
```

7. Edit the scripts header so it always runs in debug mode. Using nano, open the debug.sh. Add the following line to the header:

```
-x
```



```
  UW PICO 5.09                    File: debug.sh                      Modified

#!/bin/zsh -x

#Simple debug script

echo "This is my shell info: $SHELL"
echo "The next line is my bug"
cp me.sh you.sh
```

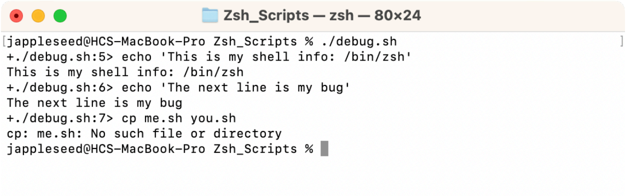
8. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.

9. To finish saving the file, press the Return key.

10. Since we are already in the directory that the script resides in, enter the following command to run the script:

    ```
    ./debug.sh
    ```

    Again, Each command in the script will be printed to the screen with a + sign next to it and the results of the command. If there was an error, it will be displayed to the screen as well.



```
[jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./debug.sh                              ]
+./debug.sh:5> echo 'This is my shell info: /bin/zsh'
This is my shell info: /bin/zsh
+./debug.sh:6> echo 'The next line is my bug'
The next line is my bug
+./debug.sh:7> cp me.sh you.sh
cp: me.sh: No such file or directory
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

11. Using nano, open the debug.sh. Remove the following line from the header:

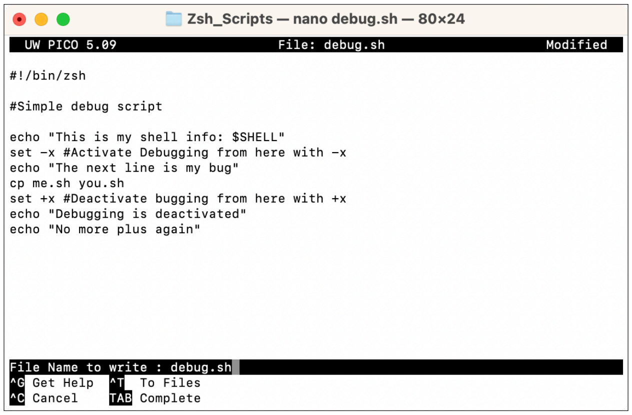
    ```
    -x
    ```

    We will use set -x and set +x to debug a section of the script. When debugging starts, you will see a + sign before each command, when debugging ends, the plus sign will be gone before each command. This is useful when you want to debug a section of your script. Adjust your script so it looks like the following:

    ```
    #!/bin/zsh
    #Simple debug script
    echo "This is my shell info: $SHELL"
    set -x #Activate Debugging from here with -x
    echo "The next line is my bug"
    cp me.sh you.sh
    set +x #Deactivate bugging from here with +x echo "Debugging is
    deactivated"
    echo "No more plus sign"
    ```

12. Save the script by pressing the following keys: Control and x. This will prompt you to save the file. Enter Y for yes.

13. To finish saving the file, press the Return key.

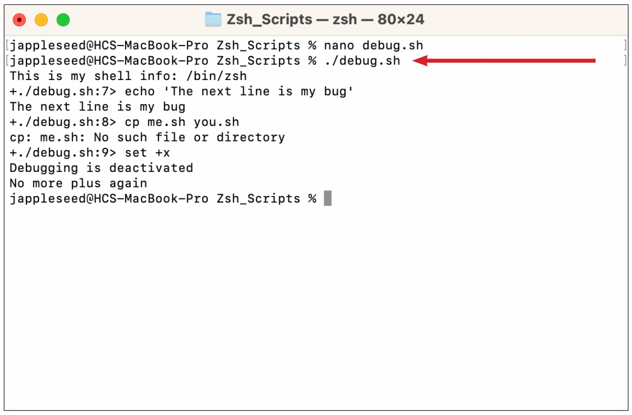


```
  UW PICO 5.09                     File: debug.sh                      Modified

#!/bin/zsh

#Simple debug script

echo "This is my shell info: $SHELL"
set -x #Activate Debugging from here with -x
echo "The next line is my bug"
cp me.sh you.sh
set +x #Deactivate bugging from here with +x
echo "Debugging is deactivated"
echo "No more plus again"




File Name to write : debug.sh
^G Get Help   ^T  To Files
^C Cancel     TAB Complete
```

14. Run the script with the following command:

    ```
    ./debug.sh
    ```

    You will see the following output. Notice the + sign starts at the echo command and ends after the set +x command is ran.



```
jappleseed@HCS-MacBook-Pro Zsh_Scripts % nano debug.sh
jappleseed@HCS-MacBook-Pro Zsh_Scripts % ./debug.sh
This is my shell info: /bin/zsh
+./debug.sh:7> echo 'The next line is my bug'
The next line is my bug
+./debug.sh:8> cp me.sh you.sh
cp: me.sh: No such file or directory
+./debug.sh:9> set +x
Debugging is deactivated
No more plus again
jappleseed@HCS-MacBook-Pro Zsh_Scripts % 
```

This completes the guide.