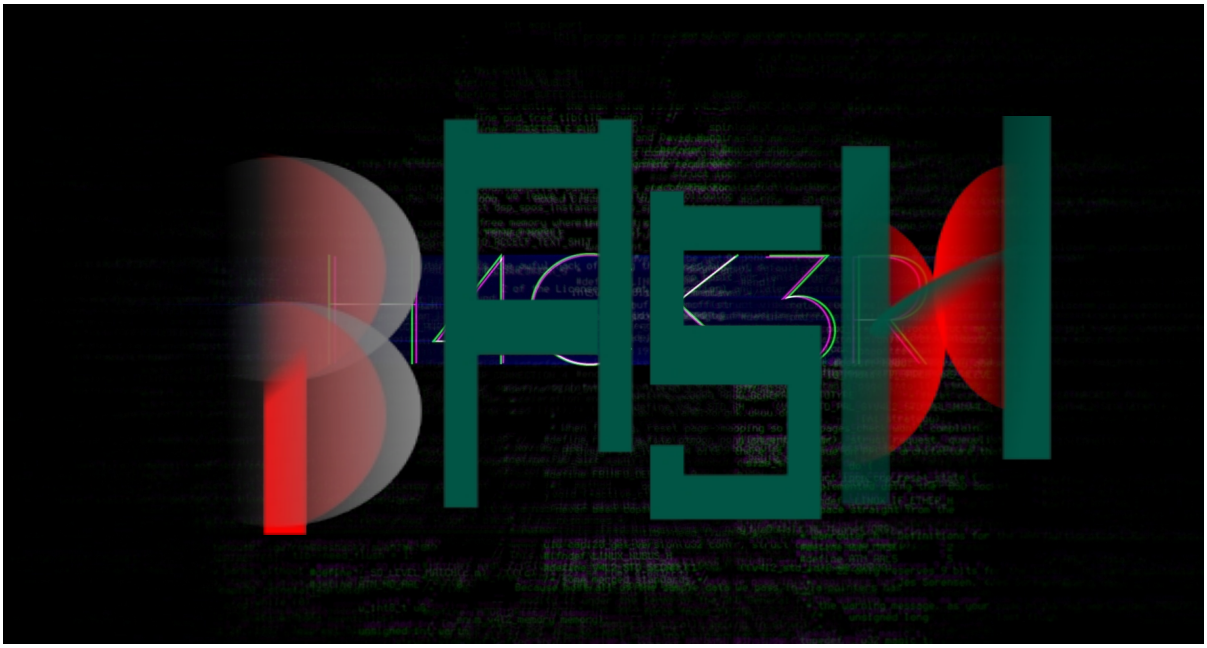
###Bash Scripting###

25-8-2023

Pen-tester

Edited by CS-NINJA

Uttora university BD,Dhaka



BEGINNING

1. **Shebang**: This is the first line in a Bash script that specifies the interpreter to be used.

```
bashCopy code
#!/bin/bash
```

2. **Variables**: Variables hold values. Bash variables are untyped.

```
bashCopy code
name="John"
```

```
age=30
```

3. **User Input**: Use `read` to get user input.

```
bashCopy code
echo "Enter your name: "
read name
echo "Hello, $name!"
```

4. **Conditional Statements**:

- **if-else**:

```
bashCopy code
if [ $age -ge 18 ]; then
    echo "You are an adult."
else
    echo "You are a minor."
fi
```

- **case**:

```
bashCopy code
case $option in
    "start")
        echo "Starting the process."
        ;;
    "stop")
        echo "Stopping the process."
        ;;
    *)
        echo "Unknown option."
        ;;
esac
```

5. **Loops**:

- **for loop**:

```
bashCopy code
for i in {1..5}; do
    echo "Number: $i"
done
```

- **while loop**:

```
bashCopy code
count=0
while [ $count -lt 5 ]; do
    echo "Count: $count"
```

```
    ((count++))
done
```

6. **Functions**:

```
bashCopy code
greet() {
    echo "Hello, $1!"
}

greet "Alice"
```

7. **Command Substitution**: Running a command and using its output.

```
bashCopy code
files=$(ls)
echo "Files: $files"
```

8. **Arrays**:

```
bashCopy code
fruits=("apple" "banana" "orange")
echo "First fruit: ${fruits[0]}"
echo "All fruits: ${fruits[@]}"
```

9. **Exit Status**: Checking the success or failure of a command.

```
bashCopy code
ls /nonexistent-folder
if [ $? -ne 0 ]; then
    echo "Error: Folder does not exist."
fi
```

10. **File Operations**:

- **File Existence**:

```
bashCopy code
if [ -e "file.txt" ]; then
    echo "File exists."
else
    echo "File does not exist."
fi
```

- **File Permissions**:

```
bashCopy code
if [ -w "file.txt" ]; then
    echo "File is writable."
```

```
fi
```

LITTLE MORE

1. **Command Line Arguments**:

```
bashCopy code
#!/bin/bash

echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
echo "All arguments: $@"
```

2. **Redirecting Input/Output**:

- Redirecting output to a file:

```
bashCopy code
echo "Hello, world!" > output.txt
```

- Appending output to a file:

```
bashCopy code
echo "Appended line" >> output.txt
```

- Redirecting input from a file:

```
bashCopy code
while read line; do
    echo "Line: $line"
done < input.txt
```

3. **Pipes**:

Pipes allow you to use the output of one command as the input of another.

```
bashCopy code
ls -l | grep "file"
```

4. **Functions with Return Values**:

```
bashCopy code
calculate_sum() {
    local sum=$(( $1 + $2 ))
    echo $sum
```

```
}

result=$(calculate_sum 10 20)
echo "Sum: $result"
```

5. **String Manipulation**:

```
bashCopy code
str="Hello, world!"

# Length of the string
length=${#str}
echo "Length: $length"

# Substring extraction
substring=${str:7:5}
echo "Substring: $substring"

# Replace a substring
new_str=${str//world/there}
echo "New string: $new_str"
```

6. **Arithmetic Operations**:

```
bashCopy code
num1=10
num2=5

sum=$((num1 + num2))
echo "Sum: $sum"

# Using bc for floating point calculations
result=$(echo "scale=2; $num1 / $num2" | bc)
echo "Result: $result"
```

7. **Arrays**:

- Associative arrays:

```
bashCopy code
declare -A fruits
fruits["apple"]="red"
fruits["banana"]="yellow"

echo "Color of apple: ${fruits["apple"]}"
```

8. **Error Handling**:

```
bashCopy code
#!/bin/bash
```

```
if ! command_exists "some_command"; then
    echo "Error: some_command not found"
    exit 1
fi

# Rest of the script
```

9. **Advanced Conditional Statements**:

```
bashCopy code
# Ternary operator
age=25
status=""

status=$(($age >= 18 ? "adult" : "minor"))
echo "Status: $status"
```

10.**Debugging**:

```
bashCopy code
#!/bin/bash

set -x  # Enable debugging

# Your script here

set +x  # Disable debugging
```

These advanced concepts provide more powerful tools for creating complex and sophisticated Bash scripts. Keep in mind that Bash scripting can become quite intricate depending on the tasks you want to automate or the problems you want to solve.

DATA TYPES OF BASH:

1. **Strings**:

Strings are the most common data type in Bash. Variables hold strings by default.

```
bashCopy code
name="John"
```

Concatenation:

```
bashCopy code
greeting="Hello, "
```

```
name="John"
message="$greeting$name"
```

2. **Integers**:

Bash doesn't have native support for integers, but you can perform integer arithmetic using the `(( ))` construct.

```
bashCopy code
x=5
y=3
sum=$((x + y))
```

3. **Floating-Point Numbers**:

Bash can't handle floating-point arithmetic natively. You might use external tools like `bc` for calculations involving floating-point numbers.

```
bashCopy code
result=$(echo "scale=2; 5 / 3" | bc)
```

4. **Arrays**:

Bash supports indexed arrays (similar to lists).

```
bashCopy code
fruits=("apple" "banana" "orange")
echo "First fruit: ${fruits[0]}"
echo "All fruits: ${fruits[@]}"
```

Associative arrays (using key-value pairs):

```
bashCopy code
declare -A colors
colors["apple"]="red"
colors["banana"]="yellow"
echo "Color of apple: ${colors["apple"]}"
```

5. **Booleans**:

Bash doesn't have a dedicated boolean type, but you can use integer values to simulate boolean conditions (0 for false, 1 for true).

```
bashCopy code
is_true=1
if [ $is_true -eq 1 ]; then
    echo "It's true!"
fi
```

6. **Null or Empty**:

To represent an empty or null value, you can use an empty string or unset a variable.

```
bashCopy code
empty_string=""
unset some_variable
```

Mathematical operations in Bash:

1. **Using `(( ))` for Basic Arithmetic**:

```
bashCopy code
# Addition
result=$((5 + 3))

# Subtraction
result=$((10 - 4))

# Multiplication
result=$((6 * 2))

# Division
result=$((20 / 4))

# Modulus
result=$((15 % 7))
```

2. **Using `expr`**:

```
bashCopy code
# Addition
result=$(expr 5 + 3)

# Subtraction
result=$(expr 10 - 4)

# Multiplication
result=$(expr 6 \* 2)  # Note: * needs to be escaped

# Division
result=$(expr 20 / 4)

# Modulus
result=$(expr 15 % 7)
```

3. **Using `bc` for Floating-Point Calculations**:

```
bashCopy code
# Basic arithmetic with floating-point result
```

```
result=$(echo "5.5 + 3.2" | bc)

# Division with scale precision
result=$(echo "scale=2; 10 / 3" | bc)

# Square root
result=$(echo "scale=2; sqrt(25)" | bc)
```

4. **Using `let` for Increment/Decrement**:

```
bashCopy code
count=5
let "count+=1"  # Increment
echo "Count: $count"

let "count-=2"   # Decrement
echo "Count: $count"
```

5. **Using Arithmetic Operators in Comparison**:

```
bashCopy code
x=10
y=5

if ((x > y)); then
    echo "x is greater than y"
fi
```

Remember to consider the limitations of using arithmetic expressions in Bash:

- They don't support floating-point arithmetic natively (except for `bc`).
- For complex math or precision, consider using `bc` or other external tools.

For example, if you want to do advanced calculations like trigonometry or logarithms, you would typically use `bc`.

FILE MANAGE MENT

Bash provides a wide range of commands and techniques for managing files and directories. Here

are some common operations for file management using Bash:

1. **Creating Files and Directories**:

- Create a file:

```
bashCopy code
touch myfile.txt
```

- Create a directory:

```
bashCopy code
mkdir mydir
```

2. **Listing Files and Directories**:

- List files and directories in the current directory:

```
bashCopy code
ls
```

- List files and directories with detailed information:

```
bashCopy code
ls -l
```

3. **Copying and Moving Files**:

- Copy a file:

```
bashCopy code
cp source_file.txt destination_file.txt
```

- Copy a directory and its contents:

```
bashCopy code
cp -r sourcedir/ destinationdir/
```

- Move or rename a file:

```
bashCopy code
mv oldname.txt newname.txt
```

- Move a file to another directory:

```
bashCopy code
mv myfile.txt mydir/
```

4. **Deleting Files and Directories**:

- Delete a file:

```
bashCopy code
rm myfile.txt
```

- Delete an empty directory:

```
bashCopy code
rmdir mydir
```

- Delete a directory and its contents:

```
bashCopy code
rm -r mydir/
```

5. **Viewing File Content**:

- Display the content of a file:

```
bashCopy code
cat myfile.txt
```

- Display the content with line numbers:

```
bashCopy code
nl myfile.txt
```

6. **File Permissions**:

- Change file permissions:

```
bashCopy code
chmod permissions filename
```

7. **Searching for Files**:

- Find files by name:

```
bashCopy code
find /path/to/search -name "filename"
```

- Find files by type:

```
bashCopy code
find /path/to/search -type f
```

8. **File Compression and Archiving**:

- Compress a file using gzip:

```
bashCopy code
gzip myfile.txt
```

- Compress a directory and its contents using tar:

```
bashCopy code
tar -czvf archive.tar.gz mydir/
```

9. **File Information**:

- Get information about a file:

```
bashCopy code
stat myfile.txt
```

- Get file size:

```
bashCopy code
du -h myfile.txt
```

These are just a few examples of the file management operations you can perform using Bash. Remember to exercise caution when performing actions like deletion or moving files, as these operations are irreversible. Always make sure you have backups or are confident in your actions before proceeding

SOME ADITIONAL

System Control:

- **Shutdown**:

```
bashCopy code
sudo shutdown -h now  # Shutdown immediately
sudo shutdown -r now  # Reboot immediately
```

- **Restart**:

```
bashCopy code
sudo reboot
```

- **View System Information**:

```
bashCopy code
uname -a          # Display system information
df -h             # Display disk space usage
free -h           # Display memory usage
```

2. **Package Management**:

- **APT (Advanced Package Tool)** for Debian-based systems (Ubuntu):

```
bashCopy code
sudo apt update       # Update package lists
sudo apt upgrade      # Upgrade installed packages
sudo apt install pkg  # Install a package
sudo apt remove pkg   # Remove a package
```

- **DNF (Dandified YUM)** for Fedora-based systems:

```
bashCopy code
sudo dnf update       # Update packages
sudo dnf install pkg  # Install a package
sudo dnf remove pkg   # Remove a package
```

- **Pacman** for Arch Linux:

```
bashCopy code
sudo pacman -Syu      # Synchronize and update packages
sudo pacman -S pkg    # Install a package
sudo pacman -R pkg    # Remove a package
```

3. **Navigation within Filesystem**:

- **Change Directory**:

```
bashCopy code
cd /path/to/directory   # Change to an absolute path
cd ~                    # Change to home directory
cd ..                   # Move to the parent directory
cd directory_name       # Change to a relative directory
```

- **Working with Directories**:

```
bashCopy code
pwd                     # Print current working directory
ls                      # List files and directories
ls -l                   # List files and directories with details
ls -a                   # List all files, including hidden ones
```

- **Path Completion**:

Pressing the `Tab` key will automatically complete filenames and directory paths.

- **Wildcards**:

```
bashCopy code
*.txt                  # Matches all files with a .txt extension
file*.txt              # Matches files starting with "file" and ending
with .txt
```

- **Copying and Moving Files and Directories**:

```
bashCopy code
cp source destination  # Copy file or directory
mv source destination  # Move or rename file or directory
```

These are just a few examples of system control, package management, and navigation tasks you can perform using Bash. Remember to use these commands with caution, especially system control and package management commands, as they can have a significant impact on your system. Always double-check your commands before executing them.

More

Here are some additional commands and techniques for system control, package management, and navigation using Bash:

4. **File Permissions and Ownership**:

- **Change Permissions**:

```
bashCopy code
chmod +x script.sh      # Make a script executable
chmod 644 file.txt     # Set read and write permissions for the owner, and read
permissions for others
```

- **Change Ownership**:

```
bashCopy code
sudo chown user:group file.txt   # Change file ownership
```

5. **Text Manipulation and Viewing**:

- **Viewing Files**:

```
bashCopy code
cat file.txt          # Display the entire file
head file.txt         # Display the beginning of the file
tail file.txt         # Display the end of the file
```

- **Searching for Text**:

```
bashCopy code
grep "pattern" file.txt    # Search for a pattern in a file
grep -r "pattern" dir/     # Search for a pattern recursively in a directory
```

- **Piping and Redirection**:

```
bashCopy code
command1 | command2    # Pipe the output of command1 as input to command2
command > output.txt   # Redirect output to a file (overwrite)
command >> output.txt  # Redirect output to a file (append)
```

6. **Archiving and Compression**:

- **Extracting Archives**:

```
bashCopy code
tar -xvf archive.tar.gz    # Extract files from a tar.gz archive
unzip archive.zip          # Extract files from a zip archive
```

7. **Environment Variables**:

- **Display Environment Variables**:

```
bashCopy code
echo $HOME           # Display the home directory
echo $PATH           # Display the current PATH variable
```

- **Setting Environment Variables**:

```
bashCopy code
export MY_VAR="value"    # Set an environment variable
```

8. **Process Management**:

- **View Running Processes**:

```
bashCopy code
ps aux          # Display detailed information about running processes
```

- **Kill a Process**:

```
bashCopy code
```

```
kill process_id    # Terminate a process by its ID
```

9. **Permissions and Access Control**:

- **Change Permissions Recursively**:

```
bashCopy code
chmod -R permissions dir/    # Change permissions recursively for a directory
and its contents
```

10. **Searching for Files and Text**:

- **Find Files**:

```
bashCopy code
find /path/to/search -name "filename"    # Find files by name
```

- **Find Text in Files**:

```
bashCopy code
grep -r "pattern" /path/to/search       # Search for a pattern in files
recursively
```

11. **Network Commands**:

- **Ping**:

```
bashCopy code
ping google.com    # Send ICMP echo requests to a host
```

- **Traceroute**:

```
bashCopy code
traceroute google.com    # Display the route packets take to a host
```

12. **Background Jobs and Foreground**:

- **Run a Command in the Background**:

```
bashCopy code
command &     # Run a command in the background
```

- **Bringing a Background Job to the Foreground**:

```
bashCopy code
fg %1         # Bring the job with job number 1 to the foreground
```

These additional commands and techniques expand on system control, package management, and navigation in a Bash environment. Keep in mind that Bash provides a vast array of tools, and

mastering them can greatly enhance your ability to work efficiently within a Unix-like operating system.

Lastly

here are a few more commands and concepts that you might find useful in your journey with Bash scripting:

1. **Redirecting Input/Output**:

- Redirecting input from a file:

```
bashCopy code
command < input.txt
```

- Redirecting output to a file:

```
bashCopy code
command > output.txt
```

2. **Command Substitution**:

- Using the output of one command as an argument for another:

```
bashCopy code
result=$(ls -l)
echo "Listing: $result"
```

3. **Shell Variables**:

- Accessing command line arguments:

```
bashCopy code
echo "First argument: $1"
echo "Second argument: $2"
```

4. **Logical Operators**:

- Using logical AND (`&&`) and OR (`||`) operators:

```
bashCopy code
command1 && command2    # Execute command2 only if command1 succeeds
command1 || command2    # Execute command2 only if command1 fails
```

5. **Exit Status**:

- Using exit status to check the success of a command:

```
bashCopy code
if command; then
    echo "Command succeeded"
else
    echo "Command failed"
fi
```

6. **Aliases**:

- Creating shortcuts for commands:

```
bashCopy code
alias ll='ls -l'
```

7. **History**:

- Accessing command history:

```
bashCopy code
history                    # View command history
!!                         # Execute the last command
!n                         # Execute the nth command in history
```

8. **Running Scripts**:

- Running a script:

```
bashCopy code
chmod +x script.sh          # Make the script executable (if not already)
./script.sh                 # Run the script
```

9. **Environment Variables**:

- Accessing predefined environment variables:

```
bashCopy code
echo $USER                  # Display the username
echo $SHELL                 # Display the current shell
```

10. **Debugging**:

- Printing messages for debugging:

```
bashCopy code
set -x                       # Enable debugging
# Your script here
set +x                       # Disable debugging
```

11. **Getting Help**:

- Accessing manual pages for commands:

```
bashCopy code
man command                 # Display the manual for a command
```

12. **Scripting Practices**:

- Commenting your code for clarity:

```
bashCopy code
# This is a comment
```

- Using meaningful variable names and organizing your code.

Remember that Bash scripting is a skill that grows with practice. As you encounter new tasks and challenges, you'll naturally learn and discover new commands and techniques to accomplish them. Don't hesitate to refer to official documentation and resources whenever you encounter unfamiliar concepts or commands.

Therefore thank you very much learner....

If you intersted connect me on

fb/ahamedrashid.me

github/ahamedrashid-me

mail/ahamedrashid.me

hackerone/ahamedrashid-me

Community/whatsapp